1 Constraint Satisfaction Problems

In standard search problems (like pathfinding), we care about the sequence of actions (the path) to reach a goal. In **Constraint Satisfaction Problems (CSPs)**, the path is irrelevant. We only care about the **goal state** itself.

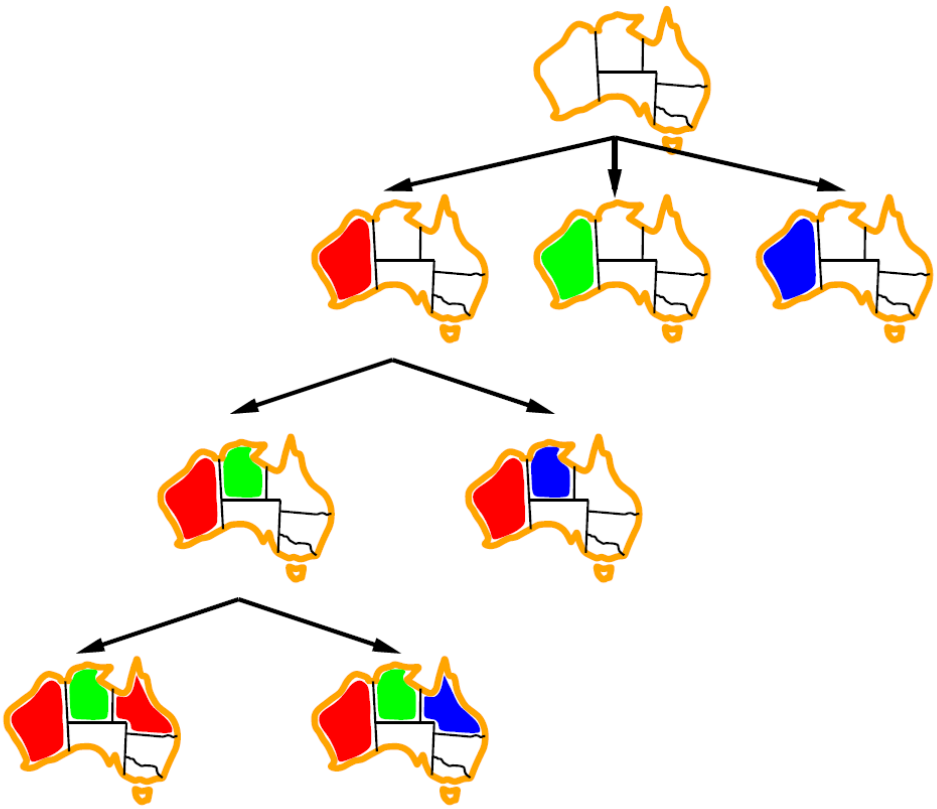
Definition: CSP

A CSP is defined by three components:

- **Variables (X):** A set of variables $\{X_1, \ldots, X_n\}$.
- **Domains (D):** A set of domains $\{D_1, \ldots, D_n\}$, where each variable X_i has a set of possible values D_i.
- **Constraints (C):** A set of constraints specifying allowable combinations of values for subsets of variables.

1.1 Types of Assignments

- **Partial Assignment:** Values are assigned to only some variables.
- **Consistent (Legal) Assignment:** An assignment that does not violate any constraints.
- **Complete Assignment:** Every variable is assigned a value.
- **Solution:** A consistent, complete assignment.



1.2 Constraint Graphs

Problems are often visualized as a **Constraint Graph**:

- **Nodes:** Represent variables.

- **Edges:** Represent constraints between variables.

This abstraction helps us understand the structure of the problem (e.g., independent subproblems).

1.3 Types of Constraints

1. **Unary Constraint:** Restricts the value of a single variable (e.g., $SA \neq$ green).
2. **Binary Constraint:** Relates two variables (e.g., $SA \neq WA$).
3. **Higher-order Constraint:** Involves 3 or more variables (e.g., Cryptarithmetic puzzles like $TWO + TWO = FOUR$).
4. **Soft Constraints (Preferences):** Constraints that are not binding but preferred (e.g., "Red is better than Green"). These turn the CSP into a **Constrained Optimization Problem**.

1.4 Solving CSPs: Search Strategies

We can view CSPs as a search problem where the initial state is empty, and the successor function assigns a value to an unassigned variable.

1.4.1 Naïve Search vs. Commutativity

A naïve Breadth-First or Depth-First search would branch on every variable and every value, leading to a massive search space ($n! \cdot v^n$). However, CSP assignments are **commutative**. The order in which we assign variables does not matter ($WA =$ red then $NT =$ green is the same state as $NT =$ green then $WA =$ red).

- **Implication:** We only need to consider assigning a value to *one* variable at each node.
- This reduces the search space to v^n (where v is domain size, n is number of variables).

