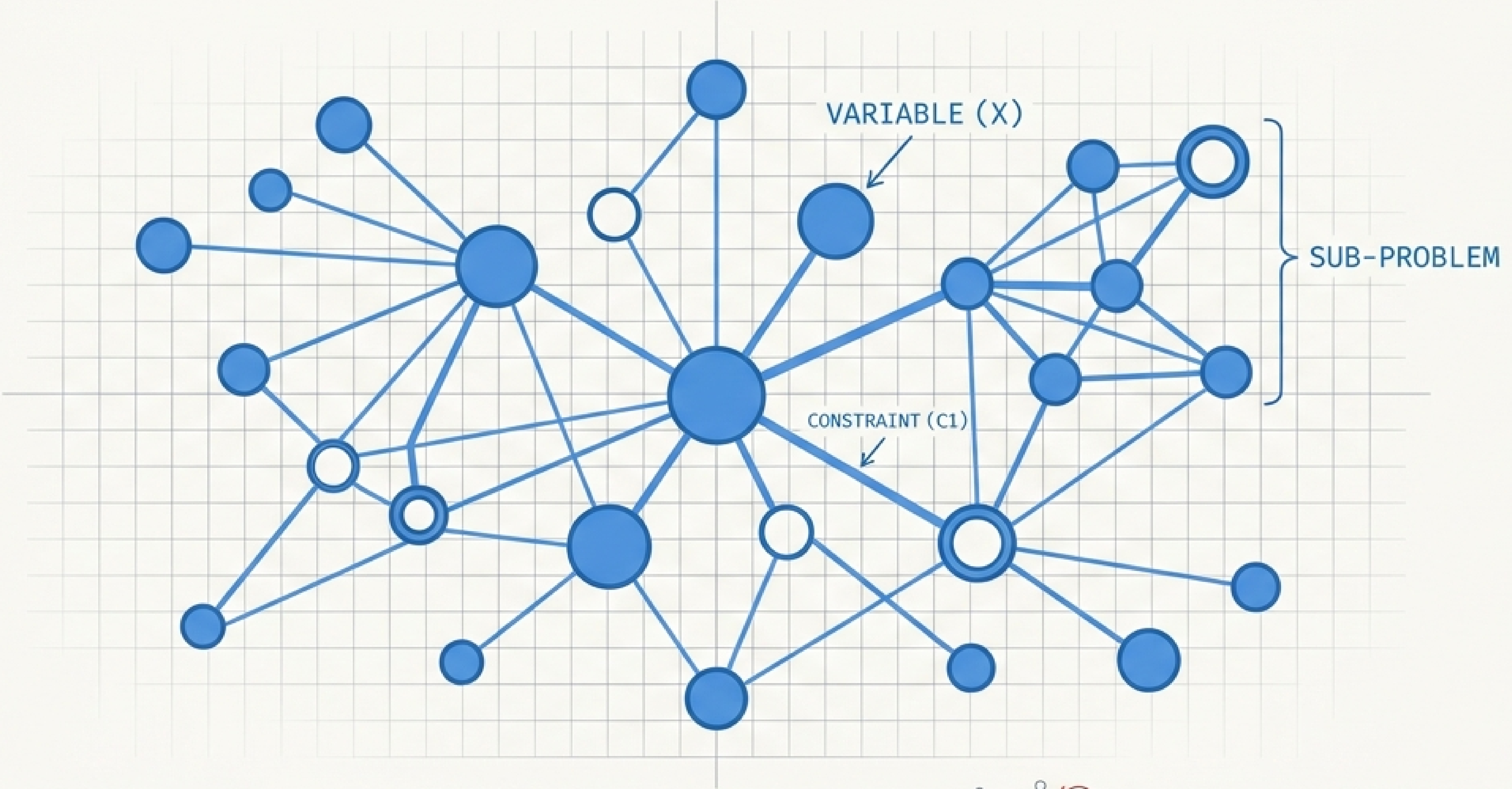
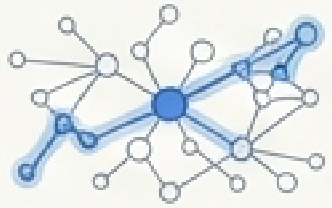
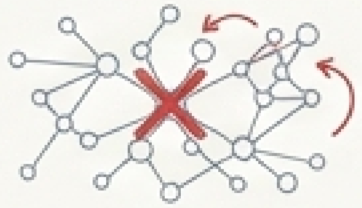
Fundamentals of AI: Constraint Satisfaction Problems

A Comprehensive Guide to Theory, Algorithms, and Application





SEARCH SPACE
Exploration of Possible Assignments



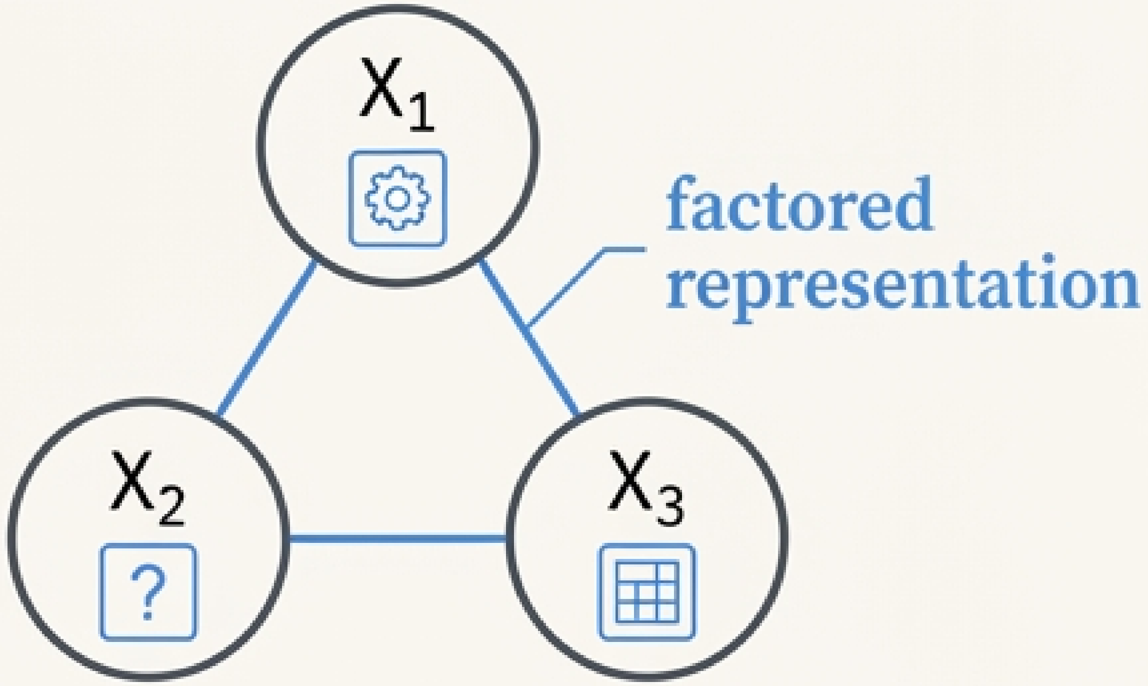
CONSTRAINT VIOLATION
Inconsistent Assignment & Pruning

Beyond Standard Search: A Factored View of States

Standard Search Problems

States are treated as atomic, indivisible black boxes with no internal structure.

