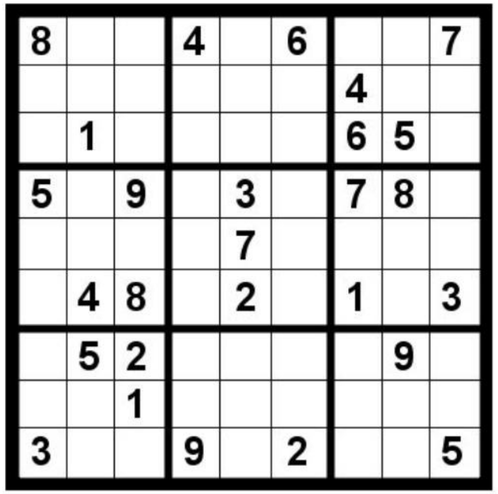
# CS 6511: Artificial Intelligence

## Constraint Satisfaction Problems



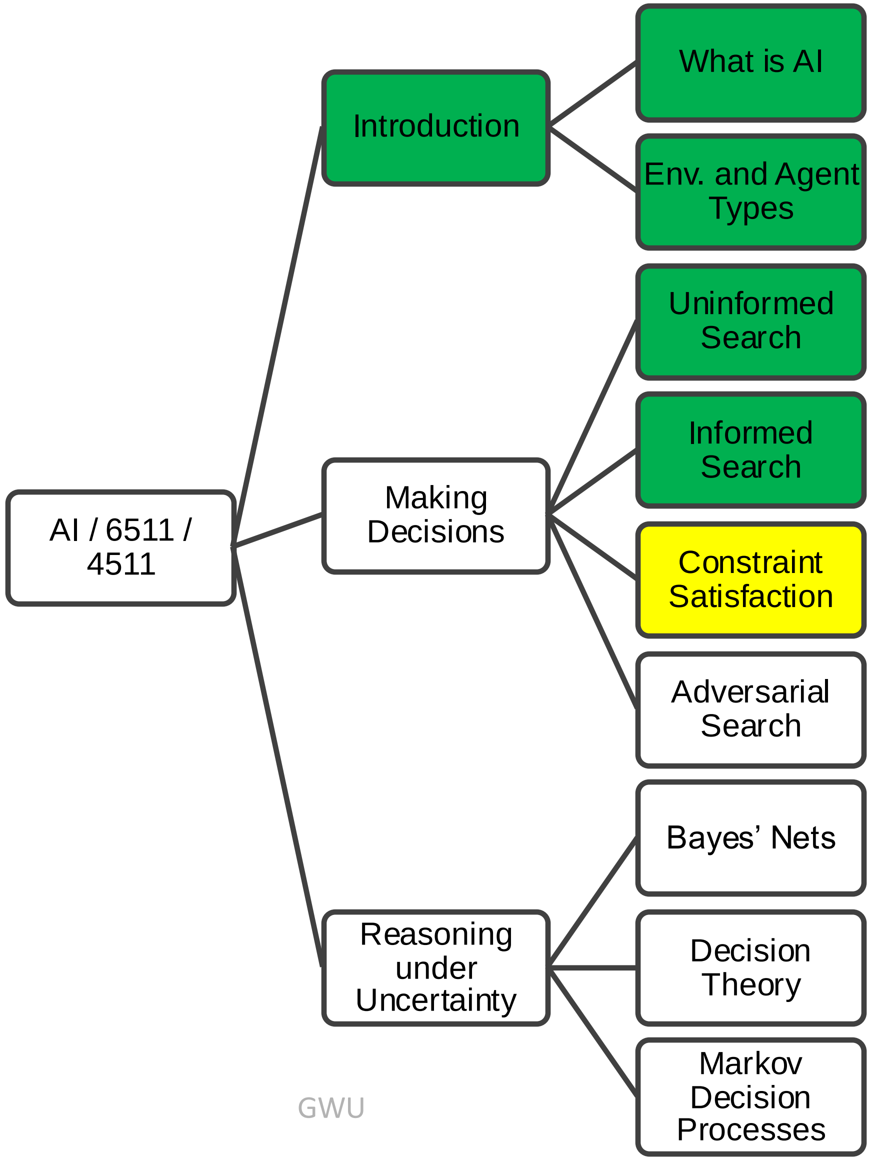


Amrinder Arora
The George Washington University



Attributions:
[A previous version of these slides was created by Dan Klein and Pieter Abbeel for Intro to AI at UC Berkeley. http://ai.berkeley.edu]
https://ktiml.mff.cuni.cz/~bartak/constraints/propagation.html

# Course Outline



# CSPs: Learning Objectives

- Basics and Definitions
  - Modeling as a graph
- Formulation as a Search
  - Backtracking with Heuristics / MRV / LCV
- Constraint Propagation
  - Forward Checking / Arc Consistency / K-Consistency
- Graph Decomposition
  - Cutsets / Vertex Ordering

# Constraint Satisfaction Problems

- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables $\boldsymbol{X_i}$ with values from a domain $\boldsymbol{D}$ (sometimes $\boldsymbol{D}$ depends on $\boldsymbol{i}$)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Simple example of a *formal representation language*

- Allows useful general-purpose algorithms with more power than standard search algorithms

# CSPs can be considered a special case of Search

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

# CSP Examples

- Coloring
- Scheduling – Assignment / Timetabling / Transportation / Factory
- Hardware configuration / Circuit layout
- Fault diagnosis

- Variables can be:
  - Finite (Colors 1,2,3..k)
  - Infinite, but countable (natural numbers)
  - Infinite, uncountable (real numbers)

# Example: Map Coloring

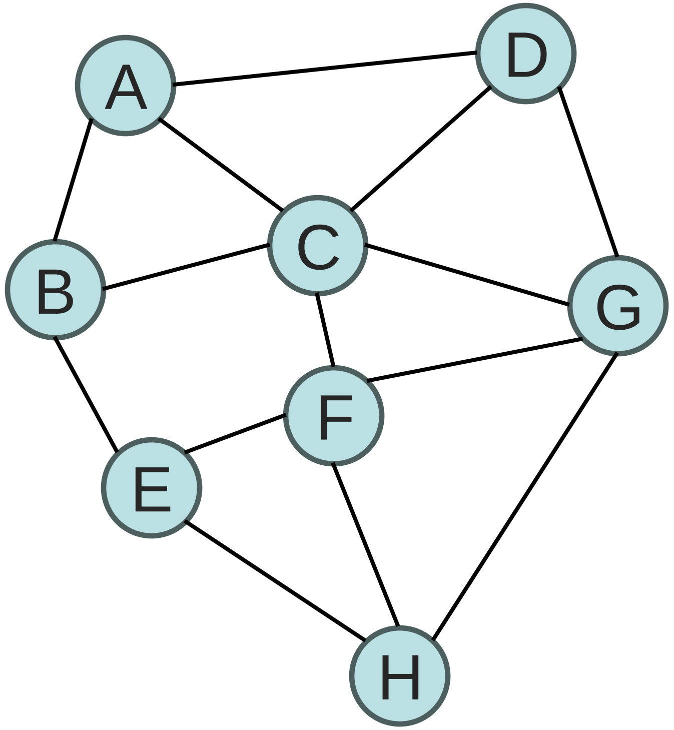
- Variables: A, B, C, D, E, F, G, H
- Domains: D = {r, g, b}
- Constraints: adjacent regions must have different colors

  Implicit:

  Explicit: $(\mathrm{WA}, \mathrm{NT}) \in \{(\mathrm{red}, \mathrm{green}), (\mathrm{red}, \mathrm{blue}), \ldots\}$

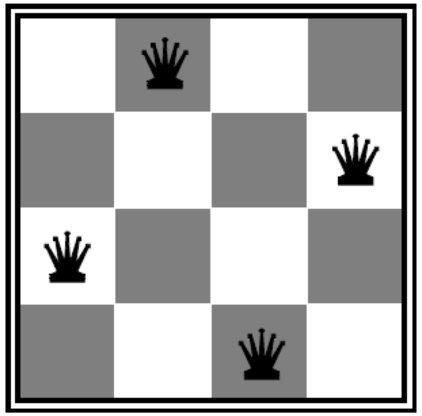
- Solutions are assignments satisfying all constraints, e.g.:

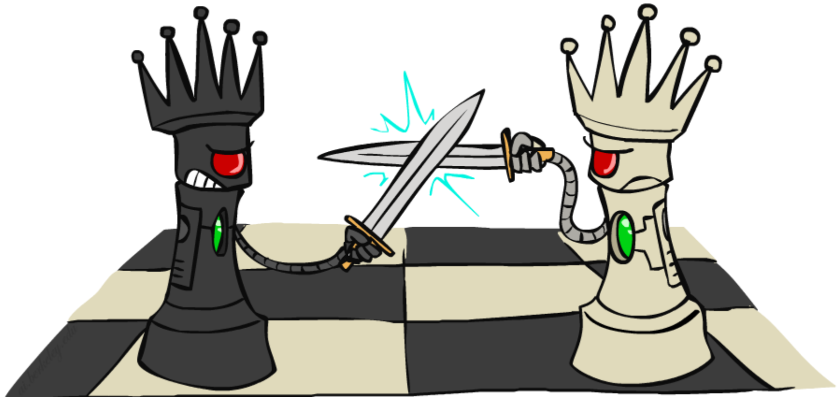
  $\{\mathrm{WA}{=}\mathrm{red},\ \mathrm{NT}{=}\mathrm{green},\ \mathrm{Q}{=}\mathrm{red},\ \mathrm{NSW}{=}\mathrm{green},\ \mathrm{V}{=}\mathrm{red},\ \mathrm{SA}{=}\mathrm{blue},\ \mathrm{T}{=}\mathrm{green}\}$



# Example: N-Queens

- Formulation 1:
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints



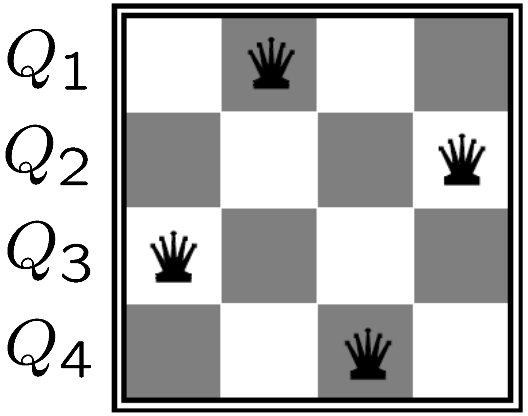


$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

# Example: N-Queens

- Formulation 2:
  - Variables: $Q_k$
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:




Implicit: $\forall i, j \ \text{non-threatening}(Q_i, Q_j)$

Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$

$\ldots$

- 2^n^2 vs. n^n
- N = 10
  - 2^100 >>10^10

- N = 100
  - 2^10000 ? 100^100. 1 with 200 zeros
  - (2^10)^1000
  - 10^3^1000
  - 3000 zeroes. Vs. 200 zeros.

