CS324: Constraint Satisfaction Problems. **Spring 2015**

Lecture 17: April 13, 2015

Lecturer: K.R. Chowdhary *: Professor of CS (VF)*

17.1 Introduction

In Constraint Satisfaction Problems(CSP) one need to search the state space, but every move is subject to the fulfillment of certain constraint, which is different from selection of best fitting state in best-first search or in hill-climbing. A general constraint satisfaction problem is known to be *NP-hard*. The constraints imposed on the previous states also propagates to next states. If there is violation of constraints, we need to backtrack and try other branches in the tree constructed.

Example 17.1 *Time-table for class-room teaching.*

Consider the problem of preparing a time-table for classes in a department, where there are courses, teachers, class-rooms, periods, and days of week. The teachers are assigned subjects, rooms are assigned classes, the time-slots in days and weeks (periods) are assigned the classes, teachers, and subjects, etc. It should be done to satisfy many things, like, all the teachers should get the courses of their choice (as far as possible), there is uniform distribution of teaching load among the teachers, no teacher should be assigned two classes at the same time of a day, neither it should happen with the room, or even two teachers teaching a class at the same time ! This is an example of Constraint satisfaction Problem. □

It is usually common, to encounter many similar problems to be solved, in job scheduling, supply chain management, CPU job scheduling, and other optimization problems.

Example 17.2 *Map Colouring Problem.*

Consider the case of boundary relations between states of Indian territory, where, Madhya-Pradesh (MP) is connected to Uttar-Pradesh (UP), Rajasthan (RAJ), Gujrat (GUJ), Maharastra (MAH), and Chattisgardh (CHG) all surrounded to MP, as shown in figure 17.1.

Assuming that we have been given a task of colouring each region either red, green, or blue in such a way that no neighbouring regions have the same color. To formulate this as a CSP, we define the variables to be the regions: MP, RAJ, GUJ, MAH, CHG, and UP. The domain of each variable is the set $\{red, green, blue\}$. The constraints require neighbouring regions to have distinct colors; for

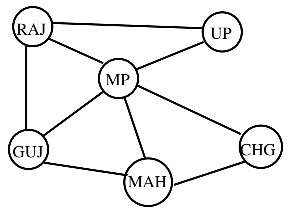


Figure 17.1: Cities connected as directed graph.

example, the allowable combinations for RAJ
and GUJ is set of pairs:

$$\{(red, green), (red, blue), (green, red),\\ (green, blue), (blue, red), (blue, green)\}$$

A constraint can also be represented as an inequality $RAJ \neq GUJ$, provided that constraint satisfaction algorithm has some way to evaluate such expressions. There is a solution for this problem as: $\{RAJ = red, GUJ = green, MAH = red, CHG = green, UP = red, MP = blue\}$. However, there are many other possible solutions. □

A constraint satisfaction problem is defined as a triple,

$$\langle V, D, C \rangle \tag{17.1}$$

where,

V is a finite set of variables;

D is a set (domain) of values (which may be finite or infinite); and

$C = \{c_1, c_2, \ldots, c_l\}$ is a finite set of constraints.

17.2 Constraints in CSP

A constraint can often be regarded as a mathematical equation, and the constraint network as a system of simultaneous equations. The solutions of simultaneous equations corresponds to the propagation of constraints. However, the constraints are not always restricted to equations, they can also be expressed as *inequalities* and non-numeric relationships.

Following are types of constraints:

1. *Unary Constraints*: Involves single variable, e.g., $MP \neq green$.
2. *Binary Constraints*: Constraints involve pair of variables, e.g., $MP \neq UP$.
3. *Higher order Constraints*: Contains the{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)} or more variables, e.g., cryptographic column constraints, and Professors A, B, C cannot be in the same committee together.
4. *Preference or Soft constraints:* For example, the red color is better than green, cost for each variable assignment, etc.

