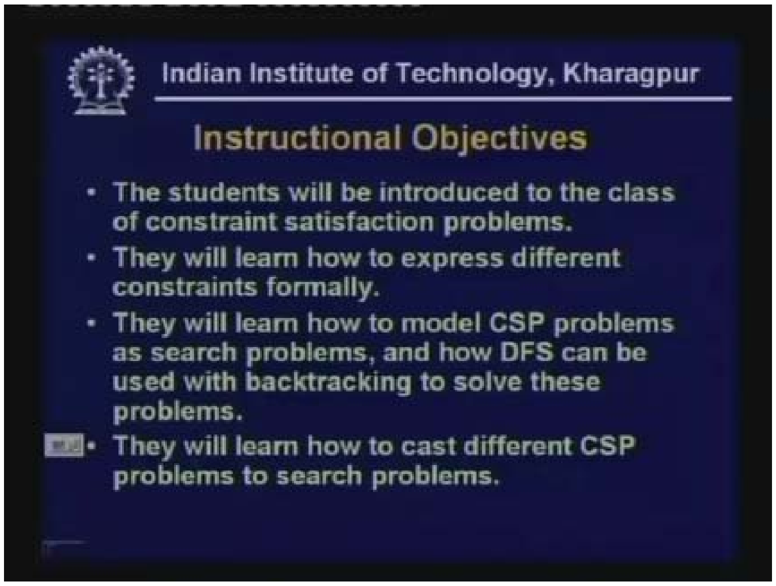
Artificial Intelligence

Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 9
Constraint Satisfaction Problems - 1

So today we start with lecture 9. This is the first lecture on constraint satisfaction problems. The instructional objectives of today's lecture are as follows:

(Refer Slide Time: 01:02)



Indian Institute of Technology, Kharagpur

Instructional Objectives

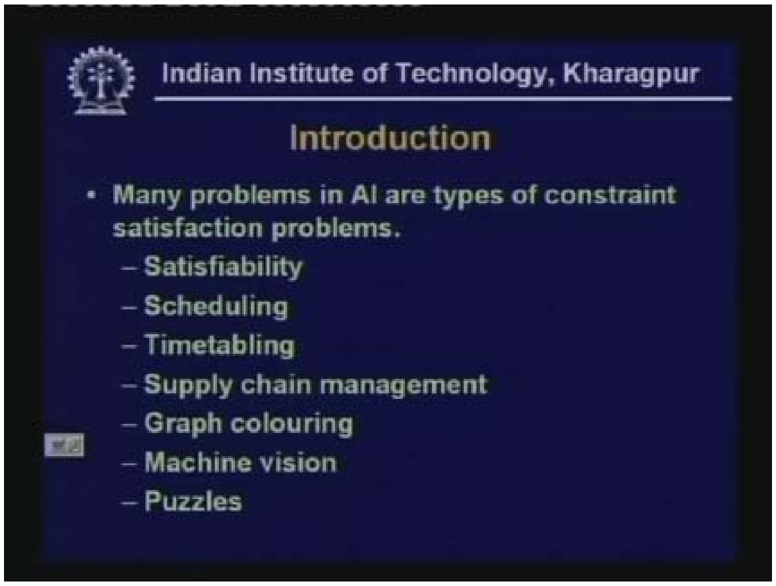
- The students will be introduced to the class of constraint satisfaction problems.
- They will learn how to express different constraints formally.
- They will learn how to model CSP problems as search problems, and how DFS can be used with backtracking to solve these problems.
- They will learn how to cast different CSP problems to search problems.

Students will be introduced to the class of problems which we call constraint satisfaction problems. Students will learn how different types of constraints can be expressed in the formal manner. Students will also learn how constraint satisfaction problems can be modeled as search problems and they will also see how depth first search can be used with backtracking to solve these problem. We will also discuss how different heuristics can be used to make this search process more efficient. Students should be able to cast different types of constraint satisfaction problems as search problems in this framework.

Many problems that occur in Artificial Intelligence as well as in many other areas of computer science are different types of constraint satisfaction problems. Many of you must be familiar with the satisfyability problem the 3-sat. You are given a propositional formula and you want to know whether this formula is satisfiable. That is, does there an assignment of values to the different propositions so that the formula evaluates to true?
So satisfyability problem is a type of constraint satisfaction problem.

Every variable can take exactly one value true or false and the formula must evaluate to true. So, if you have a formula which is a conjunction of different clauses then each of these different clauses must be individually true for the entire formula to be true.

(Refer Slide Time: 03:23)



Indian Institute of Technology, Kharagpur

Introduction

- Many problems in AI are types of constraint satisfaction problems.
  - Satisfiability
  - Scheduling
  - Timetabling
  - Supply chain management
  - Graph colouring
  - Machine vision
  - Puzzles

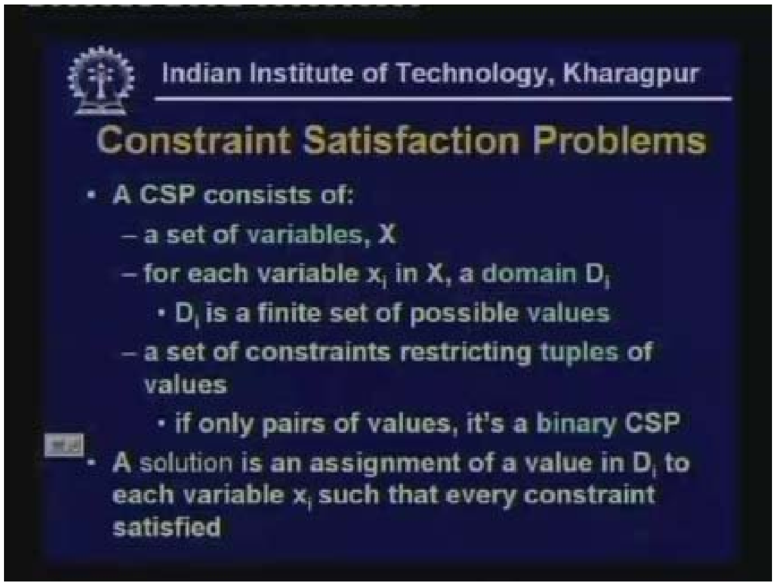
Different types of scheduling problem like the time tabling problem, different types of job shop scheduling can also we looked upon as constraint satisfaction problems. There can be different constraints among the different jobs. For example, job one may need to precede job five. So the precedence relationships among the jobs imposed are constraints among these jobs.

Other types of constraints can also be imposed. For example, in the time tabling problem our objective is, we are given a list of class rooms, a list of courses, a list of teachers and students taking the courses. Our objective is to schedule courses to time slots and to class rooms so that at any time slot not more than one class can be scheduled at a class room at a given time slot.

A teacher at the same time slot cannot be teaching two courses. A student at a same time slot cannot be taking two courses. So these are the different types of constraints in the time tabling problem. There are other problems like supply chain management, the graph coloring problem, etc. Then there is constraint satisfaction arising in machine vision in the age detection walls filtering and then different types of puzzles can also be looked upon as constraint satisfaction problems.

Many of you have worked on crossword puzzles. You are given a rectangular grid and you have to fill up words row wise and column wise. And there are constraints because several words may share a common letter. So formally a constraint satisfaction problem consists of a set of variables x.

(Refer Slide Time: 05:42)



Indian Institute of Technology, Kharagpur

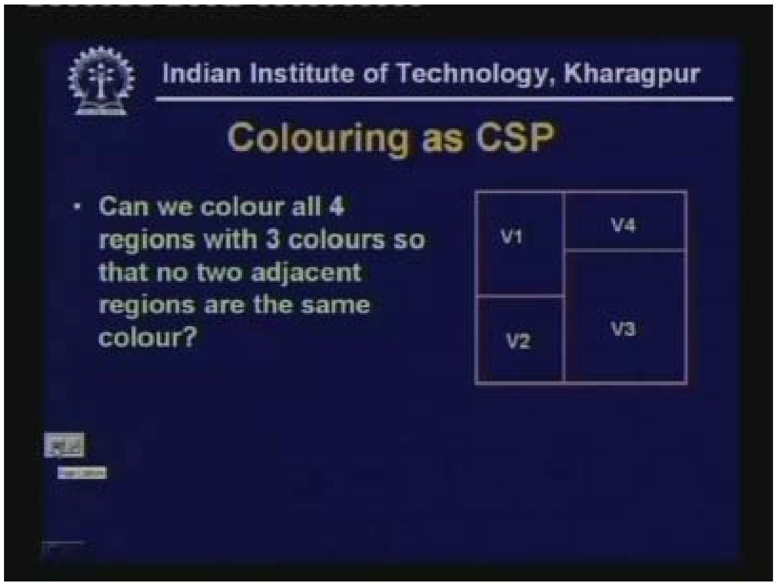
Constraint Satisfaction Problems

- A CSP consists of:
  - a set of variables, X
  - for each variable x_i in X, a domain D_i
    - D_i is a finite set of possible values
  - a set of constraints restricting tuples of values
    - if only pairs of values, it's a binary CSP
- A solution is an assignment of a value in D_i to each variable x_i such that every constraint satisfied

These are the variables X. For each variable x_i belonging to X the variable x_i can take its values from a domain D_i. And D_i is a finite set of possible values. So we assume today that the domain of every variable is discrete and finite. We are also given a set of constraints restricting tuples of values. For example, we can have binary constraints. We may say that x_5 and x_7 cannot take the same value. So this is an example of a binary constraint or we can say that the value of x_i must be numerically less than the value of the variable x_j. So these are binary constraints.