1.4.2 Backtracking Search

Backtracking is the fundamental uninformed search algorithm for CSPs. It performs a Depth-First Search (DFS).

Backtracking Logic

1. Select an unassigned variable.
2. Try a value from its domain.
3. If the value is consistent with current assignments, proceed recursively.
4. If a conflict arises, **backtrack** (undo the assignment and try the next value).
5. If all values fail, return failure.

1.5 Heuristics: Improving Backtracking

To make backtracking efficient, we need to make smart decisions about *which* variable to assign next and *which* value to try first.

1.5.1 Variable Selection Heuristics (Which variable next?)

1. **Minimum Remaining Values (MRV):**
   - Also known as the "Fail-First" heuristic.
   - **Rule:** Choose the variable with the *fewest* legal values remaining in its domain.
   - **Reasoning:** If a variable has only 1 option left, we should assign it immediately to detect inevitable failures early.
2. **Degree Heuristic:**
   - Used as a tie-breaker for MRV.

- **Rule:** Choose the variable involved in the largest number of constraints on *unassigned* variables.
- **Reasoning:** Assigning this variable reduces the branching factor of future steps the most.

1.5.2 Value Selection Heuristics (Which value first?)

1. **Least Constraining Value (LCV):**
   - **Rule:** Given a variable, choose the value that rules out the *fewest* choices for the neighboring variables.
   - **Reasoning:** We want to leave maximum flexibility for subsequent assignments to find a solution.

1.6 Constraint Propagation

Search implies "trying" values. Constraint propagation implies "inferring" which values are impossible and removing them before search to reduce the search space.

1.6.1 Levels of Consistency

- **Node Consistency:** Every single variable satisfies its unary constraints.
- **Arc Consistency (2-Consistency):** A variable X is arc-consistent with respect to Y if for every value $x \in D_X$, there exists some value $y \in D_Y$ that satisfies the binary constraint.
- **Path Consistency (3-Consistency):** Looks at triples of variables.
- **k-Consistency:** Generalization to k variables. Checking high k is computationally expensive ($O(d^k)$).

1.6.2 Algorithms for Propagation

Forward Checking When variable X is assigned a value:

1. Look at all unassigned neighbors Y.
2. Remove any values from D_Y that conflict with the assignment of X.
3. If any domain becomes empty, backtrack immediately.

Limitation: It only checks immediate consequences of an assignment. It fails to detect failures further down the chain (e.g., if two neighbors are forced to take the same value, Forward Checking won't see it until one is assigned).

Arc Consistency (AC-3 Algorithm) This is more powerful than Forward Checking. It propagates constraints through the whole graph until stability.

AC-3 Algorithm

1. Initialize a **Queue** with all arcs (constraints) in the CSP.
2. While the Queue is not empty:
   - Remove an arc (X_i, X_j).
   - Check if X_i is consistent with X_j.
   - If we remove a value from D_i to make it consistent:
     - We must re-check all neighbors of X_i (add arcs (X_k, X_i) back to the Queue).

1.7 Local Search for CSPs

Instead of building a partial assignment (constructive search), Local Search starts with a complete (but likely inconsistent) assignment and tries to repair it.

- **State:** Complete assignment of all variables.
- **Goal:** Eliminate all violated constraints.
- **Min-Conflicts Heuristic:**

1. Choose a random variable that is currently involved in a conflict.
2. Change its value to the one that violates the *fewest* constraints (minimizes conflicts).

This is surprisingly effective for problems like N-Queens (can solve for millions of queens).

1.8 Problem Structure and Decomposition

1.8.1 Independent Subproblems

If the constraint graph has connected components that are not connected to each other, we can solve them separately. This reduces complexity from $O(d^n)$ to $O(n/c \cdot d^c)$ where c is the size of the subproblem.

1.8.2 Tree-Structured CSPs

If the constraint graph is a tree (no loops), the CSP can be solved in **linear time** $O(n \cdot d^2)$.

Algorithm for Tree CSPs:

1. **Topological Sort:** Pick a root and order variables such that parents appear before children $(X_1 \dots X_n)$.
2. **Backward Pass (Constraint Propagation):** From X_n down to X_2, make each parent arc-consistent with its child. This removes all inconsistent values.
3. **Forward Pass (Assignment):** From X_1 to X_n, assign any valid value consistent with the parent. Since the graph is arc-consistent, a solution is guaranteed without backtracking.

1.8.3 Nearly Tree-Structured Problems

Most real problems are not trees, but we can transform them.

Cutset Conditioning Idea: Remove a set of variables (the **Cycle Cutset**) so that the remaining graph is a tree.

1. Identify a subset of variables S (the cutset).
2. Assign values to variables in S.
3. These assignments act as constraints on the remaining variables.
4. Solve the remaining (now tree-structured) problem efficiently.
5. If no solution, try a different assignment for S.