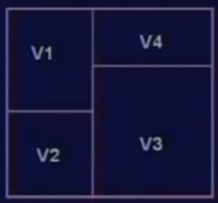
**Constraint Satisfaction Problems (CSPs)** are mathematical problems defined by a set of variables, constraints, and domains. They are used to find a solution that satisfies all constraints while assigning values from specified domains to each variable.

A finite set of variables 𝑉1, 𝑉2, …, 𝑉𝑛

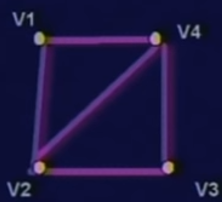
Each variable Vi has nonempty domain , which is set of possible values that it can take.

A finite set of constraints C1, C2, …, Cm. Each constraint limits the values that the variables can take simultaneously. Constraints:-

1. Unary constraints involve a single variable.e.g. SA ≠ green
2. Binary constraints involve pairs of variables. e.g. SA ≠ WA
3. Higher-order: involve 3 or more Variables. e.g. cryptharithmetic column constraints.
4. Preference (soft constraints) e.g. red is better than green

**Applications of CSPs**

1. Scheduling
2. Floor Planning
3. Map Coloring
4. Cryptography (encoding and decoding info securely)



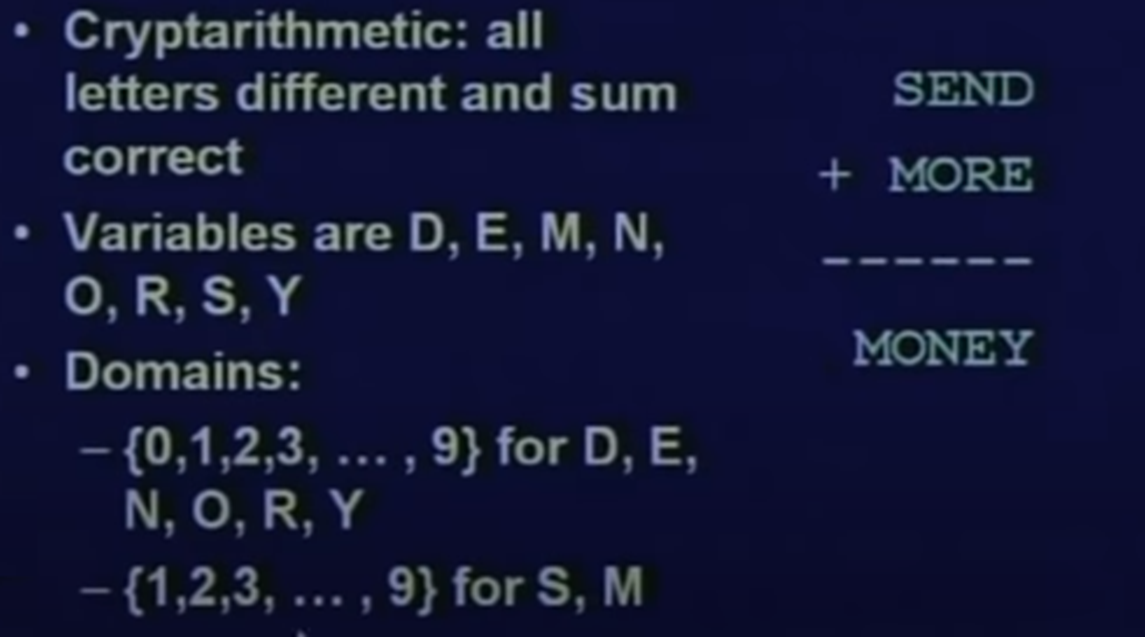
***Coloring as CSP:*** To color four regions using three colors (red, green, and blue) such that no two adjacent regions share the same color, Variable for each node, D₁ = { red, green, blue}. Constraint for each edge Xi≠Xj. The solution gives a coloring. It's binary.

