32002: AI (CSP: Constraint Satisfaction Problems) **Spring 2014**

Lecture 31-32: March 26-27, 2014

Instructor: K.R. Chowdhary *: Professor of CS*

31.1 Constraint Satisfaction Problems

In Constraint Satisfaction Problems(CSP), every move in state space is subject to fulfillment of certain constraint, which is different from selection of best fitting state in best-first search or in hill-climbing. For example, the standard propositional satisfiability problem may be viewed as a constraint satisfaction problem where the variables are to be assigned boolean values, with the constraints specified by clauses. 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. The constraints only specify the relationships without specifying a computational procedure to enforce that relationship.

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 31.1 *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 31.1. 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 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. □

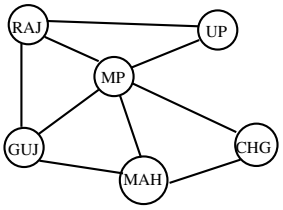

Figure 31.1: Cities connected as directed graph.

31.2 Representation of CSP

Using constraint satisfaction methods, many problems-especially in AI - can be represented in a very declarative way by identifying the variables of interest for the problem, laying-down the domains for the variables and restricting the variable assignments by constraints. More formally, a constraint network R consists of a finite set of $V = \{v_1, v_2, \ldots, v_n\}$ variables and a set $C = \{c_1, c_2, \ldots, c_l\}$ of constraints. Associated with each variable v is a finite, discrete domain D_v . A constraint c on the variable set $V_k \subseteq V$ is in its extensional form a subset of the Cartesian product of the domains of the afflicted variables. The expression $var(c)$ denotes the tuple of variables the constraint is defined on, and $rel(c) \subseteq X_v D_v$, and $v \in var(c)$ is the relational information of the constraint c . The assignment of a value $d \in D_v$ to a variable v is denoted by $v := d$. A tuple t of a assignments of variables $V_k \subseteq V$ satisfies a constraint c, if and only if $t[var(c)] \in rel(c)$.

Thus, a constraint satisfaction problem is defined as a triple,

$$\langle V, D, C \rangle \tag{31.1}$$

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.

Each constraint c_i is a pair (S_i, R_i) (usually represented as a matrix), where $S_i \subseteq V$ is a set of variables, called the constraint *scope*, and R_i is a set of (total) functions from S_i to D, called the constraint *relation* (an $n - ary$ relation). Thus an evaluation of variables is evaluation of a function u from the set of variables to the domain of values, $u : V \to D$. The evaluation function u satisfies a constraint $\langle (v_1, \ldots, v_n), R \rangle$ if $(u(v_1), \ldots, u(v_n)) \in R$. A solution is an evaluation that satisfies all constraints.

31.2.1 Variables in CSP

A constraint satisfaction problem can be represented as a *constraint graph*, where the vertices correspond to the variables and the edges to the binary constraints. The constraints specify which pairs of values are allowed. If all pairs are allowed, we say that there is no constraint. If we place the vertices in a linear order, we obtain an *ordered constraint graph*. Such an ordering corresponds to the levels of a backtrack search tree at which the variables are instantiated. The order in which a backtrack process chooses variables to instantiate is termed the *vertical order* of search, corresponding to depth in the backtrack tree.

Discrete Variables: For a finite domain with n variables, and domain size d, there are $O(d^n)$ total number of assignments. The example is Boolean Satisfiability problem, where there are n variables $(v_1, \ldots, v_n)$, and these can be assigned in total 2^n ways, for 8-queens, the complexity is $O(8^{64})$. However, for infinite domains, the data are integers, strings, etc.

Considering a CSP of *job scheduling*, where variables are start and end days of job. They need a constraint language like, $startjob_i + 5 \leq endjob_j$. There are infinitely many solutions for this problem. If the constraints are *linear*, then it is solvable, else there is no general algorithm.

Continuous variables: Building an airline schedule or class schedule uses counter variables. Linear constraints are solvable in polynomial time.

31.2.2 Constraints in CSP

Following are types of constraints:

1. *Unary*: Involves single variable, e.g., $MP \neq green$.
2. *Binary*: Constraints involve pair of variables, e.g., $MP \neq UP$.
3. *Higher order*: 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.

For the representation of constraints, such as the relationship between height and weight of a person, one of the following possibilities can be used:

- *Table*: as a lookup table.
- *Functions:* Normal weight = height (in cm) - 100.
- *Heuristics:* For example, the under weight can be specified as :

$$\frac{actual\ weight}{normal\ weight} < 0.8 \tag{31.2}$$

Propagation algorithm may be distinguished according to what can be propagated through a variable. It can be classified as:

- Fixed Value quantity: For example, $C = 5$.
- Set of values, for example, $C \in \{3, 4, 6, 7\}$.
- Symbolic expression, for example, $C = 2y$.