17.3 Solving a CSP

It is helpful to visualize a CSP as a constraint graph (network). The nodes of the graph correspond to variables of the problem and the edges correspond to constraints.

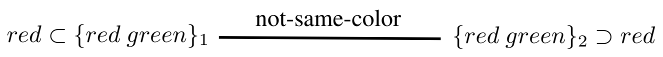
It is fairly easy to see that a CSP can be as a standard search problem as follows:

- *Initial state:* It is empty assignment {}, where all variables are unassigned.
- *Successor function:* A value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables, and does not violate constraint.
- *Goal test:* Whether the current assignment is complete?
- *Path cost:* A constant cost (e.g., 1) for every step.

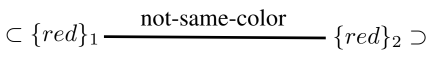
17.4 Synthesizing Constraints

A constraint expression of order n is a *conjunction* of constraints. Constraints are represented in the form of a network, where each variable is represented by a node and each constraint by a link or arc. Constraints are restricted to *unary* and *binary* constraints, i.e. predicates on one or two variables. For example, consider the problem of coloring a two-vertex complete graph using one color, say red. (A complete graph is one in which all possible edges between pairs of vertices are present.) The two-vertex complete graph contains two vertices, v_1 and v_2, and an edge between them. The coloring problem can be represented as a constraint satisfaction problem where variables X_1 and X_2 represent the colors of v_1 and v_2. There is a binary constraint, specifying that X_1 and X_2 are not the same color, and a unary constraint on both X_1 and X_2, requiring them to be red. Let us suppose we have two colours available in general, *red* and *green*; these form the initial domain of possible values for X_1 and X_2. The problem can be represented as a constraint network as given below.

The nodes $\{red\ green\}_1$ and $\{red\ green\}_2$ contain the possible values for X_1 and X_2. The loop at each node corresponds to the unary constraints, and the link between the nodes corresponds to the binary constraint. The first and most obvious level of inconsistency in constraint networks is *node inconsistency.* Say, the potential domain of values for X_1 and X_2 is given as red and green, but the unary constraints specify red. We can immediately eliminate green from both nodes, which results to the following network.


$red \subset \{red\ green\}_1$
not-same-color
$\{red\ green\}_2 \supset red$

The next level of inconsistency is *arc inconsistency.* The arc from X_1 to X_2 is inconsistent because for a value in X_1, namely "red," there does not exist any value a_2 in X_2 such that red and a_2 together satisfy the relation "red is not the same color as a_2." To remedy this inconsistency, remove red from X_1 and similarly from X_2. This cuts down the search space. Unfortunately, in this case it reflects the fact that the problem is impossible. There is no global solution, i.e. the network is what we call "unsatisfiable."



It is entirely possible for a network to have no arc inconsistencies, and still be unsatisfiable. Consider the problem of coloring a complete three-vertex graph with two colors, represented in figure 17.2.

Assume that set of possible values for each variable is $\{red\ green\}$ and the binary predicate between each pair again specifies "is not the same color as." This network is arc consistent, e.g. given a value "red" for X_1, we can choose "green" for X_2 : red is not the same color as

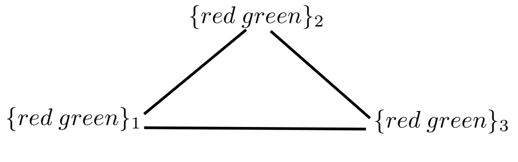


Figure 17.2: Arc and Path inconsistencies.

green. Yet obviously there is no way of choosing single values a_l, a_2, a_3, for X_1, X_2, and X_3, such that all three binary constraints are satisfied simultaneously. If we choose red, for X_1, for example, we are forced to choose green for X_2 to satisfy the constraint between X_1 and X_2. This forces a choice of red for X_3, which forces a choice of green for X_1, already picked to be red.