Constraint Satisfaction Problems (CSPs)



States are defined by a factored representation—a set of variables, each with an assigned value.

The Goal & The Benefit

Goal: Find a complete assignment of values to all variables that satisfies a given set of constraints.

Benefit: This structured representation allows for general-purpose algorithms that are significantly more powerful than standard search methods.

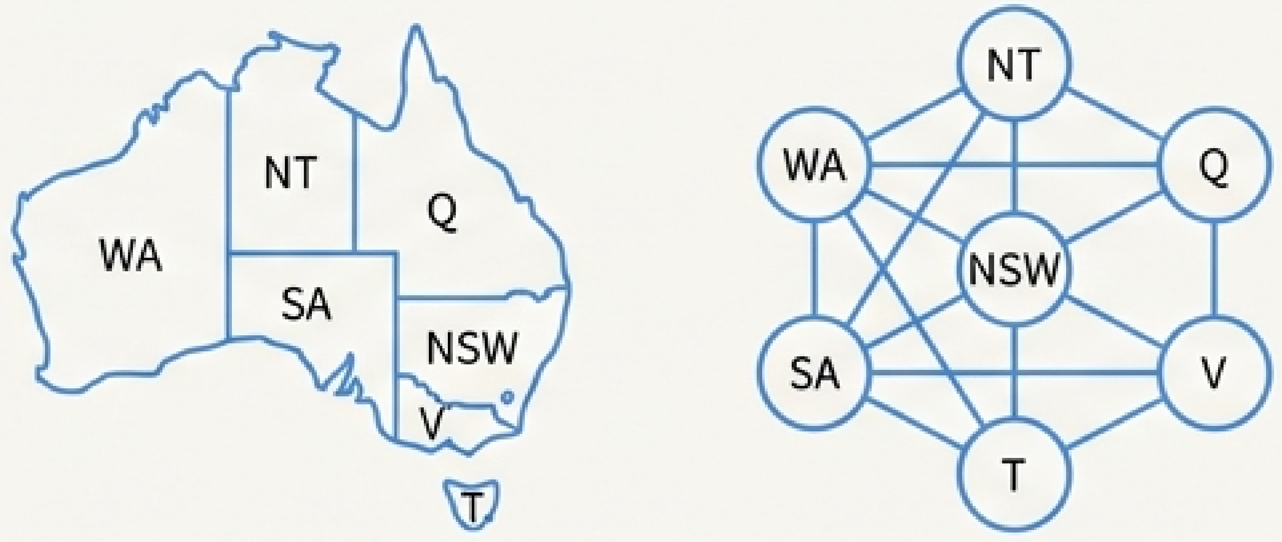
The Formal Definition: Anatomy of a CSP

The Theory

A Constraint Satisfaction Problem is a tuple (X, D, C):

- **X:** A set of variables, $\{X_1, \ldots, X_n\}$.
- **D:** A set of domains, $\{D_1, \ldots, D_n\}$, where D_i is the set of allowable values for variable X_i.
- **C:** A set of constraints, $\{C_1, \ldots, C_m\}$. Each constraint C_i is a pair $\langle$scope, rel$\rangle$, where *scope* is a tuple of variables and *rel* is a relation defining valid value combinations.

The Example (Map Coloring)



X (Variables): {WA, NT, Q, NSW, V, SA, T}

D (Domains): D_i = {red, green, blue} for all variables.

C (Constraints): Adjacent regions must have different colors. Example: {SA $\neq$ WA, SA $\neq$ NT, SA $\neq$ Q, ...}.

Note: SA $\neq$ WA is shorthand for $\langle$(SA, WA), SA $\neq$ WA$\rangle$

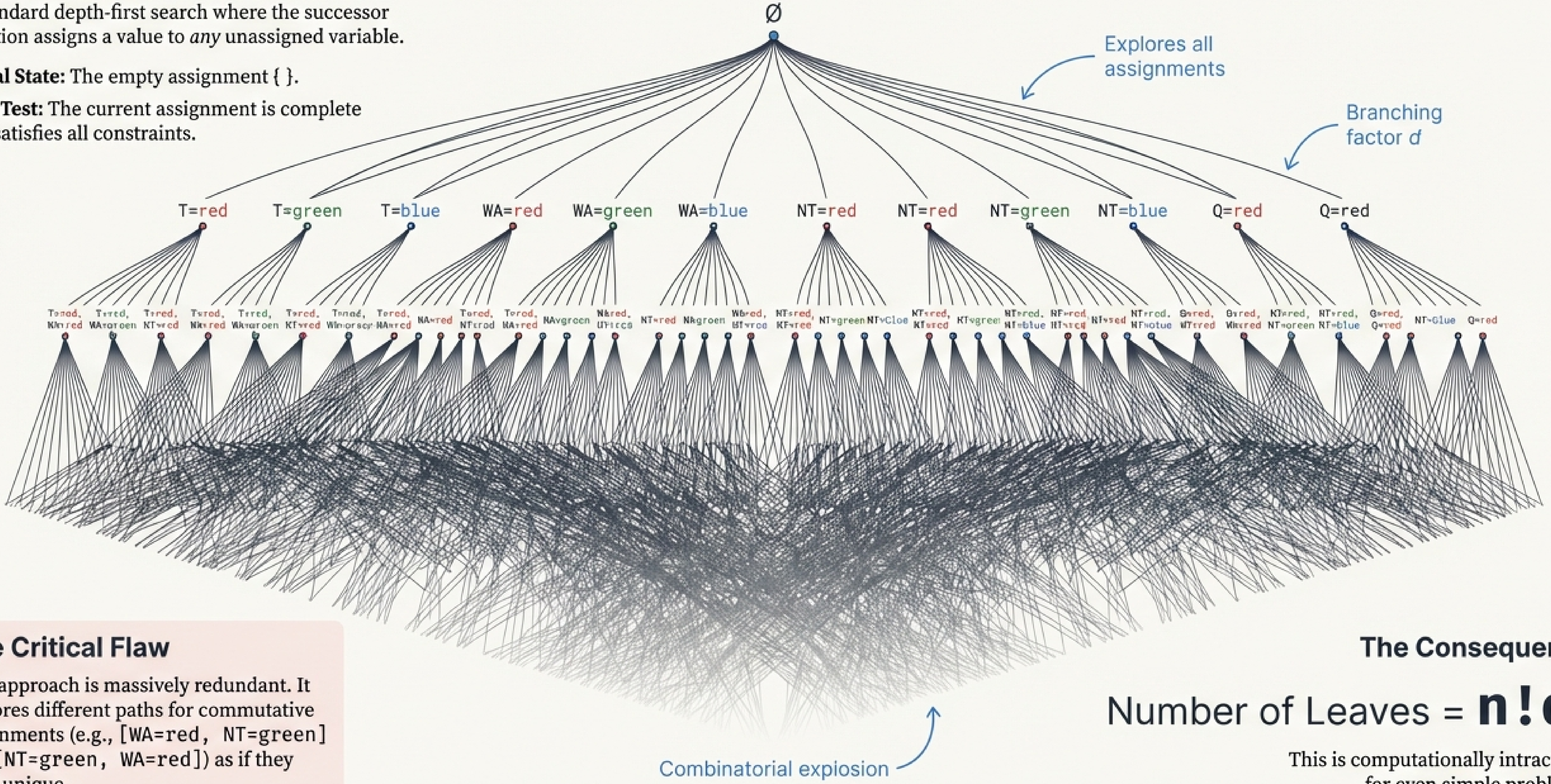
The Brute-Force Approach (And Why It Fails)

Method

A standard depth-first search where the successor function assigns a value to *any* unassigned variable.

