Constraint Satisfaction Problem

What is Constraint Satisfaction Problems (CSPs)

A CSP is a type of problem in Artificial Intelligence where:

- You are given a set of variables.
- Each variable has a domain of values it can take.
- There are constraints that specify valid combinations of values.

Comparison with Standard Search Problems

Feature	Standard Search	Constraint Satisfaction Problem (CSP)
State	Black box – arbitrary structure	Defined by variable-value pairs
Goal Test	Arbitrary function	All constraints must be satisfied
Approach	General search (e.g., DFS, A*)	Specialized algorithms (e.g., Backtracking, Forward Checking)

Types of CSP States

• Complete Assignment: All variables have values → represents a full world state.

• Partial Assignment: Only some variables have been assigned values → used during

search.

Advantages of CSPs

• Formal representation of Problem

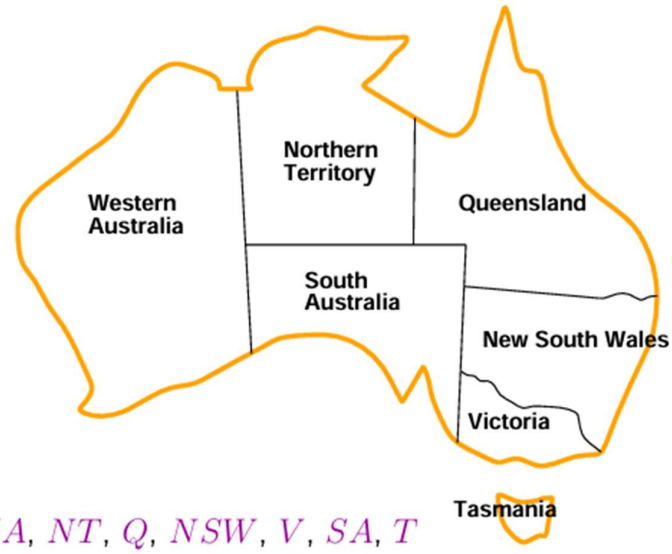
• Enable powerful general-purpose algorithms.

• Allow inference and reasoning during search

Constraint Satisfaction Problem

Example : Map Coloring Problem

- **Problem** : Assign colors to the states of Australia
- Constraints : No two adjacent states have same color.
- Colors Available : { Red , Green , Blue }



Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Sample Constraints:

- WA ≠ NT(if the language allow this)
- NT ≠ Q
- SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V
- NSW ≠ Q, NSW ≠ V
- V ≠ T (if applicable)

These can either be:

1. Expressed directly using inequality: A ≠ B
2. Enumerated: List all valid value pairs:

   e.g., (WA, NT) ∈ {(red, green), (green, blue), ...} excluding same-color pairs

Constraint Satisfaction Problem

Constraint Graph Representation

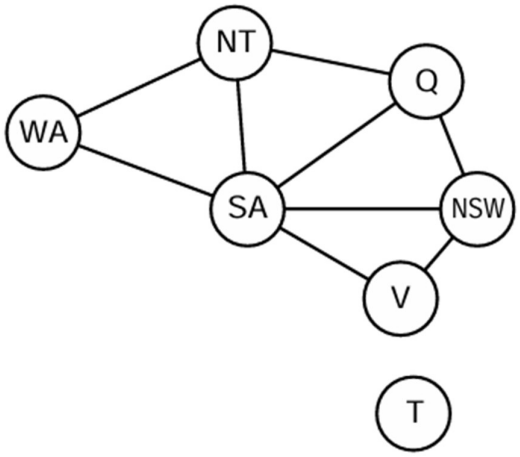
A Constraint Graph is a visual and structural way to represent CSPs.

- Nodes: Represent variables (e.g., regions in map-coloring).
- Edges: Represent constraints between variables (e.g., “Region A ≠ Region B”).

This makes it easier to see which variables interact (i.e., have constraints between them).

Binary Constraint Graphs

- A graph where each edge links at most two variables is called a Binary Constraint Graph.
- A Binary CSP only has constraints involving two variables at a time.
- Even if real problems have **non-binary constraints** (3 or more variables), they can be **converted into binary** ones.