# CSPs: Main Ideas

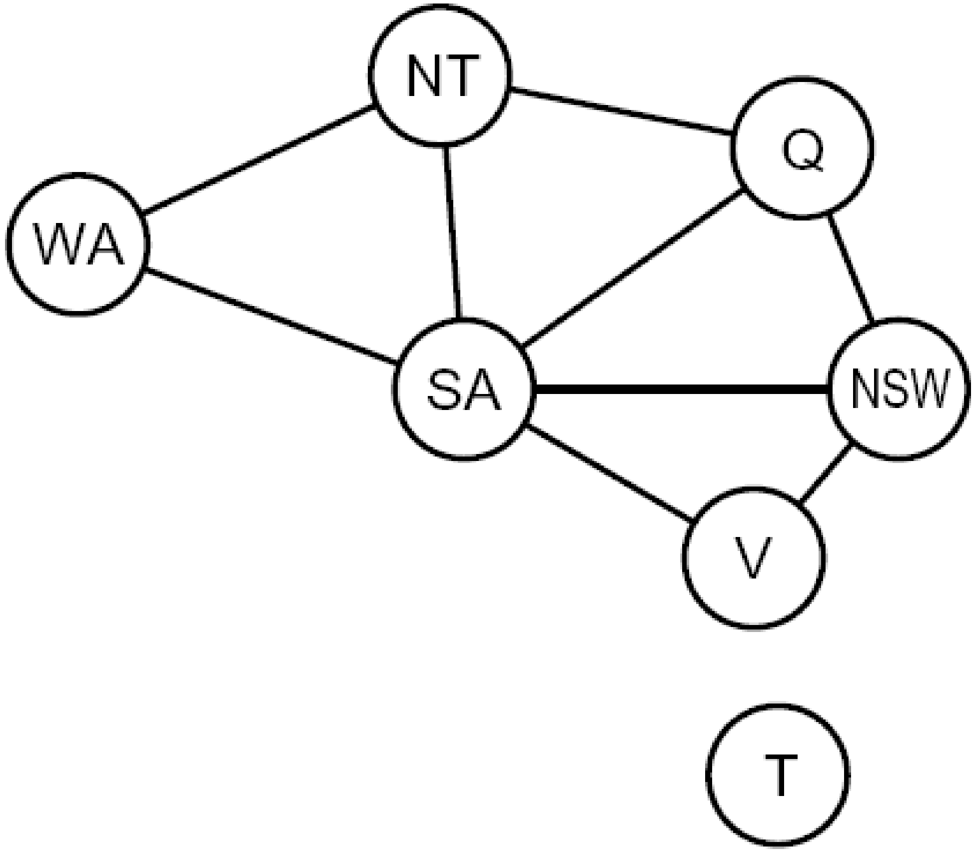
- Representing CSPs
  - Constraint Graphs
- Backtracking Search for CSPs
  - Heuristics for improving this, by
    - Ordering variables
    - Ordering values
  - Backjumping
- Constraint Propagation
  - Forward Checking
  - Arc Consistency (AC3 algorithm)
  - Using structure of constraint graph
- Local Search

# Representing CSPs: Modeling Constraints

- Constraints can be articulated using a specific language
- Or, using constraint graphs

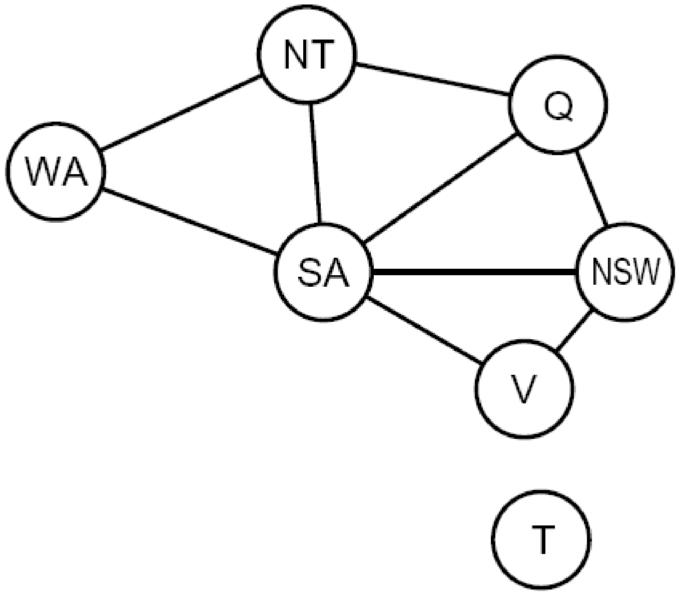
- Constraint graphs can be drawn two different ways:
  - Using variables only, lines drawn between them (Can only model binary constraints, but the graph is easy to see)
  - Using variables as circle nodes and special rectangle nodes that serve as constraints (Can model binary, ternary and in general n-ary constraints)

# Constraint Graphs



# Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



# Example: Cryptarithmetic

- Variables:

  $F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3$

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

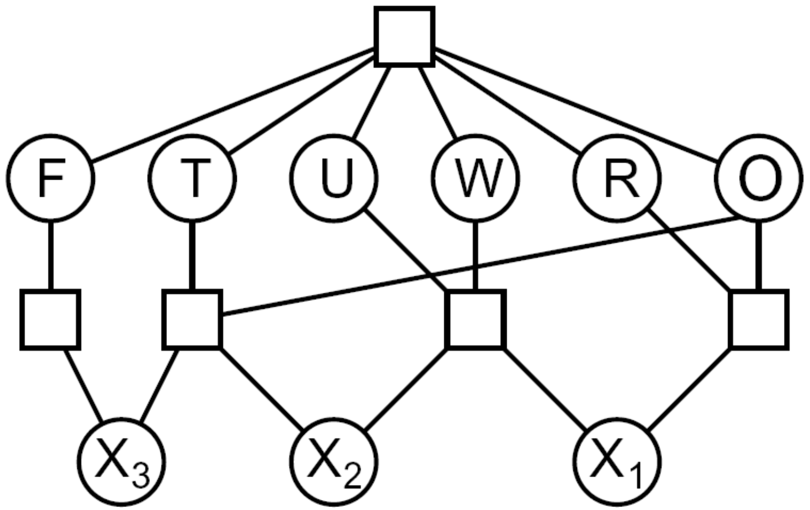
- Constraints:

  $\text{alldiff}(F, T, U, W, R, O)$

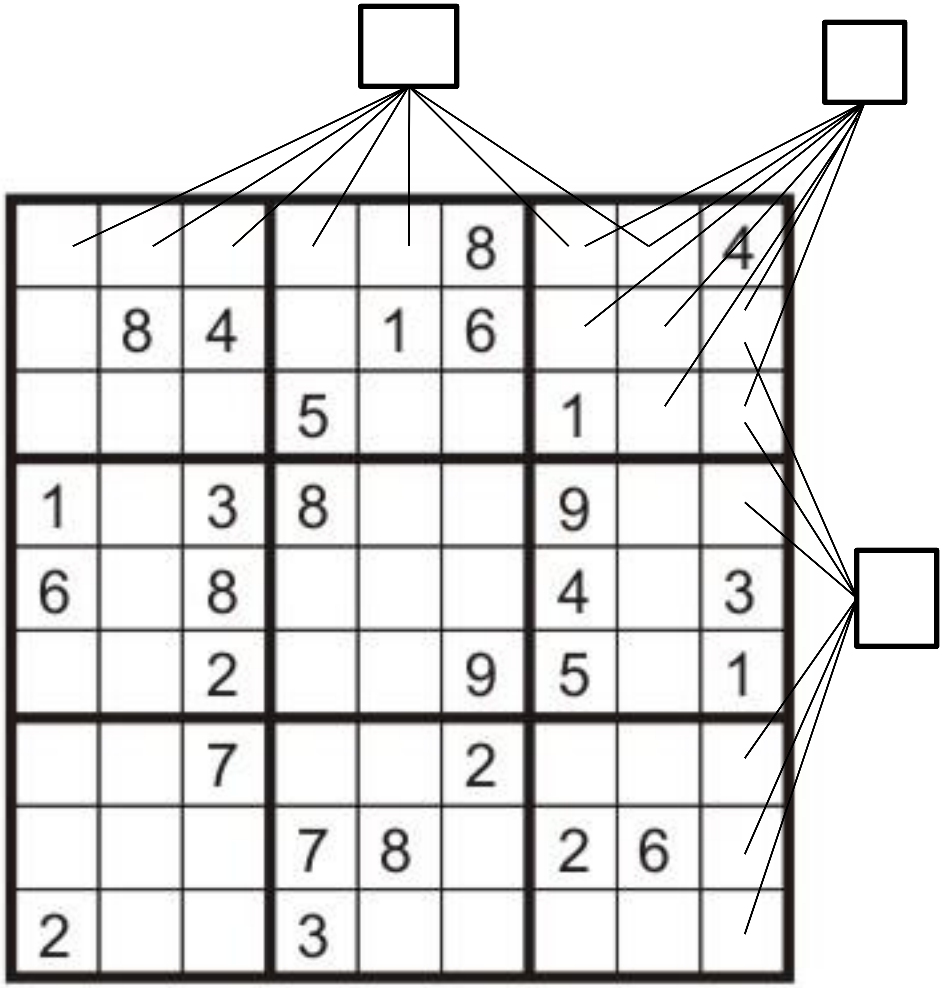
  $O + O = R + 10 \cdot X_1$

  $\cdots$

```
    T W O
+   T W O
---------
  F O U R
```



# Example: Sudoku




- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

# Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Size $d$ means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

# Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), e.g.:

$$\text{SA} \neq \text{green}$$

  - Binary constraints involve pairs of variables, e.g.:

$$\text{SA} \neq \text{WA}$$

  - Higher-order constraints involve 3 or more variables:
    e.g., cryptarithmetic column constraints

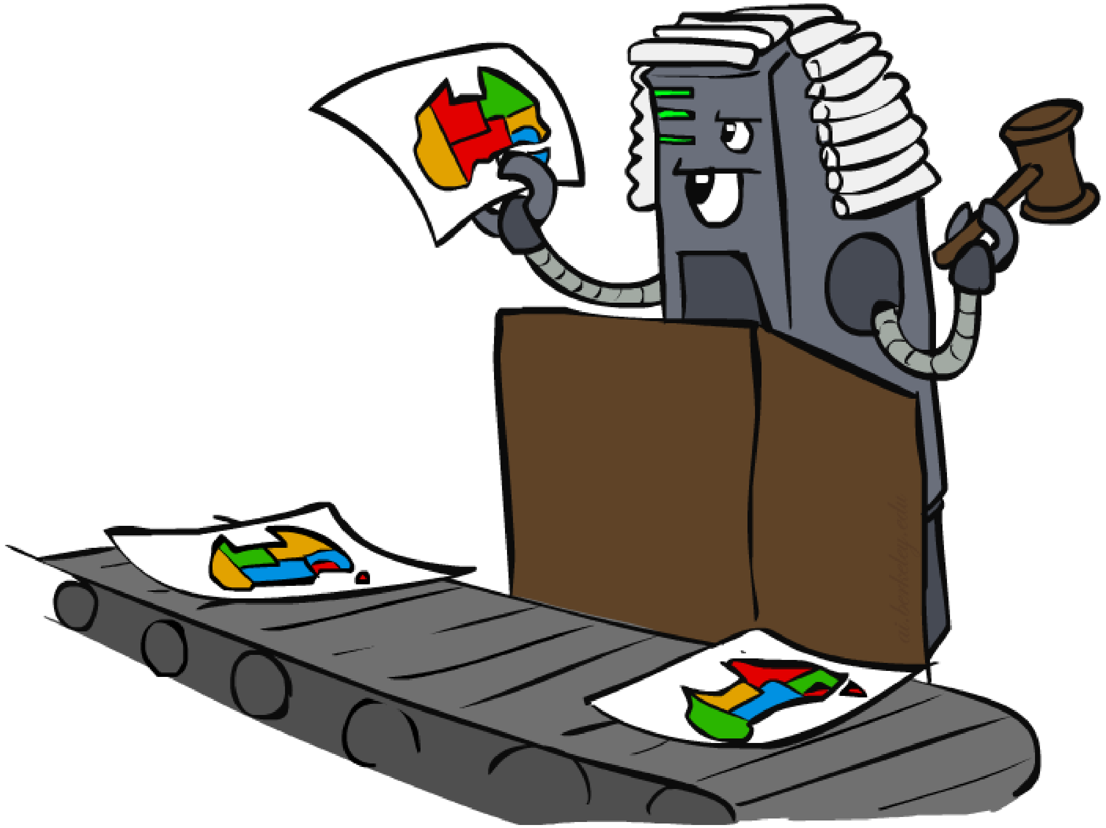
- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets)

# Standard Search Formulation

- Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

- We'll start with the straightforward, naïve approach, then improve it

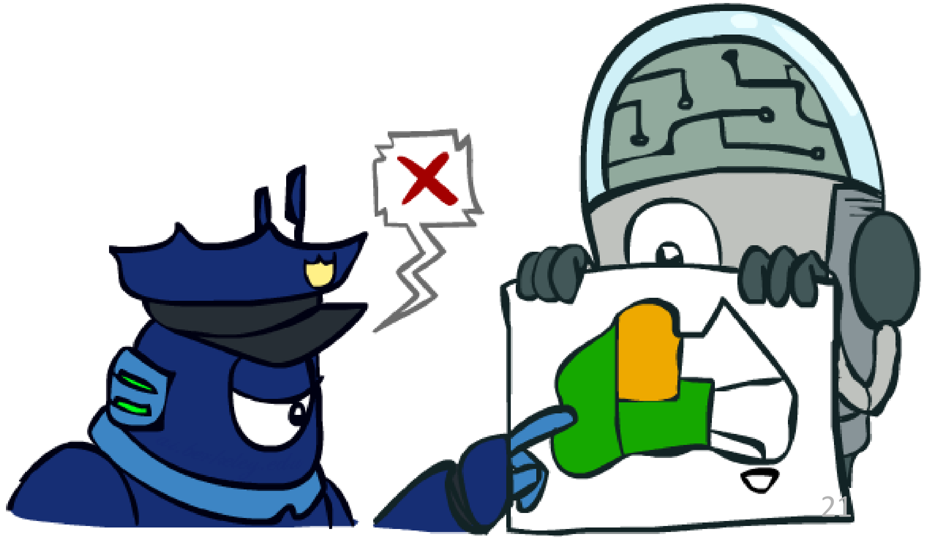


# CSPs: Main Ideas

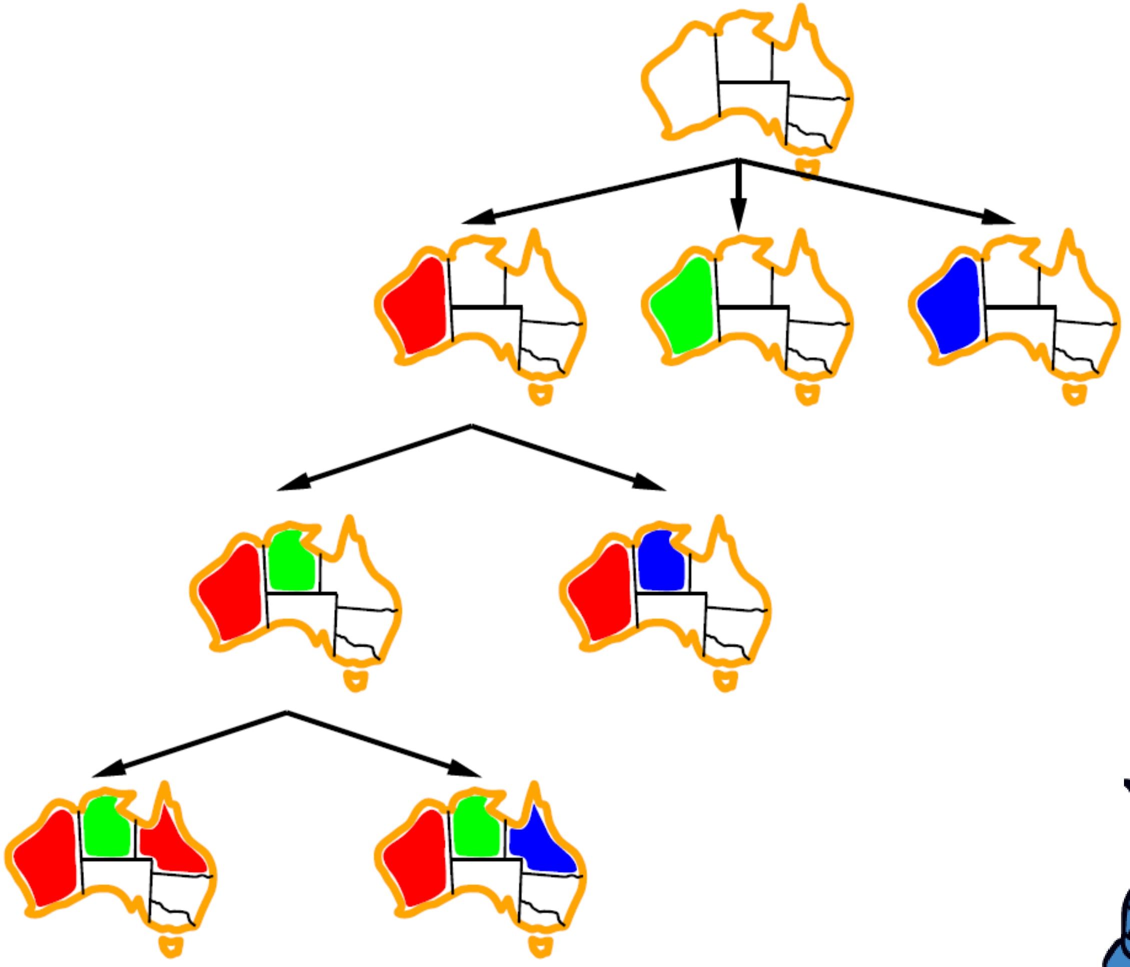
- Representing CSPs
  - Constraint Graphs
- Backtracking Search for CSPs
  - Heuristics for improving this, by
    - Ordering variables
    - Ordering values
  - Backjumping
- Constraint Propagation
  - Forward Checking
  - Arc Consistency (AC3 algorithm)
  - Using structure of constraint graph
- Local Search

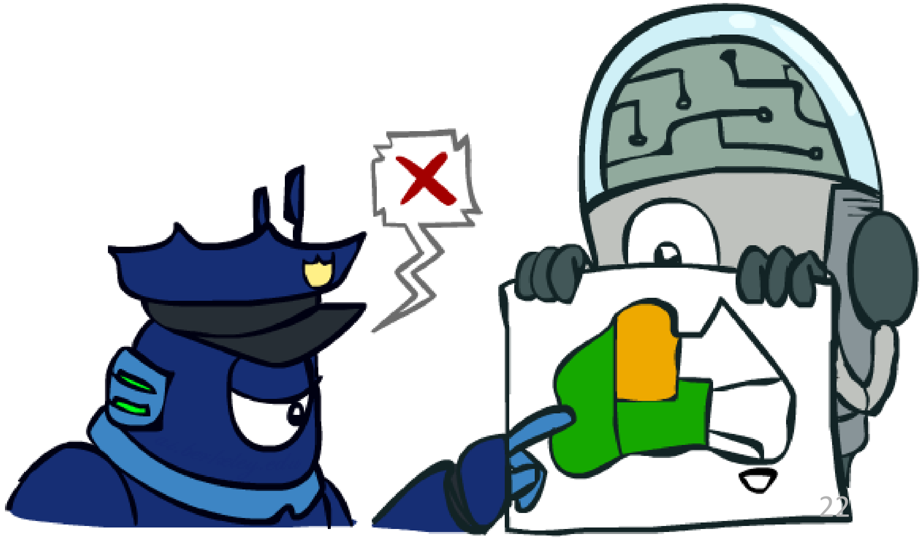
# Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - That is, consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search* (not the best name)
- Can solve n-queens for $n \approx 25$