For propagation of fixed value or set, a simple *forward chaining* is sufficient as constraint strategy for rules. The application of symbolic expressions must be controlled heuristically.

31.3 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.

- *Initial state:* the 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.

The above steps are same for all CSPs. Every solution appears at depth n with n variables, this indicates that it is a depth-first search approach. The Path is irrelevant, so can also use complete-state formulation. For domain size $|D| = d$, the branching factor is,

$$b = (n - i)d \tag{31.3}$$

at depth i, hence the total number of leaves are only $n! \cdot d^n$.

31.3.1 Synthesizing Constraint Expressions

Apart from node level consistency, where a variable is assigned a value, 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 31.2.

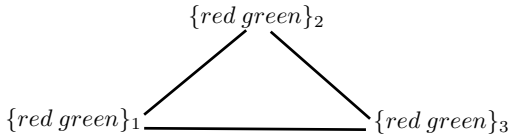

Figure 31.2: Arc and Path inconsistencies.

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 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.

Nevertheless, it may be helpful to remove arc inconsistencies from a network. This involves comparing nodes with their neighbors as we did above. 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.

Thus removing arc inconsistencies involves several distinct ideas: local constraints are globally propagated through iteration of parallel local operations. It remains to be seen which aspects of this process are most significant to its application. The parallel possibilities may prove to be particularly important.

A more powerful notion of inconsistency, called as *path inconsistency*, where a network is path inconsistent if there are two nodes X_1 and X_2 such that a satisfies X_1, b satisfies X_2, a and b together satisfy the binary constraint between them, yet there is some other path through the network from X_1 to X_2, such that there is no set of values, one for each node along the path, which includes a and b, and can simultaneously satisfy all constraints along the path. For example, the network in Figure 31.2 is path inconsistent: red satisfies X_1, green X_3, red is not the same color as green: however, there is no value for X_2 which will satisfy the constraints between X_1 and X_2, and between X_2 and X_3, while X_1 is red, X_3 is green.

Consider the problem of coloring the complete four-vertex graph with three colors (Figure 31.3). 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.

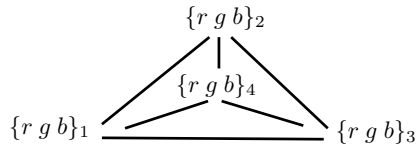

Figure 31.3: Network with path inconsistency.

In summary, arc and path consistency algorithms may reduce the search space, but do not in general fully synthesize the global constraint. When there are multiple solutions, additional search will be required to specify the several acceptable combinations of values. Even a unique solution may require further search to determine, and the consistency algorithms may even fail to reveal that no solutions at all exist.

31.3.2 An Extended Theory Synthesizing

As the coloring problem suggests, the general problem of synthesizing the global constraint is NP-complete, and thus unlikely to have an efficient (polynomial time) solution. On the other hand it is found that in specific applications it may be possible to greatly facilitate the search for solutions.

There are two key observations that motivated this approach.

1. Node, arc, and path consistency in a constraint network for n variables can be generalized to a concept of k-consistency for any $k \leq n$.

2. The given constraints can be represented by nodes, as opposed to links, in a constraint network; we can add nodes representing k-ary constraints to a constraint net-work for all $k \leq n$ (whether or not a corresponding k-ary constraint is given); and we can then propagate these constraints in this augmented net to obtain higher levels of consistency.

Example 31.2 *A Preliminary Example of the Synthesis Algorithm.*

We consider a crude example of the synthesis algorithm in operation, by way of motivation for the formal description which follows. Suppose 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 $C_{23} = \{ed\ fg\}$.

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 31.4(a).

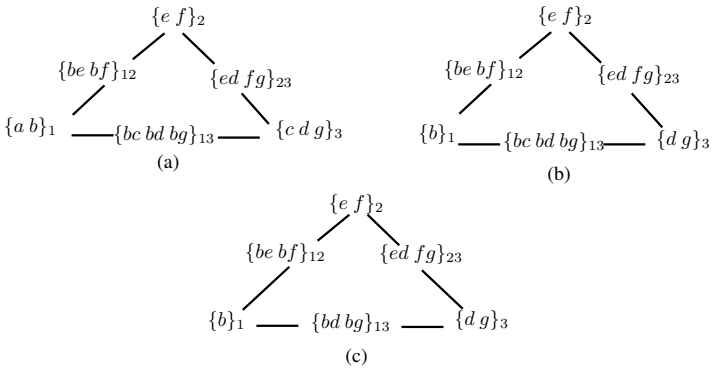

Figure 31.4: Synthesis Algorithm.

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 31.4(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 31.4(c). From this we note that the problem has two solutions as follows:

X_1	X_2	X_3
b	e	d,
b	f	g.

But, but the consistency of all the variables have not been simultaneously verified yet, which is covered using the concept of *augmented network*. □

Augmented Network: So far we have merely achieved a sort of "arc consistency" (though we indicate the restriction of the pair bc (in C_{13}), as well as the elements a and c).

