Constraint Satisfaction Problems

- **Multi-dimensional Selection 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:

$$(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge$$
$$(C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge$$
$$(E < C) \wedge (E < D) \wedge (B \neq D).$$

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:

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

Generate-and-test is always exponential.

Backtracking Algorithms

Systematically explore **D** by instantiating the variables in some order and evaluating 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 $A = 1 \wedge B = 1$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables.

CSP as Graph Searching

A CSP can be seen as a graph-searching algorithm:

- Totally order the variables, $V_1, \ldots, V_n$.
- A node assigns values to the first j variables.
- The neighbors of node $\{V_1/v_1, \ldots, V_j/v_j\}$ are the consistent nodes $\{V_1/v_1, \ldots, V_j/v_j, V_{j+1}/v_{j+1}\}$ for each $v_{j+1} \in \mathbf{D}_{V_{j+1}}$.
- 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 $B \neq 3$.

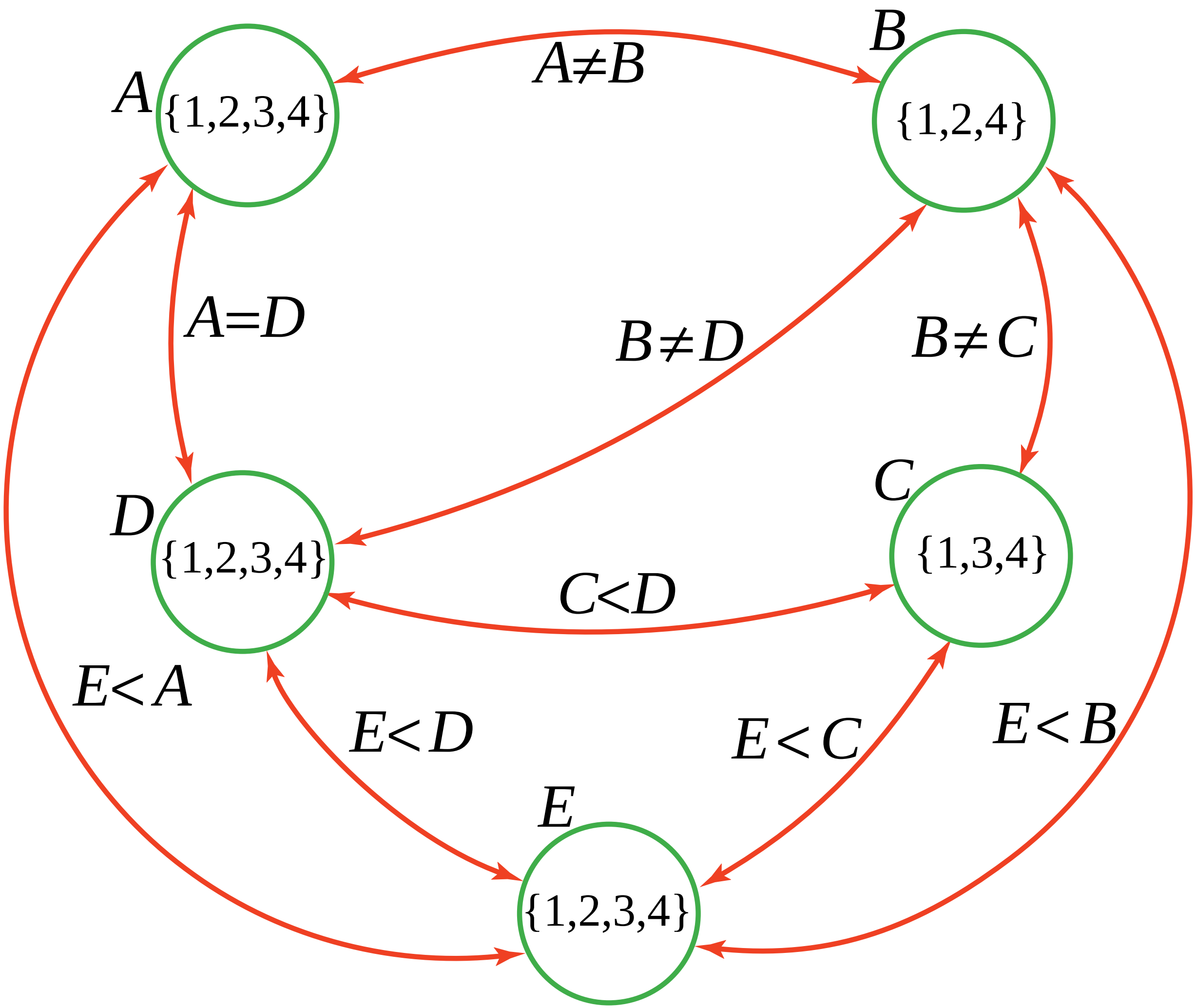
Arc Consistency

- A **constraint network** has nodes corresponding to variables with their associated domain. Each constraint relation $P(X, Y)$ corresponds to arcs $\langle X, Y \rangle$ and $\langle Y, X \rangle$.

- An arc $\langle X, Y \rangle$ is **arc consistent** if for each value of X in $\mathbf{D}_X$ there is some value for Y in $\mathbf{D}_Y$ such that $P(X, Y)$ is satisfied. A network is arc consistent if all its arcs are arc consistent.

- If an arc $\langle X, Y \rangle$ is *not* arc consistent, all values of X in $\mathbf{D}_X$ for which there is no corresponding value in $\mathbf{D}_Y$ may be deleted from $\mathbf{D}_X$ to make the arc $\langle X, Y \rangle$ consistent.

Example Constraint Network



Arc Consistency Algorithm

The arcs can be considered in turn making each arc consistent.

An arc $\langle X, Y\rangle$ needs to be revisited if the domain of Y 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 vallue $\Longrightarrow$ 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.