Initial State: The empty assignment { }.

Goal Test: The current assignment is complete and satisfies all constraints.



The Critical Flaw

This approach is massively redundant. It explores different paths for commutative assignments (e.g., `[WA=red, NT=green]` and `[NT=green, WA=red]`) as if they were unique.

The Consequence

$$\text{Number of Leaves} = n!d^n$$

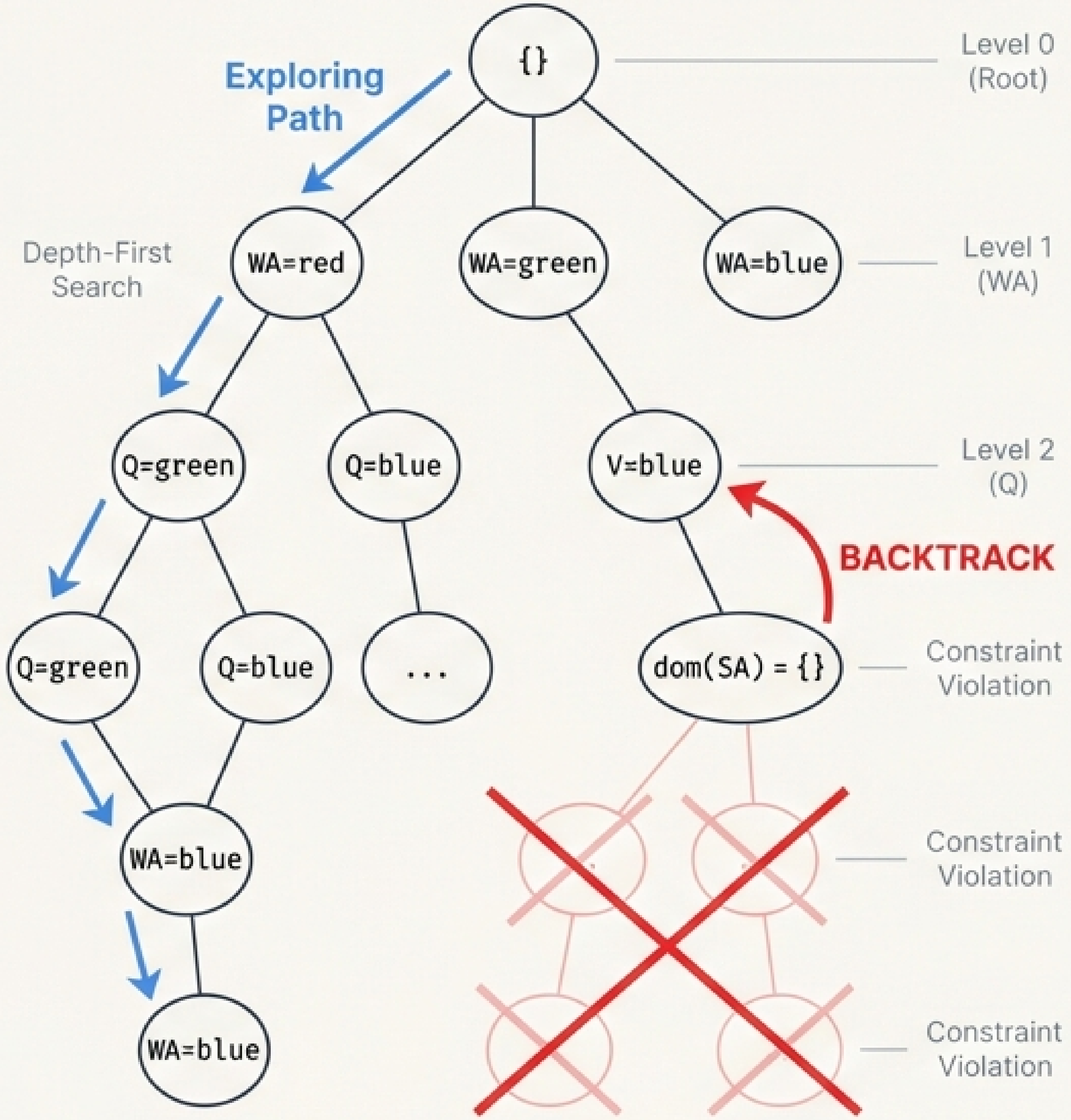
This is computationally intractable for even simple problems.

The Foundational Algorithm: Backtracking Search

Concepts

- **Key Insight:** Variable assignments are commutative. We can fix the order of assignment by considering only a *single* variable at each level of the search tree.
- **Process:**
    - A depth-first search that assigns a value to one variable at a time.
    - If a partial assignment is found to violate a constraint, the algorithm immediately "backtracks", pruning the entire subtree below that point.
- **The Improvement:**
  The search space is reduced from `$n!d^n$` to `$d^n$` leaves. While still exponential, this is a dramatic improvement.
- **Core Pseudocode:**

```
function Backtracking-Search(csp) returns solution/failure
   return Recursive-Backtracking({}, csp)
```



NotebookLM

Optimizing Backtracking: Four Guiding Questions

The core `Recursive-Backtracking` algorithm reveals four opportunities for intelligent optimization:

```
var ← Select-Unassigned-Variable(csp)
```

1. Which variable should be assigned next?

```
for each value in Order-Domain-Values(...)
```

2. In what order should its values be tried?

```
inferences ← Inference(csp, var, value)
```

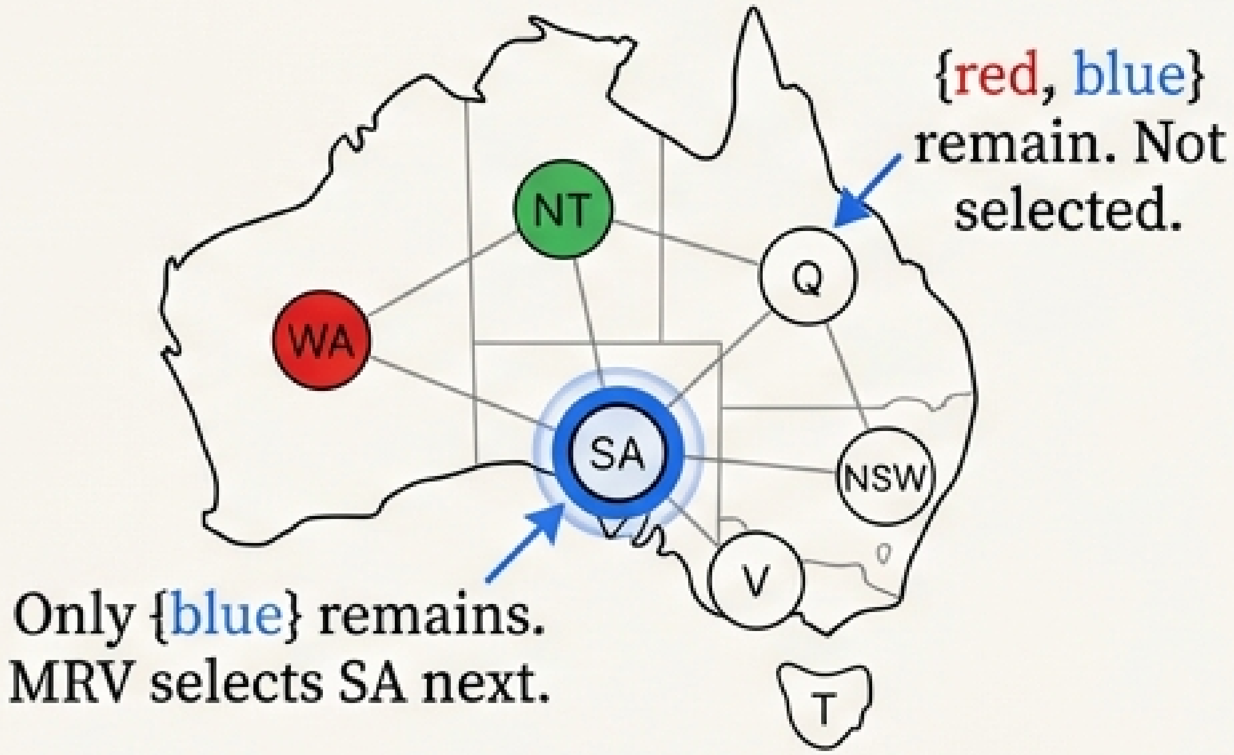
3. Can we detect inevitable failure early?