Constraint Satisfaction Problem

Varieties of CSPs :

1. Discrete Variable CSPs

➤ Finite Domains:

- Variables have limited, countable values.
    - E.g., {Red, Green, Blue}, digits 0–9.
- If there are n variables and each has d possible values → total combinations = d^n.

➤ Infinite Discrete Domains:

- Variables can take unbounded discrete values (e.g., integers, strings).
- Nonlinear constraints might become undecidable (no general solution exists).

2. Continuous Variable CSPs
    - Variables take values from real numbers (not discrete).

Varieties of constraints :

Unary constraints involve a single variable,

e.g., SA= green

Binary constraints involve pairs of variables,

e.g., SA= WA

Higher-order constraints involve 3 or more variables,

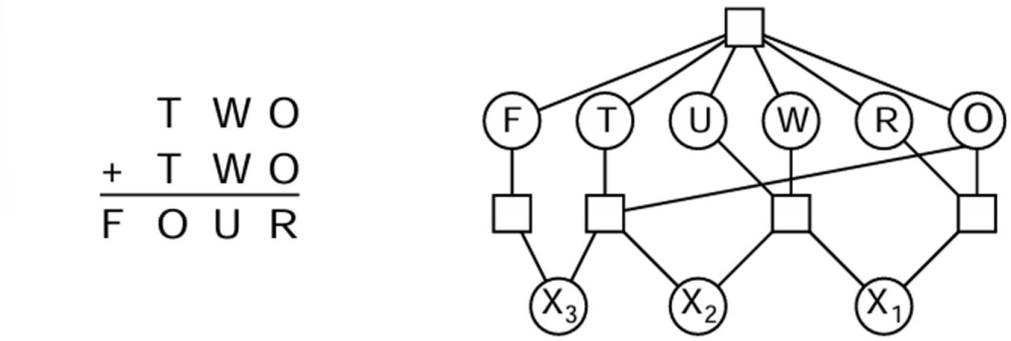
e.g., cryptarithmetic column constraints

Constraint Satisfaction Problem

Soft constraints (Preferences)

- Constraints that don't need to be strictly satisfied.
    - o Example: A student prefers sitting in front, but can sit at the back.
    - o Example: Red is better than green

Cryptarithmetic as a CSP



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints:

- All letters must represent distinct digits:
  AllDiff(F, T, U, W, R, O)
- Arithmetic constraints based on column-wise addition:
    - o O + O = R + 10 × X1
    - o W + W + X1 = U + 10 × X2
    - o T + T + X2 = O + 10 × X3
    - o X3 = F

Graph Representation:

- Variables are nodes.

Constraint Satisfaction Problem

- Constraints are square nodes connected to all involved variables.
- Can be converted to binary CSP by introducing auxiliary variables and constraints.

Real-World Examples of CSPs

- Job Scheduling
- Classroom Assignment
- Chip Design
- Timetabling

Standard Search Formulation for CSPs

Search Space:

- All possible ways to assign values to variables
- Two paradigms:
  - Systematic Search (partial assignments)
  - Local Search (complete assignments)

Systematic Search

A. State Representation

- A state is a partial assignment.
  Example: {X_1 = red, X_2 = blue, X_3 = ?}
- Initial state: Empty assignment

B. Successor Function

- Assign a value to one unassigned variable at each step
- If L variables are assigned:
  - (n - L) variables remain
  - d values per variable

- Number of successors = (n - L) × d

C. Goal Test

- An assignment is a solution if:
  - All n variables are assigned
  - All constraints are satisfied

D. Depth-First Search (DFS)

- DFS is natural for CSPs since solutions only appear at depth n

Complexity Analysis

Search Tree Nodes

- Depth 0: n × d nodes
- Depth 1: (n - 1) × d nodes
- …
- Leaves: $n! \times d^n$ nodes

Why n!?

- Because each branch represents a different order of variable assignments
- Even if the final assignment is the same, different assignment orders create new branches
- Actual number of valid complete assignments = d^n
- Search tree generates redundant paths due to permutations

Avoiding Redundant Paths

Fixed Variable Order

- Always assign in a fixed order (e.g., $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n$)
- Reduces leaf nodes to d^n

- Pros: avoids n! redundancy
- Cons: fixed order may be inefficient

Dynamic Variable Ordering (Heuristics)

- Choose next variable based on current context
- Minimum Remaining Values (MRV): pick variable with fewest legal values left
- Degree Heuristic: pick variable involved in most constraints on unassigned variables
- Improves efficiency and avoids early dead-ends

Backtracking Search in (CSPs)

What is Backtracking Search?