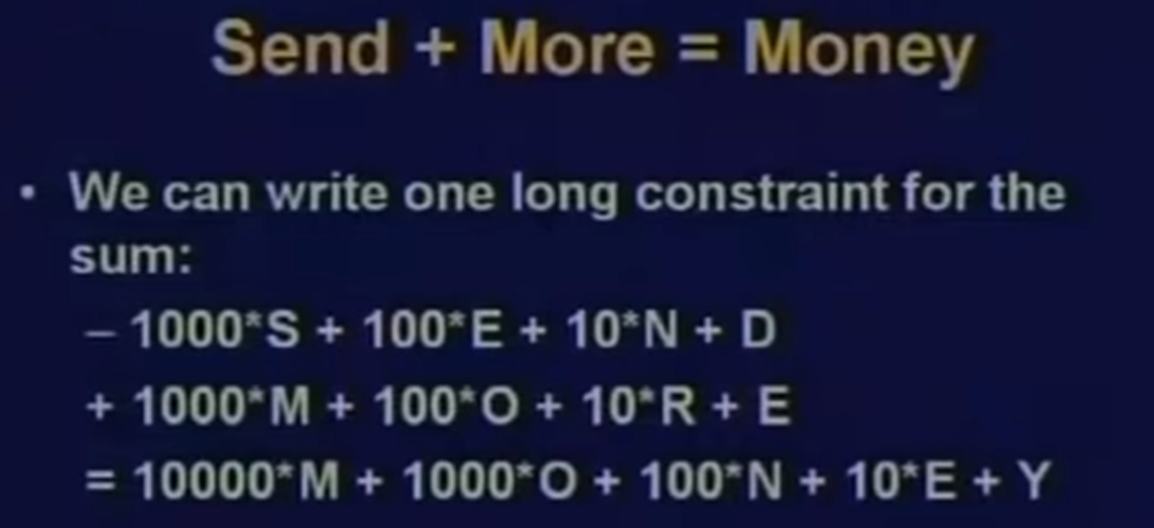
***SAT problems as CSPs*** involve assigning truth values to variables such that all logical clauses (constraints) are satisfied. Variable=each letter or variable in SAT.  
Each domain D₁ = {true, false}. Constraint corresponds to each clause and disallows unique tuples that falsify the clause. e.g. (not A) or (B) or (not C)

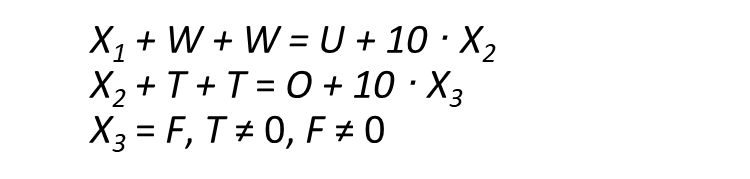
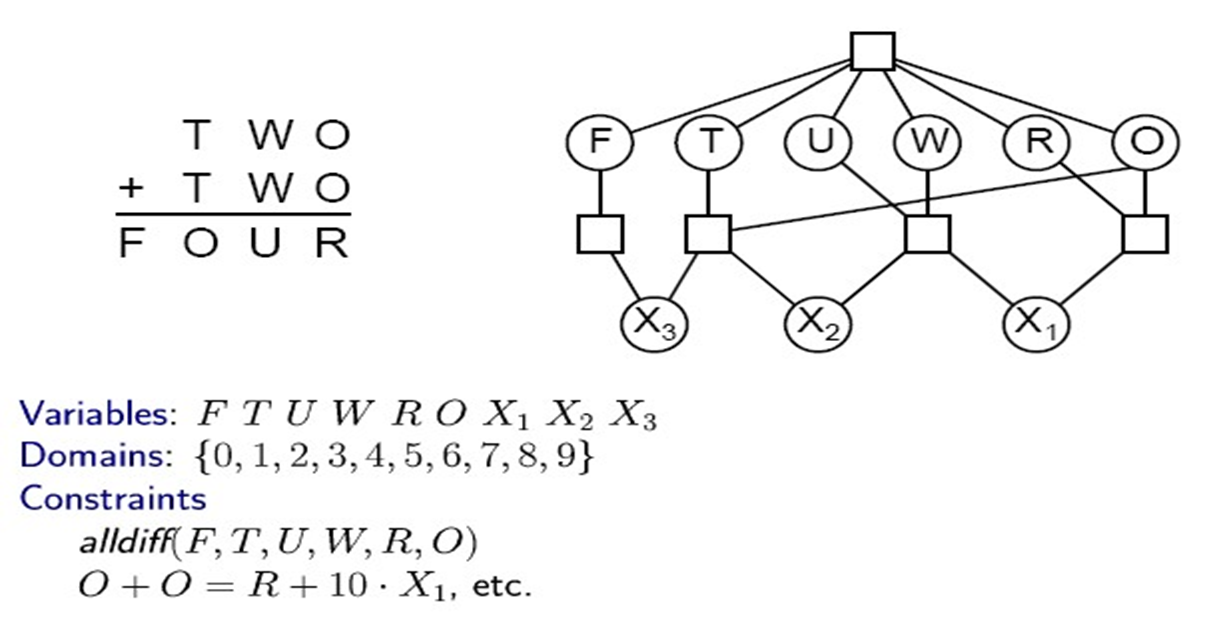
→ not < A = true, B = false, C = true >

