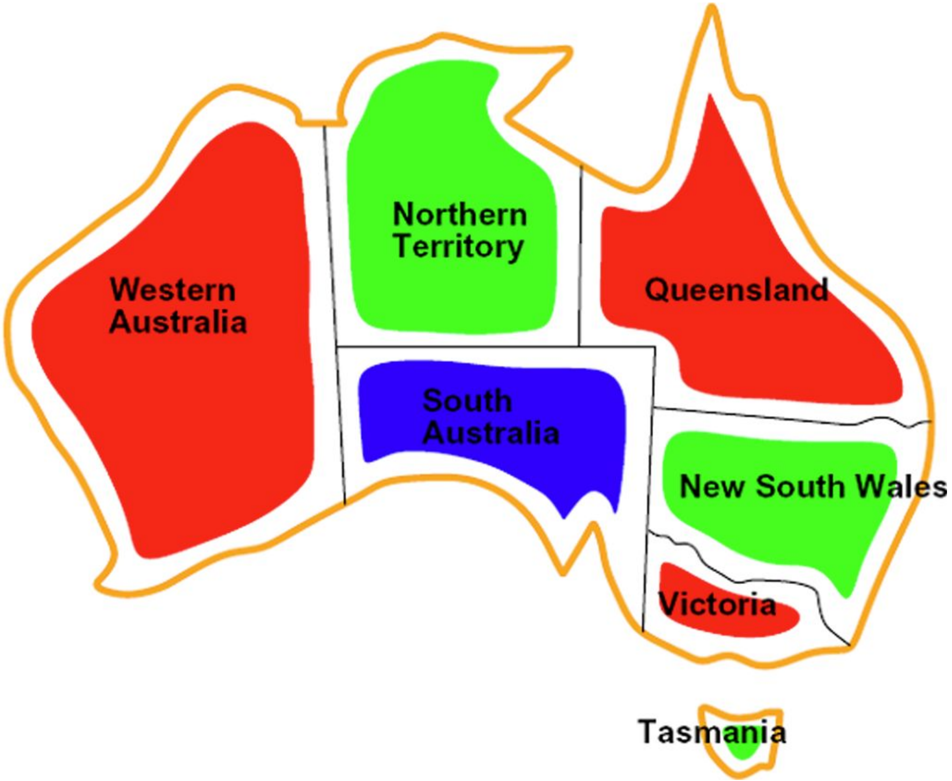
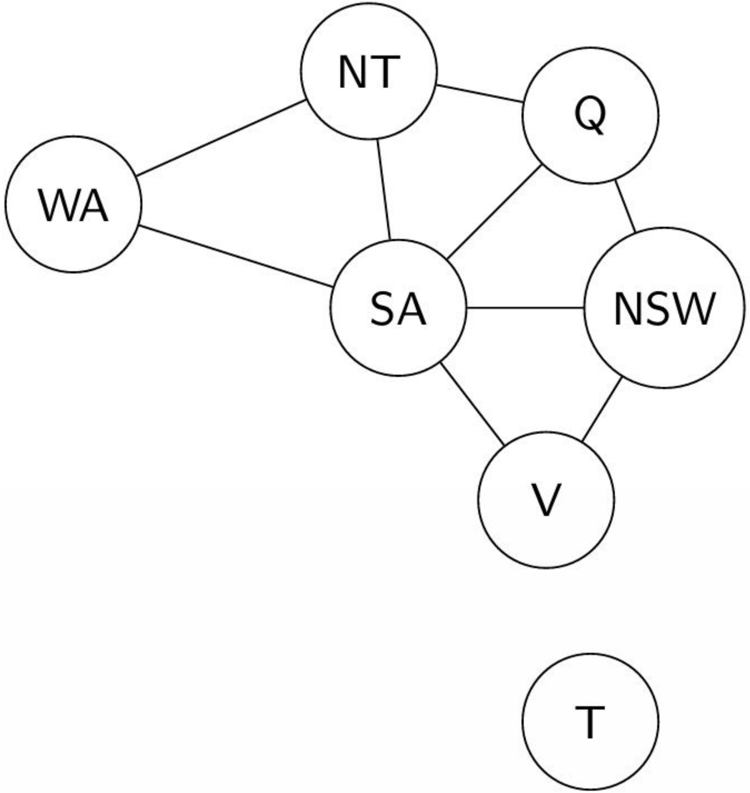
Constraint Satisfaction Problem II