- A **depth-first search** strategy used in solving CSPs.
- At each step, one variable is assigned one of its domain values.
- **Branching factor** = number of values per variable = d.
- **Depth of search tree** = number of variables = n.
- **Total leaf nodes** ≈ d^n.

Why Use It?

- Simple but surprisingly powerful.

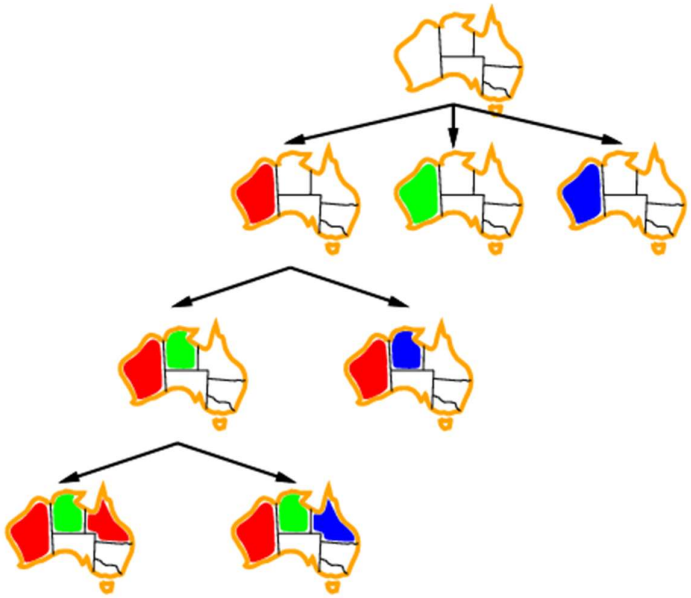
Backtracking Search Steps:

1. Select an unassigned variable.
2. Assign it a value from its domain.
3. Check constraints:
    - If **valid**, continue to next variable.
    - If **invalid**, **backtrack** and try another value.

Constraint Satisfaction Problem

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```



Improving Backtracking Efficiency

Key Questions:

1. **Which variable to assign next?** (Variable Ordering)
2. **In what order should its values be tried?** (Value Ordering)
3. **Can we detect failure early?** (Constraint Propagation)

Constraint Satisfaction Problem

4. **Can we exploit problem structure?** (Problem Decomposition)

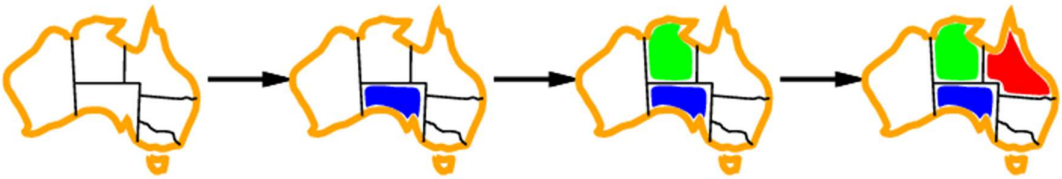
1. Variable Ordering Heuristics

a. Minimum Remaining Values (MRV) Heuristic

- Choose the variable with the **fewest legal values left**.
- **Why?**
  - "Fail-fast" approach → detects conflicts early.
  - Reduces branching factor early in the search.
- **Example:** In map coloring, if **South Australia** has only **1 valid color left**, assign it first.

b. Degree Heuristic (Tie-Breaker for MRV)

- Among variables with the same MRV, choose the one with the **most constraints on unassigned variables**.
- **Why?**
  - Reduces future branching by assigning highly constrained variables first.
- **Example:** When scheduling a meeting, ask the **busiest person first** (e.g., a director).