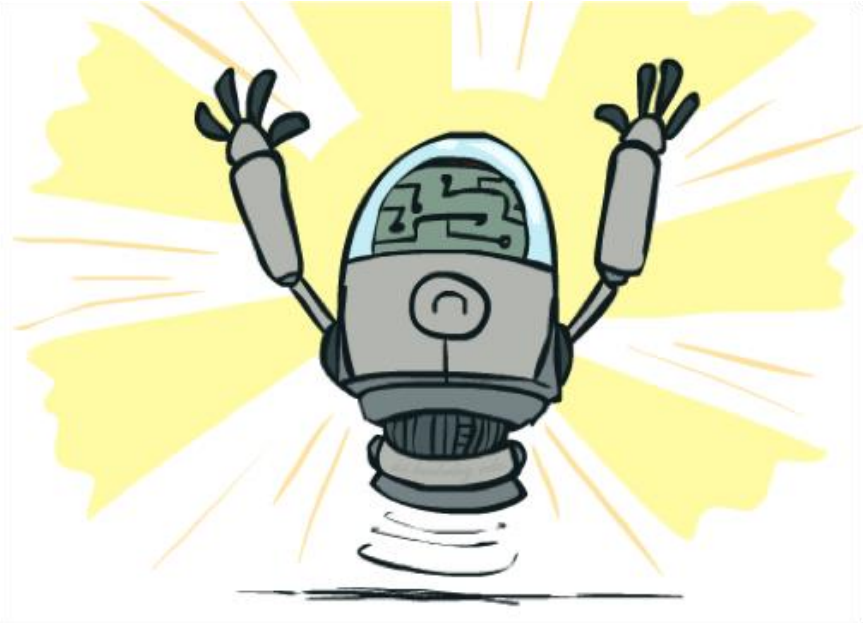
# Backtracking Example





# Improving Backtracking

- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?



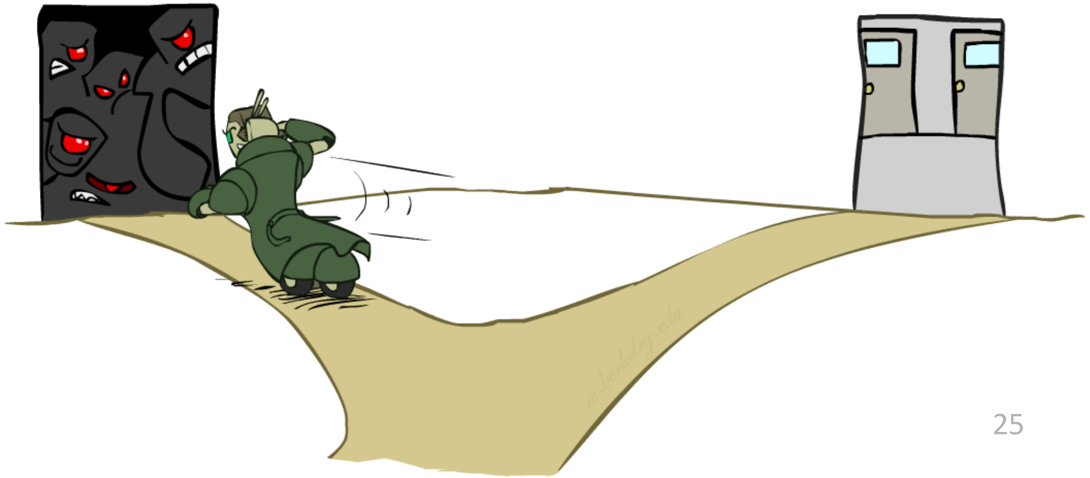
Ordering

# WHICH VARIABLE TO PICK
# WHICH VALUE TO TRY

# Ordering: Which Variable to Pick

- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain

- Why min rather than max?
- Also called “most constrained variable”
- “Fail-fast” ordering



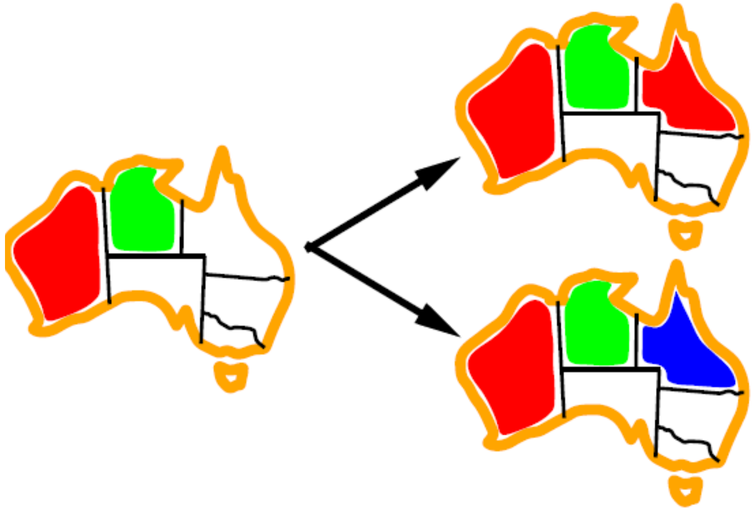
# Tie Breaking Rule

- If two variables both have minimum remaining values, then from within these two, we can consider a variable that is involved in more constraints.

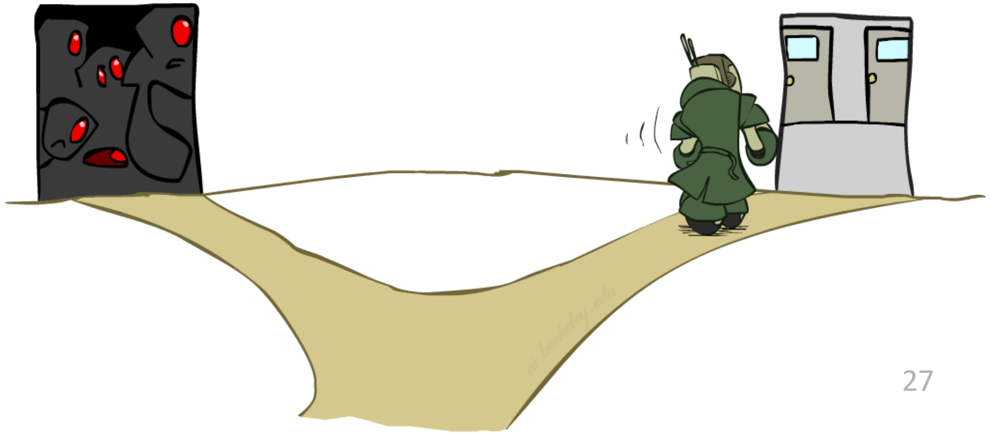
- Even after this rule, multiple variables may still be tied.

# Ordering: Which Value to Choose?

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the *least constraining value*
  - I.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)



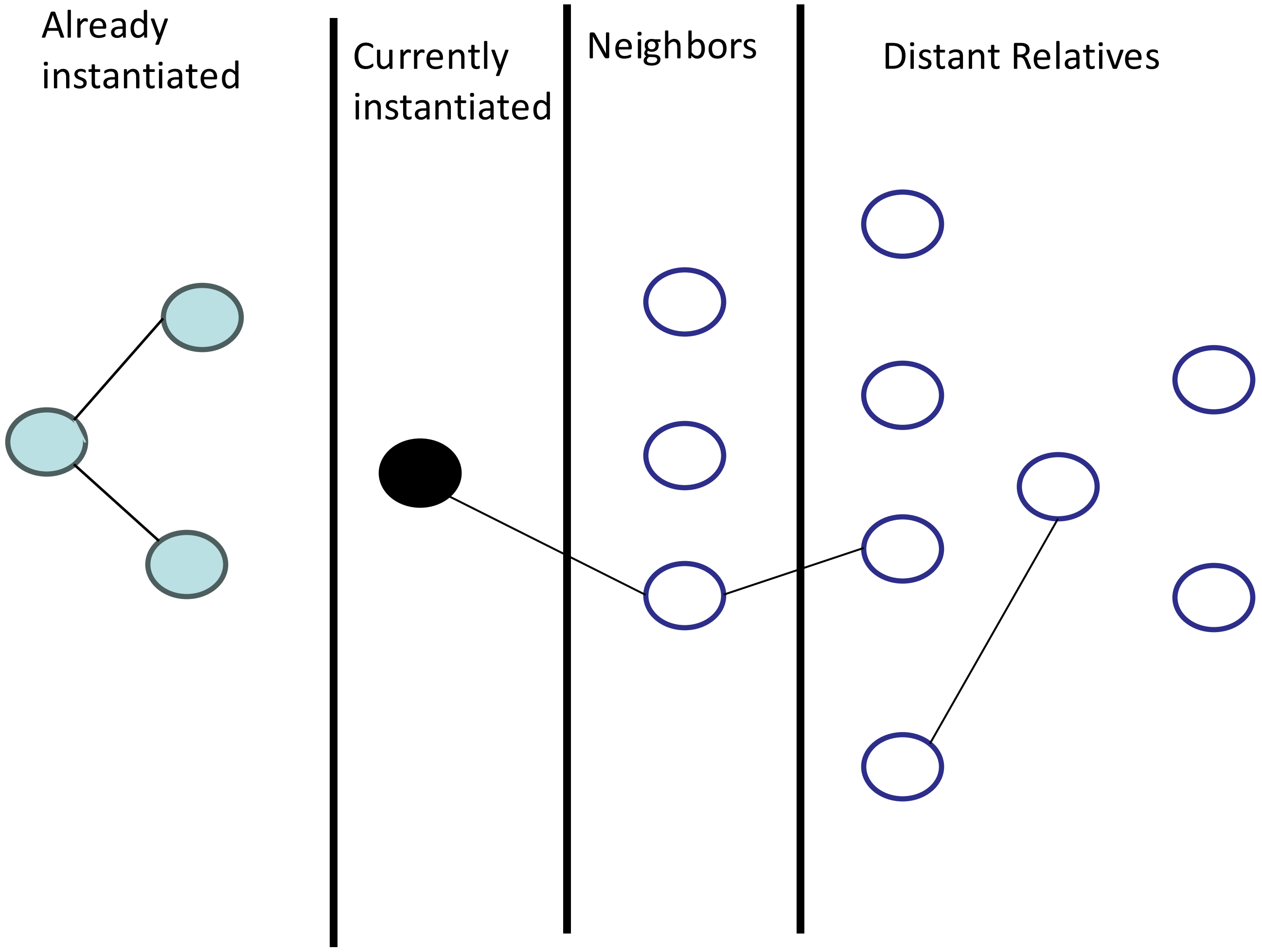
- Why least rather than most?
- Combining these ordering ideas makes 1000 queens feasible



