Learning Objectives

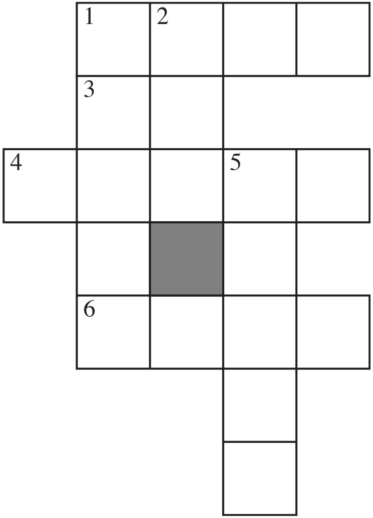
At the end of the class you should be able to:

- recognize and represent constraint satisfaction problems
- show how constraint satisfaction problems can be solved with search
- implement and trace arc-consistency of a constraint graph
- show how domain splitting can solve constraint problems

Constraint satisfaction

- A Constraint Satisfaction problem consists of:
  - a set of variables
  - a set of possible values, a domain for each variable
  - a set of constraints amongst subsets of the variables
- The aim is to find a set of assignments that satisfies all constraints, or to find all such assignments.

Example: crossword puzzle



at, be, he, it, on,
eta, hat, her, him,
one,
desk, dove, easy,
else, help, kind,
soon, this,
dance, first, fuels,
given, haste, loses,
sense, sound, think,
usage

Dual Representations

Two ways to represent the crossword as a CSP

- First representation:
  - nodes represent word positions: 1-down. . . 6-across
  - domains are the words
  - constraints specify that the letters on the intersections must be the same.
- Dual representation:
  - nodes represent the individual squares
  - domains are the letters
  - constraints specify that the words must fit

Posing a Constraint Satisfaction Problem

A CSP is characterized by

dom(V) in the book

- A set of variables $V_1, V_2, \ldots, V_n$.
- Each variable V_i has an associated domain $\mathbf{D}_{V_i}$ of possible values.
- There are hard constraints on various subsets of the variables which specify legal combinations of values for these variables.
- A solution to the CSP is an assignment of a value to each variable that satisfies all the constraints.

Example: scheduling activities

- Variables: A, B, C, D, E that represent the starting times of various activities.
- Domains: $\mathbf{D}_A = \{1,2,3,4\}$, $\mathbf{D}_B = \{1,2,3,4\}$, $\mathbf{D}_C = \{1,2,3,4\}$, $\mathbf{D}_D = \{1,2,3,4\}$, $\mathbf{D}_E = \{1,2,3,4\}$
- Constraints:

$$
\begin{aligned}
&(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge \\
&\quad (C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge \\
&\quad (E < C) \wedge (E < D) \wedge (B \neq D).
\end{aligned}
$$

Generate-and-Test Algorithm

- Generate the assignment space $\mathbf{D} = \mathbf{D}_{V_1} \times \mathbf{D}_{V_2} \times \ldots \times \mathbf{D}_{V_n}$. Test each assignment with the constraints.
- Example:

$$\begin{aligned}
\mathbf{D} &= \mathbf{D}_A \times \mathbf{D}_B \times \mathbf{D}_C \times \mathbf{D}_D \times \mathbf{D}_E \\
&= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \\
&\quad \times \{1,2,3,4\} \times \{1,2,3,4\} \\
&= \{\langle 1,1,1,1,1\rangle, \langle 1,1,1,1,2\rangle, ..., \langle 4,4,4,4,4\rangle\}.
\end{aligned}$$

- How many assignments need to be tested for n variables each with domain size d?

Backtracking Algorithms

- Systematically explore **D** by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound
- any partial assignment that doesn't satisfy the constraint can be pruned.

Example Assignment $A = 1 \land B = 1$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables.

CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node N is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable Y that isn't assigned in N.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

Example 4.13 Suppose you have a CSP with the variables A, B, and C, each with domain $\{1,2,3,4\}$. Suppose the constraints are $A < B$ and $B < C$. A possible search tree is shown in Figure 4.1 (on the next page). In this figure, a node corresponds to all of the assignments from the root to that node. The potential nodes that are pruned because they violate constraints are labeled with ✘.

The leftmost ✘ corresponds to the assignment $A = 1$, $B = 1$. This violates the $A < B$ constraint, and so it is pruned.

This CSP has four solutions. The leftmost one is $A = 1$, $B = 2$, $C = 3$. The size of the search tree, and thus the efficiency of the algorithm, depends on which variable is selected at each time. A static ordering, such as always splitting on A then B then C, is less efficient than the dynamic ordering used here. The set of answers is the same regardless of the variable ordering.

In the preceding example, there would be $4^3 = 64$ assignments tested in a generate-and-test algorithm. For the search method, there are 22 assignments generated.

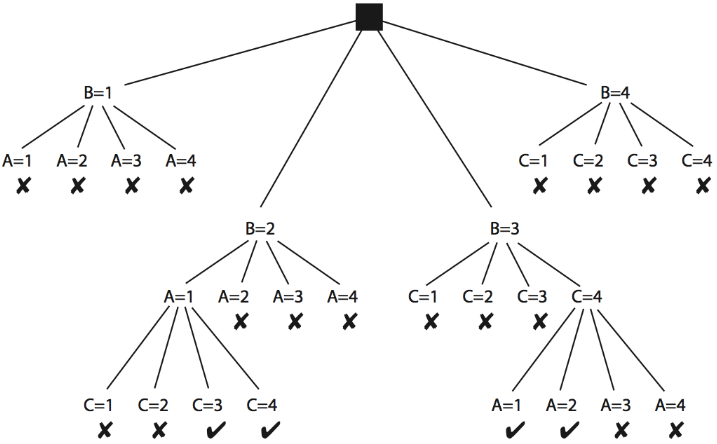


Figure 4.1: Search tree for the CSP of Example 4.13

Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?

Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?
  $\mathbf{D}_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.

Constraint Network

- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X.

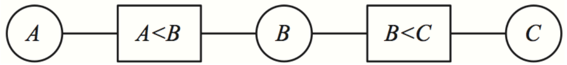
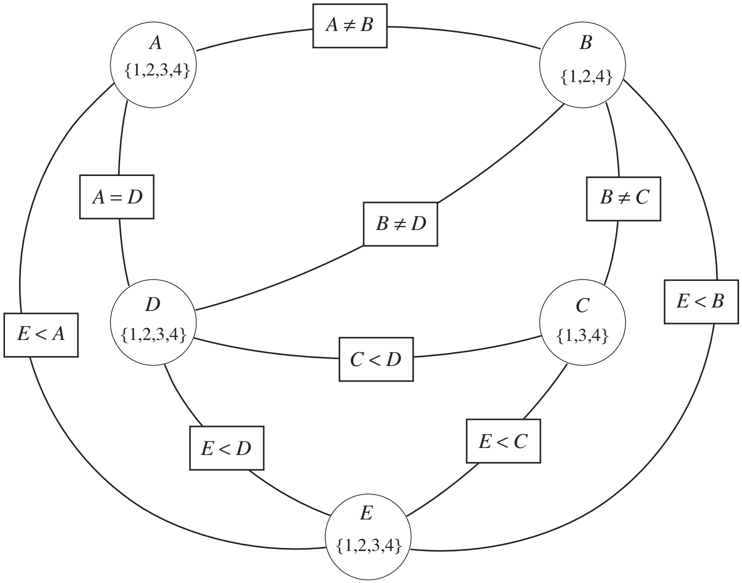


Figure 4.2: Constraint network for the CSP of Example 4.15

Example 4.15 Consider Example 4.13 (page 119). There are three variables A, B, and C, each with domain $\{1,2,3,4\}$. The constraints are $A < B$ and $B < C$. In the constraint network, shown in Figure 4.2, there are four arcs:

$\langle A, A < B\rangle$
$\langle B, A < B\rangle$
$\langle B, B < C\rangle$
$\langle C, B < C\rangle$

Example Constraint Network



Arc Consistency

- An arc $\langle X, r(X, \overline{Y})\rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What if arc $\langle X, r(X, \overline{Y})\rangle$ is *not* arc consistent?