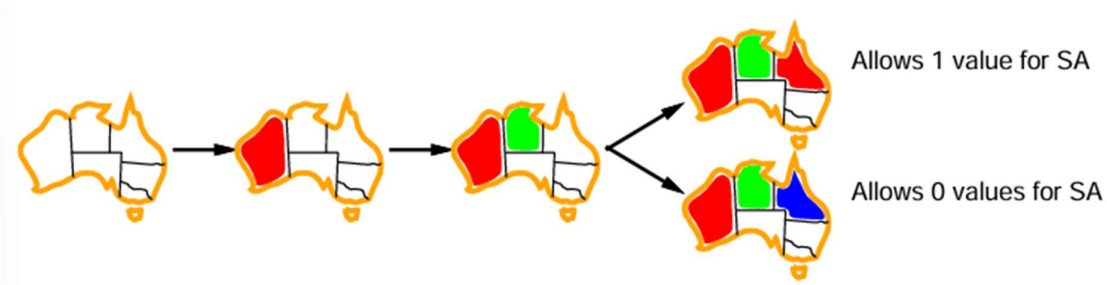
Constraint Satisfaction Problem

2. Value Ordering Heuristics

Least Constraining Value (LCV) Heuristic

- Choose the value that **rules out the fewest choices** for neighboring variables.
- **Why?**
  - Keeps future options open → reduces backtracking.
- **Example:** In map coloring, assign a color **least used by adjacent states**.

"Combining MRV, Degree, and LCV makes solving 1000-Queens feasible."

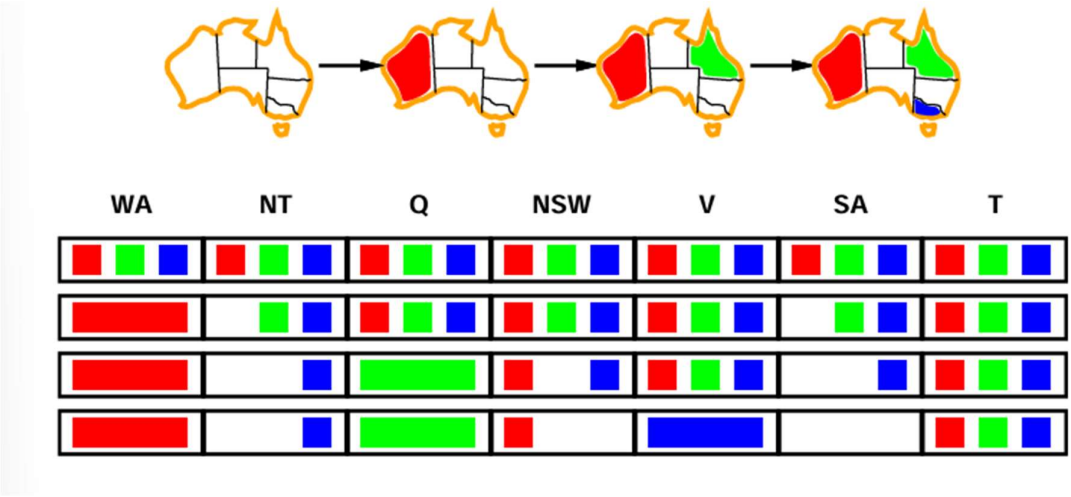


3. Detecting Failure Early

(a) Forward Checking

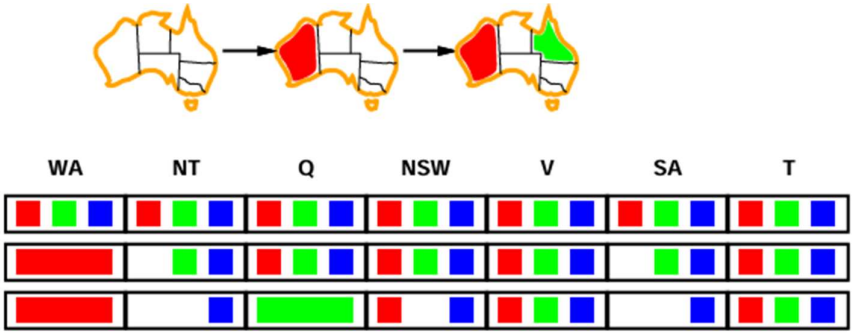
- **Definition:** After assigning a variable, prune inconsistent values from its neighbors.
- **Process:**
  - Assign a value to a variable (e.g., WA = red).
  - Remove conflicting values from adjacent variables (e.g., remove red from NT, SA).
- **If a neighbor's domain becomes empty → backtrack immediately.**
- **Limitation:** Only checks **assigned → unassigned** constraints.