CSE 4617: Artificial Intelligence

Isham Tashdeed
Lecturer, CSE

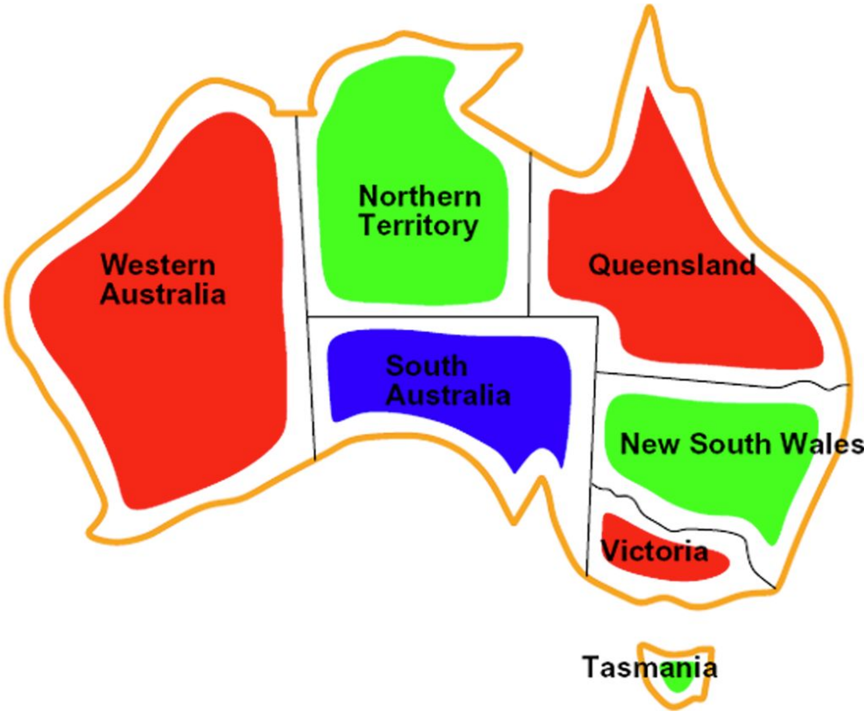
Examples of CSPs



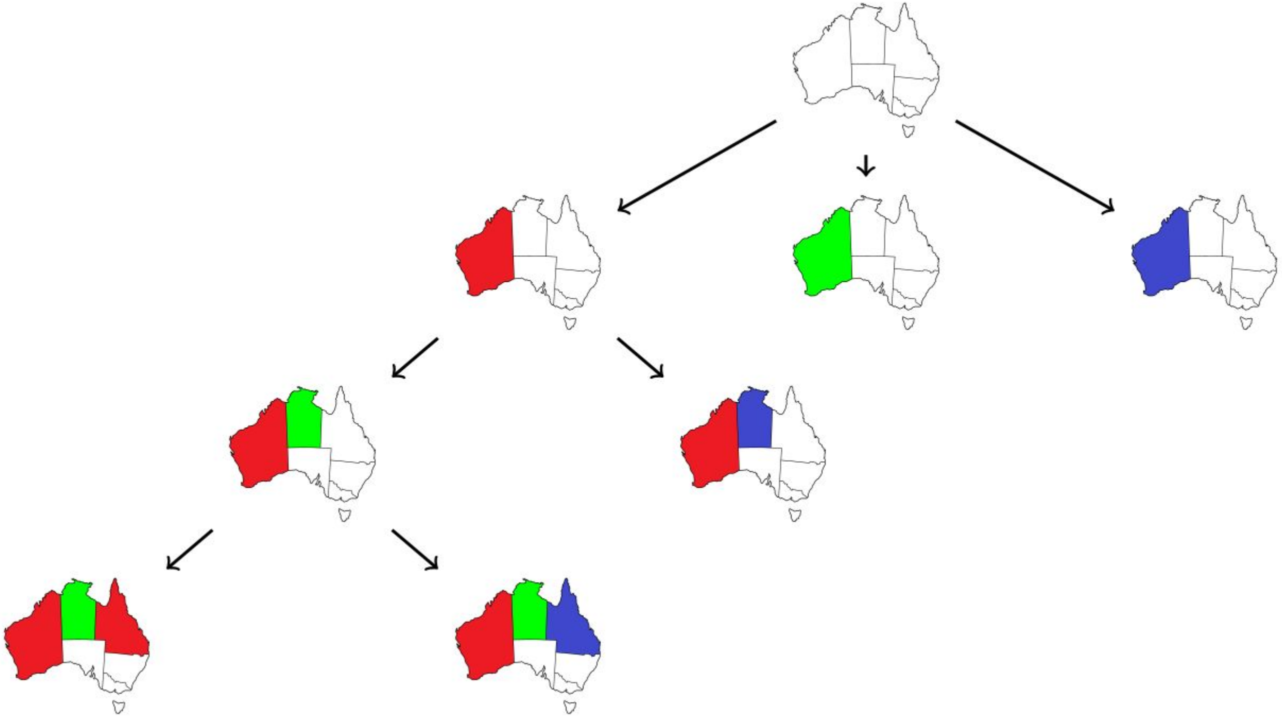


Examples of CSPs

- Variables
  - WA, NT, Q, NSW, V, SA, T
- Domains
  - $D = \{\text{red, green, blue}\}$
- Constraints → Adjacent regions must be different colored
  - Implicit: $WA \neq NT$, …
  - Explicit: $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), (\text{blue, green}), \ldots\}$
- Solution
  - An assignment of variables that satisfy all constraints



Backtracking Search



Backtracking Search

```
BACKTRACKING-SEARCH (csp) → returns a solution or failure
    return RECURSIVE-BACKTRACKING ({}, csp)


RECURSIVE-BACKTRACKING (assignment, csp) → returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for value in ORDER-DOMAIN-VALUE(var, assignment, csp):
        if value is consistent with assignment given CONSTRAINTS[csp]:
            then add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING (assignment, csp)
            if result is not failure then return result
            remove {var = value} from assignment
    return failure
```

How to Improve Backtracking Search