4. Can we exploit the problem's structure?

Question 1: Which Variable? (The 'Fail-First' Principle)

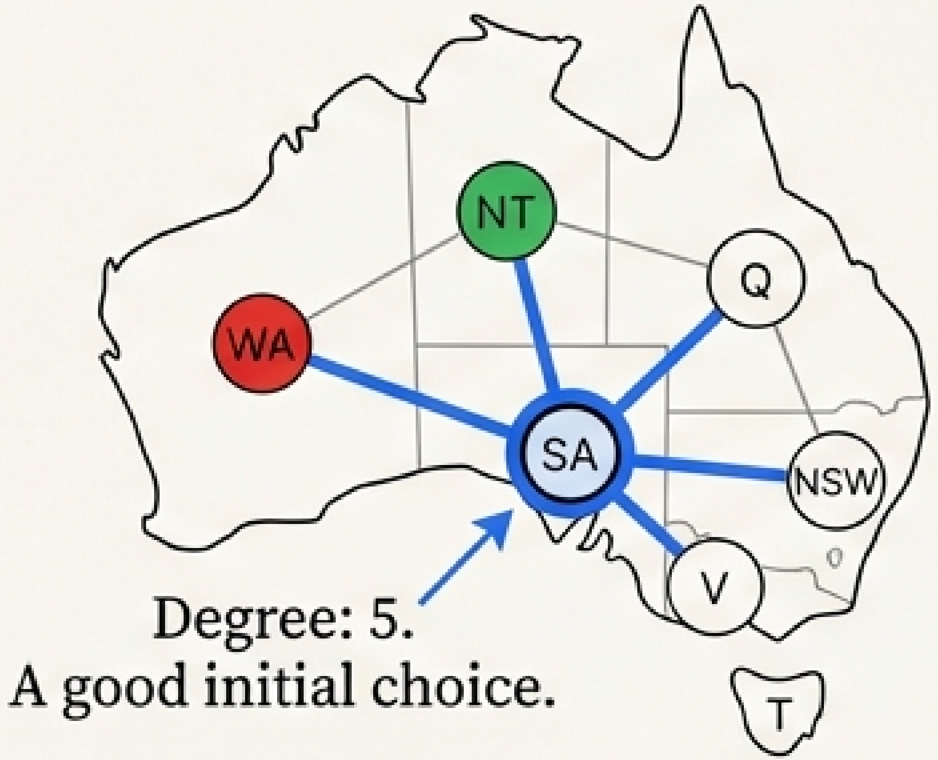
Minimum Remaining Values (MRV)

- **Rule:** "Choose the variable with the fewest legal values remaining in its domain."
- **Rationale:** "It is the most likely to cause a failure, allowing the algorithm to prune the search tree as early as possible."



Degree Heuristic (Tie-Breaker)

- **Rule:** "Choose the variable involved in the most constraints on *other unassigned variables*."
- **Rationale:** "This choice has the largest effect on reducing the domains of other variables, thus pruning future branches."



Question 2: Which Value? (The ‘Fail-Last’ Principle)

Heuristic: Least-Constraining-Value (LCV)

- **Rule:** “Prefer the value that rules out the fewest choices for neighboring variables in the constraint graph.”
- **Rationale:** “We only need one solution. This strategy attempts to find it by picking values that keep options open for the future, making backtracking less likely.”

Setup: `WA=red`, `NT=green`. We are choosing a value for `Q`.

Conclusion: LCV prefers `Q=red`.

Inference Part I: Proactive Pruning with Forward Checking

Theory

Definition: After assigning a value to variable X_i, Forward Checking makes all unassigned neighbors X_j arc-consistent with X_i.

Process: For each neighbor X_j, remove any values from its domain D_j that are inconsistent with the new assignment for X_i.

Failure Detection: If any neighbor's domain becomes empty, the assignment to X_i has failed, and the algorithm algorithm backtracks immediately.

In Action - Map Coloring

Sequential Visualization: Forward Checking on Australia Map Coloring

Step 1: Initial

Initial domains:

WA	NT	Q	NSW	V	SA	T
R G B	R G B	R G B	R G B	R G B	R G B	R G B

Step 2: Assign `WA=red`

Domains of neighbors **`NT`** and **`SA`** become **{G, B}**.

WA	~~NT~~	Q	NSW	V	~~SA~~	T
R	~~R G~~ B	R G B	R G B	R G B	~~R G~~ B	R G B

Step 3: Assign `Q=green`

`dom(NT)** and **`dom(SA) are further reduced to **{B}**.

WA	NT	**Q**	NSW	V	~~SA~~	T
R	~~R G~~ B	R ~~G~~ B	~~R G~~ B	R G B	~~R G~~ B	R G B

Step 3: Assign `Q=green`

`dom(NT) and **dom(SA)** are further reduced to **{B}**.

WA	NT	**Q**	NSW	V	SA	T
R	~~R G~~ B	R **G** B	~~R G~~ B	R G B	~~R G~~ B	R G B

Step 4: Assign `V=blue`

`dom(SA) becomes **{B}** ∩ **{R,G}** = **{}**.

WA	NT	Q	NSW	**V**	SA	T
R	R G B	R **G** B	R G B	RG**B**	{}	R G B

FAILURE DETECTED.
Backtrack from `V=blue`.