# CSPs: Main Ideas

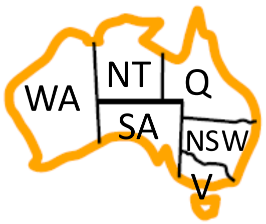
- Representing CSPs
  - Constraint Graphs
- Backtracking Search for CSPs
  - Heuristics for improving this, by
    - Ordering variables
    - Ordering values
  - Backjumping
- Constraint Propagation
  - Forward Checking
  - Arc Consistency (AC3 algorithm)
  - Using structure of constraint graph
- Local Search

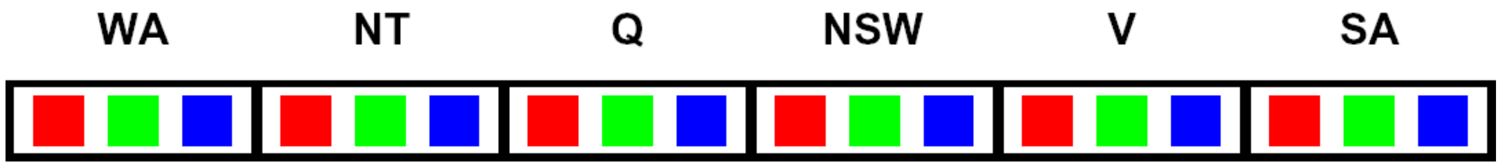
# Comparison of Propagation Techniques



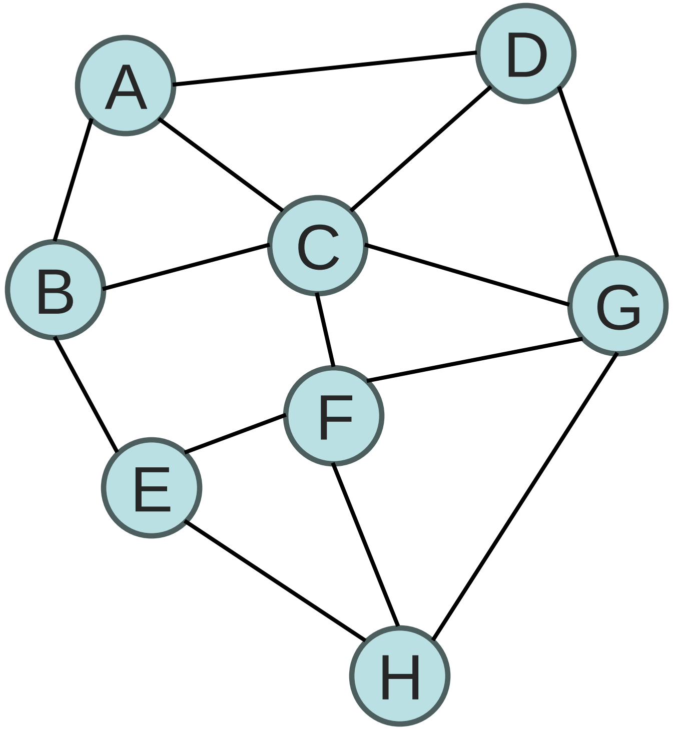
# Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment



- {r, g, b}
- MRV → “C” -> “r”
- Forward Checking
  - “A” → [“g”, ~~“r”,~~ “b”]
  - “B” → [“g”, ~~“r”,~~ “b”]
  - “D” → [“g”, ~~“r”,~~ “b”]
  - “F” → [“g”, ~~“r”,~~ “b”]
  - “G” → [“g”, ~~“r”,~~ “b”]

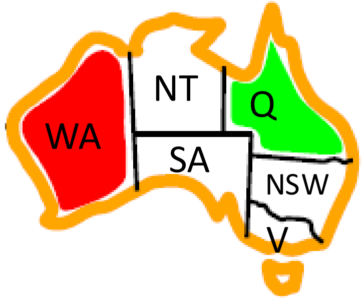


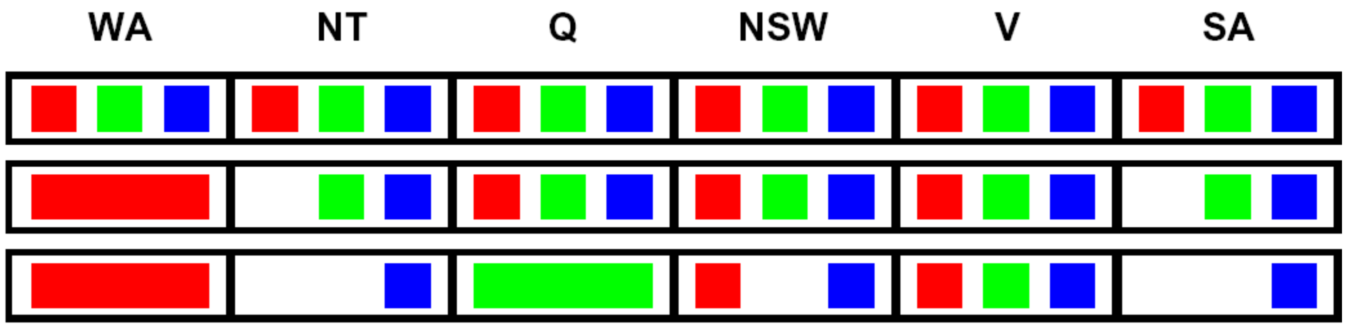
# Practical Tip

- If you implement forward checking, then, there is no need to check that the “new assignment” is valid.
- (That would be doing redundant work.)

# Filtering: Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

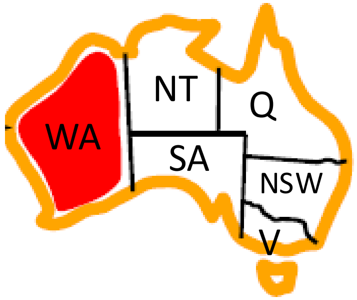


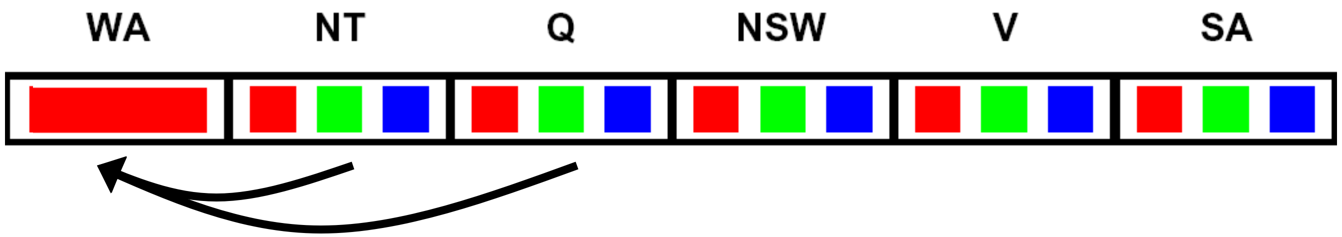



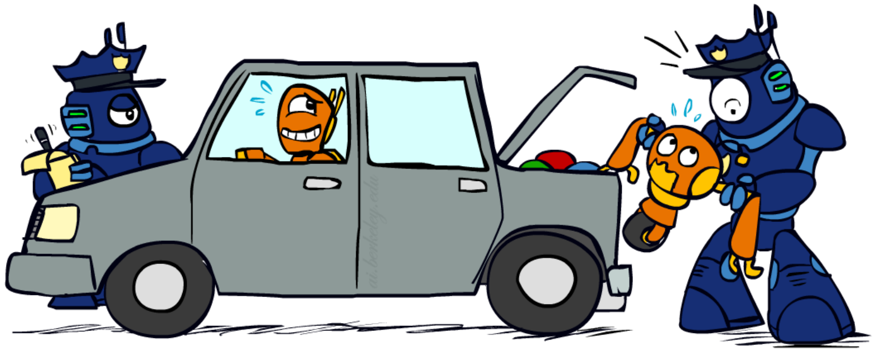

- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

# Consistency of A Single Arc

- An arc $X \rightarrow Y$ is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint



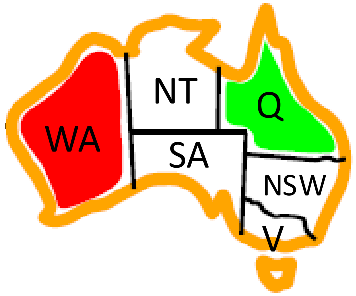



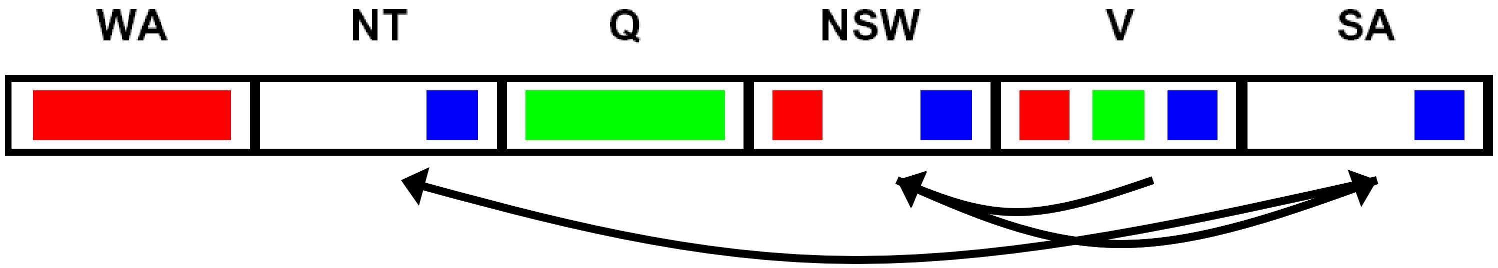
*Delete from the tail!*

- Forward checking: Enforcing consistency of arcs pointing to each new assignment

# Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:




- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- What's the downside of enforcing arc consistency?