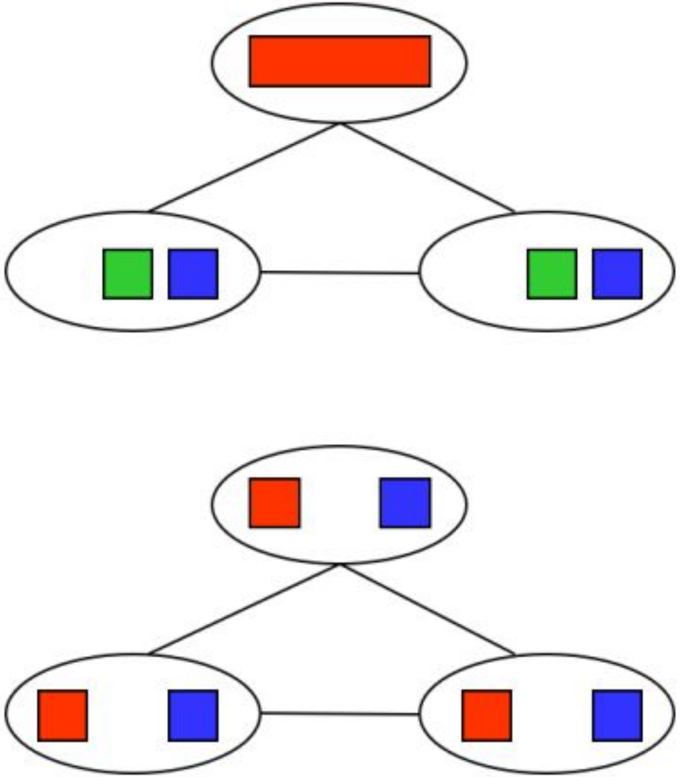
- Filtering
  - Limiting our choices for variable assignment
  - Detects an inevitable failure early
- Ordering
  - In what order should we assign variables
  - Does it actually matter?
- Structure
  - Can we exploit the structure of a problem?

Certain values as soon as arc consistency is checked

Filtering: Arc Consistency

After enforcing arc consistency:

- Can have one solution left
- Can have multiple solutions left
- Can have no solutions left (and not know it)



Arc consistency only checks pairs :(

k-Consistency

k-Consistency

Increasing degrees of consistency:

- 1-Consistency → Each node's domain has a value which meets that node's unary constraints
- 2-Consistency → For each pair of nodes, any consistent assignment to one can be extended to the other → Arc Consistency
- k-Consistency → For each k nodes, any consistent assignment to k-1^{th} can be extended to the k^{th} node
  - As the value of k increases, it gets more difficult to compute k-Consistency
- Strong k-Consistency → Nodes are k - 1, k - 2, k - 3, … , 2, 1 consistent
- Strong n-consistency → Can solve without backtracking ! → n is the number of variables

