Constraint Satisfaction Problems

- Given a set of variables, each with a set of possible values (a domain), 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.

Relationship to Search

- The path to a goal isn't important, only the solution is.
- Many algorithms exploit the multi-dimensional nature of the problems.
- There are no predefined starting nodes.
- Often these problems are huge, with thousands of variables, so systematically searching the space is infeasible.
- For optimization problems, there are no well-defined goal nodes.

Posing a Constraint Satisfaction Problem

A CSP is characterized by

- 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.
- For satisfiability problems, there are constraint relations on various subsets of the variables which give legal combinations of values for these variables.
- A solution to the CSP is an n-tuple of values for the variables that satisfies all the constraint relations.

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:

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

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:

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

- Generate-and-test is always exponential in the number of variables.

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 represented as a graph-searching algorithm:

- 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 $y_i \in dom(Y)$ there is a neighbour
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ if this assignment is consistent with the constraints on these variables.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

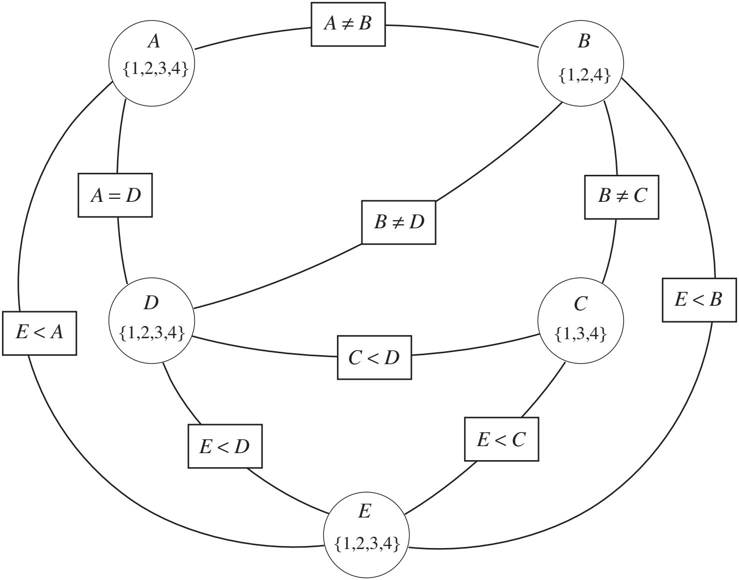
Consistency Algorithms

- 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: $\mathbf{D}_B = \{1, 2, 3, 4\}$ 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 relation.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each relation that involves X.

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.
- If an 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 $dom(\overline{Y})$ may 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.
- 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 arc consistent):
  - 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

Finding solutions when AC finishes

- If some domains have more than one element $\Longrightarrow$ search
- Split a domain, then recursively solve each half.
- We only need to revisit arcs affected by the split.
- It is often best to split a domain in half.