Not binary CSP, it depends on the number of literals in each clause.

***N-Queens Problem as CSP:*** Variable Xi for each row i of the board. Domain={1, 2, 3, …, n}  
Constraints: Xi≠Xj queens not in same column. Xi-Xj≠i-j queens not in same SE diagonal. Xj-Xi≠i-j queens not in same SW diagonal.

***More Complex Constraints:-***



***Example: cryptharithmetic:***

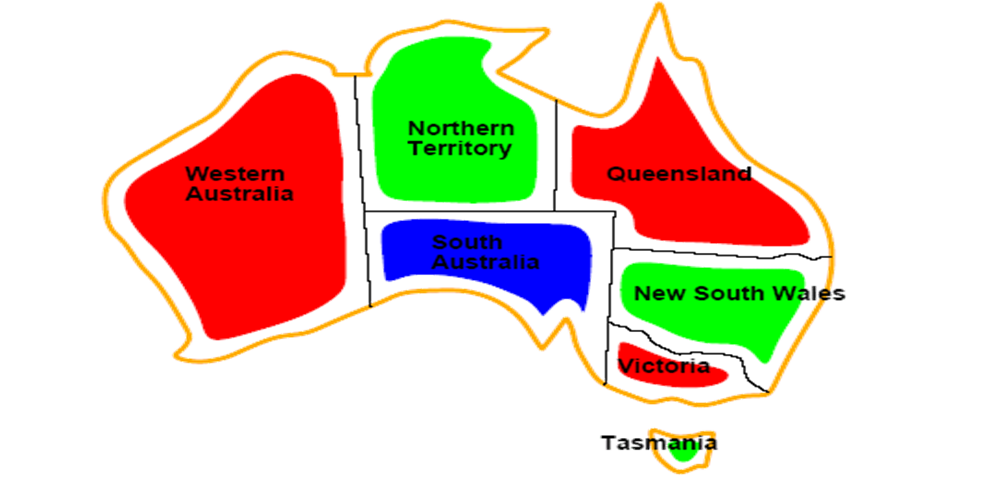
***Commutativity in CSPs:*** The order of variable assignments does not affect the final outcome if constraints are satisfied. Example: Coloring Australian territories,

{WA=red,NT=green} is the same as {NT=green,WA=red}.

**Backtracking Search**

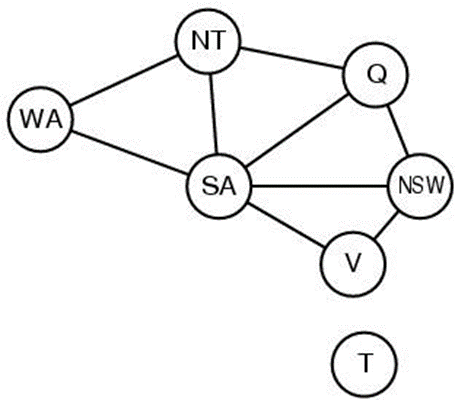
Depth-first search (DFS) algorithm. Assign values to one variable at a time. Backtrack if no legal values are left.

***Algorithm:-***

1. Initialize with an empty assignment.
2. Check completeness: If all variables are assigned, return the assignment.
3. Select a variable: Choose an unassigned variable.
4. Order domain values: For each value in the variable's domain:
   1. If consistent with current assignment:
      1. Assign the value.
      2. Recursively attempt to complete the assignment.
      3. If successful, return the assignment.
      4. If not, remove the value and backtrack.
5. If no values are consistent, return failure.