Unary constraint means constraints involving a single variable. We may say that x_i must take values which are only odd integers. So we can have unary constraints, we can have binary constraints and we may even have constraints involving more than two variables. However, we will discuss constraints mainly involving two variables or binary constraints and we will later discuss how other types of constraints can be formulated in this term. So, if the constraints concern only pairs of values we have a binary constraint satisfaction problem. A solution to a constraint satisfaction problem is an assignment of a value to each of the variables x_i. So each variable x_i can take values from its domain D_i and assignment of values to each of these variables which does not violate any of the constraints is a solution of the constraint satisfaction problem. Let us look at an example of a graph coloring problem or a map coloring problem.

(Refer Slide Time: 08:13)

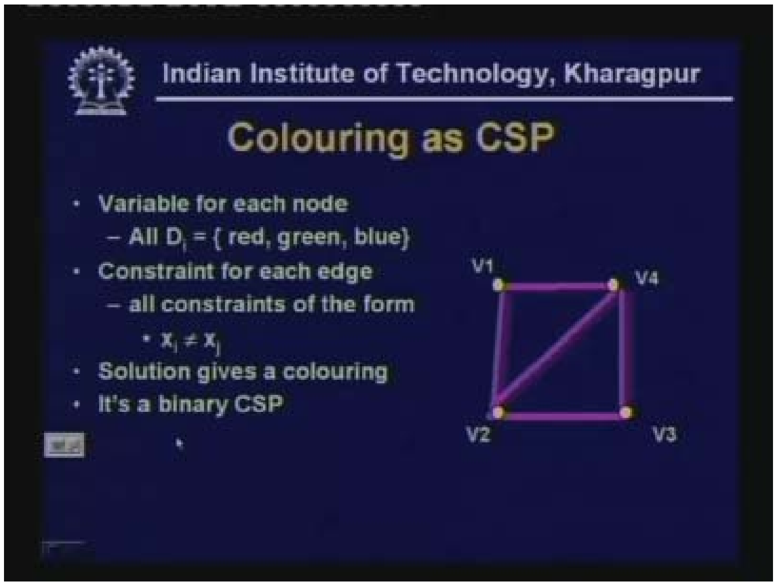

Indian Institute of Technology, Kharagpur

Colouring as CSP

- Can we colour all 4 regions with 3 colours so that no two adjacent regions are the same colour?

But I think we had occasion to discuss this problem when we discussed search. Here we have a map which consists of four regions: V1 V2 V3 and V4 and we are given three colors. Let us say the colors are red, green and blue. And we want to know, can you assign regions to colors such that two adjacent regions cannot have the same color. This problem is also called the map coloring because suppose you have a map we have different countries and you want to color the countries using colors such that two adjacent countries always have different colors. A related problem is the graph coloring problem. In fact a map coloring problem can be transformed to an instance of a graph coloring problem involving a planar graph.

(Refer Slide Time: 09:37)


Indian Institute of Technology, Kharagpur

Colouring as CSP

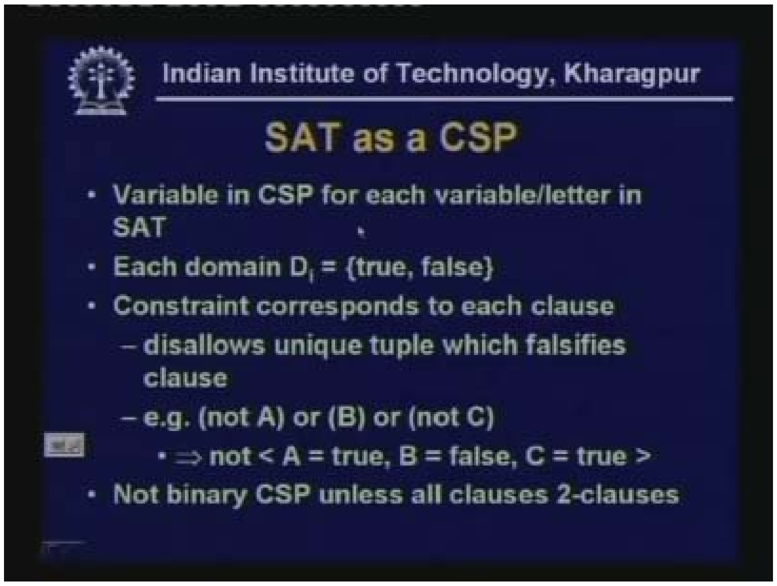
- Variable for each node
  - All D_i = { red, green, blue}
- Constraint for each edge
  - all constraints of the form
    - $x_i \neq x_j$
- Solution gives a colouring
- It's a binary CSP

If we look back at the previous slide we see that V1 is adjacent to V2, V3 and V4. V2 is adjacent to V1 and V3, V3 is adjacent to V2, V1 and V4, V4 is adjacent to V1 and V3. We can model V1 V2 V3 V4 as the vertices of the graph and we have an edge between two nodes if V1 is adjacent to V2. So we have an instance of a graph coloring problem where we want to assign colors to the vertices of the graph such that two adjacent vertices to not have the same color. So, we have a variable for each node and the domain for each of the variables is the colors red, green and blue.

V1 can be red, green or blue, V2 can be red, green or blue, V3 can be red, green or blue, V4 can be red, green or blue and so on. So there is a constraint on each edge. All constraints are of the form that the color on one end point of this edge should be different from the color of the other end point of this edge. So these two nodes must be V1 and V4 and they must have different colors.

V1 and V2 must have different colors, V2 and V4 must have different colors and so on. The solution to this problem gives the coloring of the vertices. This is an example of a binary CSP. Similarly, the satisfyability problem of propositional formula can be also looked upon as a constraint satisfaction problem. So, in the satisfyability problem in the formula we have different variables corresponding to each of these variables we have a variable in the CSP problem.

(Refer Slide Time: 11:33)



Indian Institute of Technology, Kharagpur

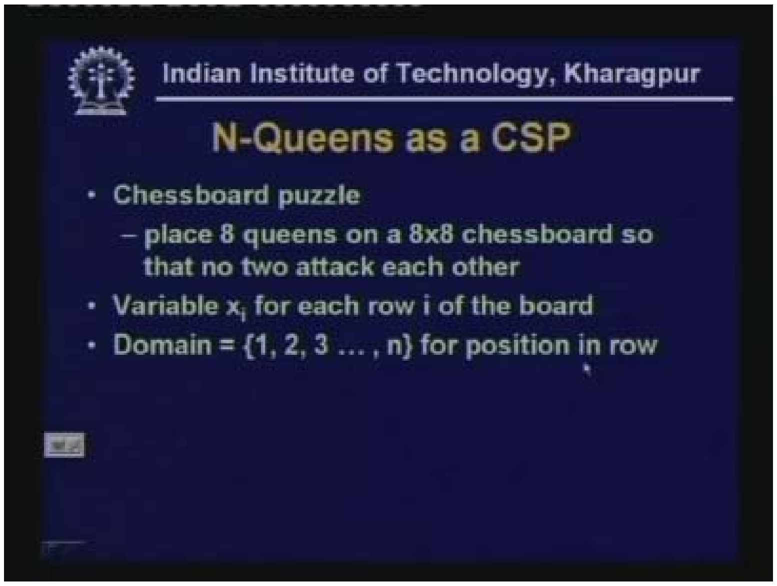
SAT as a CSP

- Variable in CSP for each variable/letter in SAT
- Each domain D_i = {true, false}
- Constraint corresponds to each clause
  - disallows unique tuple which falsifies clause
  - e.g. (not A) or (B) or (not C)
    - ⇒ not < A = true, B = false, C = true >
- Not binary CSP unless all clauses 2-clauses

The domain of each of the variables is either true or false. A proposition can be either true or false. The constraint corresponds to each clause. We disallow tuples which falsifies the clause. So, if we have a clause x_1 or x_2 bar we cannot have the situation where x_1 is 0 and x_2 is 1. If x_1 is 0 and x_2 is 1 then this clause will not be satisfyable. Therefore if you have a conjunction of clauses the entire conjunction cannot be satisfied. So, here the constraints are, for each clause we must disallow those tuples such that the clause is falsified. We have a clause x_1 bar or x_4 or x_5. We cannot have a situation that x_1

is true, x_4 is false and x_5 is false. So, Sat is an example of a constraint satisfaction problem. However, the general satisfyability problem is not a binary CSP. A clause may have k variables so the constraints involved are a set of k variables. Thirdly let us look at the n queens problem.

(Refer Slide Time: 13:34)



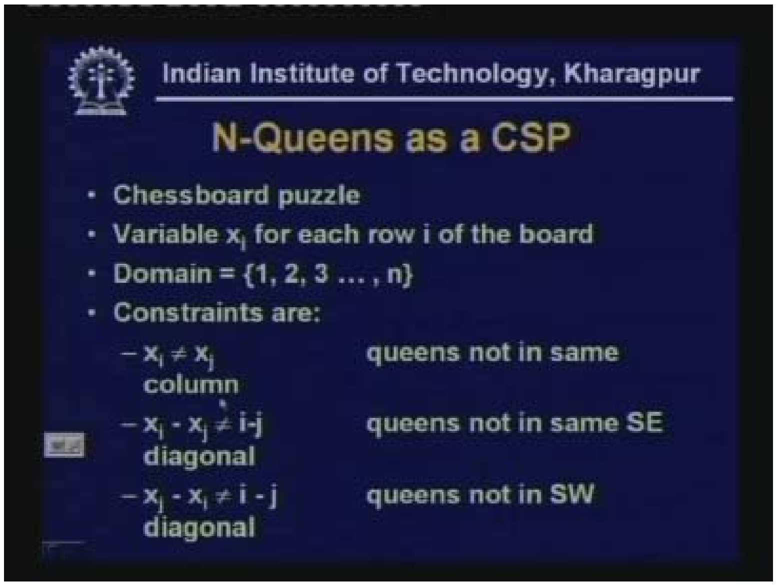
Indian Institute of Technology, Kharagpur

N-Queens as a CSP

- Chessboard puzzle
  - place 8 queens on a 8x8 chessboard so that no two attack each other
- Variable x_i for each row i of the board
- Domain = {1, 2, 3 … , n} for position in row

We are given an 8 cross 8 chess board and we have to place 8 queens in the chess board so that no 2 queens attack each other. So we can formulate this problem in this way. We have eight variables representing the positions of the 8 queens. Here 2 queens cannot be in the same row so let us say that x_1 is the queen in the first row, x_2 is the queen in the second row and x_3 is the queen in the third row. There has to be exactly 1 queen per row. So let x_1 demote the position of the queen that is the column position of the queen in the first row. So x_2 denotes the column position of the second queen in the second row. Now, because no 2 queens attack each other the domain of x_i is 1 2 3 4 up to 8 for each position in the row.