Each node must be so compared; however, comparisons can cause changes (deletions) in the network and so the comparisons must be iterated until a stable network is reached. These iterations can propagate constraints some distance through the network. The comparisons at each node can theoretically be performed in parallel and this parallel pass iterated.

Consider the problem of coloring the complete four-vertex graph with three colors (Figure 17.3). Each node contains red, green, and blue, and each arc again represents the constraint "is not the same color as." In particular, path consistency does not fully determine the set of values satisfying the global constraint, which in this inconsistent case is the empty set.

Example 17.3 *A Preliminary Example of the Synthesis Algorithm.*

we are given the following constraints on variables X_1, X_2, X_3: The unary constraint C_1 specifies that X_1 must be either a or b, i.e. $C_1 = \{ab\}$. Similarly $C_2 = \{e\ f\}$ and $C_3 = \{c\ d\ g\}$. The binary constraint on X_1 and X_2 specifies that either X_1 is b and X_2 is e, or X_1 is b and X_2 is f, $C_{12} = \{be\ bf\}$. Likewise, $C_{13} = \{bc\ bd\ bg\}$ and $C_{23} = \{ed\ fg\}$.

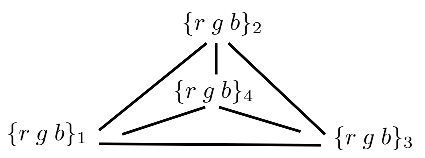


Figure 17.3: Network with path inconsistency.

We wish to determine what choices for X_1, X_2, X_3, if any, can simultaneously satisfy all these constraints. We begin building the constraint network with three nodes representing the unary constraints on the three variables. Next we add nodes representing the binary constraints, and link them to the unary constraints (e.g. $\{be\ bf\}_{12}$ represents C_{12}. The combined figure is shown as 17.4(a).

After we add and link node C_{12} we look at node C_1 and find that element a does not occur in any member of C_{12}. We delete a from C_1. Similarly, we delete c from C_3 after adding C_{23}. The constraint network now appears as in figure 17.4(b).

Now from C_3 we look at C_{13} and find that there is an element bc in C_{13} which requires c as a value for X_3, while c is no longer in C_3. We remove bc from C_{13}, as shown in figure 17.4(c). From this we note that the problem has two solutions as follows:

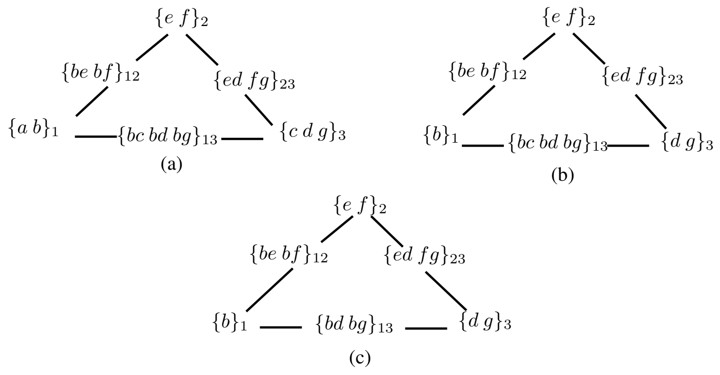


Figure 17.4: Applying Synthesis Algorithm.

X_1	X_2	X_3
b	e	d,
b	f	g.

17.5 CSP Algorithms

We will consider three solution methods for
constraint satisfaction problems: Generate-and-Test, Backtracking (possibly Dependency Directed), and Consistency Driven. Solving a CSP could mean to find:

- *One solution*, without preference as to which one,
- *All solutions*,
- An *optimal*, or at least a good solution.

The nature of methods for solving a CSP are:

1. **Combinatorial** methods for finite domain of values, D. The Solutions can be found by systematic search in D, either traversing the space of partial solutions, or explore the space of complete value assignments.
2. **Analytical** methods for infinite domain of values D. Solutions here can be found by analyzing the constraints as some (generalized) equation system, either solving the constraints simultaneously, or considering the constraints sequentially one by one.