Structure

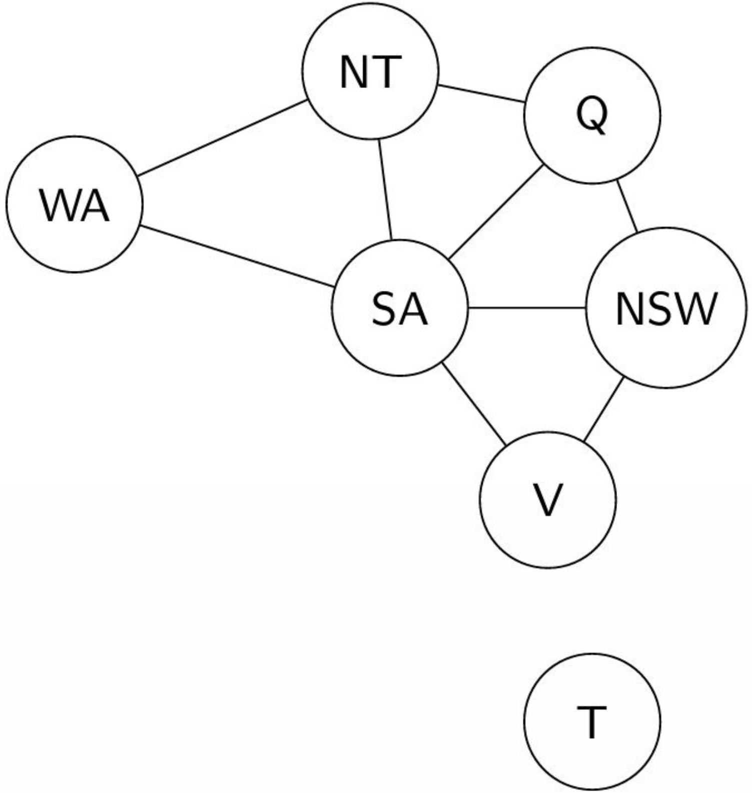
Structure

In this specific graph:

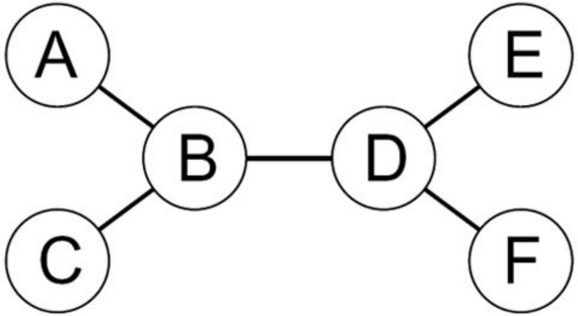
- There are 2 independent subproblems
- Independent subproblems are identifiable as connected components of constraint graph
    - How to detect independent subproblems?

If a graph of n variables can be broken down into subproblems of only c variables:

- The worst case solution cost is $O((n/c)(d^c))$
- This is much better than the worst case of naive DFS which is $O(d^n)$



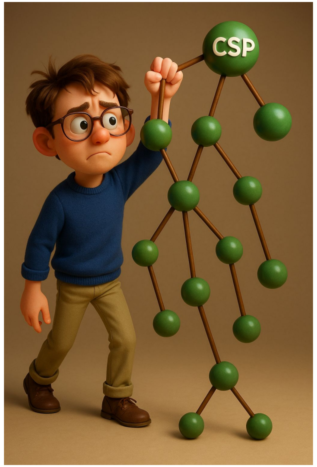
Tree-Structured CSPs



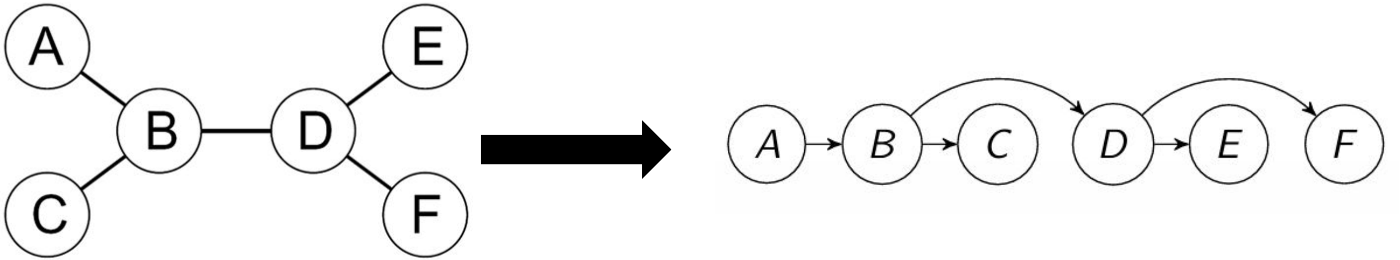
In most cases, encountered graphs have a tree- like structure

For tree-structured CSPs:

- Order: Choose a root variable, order variables so that the parents precede the children