(Refer Slide Time: 14:32)



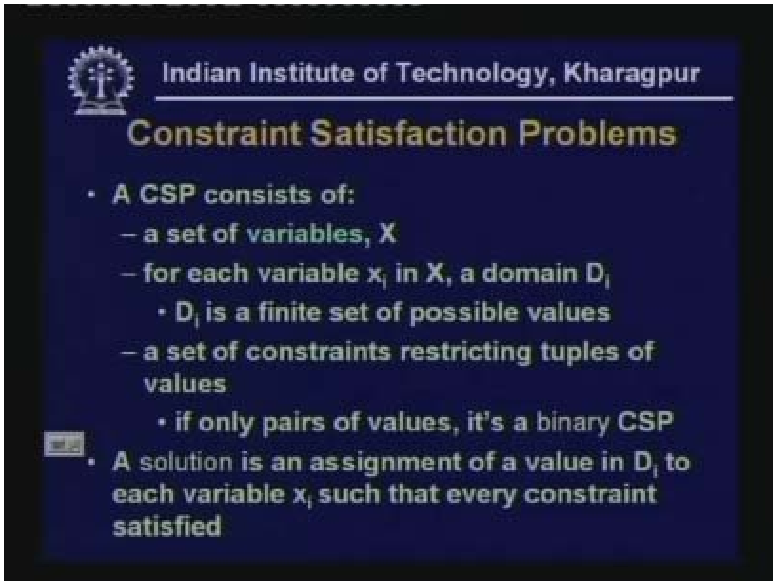
Indian Institute of Technology, Kharagpur

N-Queens as a CSP

- Chessboard puzzle
- Variable x_i for each row i of the board
- Domain = {1, 2, 3 … , n}
- Constraints are:
  - $x_i \neq x_j$ queens not in same column
  - $x_i - x_j \neq i-j$ queens not in same SE diagonal
  - $x_j - x_i \neq i - j$ queens not in SW diagonal

The constraints are, for any different i and j x_i should not be equal to x_j because we cannot have 2 queens in the same column so each of these variables must have a different value. Also 2 queens cannot be in the same column. So x_i minus x_j cannot be equal to i minus j or and x_j minus x_i cannot be equal to i minus j that is, the queens cannot be either in the right diagonal or in the left diagonal. So these are the constraints for the n queens problem or its variation the 8 queens problem. And this is the same statement we have seen.

(Refer Slide Time: 15:24)



Indian Institute of Technology, Kharagpur

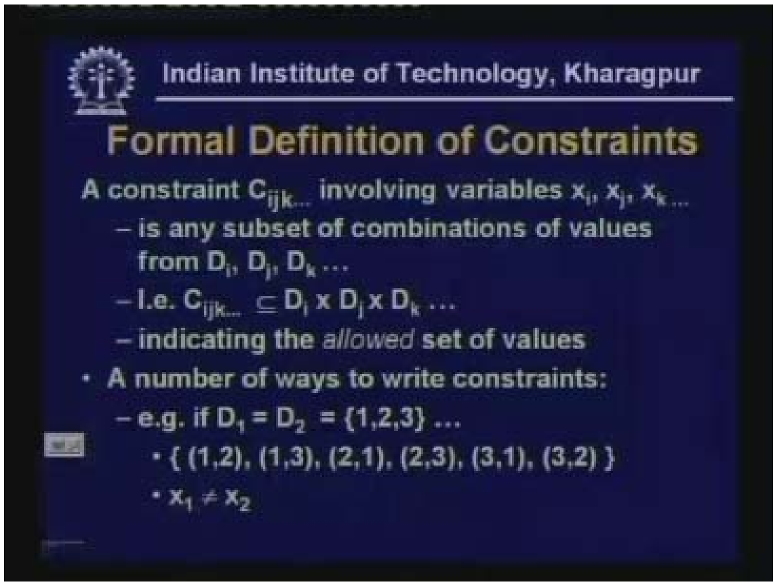
Constraint Satisfaction Problems

- A CSP consists of:
  - a set of variables, X
  - for each variable x_i in X, a domain D_i
    - D_i is a finite set of possible values
  - a set of constraints restricting tuples of values
    - if only pairs of values, it's a binary CSP
- A solution is an assignment of a value in D_i to each variable x_i such that every constraint satisfied

So a CSP to summarize consists of a set of variables x consisting of x_1 x_2 up to x_n. Each variable x_i has a domain D_i from which it takes values. And D is a finite set of possible values. We have a set of constraints restricting the tuples of values. And a solution is an assignment of a value in D_i to each variable xi such that every constraint is satisfied. So let us formally define what we mean by constraints.

A constraint C_{ijk} . . involves the variables x_i x_j x_k etc. So constraint can involve a single variable, it is called the unary constraint and unary constraint basically restricts the domain. We can have binary constraints which involve two variables, ternary constraints involving three variables and k-ary constraints in general involving k variables.

(Refer Slide Time: 16:31)



Indian Institute of Technology, Kharagpur

Formal Definition of Constraints

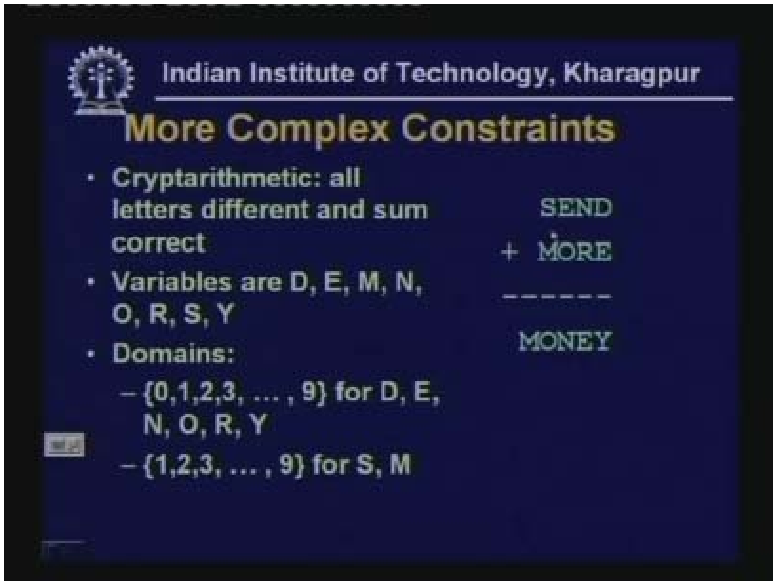
A constraint $C_{ijk...}$ involving variables x_i, x_j, x_k ...
- is any subset of combinations of values from D_i, D_j, D_k ...
- i.e. $C_{ijk...} \subseteq D_i \times D_j \times D_k$ ...
- indicating the *allowed* set of values

• A number of ways to write constraints:
- e.g. if $D_1 = D_2 = \{1,2,3\}$ ...
  • { (1,2), (1,3), (2,1), (2,3), (3,1), (3,2) }
  • $x_1 \neq x_2$

It is any subset of combinations of values from the domains of these variables xi xj xk which are allowed. That is, this constraint specifies that a subset of the Cartesian product of di dj and dk are allowed sets. There are different ways in which we can express such constraints. We can specify as to which are the valid tuples.

For example, we can say that suppose d1 and d2 are same and they are 1, 2 and 3 then let us say the valid tuples are (1,2) (1,3) (2,1) (2,3) (3,1) (3,2) and the rest of the tuples are invalid. Or we can specify constraints like x_1 is not equal to x_2 or constraints like x_1 less than x_2 or x_1 equal to x_2 plus 1 and so on. So there are different ways in which constraints can be expressed. Another example is crypt arithmetic which is a type of puzzle. We have every letter standing for a digit and every letter stands for a different digit.

(Refer Slide Time: 18:00)



Indian Institute of Technology, Kharagpur

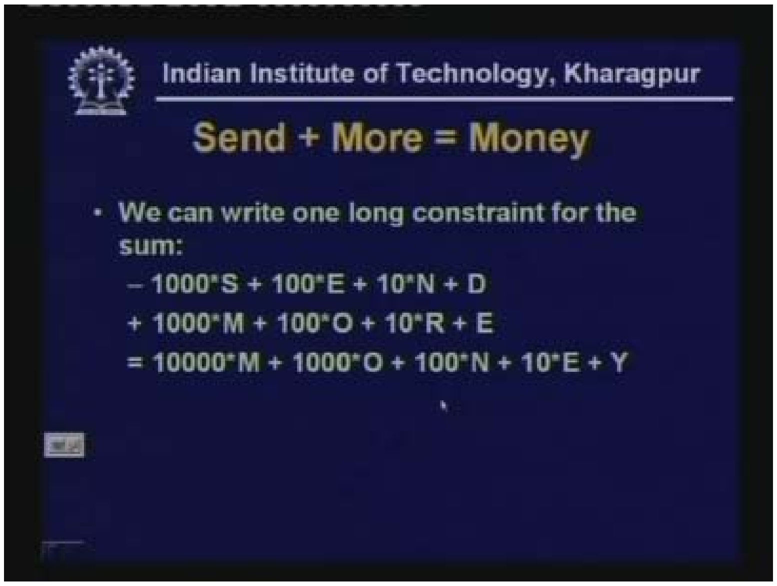
More Complex Constraints

- Cryptarithmetic: all letters different and sum correct
- Variables are D, E, M, N, O, R, S, Y
- Domains:
  - {0,1,2,3, … , 9} for D, E, N, O, R, Y
  - {1,2,3, … , 9} for S, M

```
  SEND
+ MORE
------
 MONEY
```

We have to find an assignment of letters to digits such that a given arithmetic formula is correct. For example, we have this formula:
Send plus more is equal to money and the variables are D, E, M, N, O, R, S, Y and the domains of these variables are, or rather domains for D, E, M, N, O, R, S, Y are the digits from 0 to 9. And S and M cannot be 0 so S and M must have values from 1 to 9 and each of these variables must have different values. We want to know if there is a solution to this problem. That is, is there an assignment of values to each of these variables such that this arithmetic expression is correct?