*Remember: Delete from the tail!*

# Enforcing Arc Consistency in a CSP

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

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

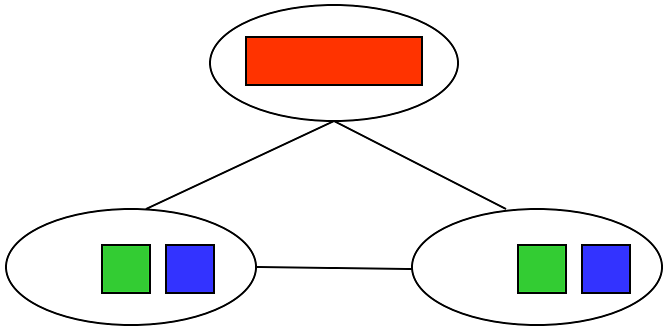
```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

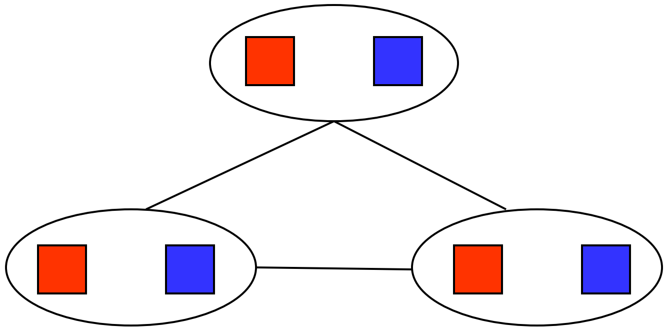
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

# Limitations of 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 still runs inside a backtracking search!





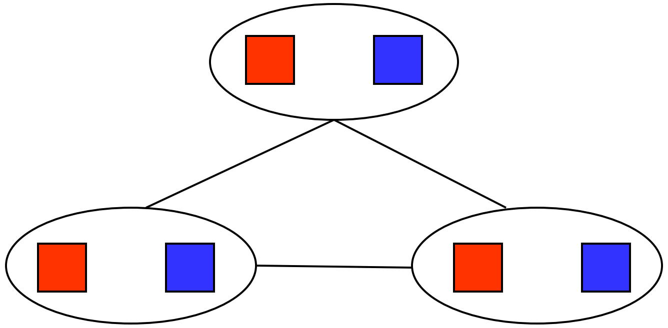
*What went wrong here?*

AC-3, Limitations and K-Consistency

# BEYOND ARC CONSISTENCY

# Limitations of Arc Consistency

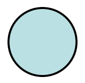
- After enforcing arc consistency:
  - Can have no solutions left (and not know it)



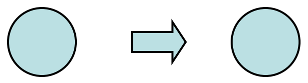
*How do we capture this component?*

# K-Consistency

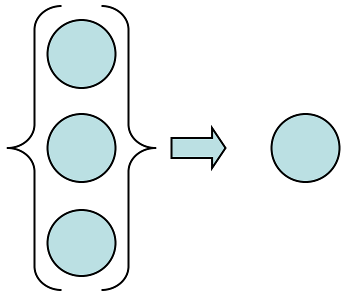
- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints



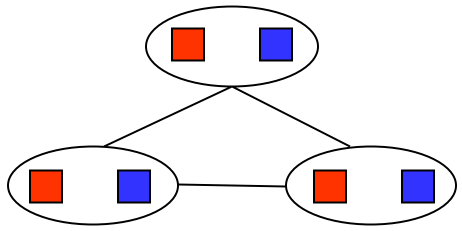
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other



  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the $k^{th}$ node.



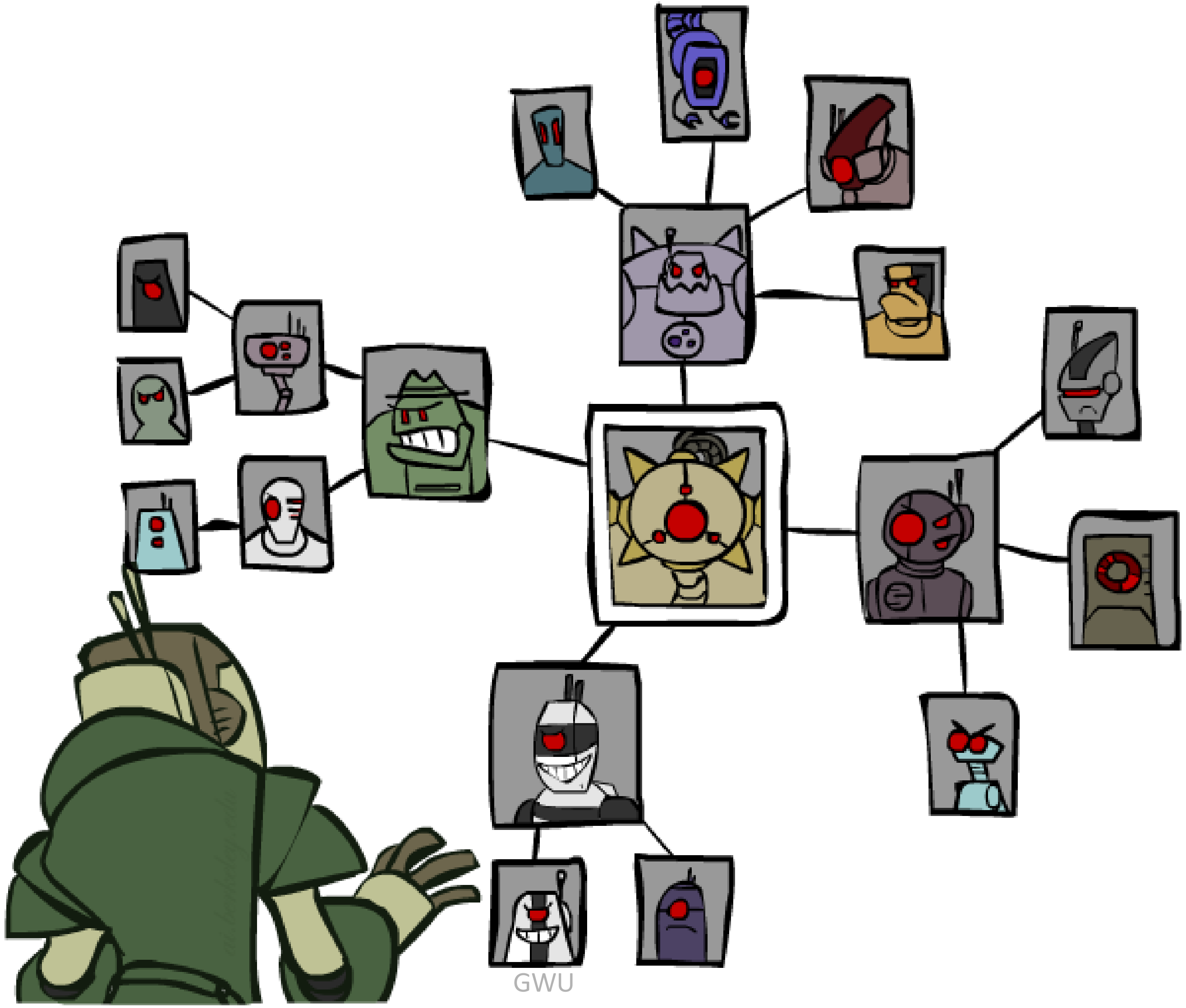
- Higher k more expensive to compute



# Strong K-Consistency

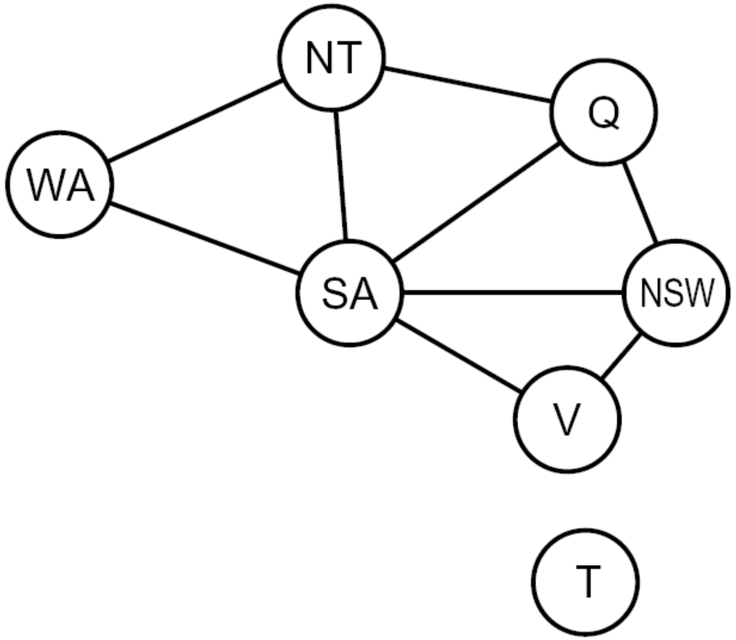
- Strong k-consistency: also k-1, k-2, … 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - …
- Lots of middle ground between arc consistency and n-consistency! (e.g., k=3, called path consistency)