**Map Coloring as a CSP:-**  
***Variables:*** WA, NT, Q, NSW, V, SA, T

***Domains:*** Di={red,green,blue}

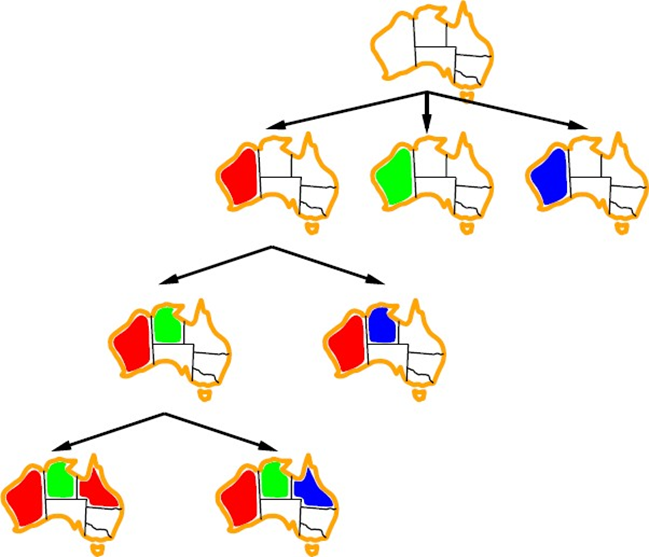
***Constraints:*** adjacent regions must have different colors.

E.g. WA ≠ NT

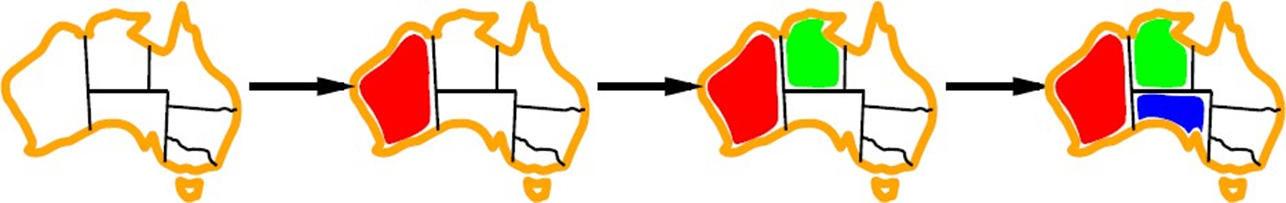
E.g. (WA,NT) ≠ {(red,green),(red,blue),(green,red),…}

Solutions are assignments satisfying all constraints, e.g.

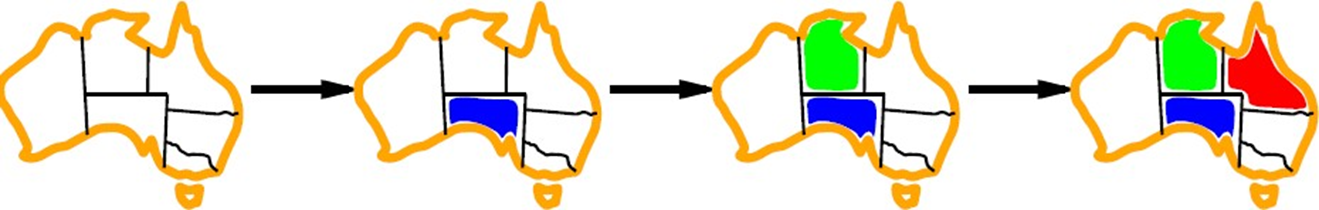
{WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue,T=green}

**Improving Backtracking Efficiency in CSPs**

1. Minimum Remaining Values (MRV): Choose the variable with the fewest legal values remaining. Detects inevitable failure early by focusing on the most constrained variables.

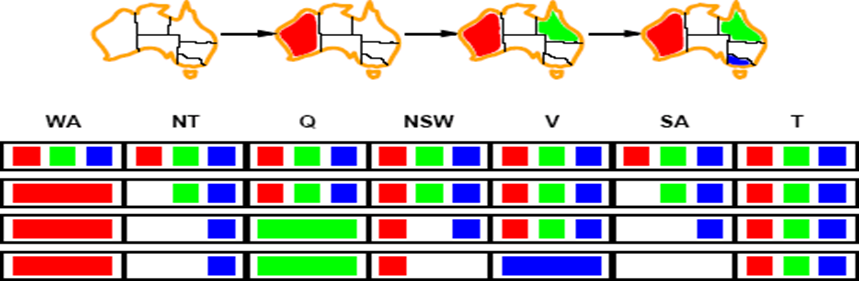


1. Degree Heuristic: Among variables tied by MRV, select the one involved in the most constraints with other unassigned variables. Acts as a tiebreaker for MRV, further focusing the search on critical areas.