(Refer Slide Time: 19:01)



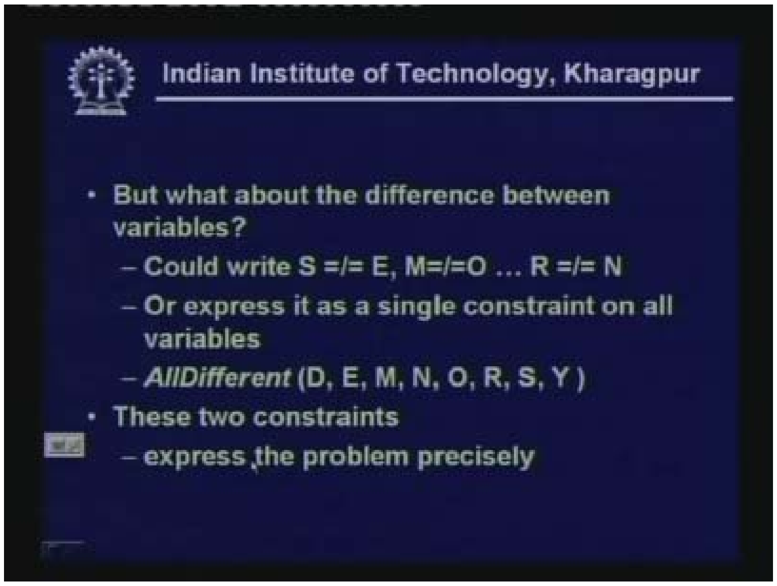
Indian Institute of Technology, Kharagpur

Send + More = Money

- We can write one long constraint for the sum:
  - $1000*S + 100*E + 10*N + D$
  $+ 1000*M + 100*O + 10*R + E$
  $= 10000*M + 1000*O + 100*N + 10*E + Y$

So how do we specify the constraint for this crypt arithmetic problem?
We can write one long constraint for the sum. It is 1000 star S plus 100 star E plus 10 star N plus D plus 1000 star M plus 100 star O plus 10 star R plus E is equal to 10000 star M plus 1000 star O plus 100 star N plus 10 star plus Y. So this is the sum constraint but there are also other constraints.

(Refer Slide Time: 19:32)

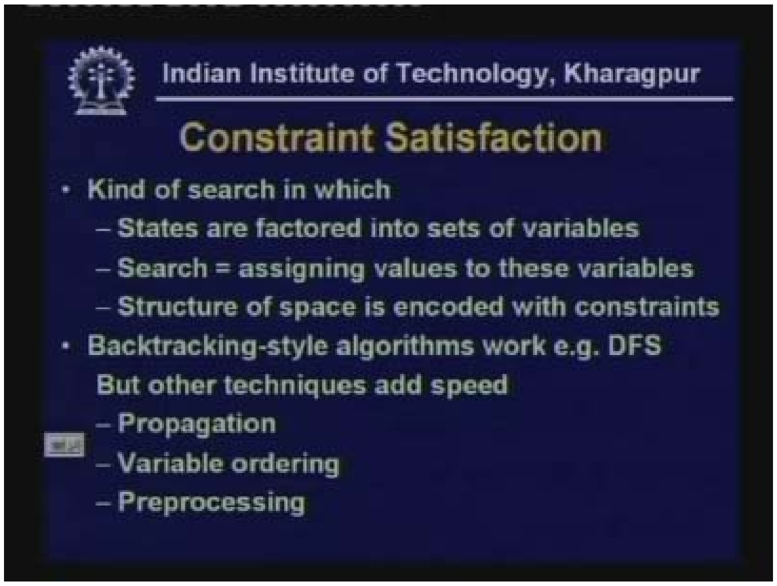


Indian Institute of Technology, Kharagpur

- But what about the difference between variables?
  - Could write S =/= E, M=/=O … R =/= N
  - Or express it as a single constraint on all variables
  - *AllDifferent* (D, E, M, N, O, R, S, Y)
- These two constraints
  - express the problem precisely

We have to specify that S is not equal to E, S is not equal to O, S is not equal to R, S is not equal N, M is not equal to 1 and so on. Or we can express it as a single constraint on all the variables. We can say that the values of each of these variables must be different by saying all different D, E, M, N, O, R, S, Y. These two constraints the sum constraint and this constraint together precisely characterize this problem. Now let us see that constraint satisfaction problems can be looked upon as search problems.

(Refer Slide Time: 20:16)



Indian Institute of Technology, Kharagpur

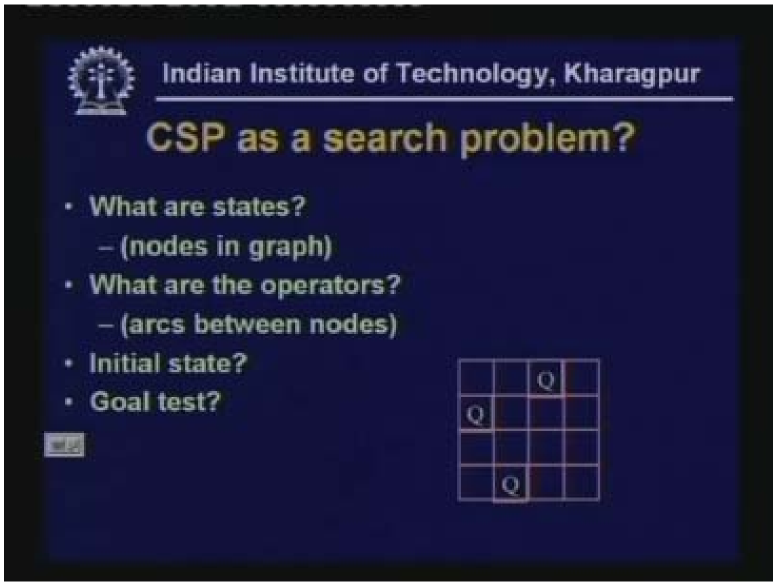
Constraint Satisfaction

- Kind of search in which
  - States are factored into sets of variables
  - Search = assigning values to these variables
  - Structure of space is encoded with constraints
- Backtracking-style algorithms work e.g. DFS

  But other techniques add speed
  - Propagation
  - Variable ordering
  - Preprocessing

It is a kind of search in which a state is not indivisible. We will look at the formulation. A state consists of assignments of values to the different variables. So state is factorized into the states of the different variables inside the state. The search involves finding an assignment of values to these variables, finding that state which corresponds to a particular assignment of values to these variables. So these constraints provide the structure to the state space. And we will discuss about backtracking algorithms which can be very well done with depth first search and that can work well for these problems. And we can use other methods along with backtracking including constraint propagation, variable ordering and different preprocessing steps to make this search more efficient.

(Refer Slide Time: 21:24)



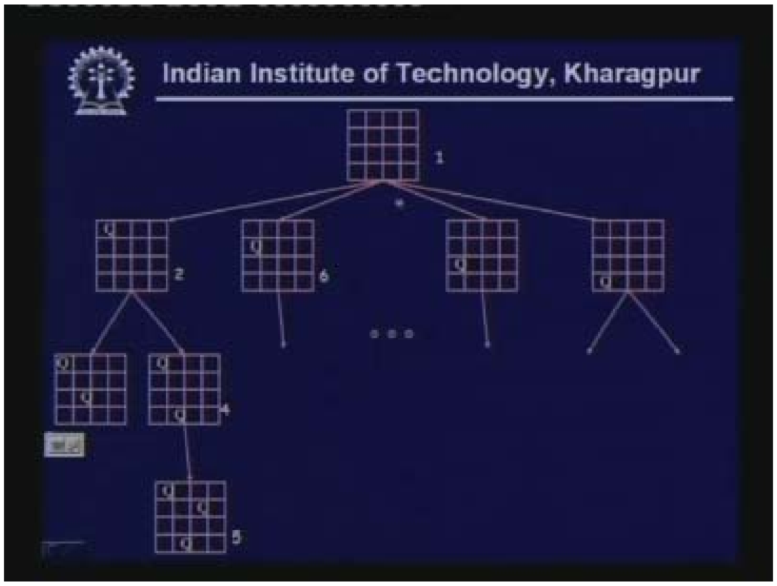
Indian Institute of Technology, Kharagpur

CSP as a search problem?

- What are states?
  - (nodes in graph)
- What are the operators?
  - (arcs between nodes)
- Initial state?
- Goal test?

Now let us see how CSP can be looked upon as a search problem. The states are the nodes in the graph, the operators are the arcs between the nodes and then we have to know what the initial state is and what the goal states are. For example, suppose we take this problem the n queens problem for n is equal to 4 and for this problem the initial state is the state where none of the variables are assigned values. That is, we have not placed any queens on the board.