# Structure

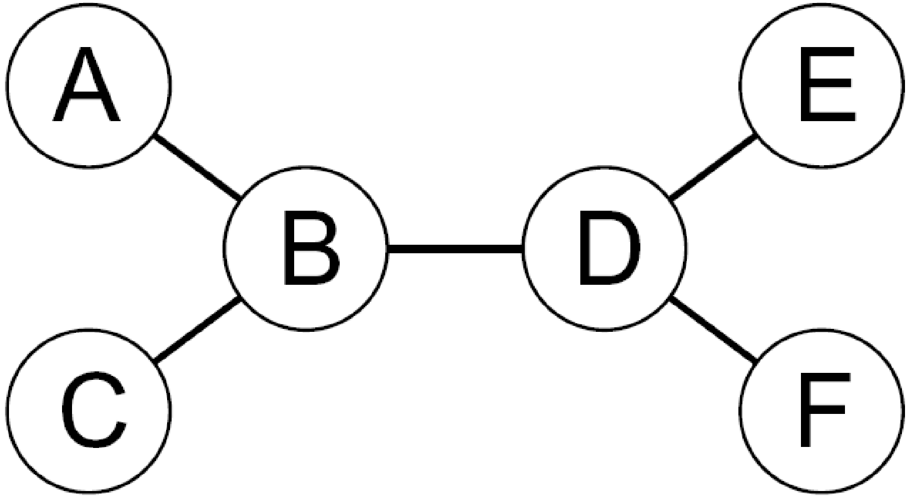


# Problem Structure

- Extreme case: independent subproblems
  - Example: Tasmania and mainland do not interact
- Independent subproblems are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into subproblems of only c variables:
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - E.g., n = 80, d = 2, c =20
  - $2^{80}$ = 4 billion years at 10 million nodes/sec
  - $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec



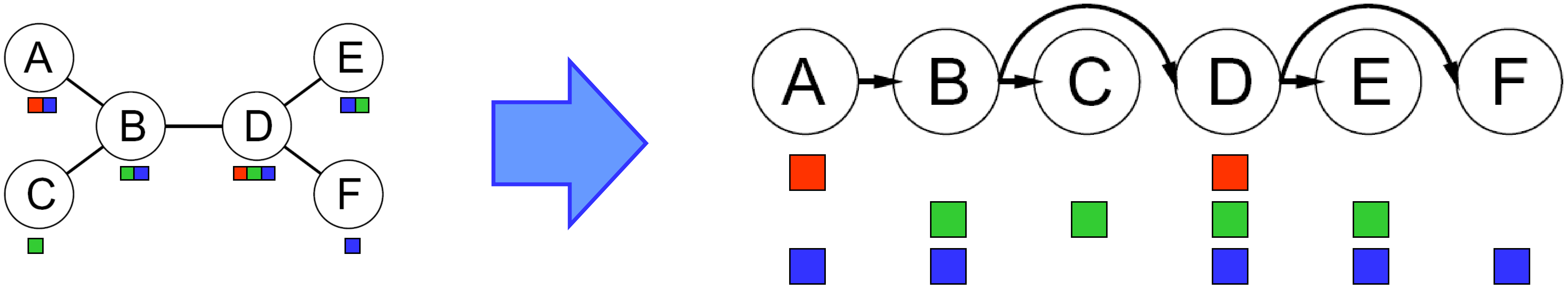
# Tree-Structured CSPs




- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning

# Tree-Structured CSPs

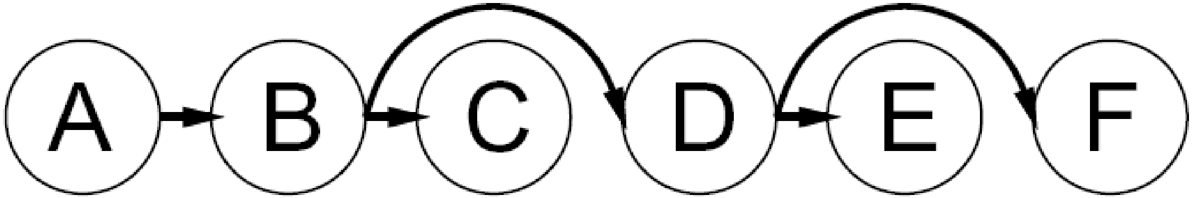
- Algorithm for tree-structured CSPs:
  - Order: Choose a root variable, order variables so that parents precede children




  - Remove backward: For i = n : 2, apply RemoveInconsistent(Parent($X_i$),$X_i$)
  - Assign forward: For i = 1 : n, assign $X_i$ consistently with Parent($X_i$)
- Runtime: O(n $d^2$)  (why?)

# Tree-Structured CSPs

- Claim 1: After backward pass, all root-to-leaf arcs are consistent
- Proof: Each X→Y was made consistent at one point and Y's domain could not have been reduced thereafter (because Y's children were processed before Y)




- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Proof: Induction on position

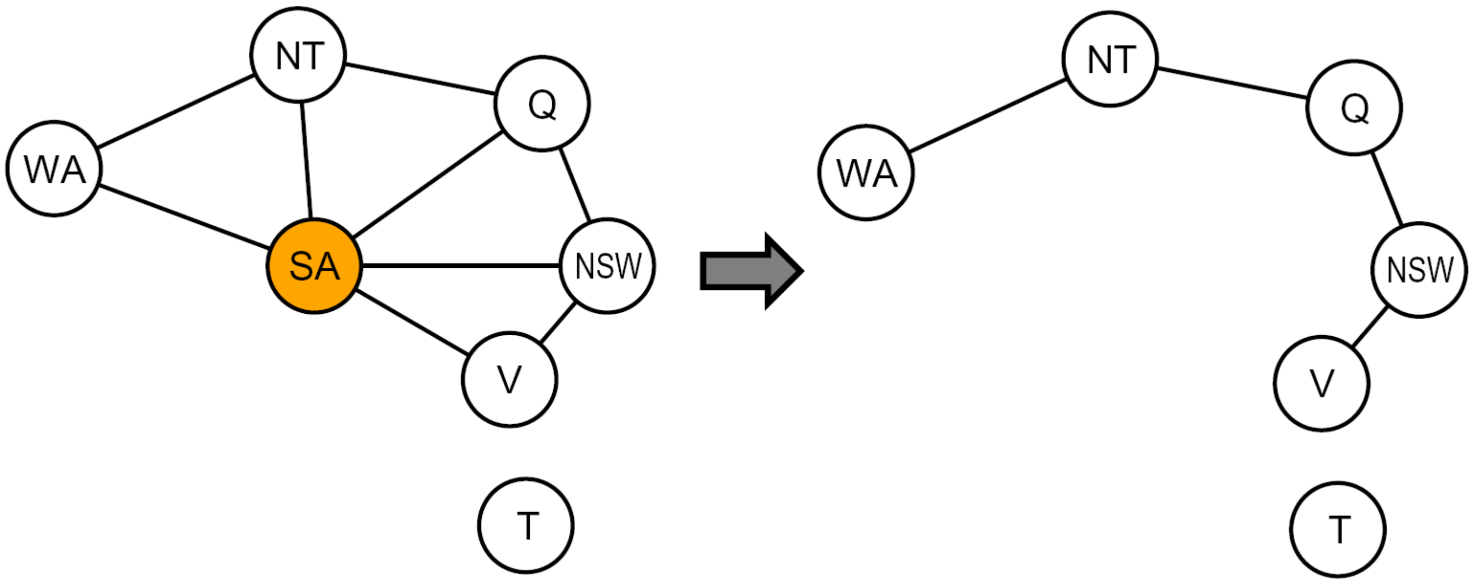
- Why doesn't this algorithm work with cycles in the constraint graph?

- We'll see this basic idea again with Bayes' nets