A systematic search can be carried out in one of the two ways, either using *Generate and Test* approach or through *Back-Tracking.*

17.5.1 Generate and Test

We generate one by one all possible complete variable assignments and for each we test if it satisfies all constraints. The corresponding program structure is very simple, just nested loops, one per variable. In the innermost loop we test each constraint. In most situation this method is intolerably slow. The steps for generate-and-test is shown as algorithm 1.

Algorithm 1 Generate-and-Test

```
1: repeat
2:     Assign a value to each variable.
3:     if it is a solution then
4:         return Success
5:     else
6:         modify the assignment
7:     end if
8: until All assignments are done
9: return fail
```

17.6 Backtracking Algorithm

Backtracking is a *recursive algorithm*. It maintains a partial assignment of the variables. Initially, all variables are unassigned. At each step, a variable is chosen, and all possible values are assigned to it in turn. For each value, the consistency of the partial assignment with the constraints is checked; in case of consistency, a recursive call is performed. When all values have been tried, the algorithm backtracks. As backtrack is no more than an "educated" exhaustive search procedure; it should be stressed that there exist numerous problems which even the most sophisticated application of backtrack will not solve in a reasonable length of time.

The algorithm 2 shows the steps for backtracking.

Algorithm 2 Recursive-backtrack()

```
1: assignment={}
2: if assignment is complete then
3:     return assignment;
4: end if
5: $v$ = select-unassigned-variable();
6: for each value $d$ in order-of-domain-values do
7:     if value is consistent with assignment according to constraints then
8:         add {var := value} to assignment;
9:         result := Recursive-backtrack();
10:        if result is NOT failure then
11:            return result;
12:        else
13:            remove {var := value} from assignment list;
14:        end if
15:    end if
16: end for
17: return fail
```

17.7 Constraint Propagation

Example 17.4 *Constraint Propagation.*

A constraint graph is shown in figure 17.5, for the domain values $\{1, 2, 3, 4\}$. There are two unary constraints: (1) variable v_1 cannot take values $3, 4$, and (2) variable v_2 cannot take value 4. There are eight binary constraints, stating that variables connected by edges cannot have the same value. It is required to find the solution using the following heuristics: *minimum value heuristics* (MVH), *degree heuristics* (DH), and *forward-checking* (FC).

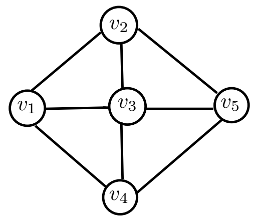


Figure 17.5: Constraint Graph.

Solution:

$MVH : v_1 = 1$

$FC + MVH : v_2 = 2$

$FC + MVH + DH : v_3 = 3$

$FC + MVH : v_4 = 4$

$FC : v_5 = 1$

The first variable v_1 is assigned the minimum value (i.e., 1) out of the available set of values in the start. The forward-checking and MVH assign next, minimum available value to v_2, while checking forward, i.e., v_3, v_5, that thee is no conflict. Next using the FC heuristics, MVH and DH, we assign 3 to v_3. Using FC and MVH we assign 4 to v_4. Finally, the variable v_5 is assigned 1, through forward checking.

17.8 Cryptarithmetics

Cryptarithmetic codes are typical of Constraint Satisfaction Problems that lie at the centre of studies of human and computer problem solving. The task is to find unique digit assignments to each of the letters so that numbers represented by words add up correctly.

Example 17.5 *Cryptographic puzzles.*

Given, a sum as below, find out the values of these 8-variables satisfying the validity of this sum, such that each variable gets a unique value from set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

```
  S E N D
+ M O R E
 ========
M O N E Y
```

For the above, the constraints can be formulated as follows:

$$\begin{aligned}
&c_1 : D + E = Y + 10 \times v_1 \\
&c_2 : v_1 + N + R = E + v_2 \times 10 \\
&c_3 : v_2 + E + O = N + v_3 \times 10 \\
&c_4 : v_3 + S + M = O + v_4 \times 10 \\
&c_5 : M = v_4
\end{aligned}$$

Since, $M \neq 0$, so, $S \neq 0$.