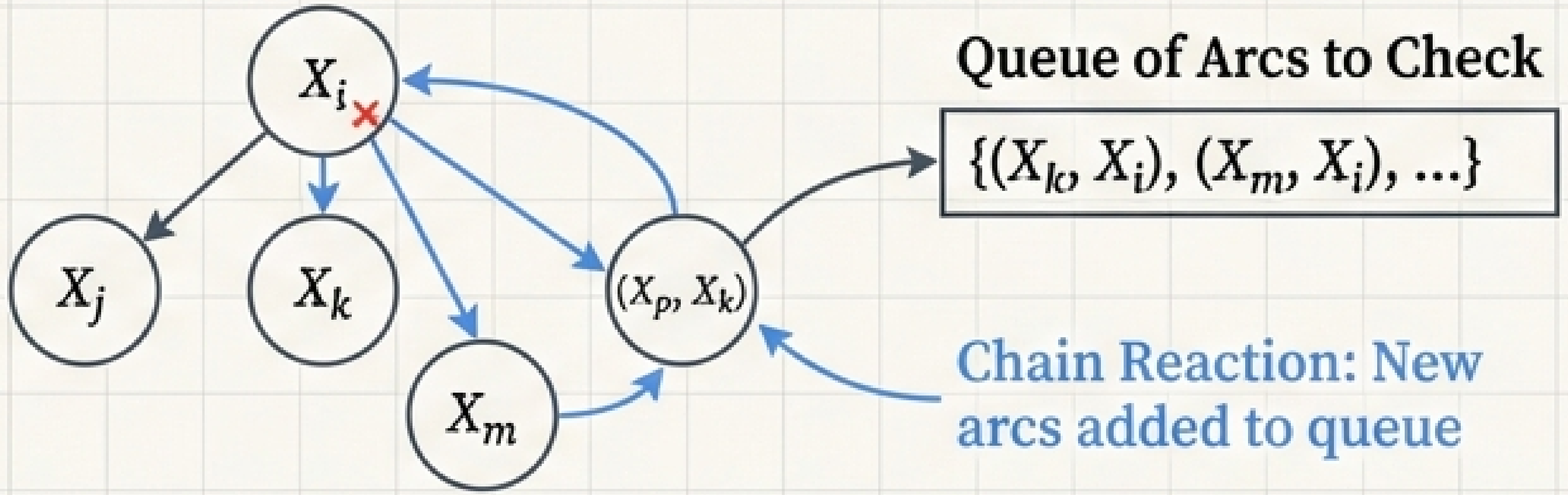
NotebookLM

Inference Part II: Full Constraint Propagation with AC-3

Definition and Concept

An arc (X_i, X_j) is **arc-consistent** if for every value x in D_i, there is at least one allowed value y in D_j.

A CSP is **arc-consistent** if every arc in its constraint graph is consistent.



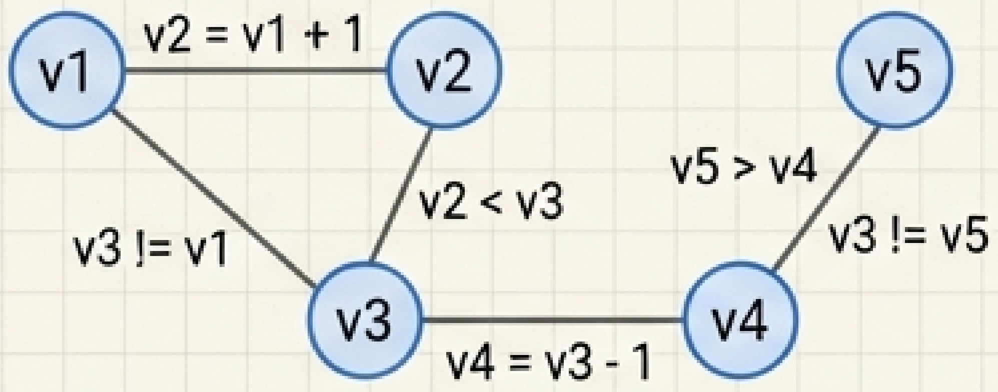
The Algorithm

- Maintains a queue of arcs to check.
- When a value is removed from D_i while checking arc (X_i, X_j), all arcs (X_k, X_i) pointing *to** X_i are added back to the queue.
- This propagation continues until the queue is empty, ensuring full arc consistency.

```
function AC-3(csp, queue) returns failure or the reduced csp
while queue is not empty do
   (Xi, Xj) ← Remove-First(queue)
   if Remove-Inconsistent-Values(Xi, Xj) then
      if size of Domain(Xi) = 0 then return failure
      for each Xk in Neighbors[Xi] \ {Xj} do
         add (Xk, Xi) to queue
return csp
```

Time Complexity: $O(cd^3)$, where c is the number of arcs and d is the maximum domain size.

Inference in Action: A Head-to-Head Comparison on Problem 3.2



The Scenario

Problem: The v1-v5 constraint graph where all initial domains are `{2, 3, 4}`.

Assignment: The search algorithm first assigns `v3 = 3`.

With Forward Checking (Problem 3.2.2)

`v3=3` is assigned. FC checks immediate neighbors.

Variable	Domain
v1	{2, 4}
v2	{2, 4}
v3	{3}
v4	{2, 4}
v5	{2, 4}

No empty domains are found. The search continues, exploring invalid paths before eventually backtracking. (Total Backtracks: 2).

With Arc Consistency (Problem 3.2.3)

`v3=3` is assigned. AC-3 propagates constraints.

Variable	Domain
v1	{2, 4}
v2	{}
v3	{3}
v4	{2, 4}
v5	{2, 4}

v1's domain change propagates to v2 via `v2 = v1 + 1`.

FAILURE IS DETECTED IMMEDIATELY. The algorithm backtracks from `v3=3`. without any further assignments. (Total Backtracks: 1).

FAILURE DETECTED. Backtrack from `v3=3`.

A Proactive Strategy: Arc Consistency as Preprocessing

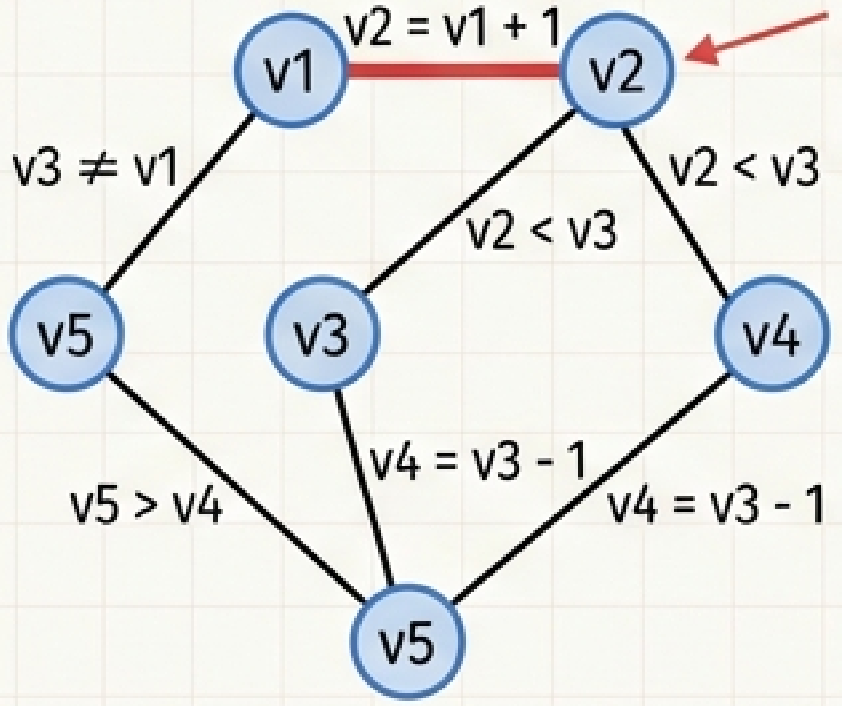
The Strategy: Run the AC-3 algorithm on the entire CSP *before* starting Backtracking Search. This makes the initial problem representation cleaner and more constrained.