Use Topological Sort

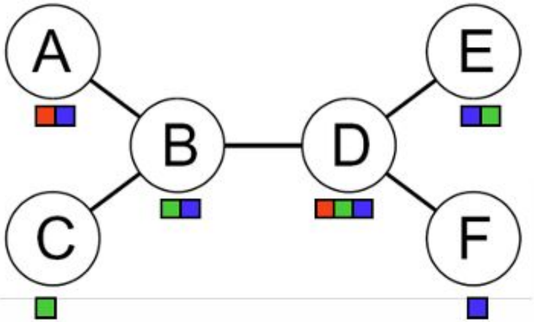


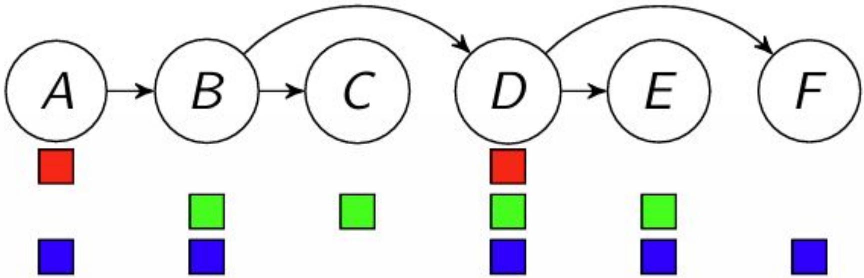
Notice that it is now a DAG, even though the original graph wasn't

Tree-Structured CSPs

For tree-structured CSPs:

- Order: Choose a root variable, order variables so that the parents precede the children
- Remove Backward: Start from the rightmost node and keep going left as you remove inconsistent values from nodes to ensure consistency of the arcs
- Assign Forward: Assign values from left to right nodes by taking valid values from the domain





Tree-Structured CSPs

For tree-structured CSPs:

- Order: Choose a root variable, order variables so that the parents precede the children
- Remove Backward: Start from the rightmost node and keep going left as you remove inconsistent values from nodes to ensure consistency of the arcs
- Assign Forward: Assign values from left to right nodes by taking valid values from the domain

Runtime: $O(nd^2)$

- Go from tail to head and then head to tail → $O(n)$
- Check pairs of values for consistency/assignment → $O(d^2)$

Tree-Structured CSPs

I can make a few claims about tree-structured CSPs:

- After a backward pass, all root-to-leaf arcs are consistent → Why?
  - Each $X \rightarrow Y$ was made consistent at one point and Y's domain could not have been reduced thereafter
- If all root-to-leaf arcs are consistent, forward assignment will not backtrack
  - Arc consistency implies for X → Y, for a consistent assignment of X so far, we have a consistent assignment of Y

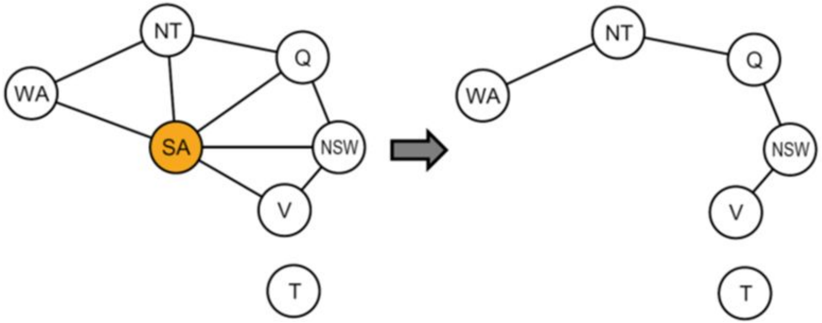
Why doesn't this algorithm work with cycles in the constraint graph?

Tree-Structured CSPs

```
TREE-CSP-SOLVER (csp) → returns a solution or failure
    X ← set of variables
    N ← number of variables in X
    root ← any random variable in X
    D ← domain of possible values

    X ← TOPOLOGICAL-SORT(X, root)
    for i in range(n → 2):
        MAKE-ARC-CONSISTENT(PARENT(X_i), X_i)
        if no consistency then return failure
    for i in range(1 → n):
        X_i ← any consistent value from D_i
        if no consistency then return failure
    return X
```