(Refer Slide Time: 21:55)


Indian Institute of Technology, Kharagpur

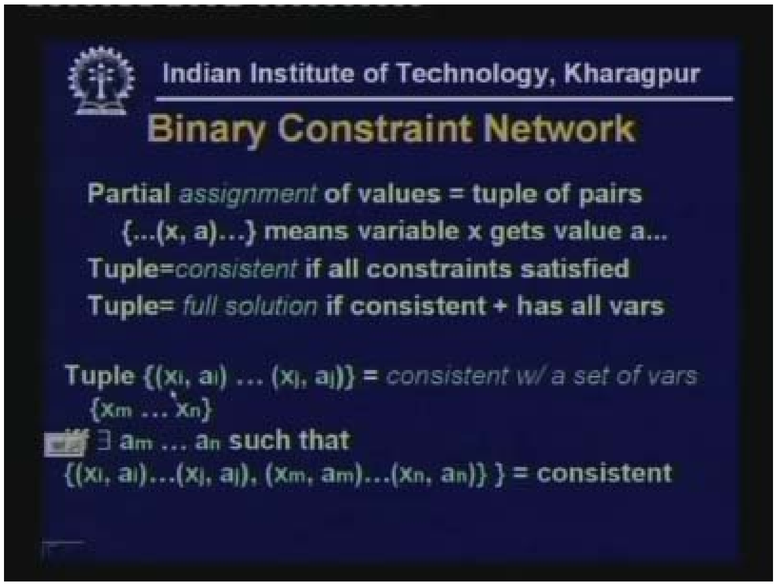
The neighbors are the cases where the first queen is assigned. The first queen can be assigned either here or here or here or here or here. So let us say the first queen corresponds to the first column. The first queen can be assigned 1 or 2 or 3 or 4. If the first queen is assigned 1 the second queen cannot be assigned 1 because the constraint is violated, 2 queens cannot be in the same row. The first queen cannot be assigned 2 because the diagonal constraints will be violated. If the first queen is in 1 the second queen can be in 3 or 4, if the second queen is in 3 the third queen cannot be in 1 or 3. It cannot be in 2 or 4 because diagonals constraints will be violated.

So, after we have placed the first queen here and the second queen here we cannot place the third queen anywhere. If we cannot place the third queen anywhere wherever we place the fourth queen the problem cannot be satisfied because the constraint of third queen is violated. So this path is fruitless so we can terminate search below this. Then we can look at this sibling that the first queen is in first row and the second queen is in fourth row.

Now the third queen can be placed neither in 1 nor in 4 nor in 3 but only in 2. So you place the third queen in 3 and then we have to see whether we can place the fourth queen anywhere. If we place the third queen in 2 then the fourth queen cannot be placed in 1 or 2 or 3 or 4. So we have to prune search below this node. Now, when you prune search below this node we have to backtrack and find out the next place where we should explore the state space. So, as you can see the nature of this search space is such that we can start from the root node and we can do a depth first search. And whenever we generate successors we only generate those successors that do not violate any constraints. If a constraint is violated at a node then further assignment of values to the other variables cannot help resolve that constraint. So, exploring that region of the search tree is fruitless. So, if a constraint is violated we can abandon that path.

If we reach a node so that we cannot place some variable then we cannot assign any value to the one of the variables, then we can prune that search portion of the search tree and we backtrack to our next choice point. So depth first search with backtracking seems to be a good solution to such search problems. Binary CSPs are the special types of constraint satisfaction problem which involve constraints between two variables. Suppose this is an assignment of values to variables

(Refer Slide Time: 26:07)



Indian Institute of Technology, Kharagpur

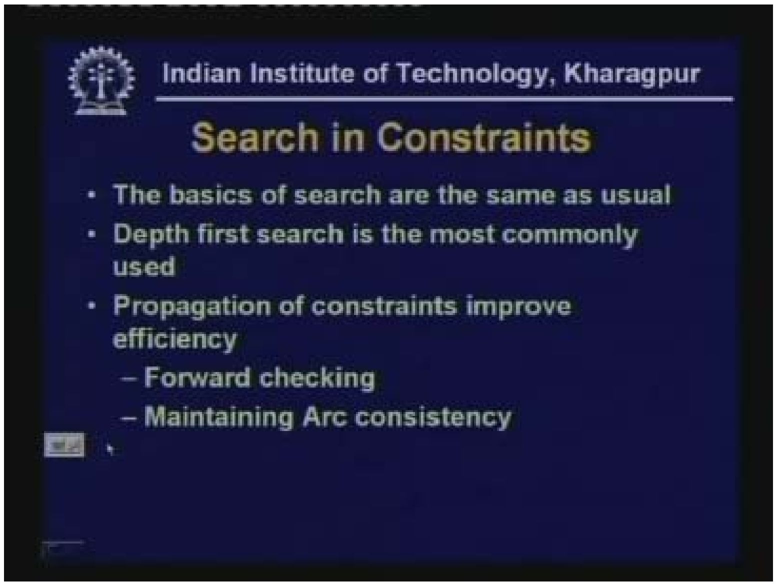
Binary Constraint Network

Partial *assignment* of values = tuple of pairs
{...(x, a)...} means variable x gets value a...
Tuple=*consistent* if all constraints satisfied
Tuple= *full solution* if consistent + has all vars

Tuple $\{(x_i, a_i) \dots (x_j, a_j)\}$ = *consistent w/ a set of vars* $\{x_m \dots x_n\}$
iff $\exists\, a_m \dots a_n$ such that
$\{(x_i, a_i)\dots(x_j, a_j), (x_m, a_m)\dots(x_n, a_n)\}$ } = consistent

Suppose x_i is assigned a_I, x_j is assigned a_j and they are consistent with a set of variables x_m to x_n if and only if there exists a tuple of values am up to an such that x_i a_I, x_i assigned to a_I, x_j assigned to a_j, x_m assigned to a_m, x_n assigned to an this whole thing is consistent. In a search tree a particular node in the middle of the search tree denotes a partial assignment of values to the variables. This partial assignment can be explored further only if there is a full assignment of values to all the variables which includes this partial assignment and which satisfies all the constraints.

We will use this property later to see where we can prune the search space even further. So, when we have a constraint satisfaction problem we can look upon this as a search problem and depth first search seems to be a very good technique to solve such search problems. But if you are able to propagate constraints then we can improve the efficiency of the search. Let us discuss different ways of constraint propagation including forward checking and maintenance of arc consistency.
(Refer Slide Time: 27:48)

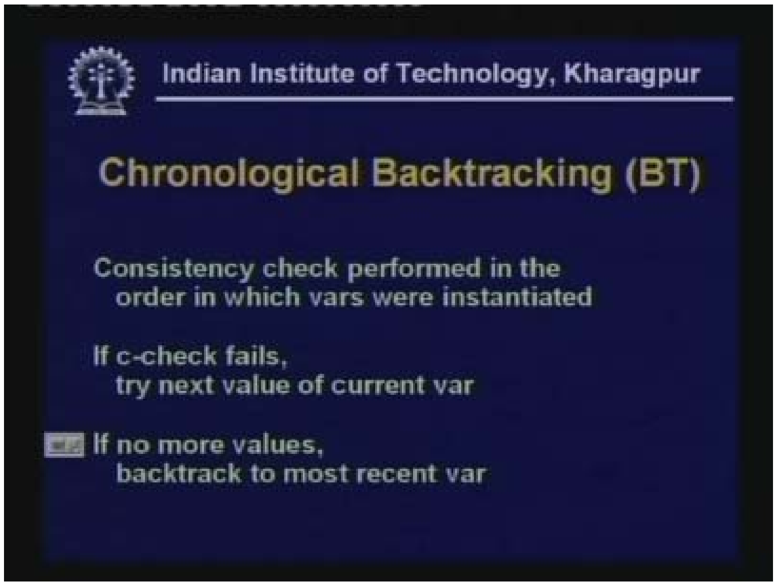

Indian Institute of Technology, Kharagpur

Search in Constraints

- The basics of search are the same as usual
- Depth first search is the most commonly used
- Propagation of constraints improve efficiency
  - Forward checking
  - Maintaining Arc consistency

So the backtracking framework we will use for constraint satisfaction problem basically involves that consistency check is performed in the order in which the variables are instantiated.

(Refer Slide Time: 28:09)


Indian Institute of Technology, Kharagpur

Chronological Backtracking (BT)

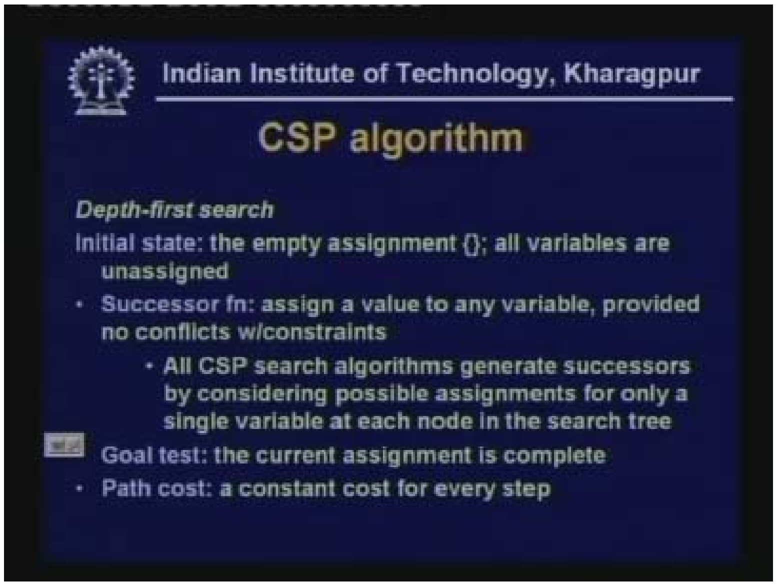
Consistency check performed in the order in which vars were instantiated

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

If no more values, backtrack to most recent var

Whenever we instantiate a variable we check its consistency with respect to the variables we have already assigned. If the consistency check fails at a particular point we look at the next possible value of the current variable. If there are no more values for the variables which are consistent with the previous assignments then we backtrack to the most recent choice point. So this is the sense of chronological backtracking which is the basic framework of search for CSPs. Let us look at the depth first search algorithm for solving CSPs.