1. Least Constraining Value (LCV): For a selected variable, choose the value that leaves the most options open for remaining variables. Avoids failure by maintaining flexibility in future assignments.





**Forward Checking**

Detect inevitable failure early by keeping track of remaining legal values for unassigned variables. Enables us to terminate search when any variable has no legal values, thus avoiding wasted effort.

Example Scenario

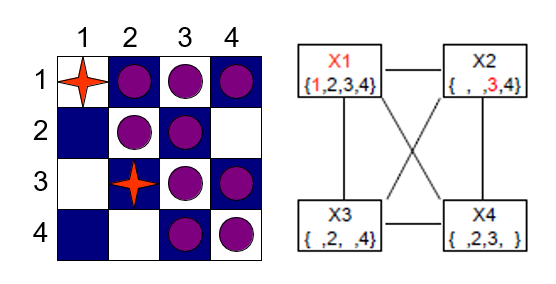
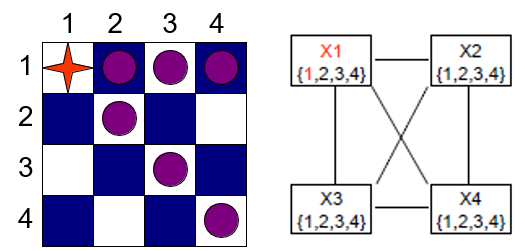
1. Assigning WA=red:   
   Effects on other variables: NT can no longer be red. SA can no longer be red.
2. Assigning Q=green:

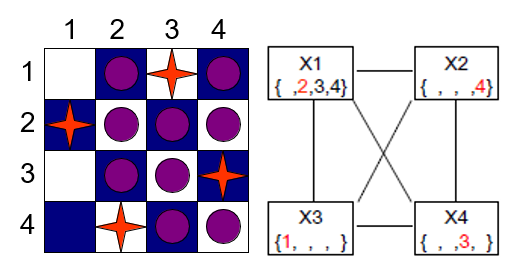
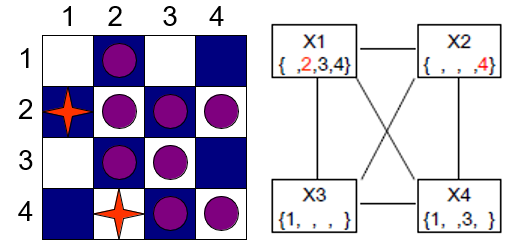
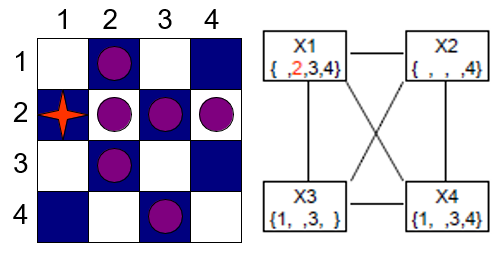
Effects on other variables: NT can no longer be green. NSW can no longer be green. SA can no longer be green.

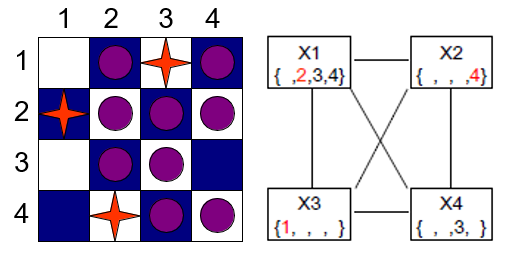
1. If V is assigned blue:  
   SA is left with no legal values. NSW can no longer be blue.

***Backtracking with Forward Checking***

Forward checking detects inconsistency between partial assignments and constraints, allowing the search to backtrack to a previous state and try alternative assignments.

**Example: 4-Queens Problem**





**Arc Consistency:**

X→Y is consistent if, for every value x of X, there is some allowed y for Y.

Example: SA→NSW is consistent if SA=blue and NSW=red.

***AC-3 Algorithm:***

1. Initialize a queue with all arcs in the CSP.

2. While the queue is not empty:

a. Remove the first arc (Xi, Xj) from the queue.

b. If removing inconsistent values from Xi with respect to Xj:

i. Add arcs from neighbors of Xi back to Xi in the queue.