Before vs. After of Problem 3.2.4

Before Preprocessing

Initial State

Is the initial v1-v5 graph arc-consistent? **No.**



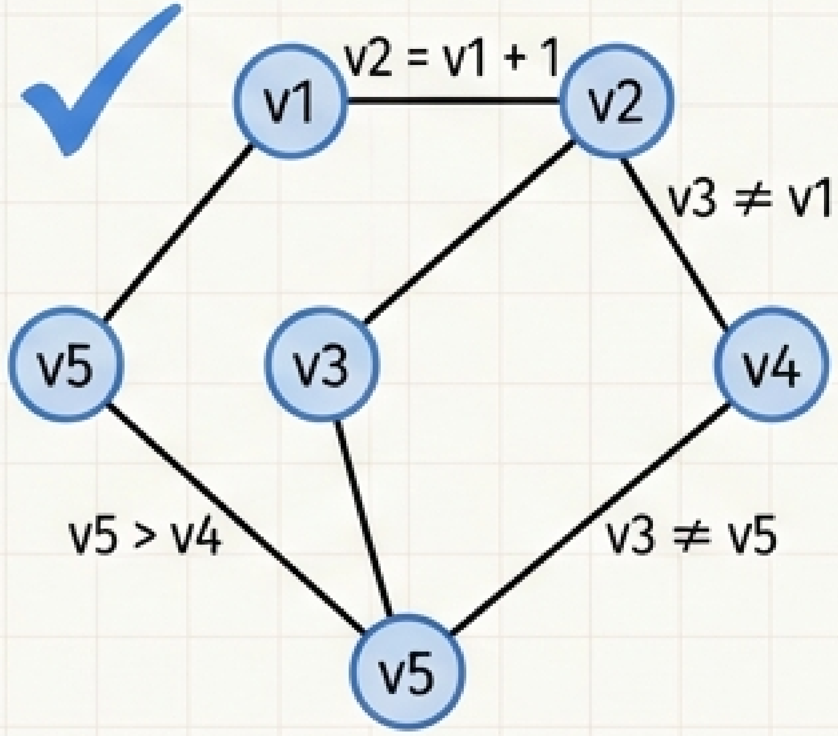
The Inconsistency: If `v1=4`, the constraint v2 = v1 + 1 requires v2=5, but 5 is not in `dom(v2)`.

```
dom(v1) = {2, 3, 4}
dom(v2) = {2, 3, 4}
dom(v3) = {2, 3, 4}
dom(v4) = {2, 3, 4}
dom(v5) = {2, 3, 4}
```

After Running AC-3

After Preprocessing

Result of Preprocessing:



dom(v1) = {2, ~~3~~}

dom(v2) = {~~3~~, 4}

dom(v3) = {2, 3, 4}

dom(v4) = {2, 3, 4}

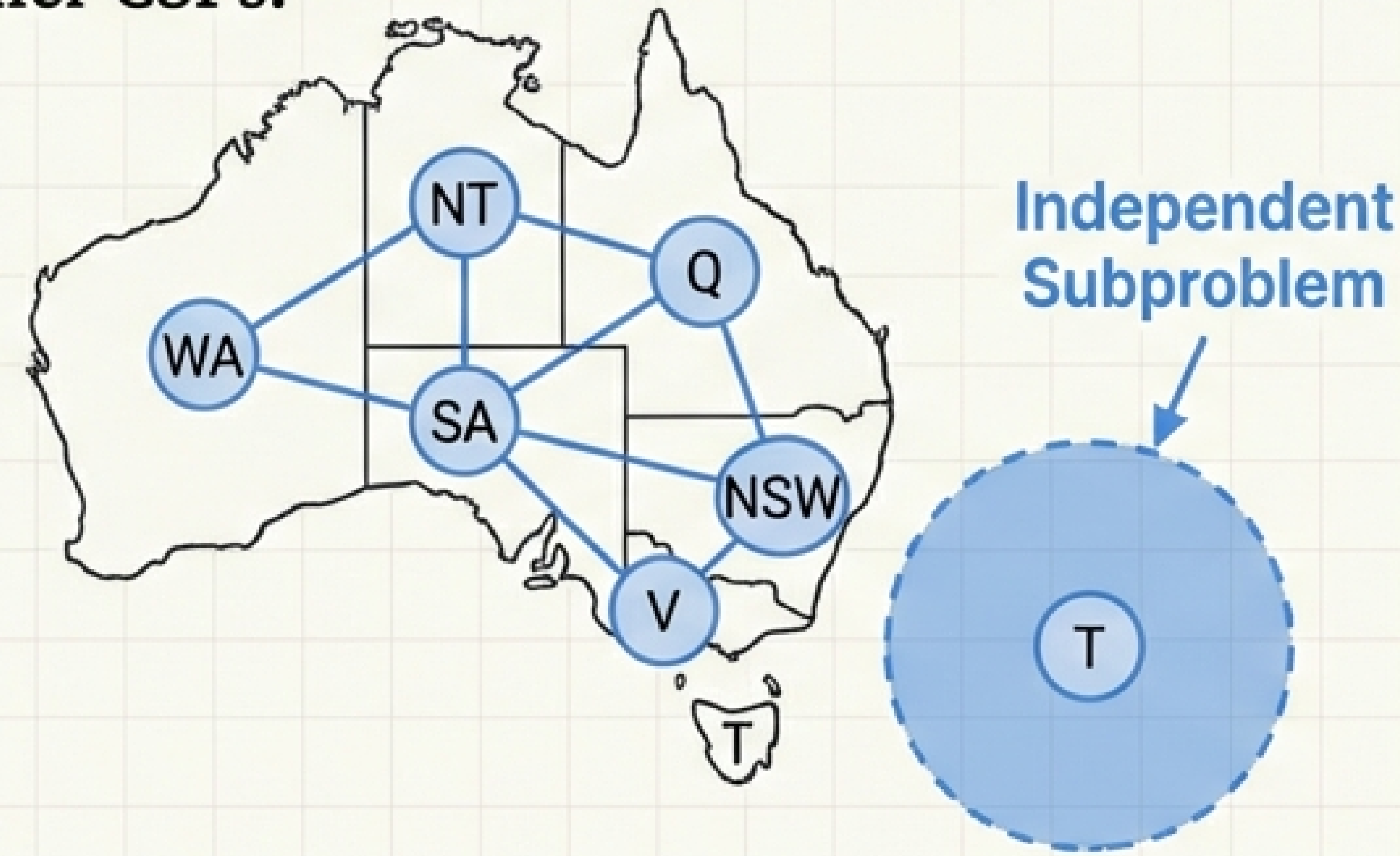
dom(v5) = {2, 3, 4}

The Benefit: This permanently reduces the branching factor for the entire search, preventing the algorithm from repeatedly rediscovering the same basic inconsistencies.