(Refer Slide Time: 28:55)



Indian Institute of Technology, Kharagpur

CSP algorithm

Depth-first search

Initial state: the empty assignment {}; all variables are unassigned

- Successor fn: assign a value to any variable, provided no conflicts w/constraints
  - All CSP search algorithms generate successors by considering possible assignments for only a single variable at each node in the search tree
- Goal test: the current assignment is complete
- Path cost: a constant cost for every step

The initial state is the empty assignment. That is, none of the variables are assigned. All variables are unassigned. In the goal state all the variables will be assigned values from their domain and all the constraints must be satisfied. The successor function assigns a value to any variable which is as yet unassigned such that this assignment does not violate any constraints with respect to what has already been assigned. All CSP search algorithms generate successors by considering possible assignments for only a single variable at each node in the search tree.

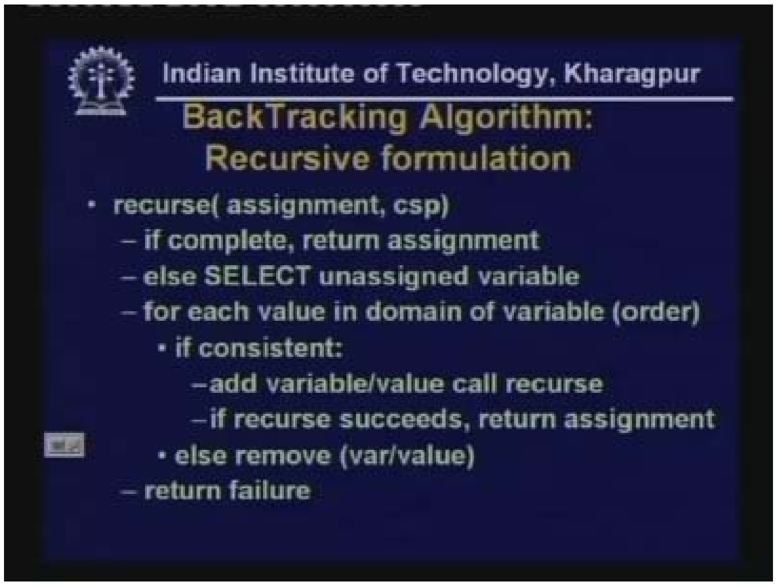
When we start from the initial node where all the variables are unassigned there are n candidate variables. Each of them can take different values. So there are potentially many successors of the first node. So what we do is, instead of trying all these values at once we pick one of the variables and for that variable we try to assign the different possible values and that is how we model the search tree for state space.

In the goal test when the assignment is complete and by the way in which we have assigned we have to make sure that no constraints are violated. So, if we have able to get a complete assignment to all the variables so that no constraints are violated we have reached a goal state. If there are n variables typically if we get to a node which is at depth n then we have found a solution to the CSP. Now in constraint satisfaction problem there may be different objectives. Your objective may be to find one solution any one solution or all solutions.

If you want to find all solutions you must explore the entire tree until except those where you have pruned the search. If you want to find only one solution you stop as soon as you get a solution. If there is not solution you have to again explore the entire tree to rule out all possibilities.

What is the path cost?
Actually path cost is not very important. So we can say that the cost is 1 for every step so assigning n variables will take a cost of one.

(Refer Slide Time: 31:38)



Indian Institute of Technology, Kharagpur

BackTracking Algorithm: Recursive formulation

```
• recurse( assignment, csp)
  – if complete, return assignment
  – else SELECT unassigned variable
  – for each value in domain of variable (order)
    • if consistent:
      –add variable/value call recurse
      –if recurse succeeds, return assignment
    • else remove (var/value)
  – return failure
```

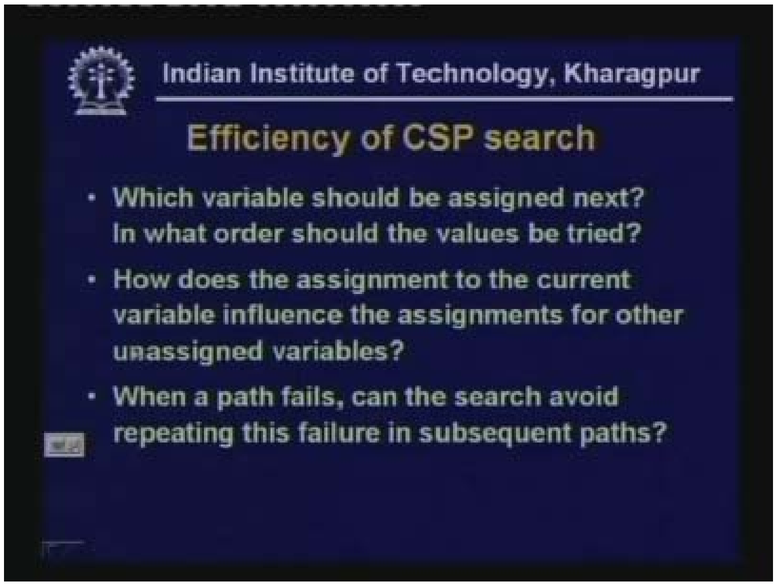
So this algorithm can be recursively formed because we are basically using depth first search we can have a recursive formulation of this algorithm. So this is a simple recursive function which captures this basic algorithm. Recurse (assignment, csp). If the assignment is complete, that is all variables are assigned values then we return the current assignment as the solution. Otherwise we select an unassigned variable.

How do we select?
We can simply select the next one in the order or we can select smart key. But we select one unassigned variable and for each value in the domain of the variable. Assuming that we order the possible values of the variable can be taken in a particular order. One by one we consider the different values or we can be smarter and we can decide in what order we should test the values so that our search effort is minimized. In any case for a given order we take up values one by one and if that assignment of that value to this variable is consistent we add this variable value tuple to the assignment and we call recurse again with this new assignment.

If recurse succeeds then we return the assignment. Otherwise if this recurse does not succeed we remove this assignment and try the next choice. If we have not been able to succeed then we return failure. If none of these assignments are successful then we return failure. This is a simple recursive structure of the basic dfs problem for backtracking which solves the csp.
If we want to consider the efficiency of the csp problem there are various things that we can consider. Among the unassigned variables so far which variable should we pick next?

And then, for a given choice of variable, in what order the values of the variables should be tried?

(Refer Slide Time: 34:30)



Indian Institute of Technology, Kharagpur

Efficiency of CSP search

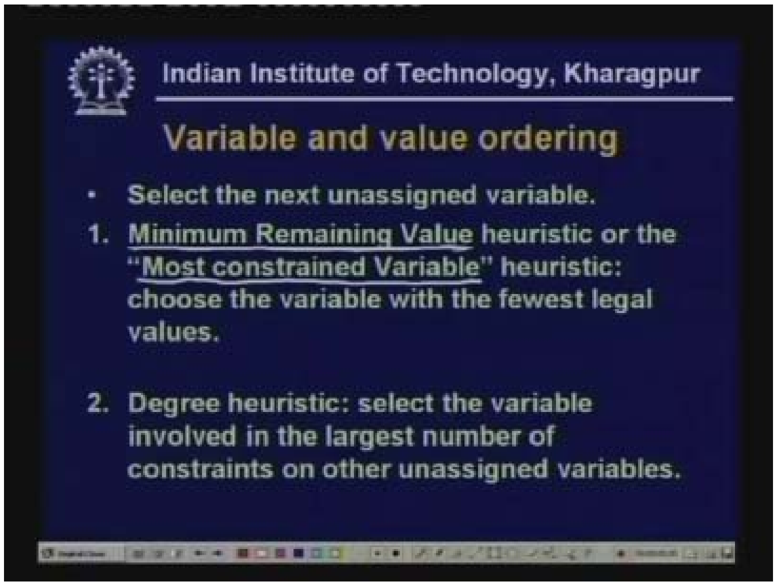
- Which variable should be assigned next? In what order should the values be tried?
- How does the assignment to the current variable influence the assignments for other unassigned variables?
- When a path fails, can the search avoid repeating this failure in subsequent paths?

Secondly, how does the assignment to the current variable influence the assignment for other unassigned variables. So, if the current variable is given a certain assignment that will affect the assignment of values to the other variables because the constraints that are imposed by this current assignment is carried over to the other variables which converts the constraints other variables including the currently unassigned variables. So we may have to consider this effect in deciding which variable or which variable value pair we will try next.

Thirdly, when a path fails, suppose at a particular point there is a failure there's an inconsistency can the search avoid repeating this failure in subsequent paths?
These issues need to be discussed to see how we can try to make this search more efficient. We have discussed a heuristic search, different types of heuristics in csp, we do not normally use heuristics. Instead we try to see how these issues can be tackled and these issues have the effect of reducing of improving search efficiency.

First let us take up the issue of variable ordering. When we want to select the next unassigned variable we can use the following heuristic which is called the minimum remaining value heuristic or which is also called the most constrained variable heuristic which states choose the variable with the fewest legal values.

(Refer Slide Time: 36:31)


Indian Institute of Technology, Kharagpur

Variable and value ordering

- Select the next unassigned variable.

1. Minimum Remaining Value heuristic or the "Most constrained Variable" heuristic: choose the variable with the fewest legal values.
2. Degree heuristic: select the variable involved in the largest number of constraints on other unassigned variables.