Improving Structure

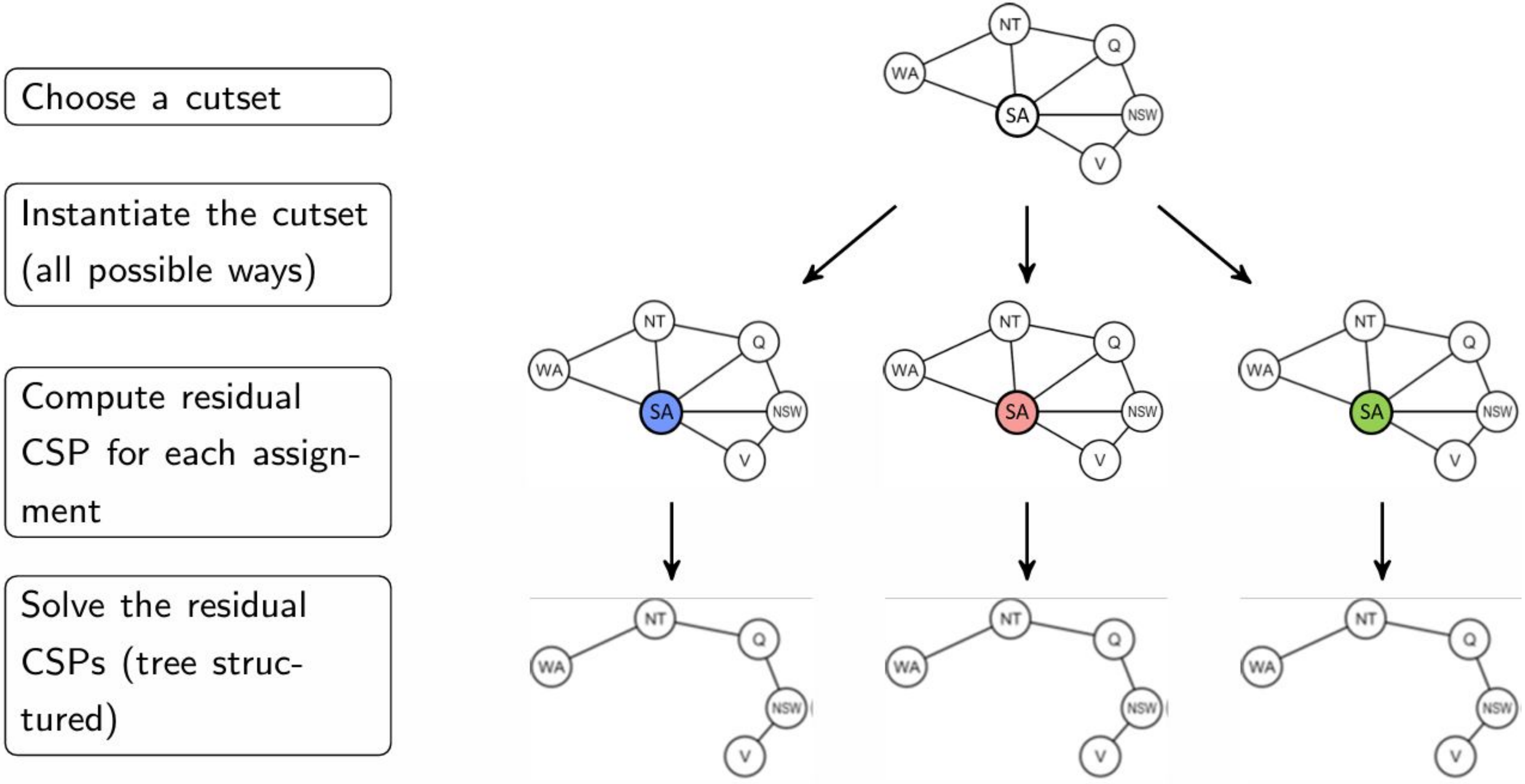


- Conditioning → Instantiate a variable, prune its neighbors' domains
- Cutset conditioning → Instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size c gives runtime $O(d^c\,(n - c)d^2)$:

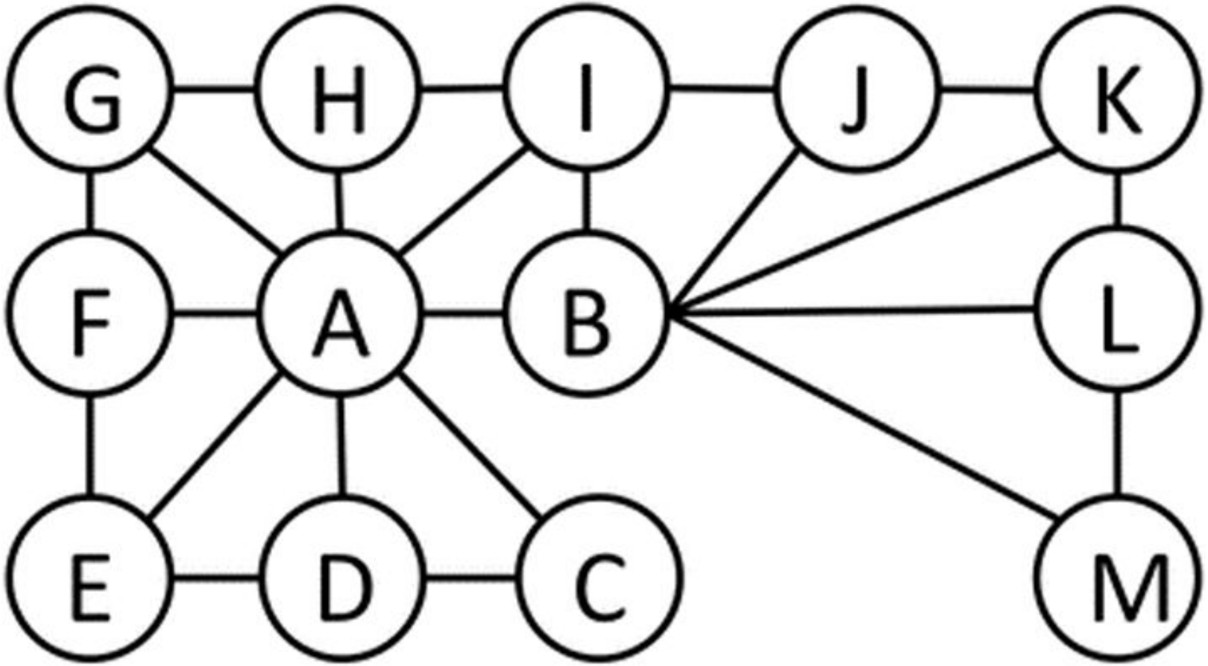
- Total instantiations → $O(d^c)$
- Total remaining subproblems → $(n - c)$

Cutset Conditioning



Cutset Conditioning

Find the smallest cutset for the following graph that gives us a tree:



Iterative Algorithm for CSPs

Iterative Algorithm for CSPs

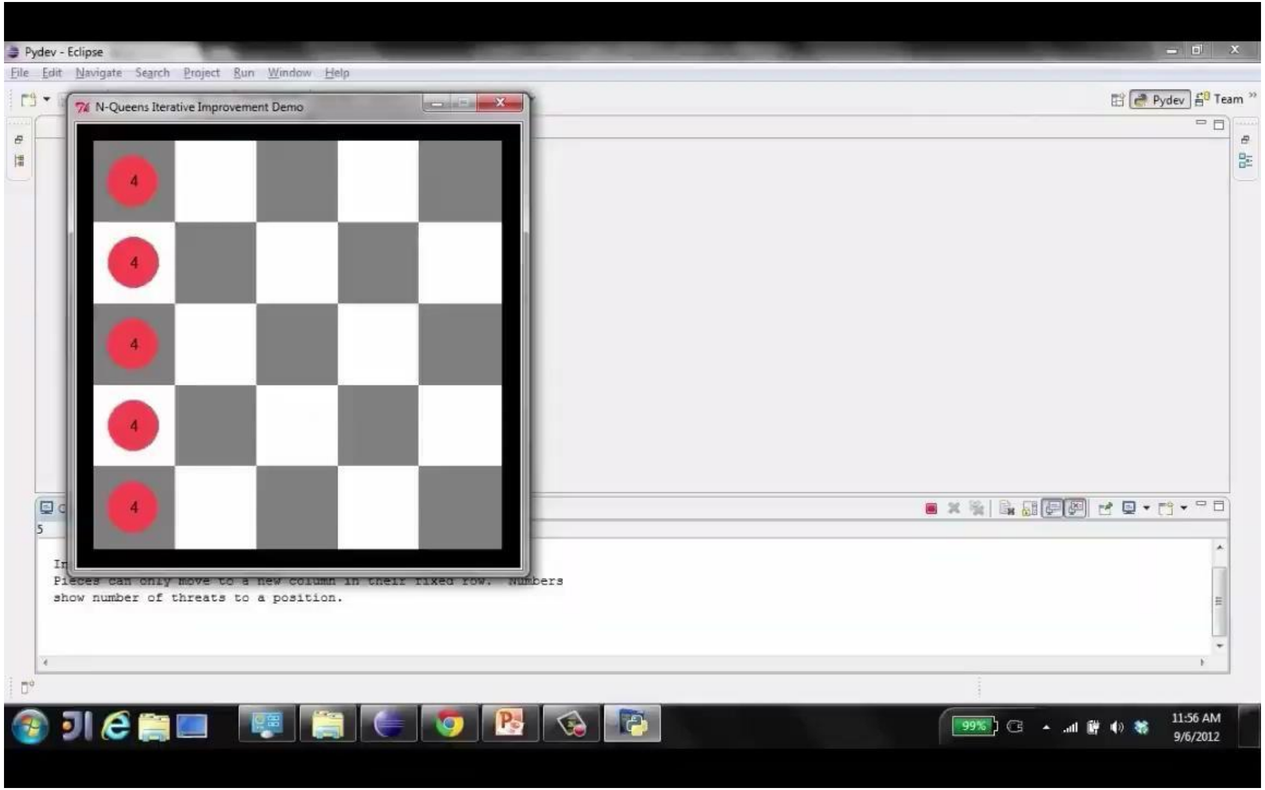
Local search methods typically work with "complete" states, i.e., all variables assigned