In the above, $c_1 \ldots c_5$ are constraints. The value range for variables is as follows:

$$\begin{aligned}&D, E, N, R, O, Y \in \{0,1,2,3,4,5,6,7,8,9\}\\&M, S \in \{1,2,3,4,5,6,7,8,9\}\\&v_1, v_2, v_3 \in \{0,1\}\end{aligned}$$

From above, we get straight forward, $v_4 = M = 1$. Thus, 1 can be removed from the value sets of all the other variables. hence:

$$\begin{aligned}&D, E, N, R, O, Y \in \{0,2,3,4,5,6,7,8,9\}\\&S \in \{2,3,4,5,6,7,8,9\}\\&c_4 : v_3 + S = O + 9.\end{aligned}$$

Assume that $v_3 = 1$.

$$\begin{aligned}&c_4 : v_3 + S = O + 9\\&S = O + 8.\end{aligned}$$

$\therefore S \in \{8, 9\}$. Since, $M = 1$ requires $O \neq 1$. So, $O = 0$, and $S = 8$.

$$\begin{aligned}&c_3 : v_2 + E + O = N + v_3 \times 10\\&\therefore v_2 + E = N + v_3 \times 10 = 10 + N.\end{aligned}$$

The above cannot be satisfied for any constraint value for E, since $E < 10$. This concludes that assumption $v_3 = 1$ was wrong. Hence, this and all the derived values must be reset, and we backtrack to select $v_3 = 0$.

$$\begin{aligned}&c_4 : v_3 + S = O + 9\\&S = O + 9.\end{aligned}$$

Therefore, the only solution is $S = 9$, and $O = 0$. C_3 is calculated as:

$$\begin{aligned}&c_3 : v_2 + E + O = N + v_3 \times 10\\&v_2 + E + 0 = N + 0 \times 10\\&\therefore v_2 + E = N.\end{aligned}$$

The assumption $v_2 = 0$ will leads to contradiction $E = N$. Hence, $v_2 = 1$. Therefore, $1 + E = N$. The c_2 reduces to:

$$\begin{aligned}
&c_2 : v_1 + N + R = E + v_2 \times 10\\
&v_1 + 1 + E + R = E + 1 \times 10\\
&v_1 + R = 9.
\end{aligned}$$

Because, S is already 9, R cannot be 9. Hence, $v_1 = 1$ and $R = 8$.

Let us try to propagate the set of values. The present sets are:

$$D, E, N, Y \in \{2, 3, 4, 5, 6, 7\} \tag{17.2}$$

From the constraint c_1, since $v_1 = 1$, we have,

$$c_1 : D + E = Y + 10 \times 1$$

That is,

$$D + E = Y + 10 \tag{17.3}$$

We note that already assigned values to variables are $\{S = 9, O = 0, M = 1, R = 8\}$. From equation 17.2 as well as equation 17.3, we note that $D + E \geq 12$, Y can be assigned value ≥ 2. Various possibilities are:

- $(D, E) = (6, 7)$. Thus, $1 + E = 1 + 7 = 8 = N$. This is not possible, because R is already 8.
- $(D, E) = (7, 6)$. Thus, $N = 1 + E = 7$. This fails, due to conflict with $D = 7$.
- $(D, E) = (7, 5)$. Thus, $N = 1 + E = 6$. $Y = D + E - 10 = 2$

The unique solution of the problem is give below.

```
  S E N D         9 5 6 7
+ M O R E       + 1 0 8 5
 =========    =>  =========
M O N E Y       1 0 6 5 2
```

$\square$