Constraint Satisfaction Problem



(b) Constraint Propagation

- **Goes beyond forward checking** by enforcing constraints between **all variables** (including unassigned pairs).
- **Example:** If NT and SA both only have blue left (but are adjacent), forward checking won't detect the conflict. Constraint propagation does.



Arc Consistency (AC-3 Algorithm)

Key Idea:

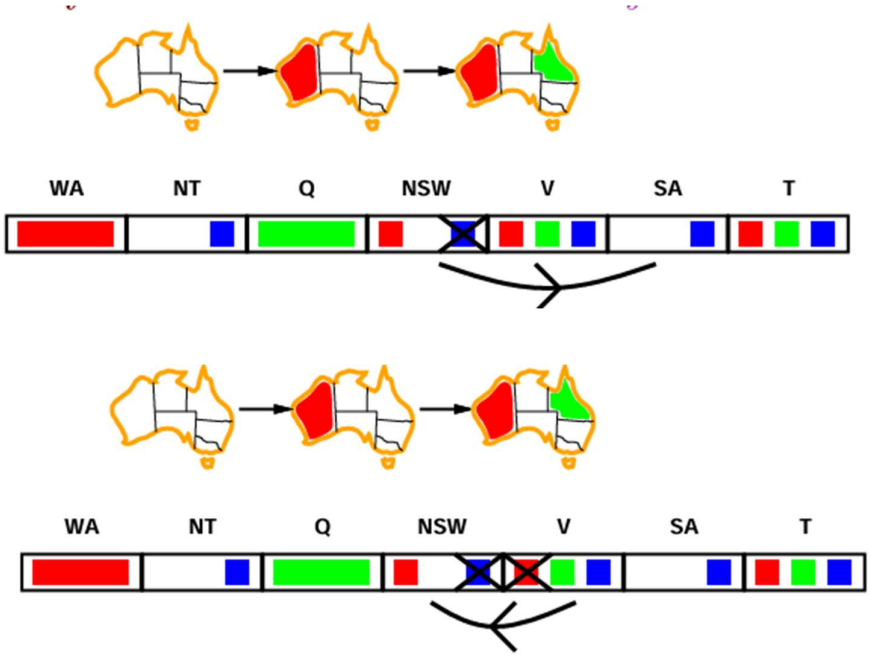
- Ensures every value in a variable's domain has at least one compatible value in its neighbor's domain.
- Operates on arcs (directed constraint pairs, e.g., X → Y).

Constraint Satisfaction Problem

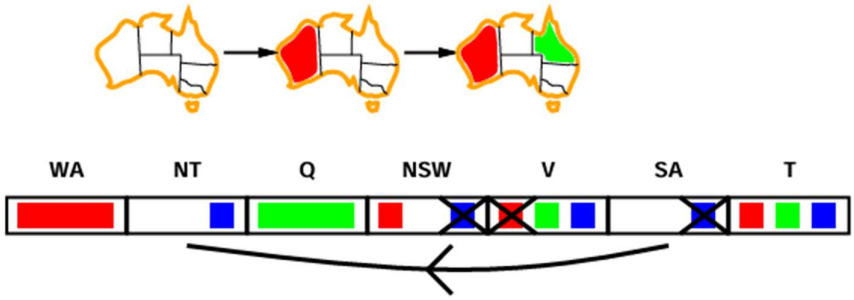
1. **What is an Arc?**
    - A directed constraint between two variables (e.g., X → Y means "X's value must be compatible with Y's value").
    - Example: In map coloring, NT → SA means "Northern Territory's color ≠ South Australia's color."
2. **Arc Consistency Definition:**
   An arc X → Y is consistent **iff**:
    - For *every* value x in X's domain, there exists *at least one* value y in Y's domain that satisfies the constraint.
    - If not, x is removed from X's domain (inference, not guesswork).
3. **Process (AC-3 Algorithm):**
    - **Step 1:** Check all arcs (e.g., SA → NSW, NSW → V).
    - **Step 2:** For each arc, remove values in one variable that lack support in the other.
    - **Step 3:** If a domain changes, recheck neighboring arcs (chain reaction).
    - **Terminate:** When no more values can be removed.
4. **Example (Map Coloring):**
    - Assign WA = red → Remove red from NT and SA.
    - Check NT → SA: If NT = blue but SA cannot be blue → Remove blue from NT.