To apply the same idea to CSPs:

- Take an assignment with unsatisfied constraints
- Operators reassign variable values
- No need for fringe!

Keep randomly selecting any conflicted variable and set the value to the one with minimum conflicts

Iterative Algorithm for CSPs



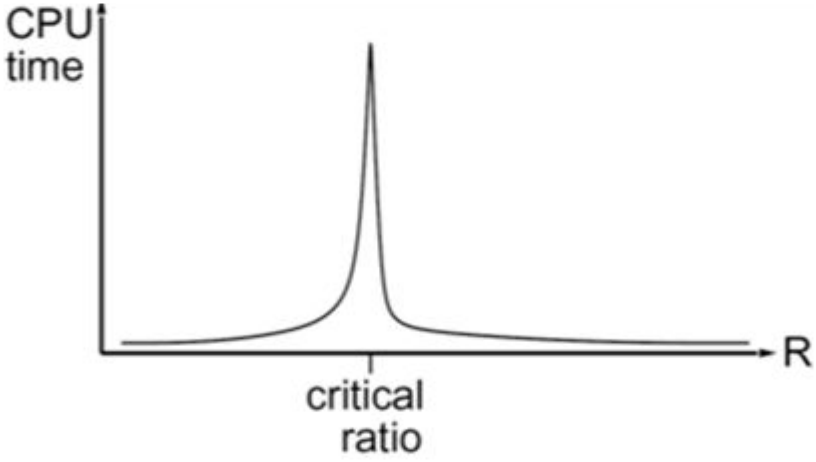
Iterative Algorithm for CSPs

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability

- Such as $n = 10{,}000{,}000$

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

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



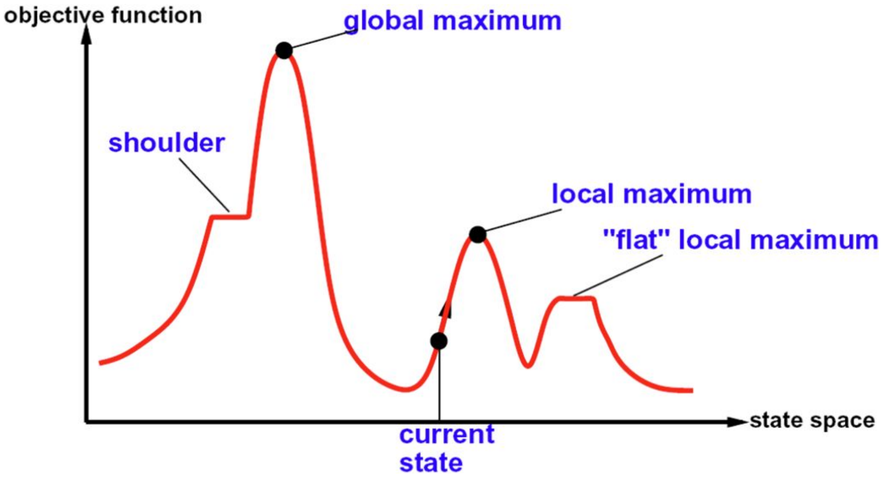
Local Search

Local Search

- Tree search keeps unexplored alternatives on the fringe → This ensures completeness
- Local search improves a single option until you can't make it better → no fringe!

General idea:

- Randomly start
- Repeat: move to the best neighboring state
- If no neighbors better than current, quit
- Not complete
- Not optimal
- Very efficient in finding good-enough solutions



Additional Resources

- [Backtracking Search Simulator]
- [What are Genetic Algorithms?]
- [A.I. Learns To Walk]

Thank you