Question 4: Exploiting Problem Structure

Independent Subproblems

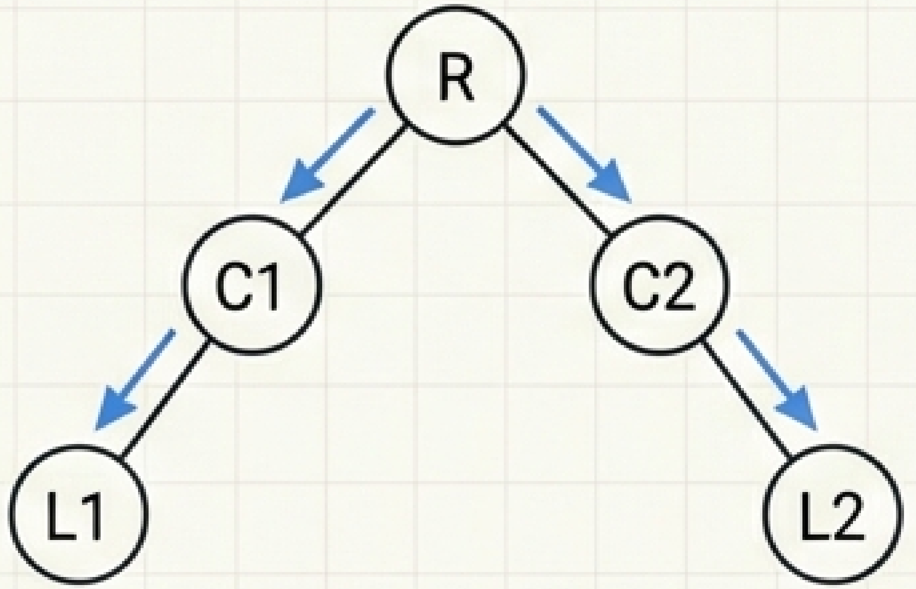
If the constraint graph has disconnected components, they can be solved as separate, smaller CSPs.



Complexity Reduction: Solving n/c problems of size c is $O(n/c * d^c)$. This is exponentially better than solving one problem of size n, which is $O(d^n)$.

Tree-Structured CSPs

If the constraint graph has no loops (i.e., it's a tree), the problem can be solved efficiently.



Complexity: $O(n*d^2)$, which is polynomial time.

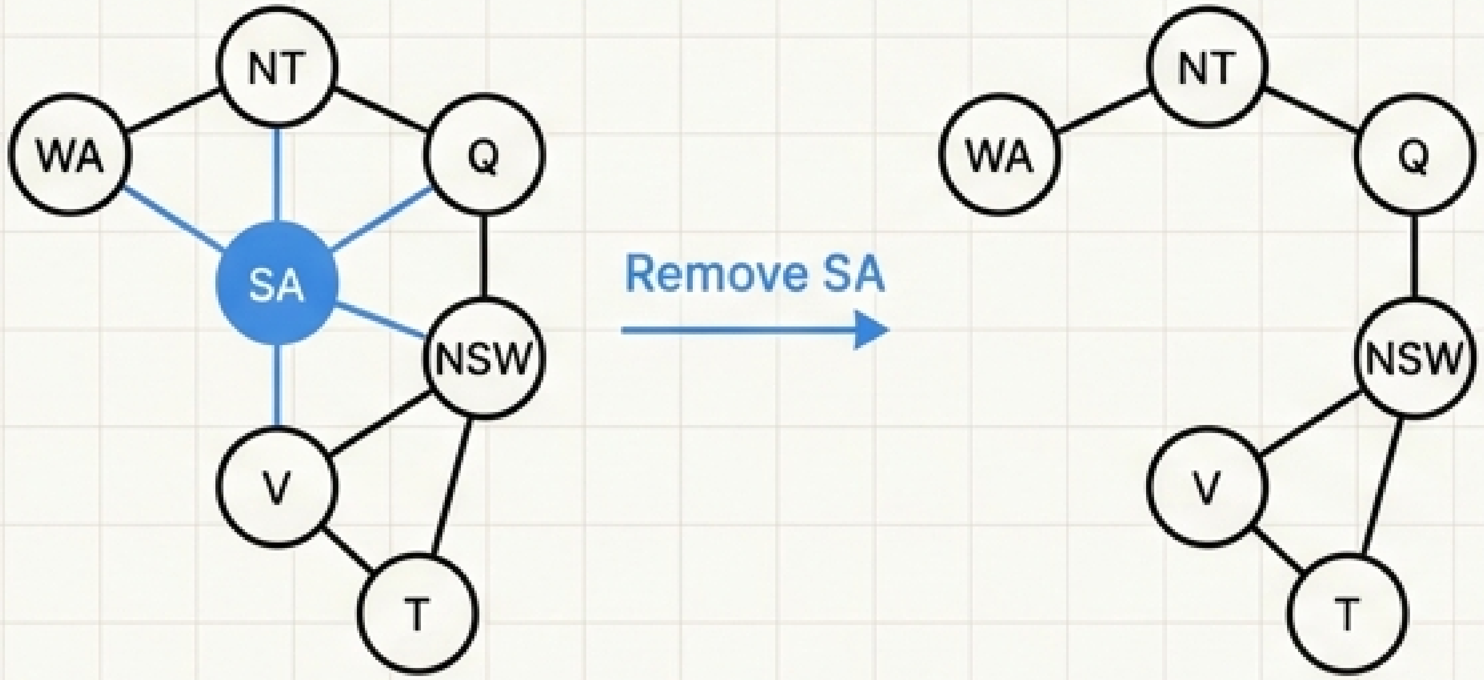
Algorithm:

1. Perform a topological sort of the variables.
2. Enforce directional arc consistency from leaves to root.
3. Assign values from root to leaves **without any need for backtracking**.

Handling Nearly Tree-Structured Problems

Method 1: Conditioning (Cycle Cutset)