Making the next leap

# IMPROVING STRUCTURE

# Nearly Tree-Structured CSPs




- 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 $O( (d^c) (n-c) d^2 )$, very fast for small c

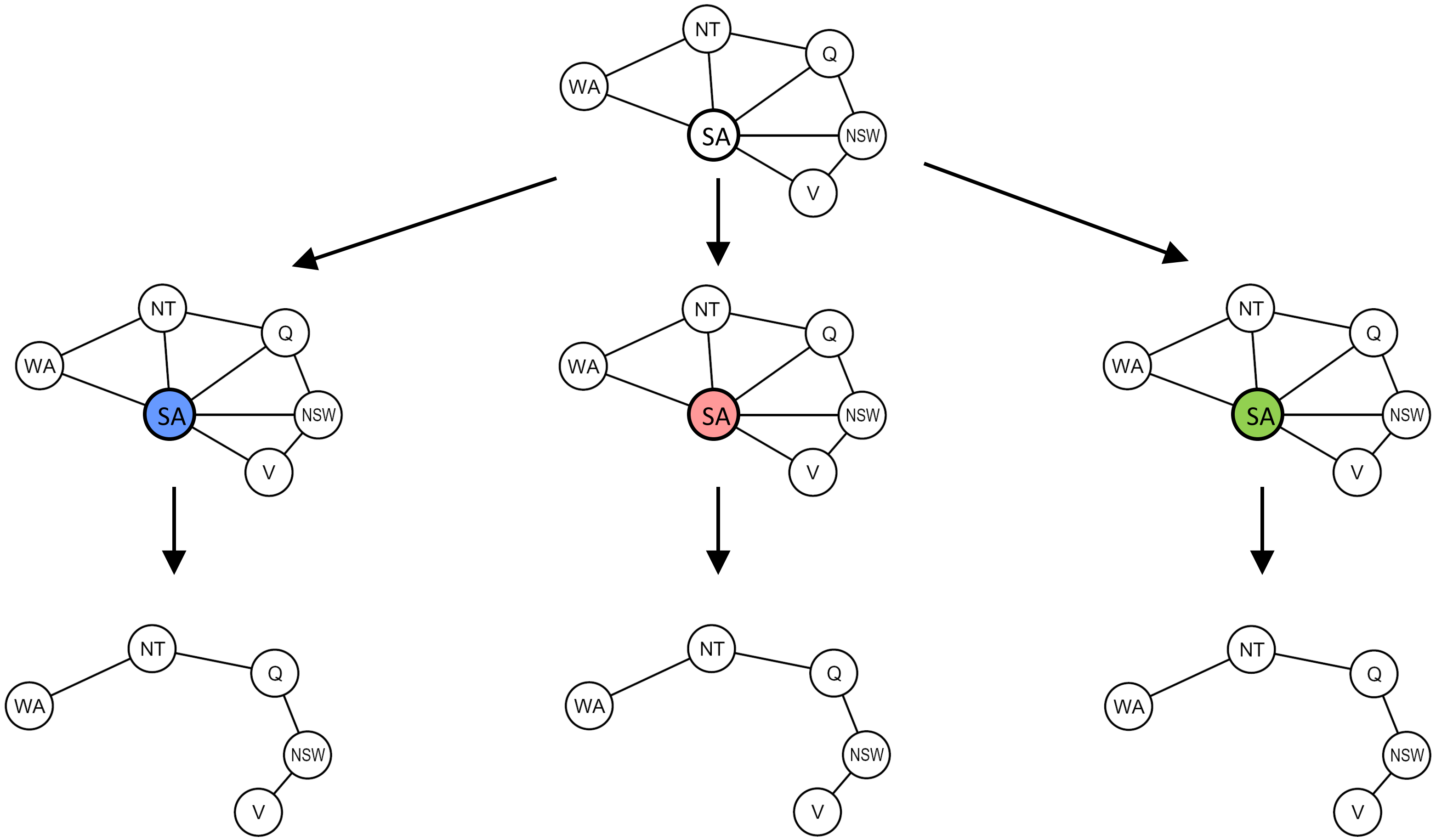
# Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

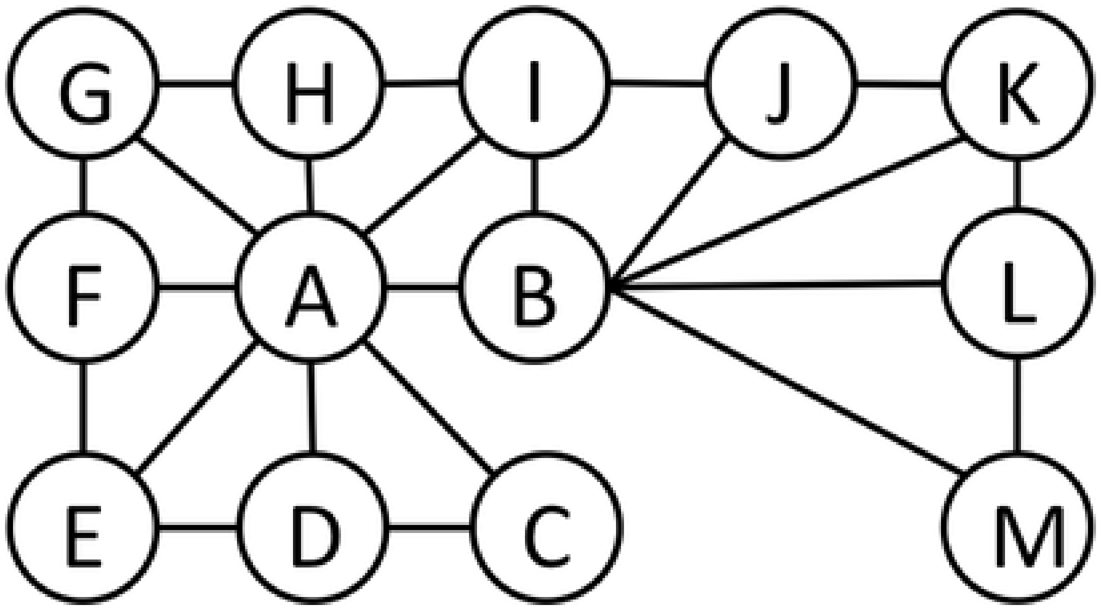
Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)



# How much time?

- 13 variables, 4 colors



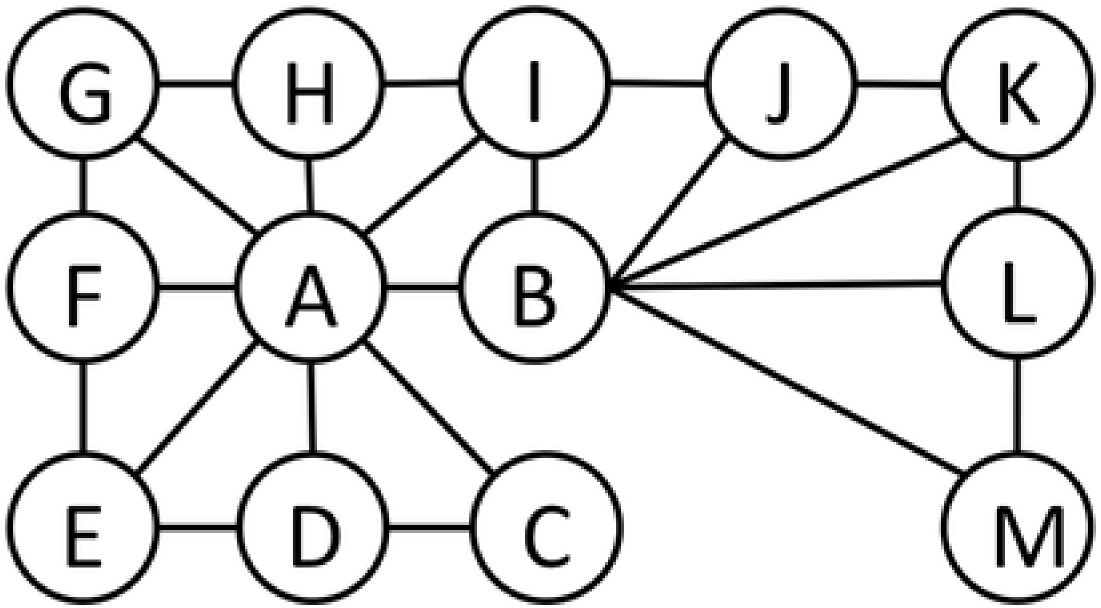

- Bruteforce backtracking: 4^13

- Step 1: Find the cutset. A and B
- Step 2: Assign colors to A and B. 12
  - Solve the tree CSP. n d^2. 11.4^2

Total time: 12 x 11 x 4^2

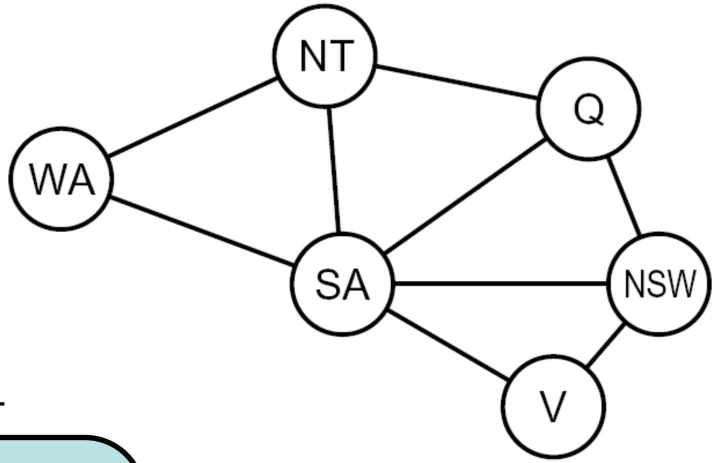
# Cutset Quiz

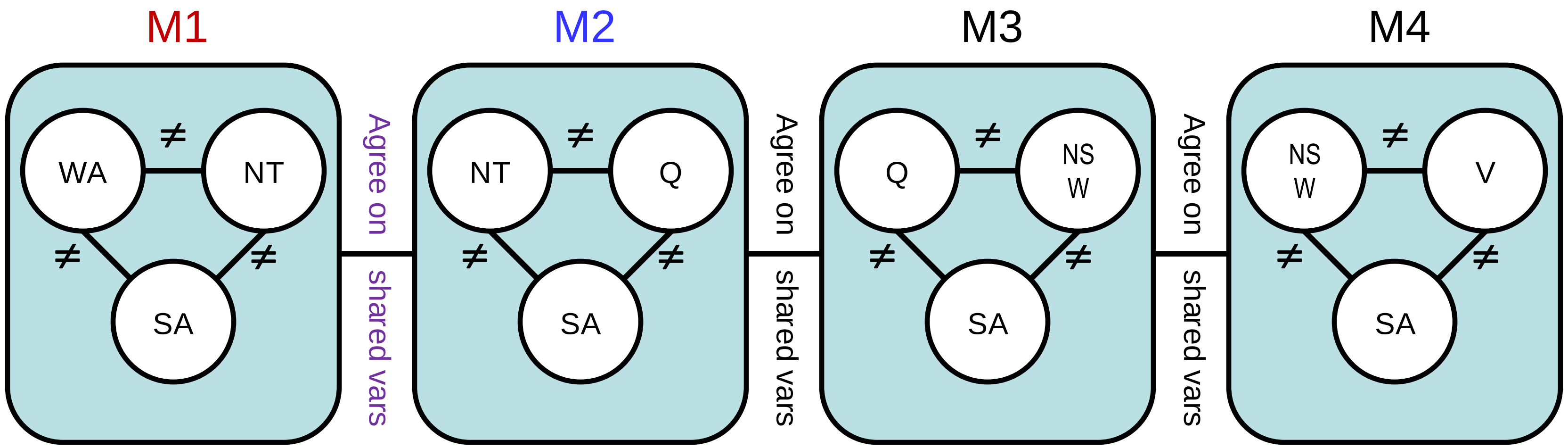
- Find the smallest cutset for the graph below.



# Tree Decomposition*

- Idea: create a tree-structured graph of mega-variables
- Each mega-variable encodes part of the original CSP
- Subproblems overlap to ensure consistent solutions




{(WA=r,SA=g,NT=b),
(WA=b,SA=r,NT=g),
…}

{(NT=r,SA=g,Q=b),
(NT=b,SA=g,Q=r),
…}

Agree: (M1,M2) ∈
{((WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g)), …}

# CSPs: Main Ideas

- Representing CSPs
  - Constraint Graphs
- Backtracking Search for CSPs
  - Heuristics for improving this, by
    - Ordering variables
    - Ordering values
  - Backjumping
- Constraint Propagation
  - Forward Checking
  - Arc Consistency (AC3 algorithm)
  - Using structure of constraint graph
- Local Search

# Course Outline