Next, we add a node for the *ternary* constraint (i.e., k-ary constant $k \leq n$. $k = 3$, $n = 3$). No order-three constraint was given originally, so we could assume initially the "nonconstraint," all possible triples. However, we will take advantage of the restrictions available from the binary and unary predicates to construct a more limited set of possibilities. C_1 and C_{23} together allow only the following set of triples: $\{bed\ bfg\}$. We use this as the ternary node and link it to the binary nodes as shown in figure 31.5, which shows the *augmented* network.

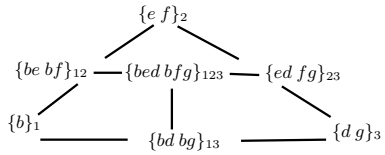

Figure 31.5: Synthesis Algorithm.

We look at the new node from its neighbors and vice versa, as we did earlier, to insure consistency of the sort we obtained earlier between neighboring nodes. C_{13} is consistent with the new node: bd is part of bed, bg part of bfg. Similarly C_{12} and C_{23} are consistent with the ternary, node. If necessary, we could propagate deletions around until local consistency is achieved on this augmented network. However, in this case, the network is already stable; no further changes are required. The ternary node represents the synthesis of the given constraints. There are two ways to simultaneously satisfy the given constraints: $X_1 = b$, $X_2 = e$, $X_3 = d$; or $X_1 = b$, $X_2 = f$, $X_3 = g$.

31.4 CSP Algorithms

31.4.1 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
```

The algorithm searches all of the domain of values, D. However, improvements are possible, like, use an informed/smart generator such that the conflicts found by the tester are minimized. The example is, *stochastic* algorithms. Merging the generator with the tester results to a new algorithm, called *backtracking* algorithm.

31.4.2 Backtracking

Using a computer to answer a question such as "How many ways are there to...,", "list all possible ...", usually requires an exhaustive search of the set of all potential solutions. One general technique for organizing such searches is backtrack, which works by continually trying to extend a partial solution. At each stage of

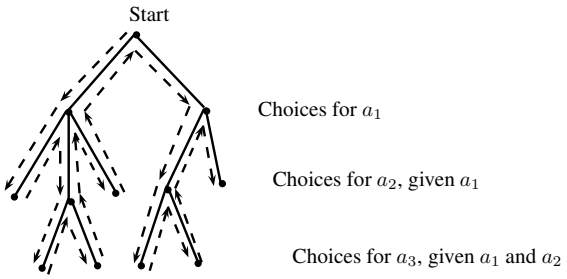

Figure 31.6: Backtrack-tree searches as depth-first search.

the search, if an extension of the current partial solution is not possible, we *backtrack* to a shorter partial solution and try again. Currently, this method is used in a wide range of combinatorial problems including, for example, parsing, game playing, and optimization.

It is helpful to picture the backtrack process in terms of a tree traversal in figure 31.6. The subset of $A_1 \times A_2 \times \times ... \times A_i$, $i = 0, 1, 2, ...$ which is searched is represented as a tree as follows: The root of the tree (the 0th level) is the null vector. Its sons are the choices for a a_1, and general the nodes at the k^{th} level are the choices for a_k, given the choices made for $a_1, a_2, ..., a_{k-1}$ as indicated by the ancestors of these nodes. In the tree shown, backtrack traverses the nodes of the tree as shown by the dotted lines. Since the traversal goes as deep as possible in the tree before backing up to explore other parts of the tree, such a search is called depth-first.

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
```

Example 31.3 *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:

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

and

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

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 $S = 8$.

$$c_3 : v_2 + E + O = N + v_3 \times 10$$
$$\therefore v_2 + E = N + v_3 \times 10 = 10 + N.$$

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 $v_3 = 0$.

$$c_4 : v_3 + S = O + 9$$
$$S = O + 9.$$

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

$$c_3 : v_2 + E + O = N + v_3 \times 10$$
$$v_2 + E + 0 = N + 0 \times 10$$
$$\therefore v_2 + E = N.$$

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

$$c_2 : v_1 + N + R = E + v_2 \times 10$$
$$v_1 + 1 + E + R = E + 1 \times 10$$
$$v_1 + R = 9.$$

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:

$$D, E, N, Y \in \{2, 3, 4, 5, 6, 7\} \tag{31.4}$$

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

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

That is,

$$D + E = Y + 10 \tag{31.5}$$

We note that already assigned values to variables are $\{S = 9, O = 0, M = 1, R = 8\}$. From equation 31.4 as well as equation 31.5, 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
```

□

References

[1] Chowdhary K.R. (2020) Logic and Reasoning Patterns. In: Fundamentals of Artificial Intelligence. Springer, New Delhi. https://doi.org/10.1007/978-81-322-3972-7_10