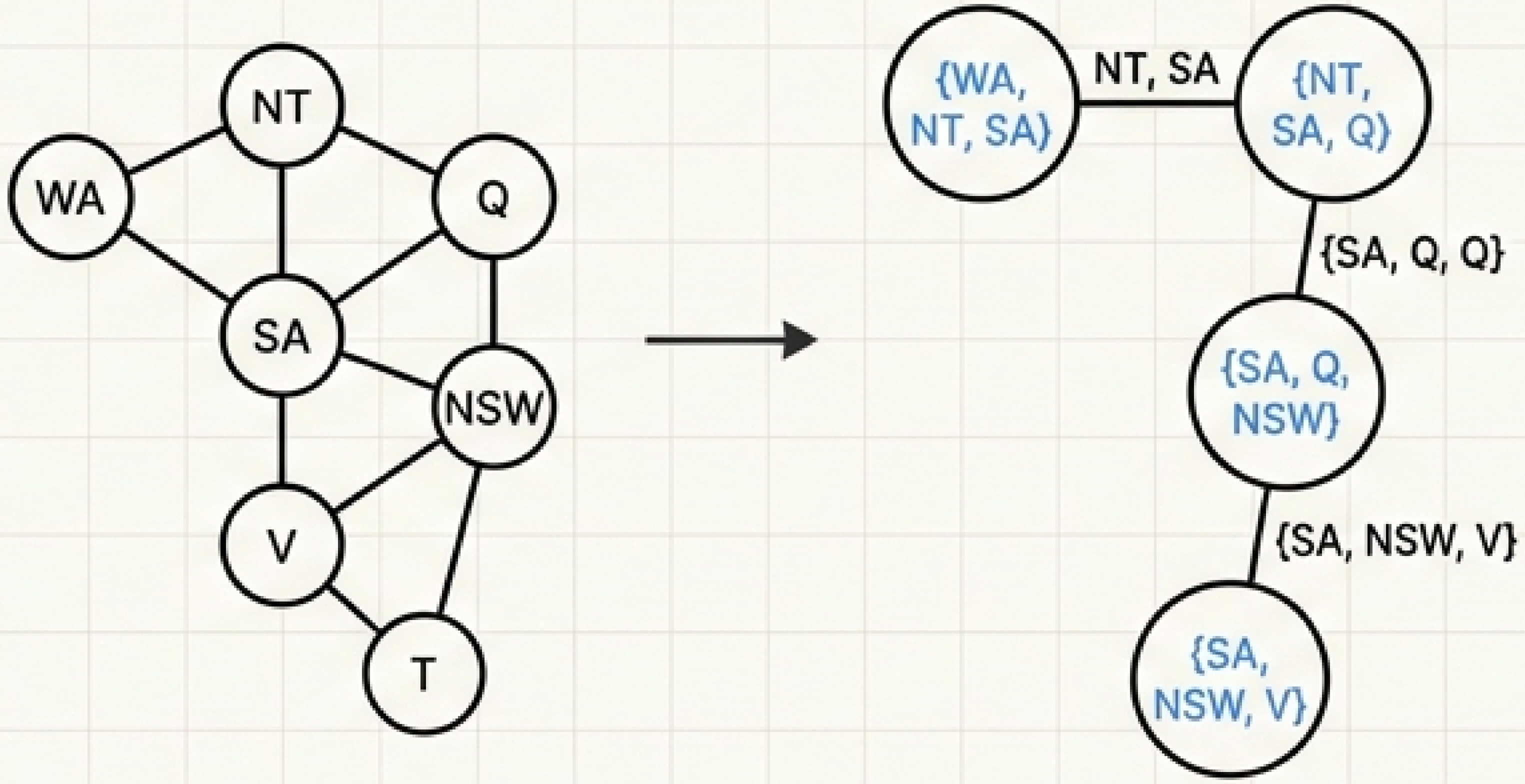
1. Identify a small set of variables S (the "cycle cutset") that, once assigned, break all cycles in the graph, leaving a tree.
2. For each valid assignment to the variables in S, solve the remaining tree-structured CSP.



Complexity: $O(d^c * (n-c)d^2)$, where c is the size of the cutset S.

Method 2: Tree Decomposition

Decompose the graph into a set of connected subproblems, which themselves form a "super-tree".



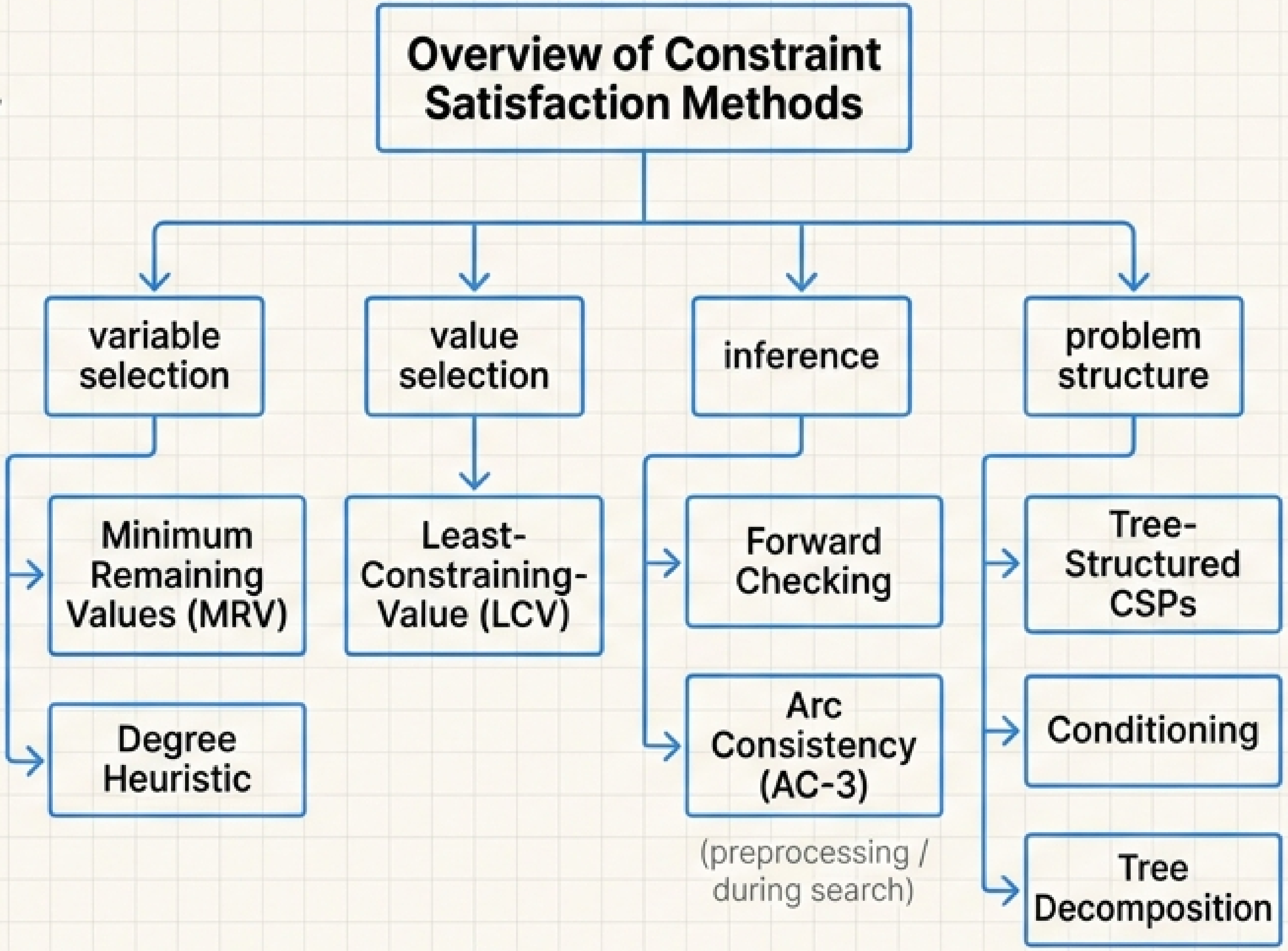
Solve each subproblem, then solve the constraints between the subproblems using the efficient tree algorithm.

Mastering CSPs: The Toolkit for Intelligent Search

Foundation: CSPs use a factored state representation, solved fundamentally by Backtracking Search.

Optimization Layer 1: Heuristics (Smart Selection)

- Variables (Fail-First): Minimum Remaining Values (MRV), Degree Heuristic.
- Values (Fail-Last): Least-Constraining-Value (LCV).



Optimization Layer 2: Inference (Smart Pruning)

- Basic: Forward Checking.
- Advanced: Arc Consistency (AC-3), used during search or as preprocessing.

Optimization Layer 3: Structure (Smart Analysis)

- Exploit independent subproblems and fast algorithms for Tree-Structured CSPs.
- Handle complex graphs with Conditioning and Tree Decomposition.