Every variable has a domain to start with. When we assign values to some of the variables the domain of the other variables get restricted. Suppose V1 and V2 are adjacent in the graph coloring problem, if I assign V1 to red V2 could initially be either red or green or blue but if V1 is assigned red and V2 is adjacent to V1 then V2 cannot be red so V2 must be either green or blue. If V2 is also adjacent to V3 and V3 is green then V2 also cannot be green so V2 cannot be red or green it has to be only blue if at all. Therefore the domain of V2 has to be restricted.

After we have made partial assignment to some of the variables we look at the remaining variables and their valid domain with respect to the current assignment. We choose that variable whose domain is smallest. Suppose V2 has only one value in its domain blue so we can just assign V2 to blue and that is the only choice. So we choose the variable which has the least number of legal values. In the beginning suppose when we have not assigned any variable to any value we can use another heuristic which is called the degree heuristic.

In the degree heuristic we select the variable which is involved in the largest number of constraints with other unassigned variables. We select variables one by one but finally we have to select all the variables. If we choose to select a variable so that it constrains severely the domains of the other unassigned variables it can help to reduce future search. The degree heuristic is useful especially in the beginning or in the case where the minimum remaining value heuristic is a very good heuristic and we usually try it first. But we can use the degree heuristic to resolve ties between two variables which have the same importance according to the MRV heuristic. So minimum remaining value heuristic or the most constrained variable heuristic is a heuristic which is used for choosing the next variable to a sign and it is very effective.

If you run different instances of constraint satisfaction problem you can test the effectiveness of this heuristic. The degree heuristic is also useful and is often used to break ties. Once we have picked which variable to try let us consider the order in which we will consider the values we assign to the heuristics. Here the heuristic used is called the least constraining value heuristic.

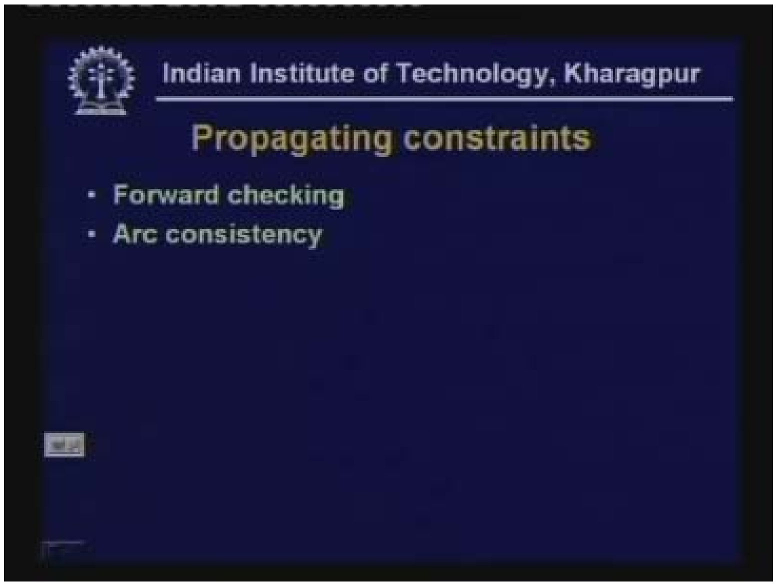
We prefer to first try that value that rules out the fewest choices for the neighboring variables in the constraint graph. So we will choose the values one by one. First time when we choose a value and suppose our objective is to find one solution to the csp problem then we will be very happy if we quickly reach a solution because after that we can stop our search.

Of course in those cases where we want all the solutions or in those cases where a solution does not exist we have to look through the entire search space. The order in which we choose the values do not matter because we have to go through the entire search space. But if we interested in only one or a few satisfying solutions then we try to choose that value for which there is a greater hope that a solution will be found. So we choose that value which is least constraint. That is which rules out the fewest choices for the remaining variables or at least the neighboring variables in the constraint graph.

Secondly, for csp problems we do depth first search and whenever we find an inconsistency we backtrack. Inconsistencies arise because the current assignment of variable value does not agree with the previous assignments. In those cases we backtrack. It is possible to take this effort one step further.

Whenever we assign a value to a variable we propagate constraints to the future variables to the unassigned variables. And then if we notice the other variables as a result of current assignment there is a variable whose domain becomes null. That means there is another variable which is not consistent with the current partial assignment. If we can discover that we can terminate our search. This can be achieved by various types of constraint propagation. There are many algorithms which do constraint propagation to different degrees.

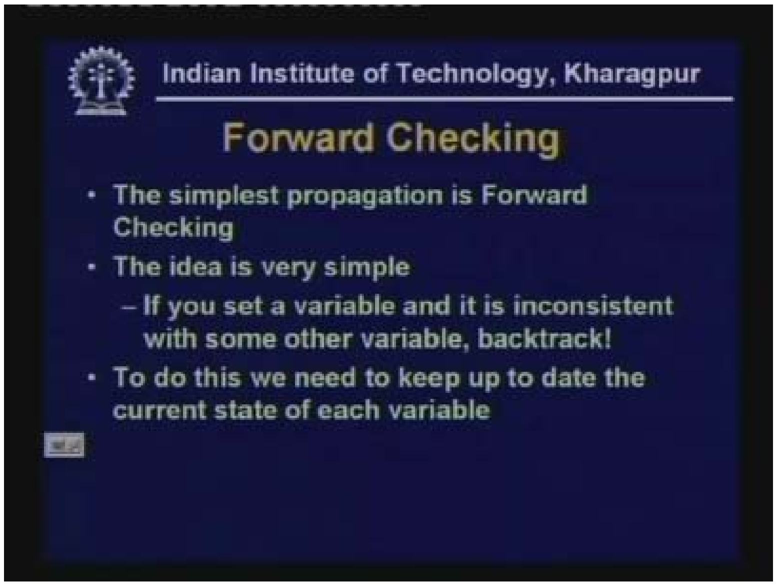
(Refer Slide Time: 42:51)


Indian Institute of Technology, Kharagpur

Propagating constraints

- Forward checking
- Arc consistency

(Refer Slide Time: 43:03)


Indian Institute of Technology, Kharagpur

Forward Checking

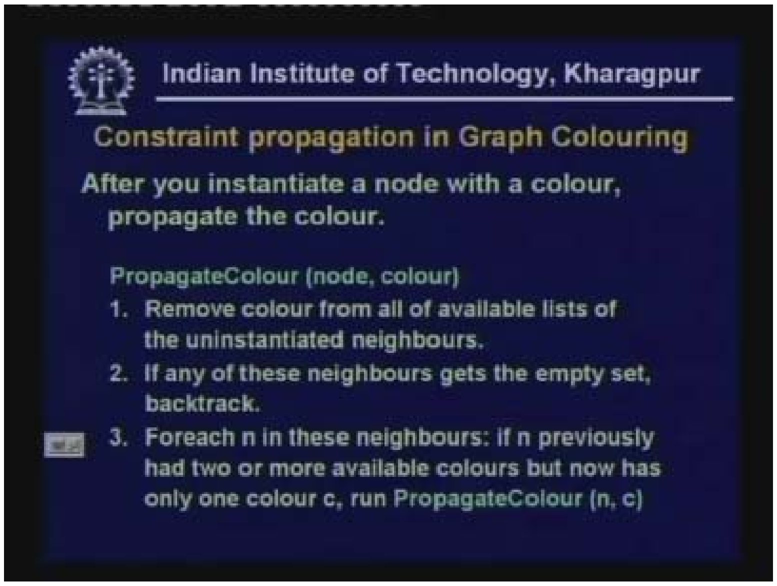
- The simplest propagation is Forward Checking
- The idea is very simple
  - If you set a variable and it is inconsistent with some other variable, backtrack!
- To do this we need to keep up to date the current state of each variable

We will discuss briefly two types of algorithms forward checking and arc consistency. Forward checking is the simplest type of constraint propagation. The idea is simple. When I assign a value to a variable and if we find that it is inconsistent with some other variable, another variable that cannot take any valid value for the current assignment then we can terminate the search at this point. This is the idea of the algorithm. So whenever we assign a value to a variable we use that assignment to restrict the domains of the unassigned variables. And if we find that the domain of some other future variable becomes null we terminate the search at this point. This is the idea of forward checking.

And how do we carry out forward checking?

With every variable we keep its current value domain. So whenever we assign a variable to a value we update the domains of the other variables. So we use a data structure. The data structure is, for every variable xi we maintain its current domain cdi. Initially cdi is equal to the whole domain of the variable, cdi is equal to D_i to start with. When we set variable x_j equal to particular value v we remove xi is equal to u from the domain of x_i if some constraint is not consistent with both xj is equal to v and x_i is equal to u. My current assignment is x_j is equal to v. Therefore x_i is an unassigned variable and u is one of the values in its domain. But if x_i is equal to u and x_j is equal to v violates some constraints then we remove u from the domain of xi. And in this way if we find that a particular variable x_i has its current domain cdi to be null then we can stop search beyond the current point.