3. Return the CSP.

***Remove Inconsistent Values Function:***

1. For each value x in the domain of Xi:

a. If there's no value y in the domain of Xj satisfying constraints with x:

i. Remove x from the domain of Xi.

ii. Mark that a value has been removed.

2. Return whether any value has been removed.

**K-Consistency:** A CSP is k-consistent if, for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable. E.g., 1-consistency (node-consistency), 2-consistency (arc-consistency), 3-consistency (path-consistency).

Ideal for faster solution finding: O(nd) instead of O(n^2d^3).

However, establishing n-consistency can be exponentially time-consuming.

In **local search for CSPs**, we use a complete-state representation where states may have unsatisfied constraints. We employ operators to reassign variable values, aiming to reach a solution or minimize conflicts.

***Algorithm: MIN-CONFLICTS***

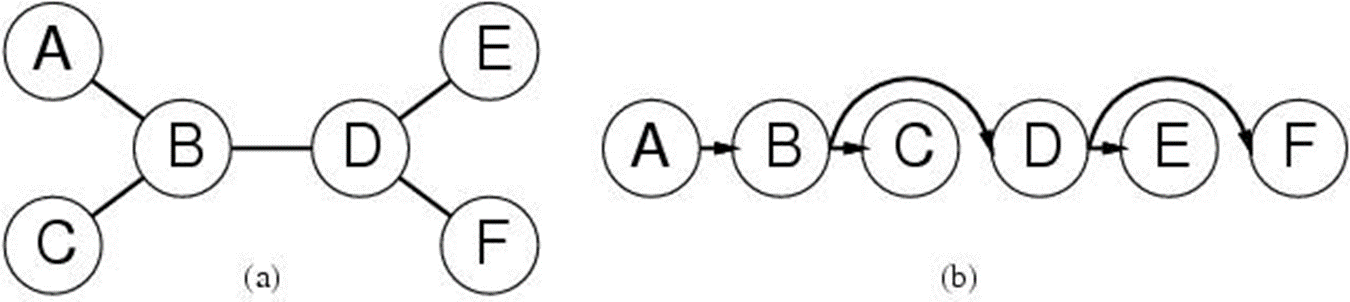
1. Initial State: Start with an initial complete assignment for the CSP.
2. Iteration: Repeat steps until reaching the maximum number of steps or finding a soln.
   1. If the current assignment is a solution, return it.
   2. Randomly choose a conflicted variable.
   3. Select a value for the variable that minimizes conflicts with other variables.
   4. Update the assignment with the selected value.
3. Result: If no solution is found within the maximum steps, return failure.

**Subproblem Identification**

1. Independent Subproblems: Identifying them improves performance.

Example: Coloring Tasmania and mainland are independent subproblems identifiable as connected components of a constrained graph.

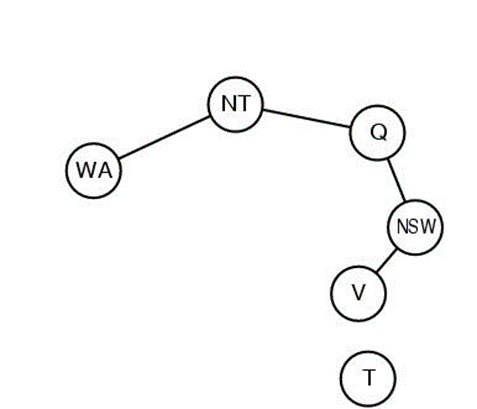
1. Worst Case Solution Cost: For a problem with n variables, worst-case soln cost is O(n/c dc), where c is the number of variables in each subproblem. Linear in n instead of exponential (O(dn)).

**Tree-Structured CSPs**

Theorem: If the constraint graph has no loops, the CSP can be solved in O(nd^2) time.

Any tree-structured CSP can be solved linearly in the number of variables.

Variables ordered from root to leaves, each node's parent precedes it.

**Nearly Tree-Structured CSPs**

General constraint graphs can be reduced to trees.

Two approaches: removing certain nodes or collapsing certain nodes.

Assign values to some variables to form a tree, then remove inconsistent values. NP-hard to find smallest cutset, but approximation algorithms exist.

Constraint graph decomposed into connected subproblems. Each subproblem solved independently and results combined.