Arc Consistency

- An arc $\langle X, r(X, \overline{Y}) \rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What if arc $\langle X, r(X, \overline{Y}) \rangle$ is *not* arc consistent? All values of X in $dom(X)$ for which there is no corresponding value in $dom(\overline{Y})$ can be deleted from $dom(X)$ to make the arc $\langle X, r(X, \overline{Y}) \rangle$ consistent.

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?
  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$
  - Each domain has a single value $\Longrightarrow$
  - Some domains have more than one value $\Longrightarrow$

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?
  An arc $\langle X, r(X, \overline{Y})\rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$ no solution
  - Each domain has a single value $\Longrightarrow$ unique solution
  - Some domains have more than one value $\Longrightarrow$ there may or may not be a solution

Example 4.21 Suppose there are three variables, A, B and C, each with the domain $\{1, 2, 3\}$. Consider the constraints $A = B$, $B = C$, and $A \neq C$. This is arc consistent: no domain can be pruned using any single constraint. However, there are no solutions. There is no assignment to the three variables that satisfies the constraints.

```
1:  procedure $GAC(V, dom, C)$
2:      Inputs
3:          $V$: a set of variables
4:          $dom$: a function such that $dom(X)$ is the domain of variable $X$
5:          $C$: set of constraints to be satisfied
6:      Output
7:          arc-consistent domains for each variable
8:      Local
9:          $D_X$ is a set of values for each variable $X$
10:         $TDA$ is a set of arcs
11:     for each variable $X$ do
12:         $D_X \leftarrow dom(X)$
13:         $TDA \leftarrow \{\langle X, c\rangle \mid c \in C$ and $X \in scope(c)\}$
14:     while $TDA \neq \{\}$ do
15:         select $\langle X, c\rangle \in TDA$;
16:         $TDA \leftarrow TDA \setminus \{\langle X, c\rangle\}$;
17:         $ND_X \leftarrow \{x \mid x \in D_X$ and some $\{X = x, Y_1 = y_1, \ldots, Y_k = y_k\} \in c$ where $y_i \in D_{Y_i}$ for all $i\}$
18:         if $ND_X \neq D_X$ then
19:             $TDA \leftarrow TDA \cup \{\langle Z, c'\rangle \mid X \in scope(c')$, $c'$ is not $c$, $Z \in scope(c') \setminus \{X\}\}$
20:             $D_X \leftarrow ND_X$
21:     return $\{D_X \mid X$ is a variable$\}$
```

Figure 4.3: Generalized arc consistency algorithm

Example 4.19 Consider the algorithm *GAC* operating on the network from Example 4.15. Initially, all of the arcs are in the *TDA* set. Here is one possible sequence of selections of arcs:

- Suppose the algorithm first selects the arc $\langle A, A < B \rangle$. For $A = 4$, there is no value of B that satisfies the constraint. Thus, 4 is pruned from the domain of A. Nothing is added to *TDA* because there is no other arc currently outside *TDA*.
- Suppose that $\langle B, A < B \rangle$ is selected next. The value 1 can be pruned from the domain of B. Again no element is added to *TDA*.
- Suppose that $\langle B, B < C \rangle$ is selected next. The value 4 can be removed from the domain of B. Because the domain of B has been reduced, the arc $\langle A, A < B \rangle$ must be added back into the *TDA* set because the domain of A could potentially be reduced further now that the domain of B is smaller.
- If the arc $\langle A, A < B \rangle$ is selected next, the value $A = 3$ can be pruned from the domain of A.
- The remaining arc on *TDA* is $\langle C, B < C \rangle$. The values 1 and 2 can be removed from the domain of C. No arcs are added to *TDA* and *TDA* becomes empty.