(Refer Slide Time: 45:36)



Indian Institute of Technology, Kharagpur

Constraint propagation in Graph Colouring

After you instantiate a node with a colour, propagate the colour.

```
PropagateColour (node, colour)
1. Remove colour from all of available lists of
   the uninstantiated neighbours.
2. If any of these neighbours gets the empty set,
   backtrack.
3. Foreach n in these neighbours: if n previously
   had two or more available colours but now has
   only one colour c, run PropagateColour (n, c)
```

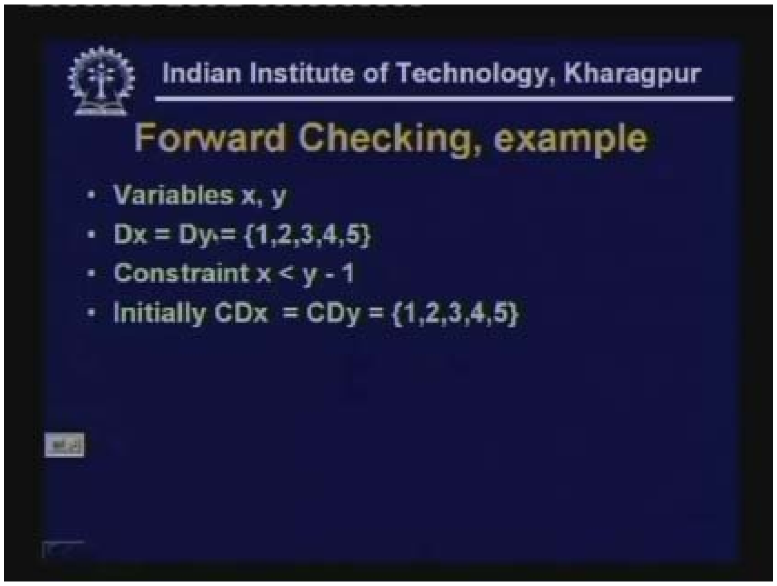
Now let us briefly discuss how constraint propagation is used in the graph coloring problem. So in a graph coloring problem we start with all the nodes unassigned then we pick a node and assign it a color. Assign it possible colors from what its current domain is. After we instantiate a node with a color we propagate the color. And how do we propagate the color?
We remove the color.

Suppose we have colored the current node red we remove red from the current domain of all its uninstantiated neighbors. By this process if any of these neighbor domains become empty then we backtrack. Now for each n in these neighbors if n previously had two or more available colors but now has only one color. Suppose there is a neighbor, earlier its domain cdi had two or more colors but as a result of this constraint propagation xi has only one valid color left on the domain, now I can use this color to propagate constraints further. So I can take n assign n that color because that is the only color n can be assigned and propagate this constraint so that its neighbors get affected. So we call propagate color nc. So this is the code for propagating constraints for the graph coloring problem.

Whenever we assign a node the color we call propagate color node color. And propagate color node color has three steps:
1) Remove color from all of available lists of the uninstantiated neighbors of node
2) If any of these neighbors gets the empty set then this is an inconsistent situation so we backtrack.

Now we look at these neighbors. Earlier its current domain had two or more colors but as a result of this propagation currently that node n has only one color c in its current domain then we call propagate color nc. So this is the algorithm for propagating constraints in the graph coloring problem. Let us look at one more example of forward checking.

(Refer Slide Time: 48:23)



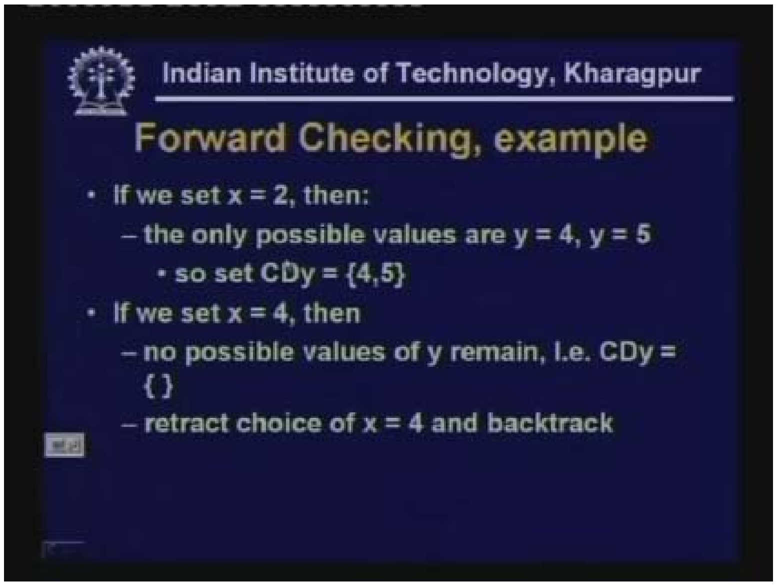
Indian Institute of Technology, Kharagpur

Forward Checking, example

- Variables x, y
- Dx = Dy = {1,2,3,4,5}
- Constraint x < y - 1
- Initially CDx = CDy = {1,2,3,4,5}

We have variables x and y. The domains of x and y are 1, 2, 3, 4, 5. The constraint is that x is less than y minus 1. So initially the current domain of both x and y are 1, 2, 3, 4, 5.

(Refer Slide Time: 48:37)

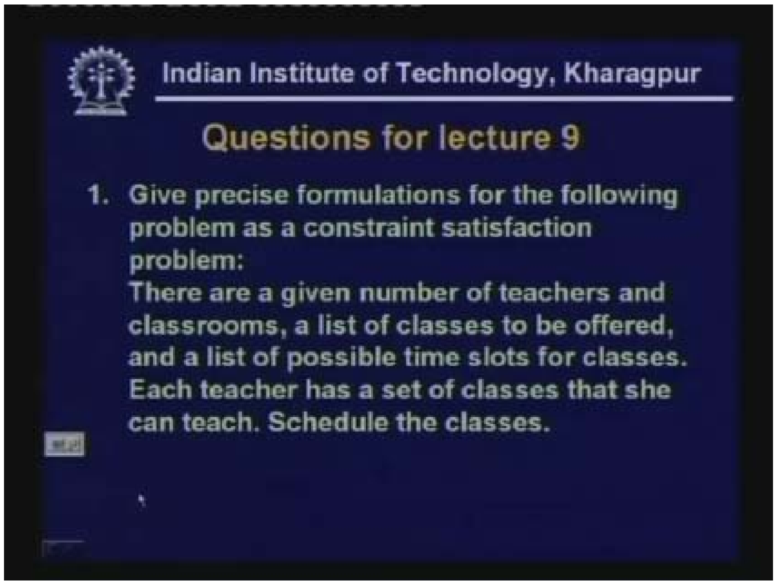

Indian Institute of Technology, Kharagpur

Forward Checking, example

- If we set x = 2, then:
  - the only possible values are y = 4, y = 5
    - so set CDy = {4,5}
- If we set x = 4, then
  - no possible values of y remain, I.e. CDy = { }
  - retract choice of x = 4 and backtrack

Suppose we set x is equal to 2 then y has to be greater than equal to x plus 1 so y can only be either 4 or 5 so cDy becomes 4 or 5. Now, if we set x is equal to 4 then cDy becomes null because there is no value in its domain. So, if we set x is equal to 4 we have to retract this choice and then backtrack.

(Refer Slide Time: 49:08)


Indian Institute of Technology, Kharagpur

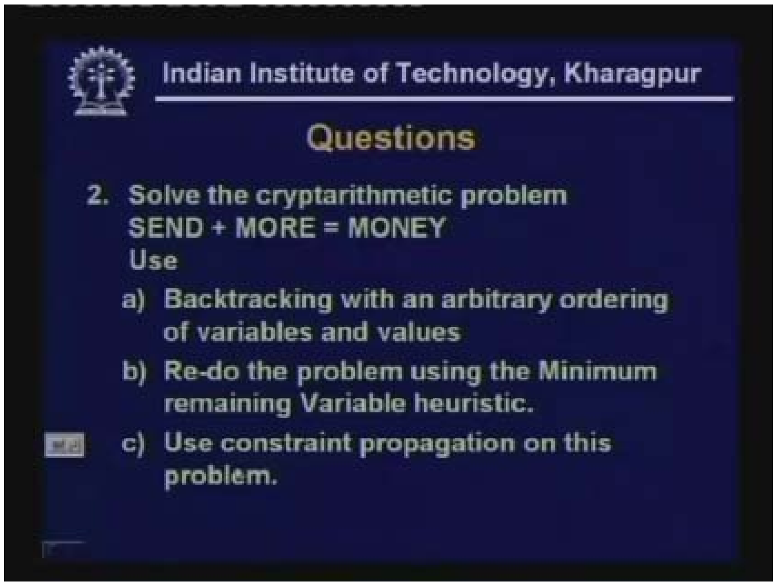
Questions for lecture 9

1. Give precise formulations for the following problem as a constraint satisfaction problem:
   There are a given number of teachers and classrooms, a list of classes to be offered, and a list of possible time slots for classes. Each teacher has a set of classes that she can teach. Schedule the classes.

I will end this lecture by setting two questions for you to consider answering.
1) Give precise formulations for the following problem as a constraint satisfaction problem.

This is the class time tabling problem. You are given a situation where there are a number of teachers and given number of classrooms. You have a list of courses that have to be offered and you are also given a list of time slots for these courses. Each teacher has a set of classes which the teacher can teach. Now your objective is to schedule the classes, schedule the courses to time slots, to teachers, to class rooms and so on. So you have to consider the different constraints in this problem and pose it as a constraint satisfaction problem and indicate how you will go about solving such problems.

(Refer Slide Time: 50:40)



Indian Institute of Technology, Kharagpur

Questions

2. Solve the cryptarithmetic problem
   SEND + MORE = MONEY
   Use
   a) Backtracking with an arbitrary ordering of variables and values
   b) Re-do the problem using the Minimum remaining Variable heuristic.
   c) Use constraint propagation on this problem.

2) You are given the following crypt arithmetic problem:
SEND plus MORE is equal to MONEY, and your job is to assign digits to s e n d to each of these characters so that each of these characters corresponds to unique digits m and s cannot be 0 and this arithmetic relation is satisfied. We have already seen how we can pose this as a constraint satisfaction problem. Your job is, firstly to solve this problem on paper using backtracking and for a particular ordering of the variables and values. You choose a particular ordering of the variables. And given with that particular ordering you draw the dfs tree and indicate where backtracking will occur.

The second part to this question, you have to do this problem again but this time you have to use the minimum remaining variable heuristic to choose which variable you should consider next. So use the minimum remaining variable heuristic and check if it has any effect on pruning of the search tree for this particular problem.

The third part to this question, you use forward checking of the type that we discussed today. We use constraint propagation or forward checking on this problem and see how you can work on it.