Advantage of AC-3:
Detects conflicts earlier by propagating constraints between *unassigned* variables (e.g., NT ≠ SA even if neither is assigned yet).

Constraint Satisfaction Problem



If X loses a value, neighbors of X need to be rechecked



Limitations of Arc Consistency

1. **Only Handles Binary Constraints:**
    - Cannot directly enforce constraints involving 3+ variables (e.g., "A ≠ B ≠ C").
2. **Not a Complete Solver:**
    - An arc-consistent CSP may still have no solution.
    - AC reduces domains but doesn't guarantee a valid assignment exists.

Constraint Satisfaction Problem

Arc Consistency vs. Forward Checking

Feature	Forward Checking	Arc Consistency (AC-3)
Scope	Assigned → Unassigned variables	All pairs of variables (assigned/unassigned)
Power	Weak: Only immediate neighbors	Stronger: Propagates through all constraints
Failure Detection	Late (after assignment)	Early (during domain reduction)
Time Complexity	O(n·d)	$O(n^2d^3)$ for AC - 3

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
      for each X_k in NEIGHBORS[X_i] do
        add (X_k, X_i) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[X_i] do
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

4. Exploiting Problem Structure in CSPs

1. Decomposable CSPs (Independent Subproblems)

Key Idea

- **If a CSP can be split into disconnected subproblems, solve each independently.**
- **Complexity Reduction: From exponential ($O(d^n)$) to linear ($O(n/c \cdot d^c)$).**

Example

- **Variables (n): 80**
- **Domain size (d): 2**
- **Subproblem size (c): 20**
- **Brute-Force: 2^{80} ≈ 4 billion years (at 10M nodes/sec).**
- **Decomposed: 4×2^{20} ≈ 0.4 seconds.**

When to Use

- **Constraint graph has isolated components (e.g., Tasmania vs. mainland Australia in map coloring).**

2. Tree-Structured CSPs

Key Theorem

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

Algorithm Steps

1. **Ordering:**
    - **Pick a root, order variables parent → children (topological sort).**
    - **Example: A → B → C → D → E → F.**
2. **Backward Pass (Arc Consistency):**

- **From leaves to root, enforce consistency between each node and its parent.**
- **Example: Remove inconsistent values from D → B, then B → A.**

3. **Forward Pass (Assignment):**
   - **From root to leaves, assign values consistent with parents.**
   - **No backtracking needed!**

Why It Works

- **Trees have no cycles → no back-and-forth constraints.**
- **Ensures global consistency via local checks.**

3. Nearly Tree-Structured CSPs

Cutset Conditioning

1. **Identify a Cutset:**
   - **A set of variables whose removal turns the graph into a tree.**
   - **Example: Removing SA in Australia map coloring.**
2. **Brute-Force Cutset Assignments:**
   - **Try all possible combinations of cutset values ($O(d^c)$ for cutset size c).**
3. **Solve the Remaining Tree:**
   - **For each cutset assignment, solve the tree-structured subproblem in $O((n-c)\cdot d^2)$.**

- **Total Complexity: $O(d^c \cdot (n-c)\cdot d^2)$.**

Example

- **Cutset size (c): 1 (e.g., SA).**
- **Runtime: $O(d \cdot (n-1)\cdot d^2)$ → still far better than $O(d^n)$.**

4. Iterative Improvement (Local Search)

Min-Conflicts Heuristic

- **For large CSPs** (e.g., N-Queens with $N=10^7$).
- **Steps**:
    1. Start with a **random complete assignment** (may violate constraints).
    2. **Pick a conflicted variable** (e.g., a queen under attack).
    3. **Assign the value minimizing conflicts** (hill-climbing).

Structure	Approach	Complexity	When to Use
Disconnected	Solve subproblems independently	$O(n/c \cdot d^c)$	Graph has isolated components.
Tree	Backward + forward passes	$O(n \cdot d^2)$	No cycles in constraint graph.
Near-Tree	Cutset conditioning	$O(d^c \cdot (n-c) \cdot d^2)$	Small cutset exists.
General CSPs	Min-conflicts heuristic	O(1) avg. per step	Large problems (e.g., N-Queens).

Performance of min-conflicts

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10{,}000{,}000$)

The same appears to be true for any randomly-generated CSP
except in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$