The algorithm then terminates with $D_A = \{1,2\}$, $D_B = \{2,3\}$, $D_C = \{3,4\}$. Although this has not fully solved the problem, it has greatly simplified it.

Example 4.20 Consider applying *GAC* to the scheduling problem of Example 4.8 (page 117). The network shown in Figure 4.4 (on the next page) has already been made domain consistent (the value 3 has been removed from the domain of B and 2 has been removed from the domain of C). Suppose arc $\langle D, C < D \rangle$ is considered first. The arc is not arc consistent because $D = 1$ is not consistent with any value in D_C, so 1 is deleted from D_D. D_D becomes $\{2,3,4\}$ and arcs $\langle A, A = D \rangle$, $\langle B, B \neq D \rangle$, and $\langle E, E < D \rangle$ could be added to *TDA* but they are on it already.

Suppose arc $\langle C, E < C \rangle$ is considered next; then D_C is reduced to $\{3,4\}$ and arc $\langle D, C < D \rangle$ goes back into the *TDA* set to be reconsidered.

Suppose arc $\langle D, C < D \rangle$ is next; then D_D is further reduced to the singleton $\{4\}$. Processing arc $\langle C, C < D \rangle$ prunes D_C to $\{3\}$. Making arc $\langle A, A = D \rangle$ consistent reduces D_A to $\{4\}$. Processing $\langle B, B \neq D \rangle$ reduces D_B to $\{1,2\}$. Then arc $\langle B, E < B \rangle$ reduces D_B to $\{2\}$. Finally, arc $\langle E, E < B \rangle$ reduces D_E to $\{1\}$. All arcs remaining in the queue are consistent, and so the algorithm terminates with the *TDA* set empty. The set of reduced variable domains is returned. In this case, the domains all have size 1 and there is a unique solution: $A = 4$, $B = 2$, $C = 3$, $D = 4$, $E = 1$.

Finding solutions when AC finishes

- If some domains have more than one element $\Longrightarrow$ search
- Split a domain, then recursively solve each half.
- It is often best to split a domain in half.
- Do we need to restart from scratch?

One effective way to solve a CSP is to use arc consistency to simplify the network before each step of domain splitting. That is, to solve a problem,

- simplify the problem using arc consistency; and,
- if the problem is not solved, select a variable whose domain has more than one element, split it, and recursively solve each case.

One thing to notice about this algorithm is that it does not require a restart of the arc consistency from scratch after domain splitting. If the variable X has its domain split, *TDA* can start with just the arcs that are possibly no longer arc consistent as a result of the split; these are only the arcs of the form $\langle Y, r\rangle$, where X appears in r and Y is a variable, other than X, that appears in constraint r.

Example 4.22 In Example 4.19 (page 122), arc consistency simplified the network, but did not solve the problem. After arc consistency had completed, there were multiple elements in the domains. Suppose B is split. There are two cases:

- $B = 2$. In this case $A = 2$ is pruned. Splitting on C produces two of the answers.
- $B = 3$. In this case $C = 3$ is pruned. Splitting on A produces the other two answers.

This search tree should be contrasted with the search tree of Figure 4.1 (page 120). The search space with arc consistency is much smaller, and it was not as sensitive to the selection of variable orderings. [Figure 4.1 (page 120) would be much bigger with different variable orderings.]

Hard and Soft Constraints

- Given a set of variables, assign a value to each variable that either
  - satisfies some set of constraints: satisfiability problems — "hard constraints"
  - minimizes some cost function, where each assignment of values to variables has some cost: optimization problems — "soft constraints"
- Many problems are a mix of hard and soft constraints (called constrained optimization problems).