

# Artificial Intelligence

## CE-417, Group 1

### Computer Eng. Department

### Sharif University of Technology

Spring 2020

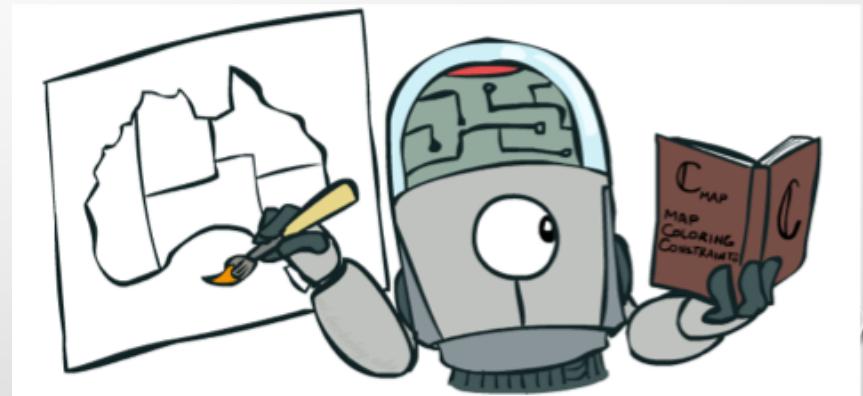
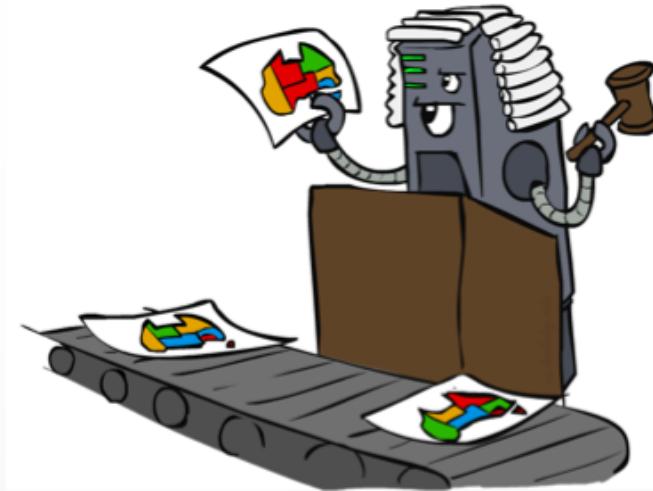
By Mohammad Hossein Rohban, Ph.D.

Courtesy: Most slides are adopted from CSE-573 (Washington U.), original  
slides for the textbook, and CS-188 (UC. Berkeley).

# Constraint Satisfaction Problems

# Constraint satisfaction problems (CSPs)

- Standard search problem:
  - state is a “black box”
  - any old data structure that supports goal test, evaluation, and successor
- CSP:
  - state is defined by **variables**  $X_i$  with values from **domain**  $D_i$
  - goal test is a set of **constraints** specifying allowable combinations of values for subsets of variable
  - Allows useful **general-purpose** algorithms with more power than standard search algorithms

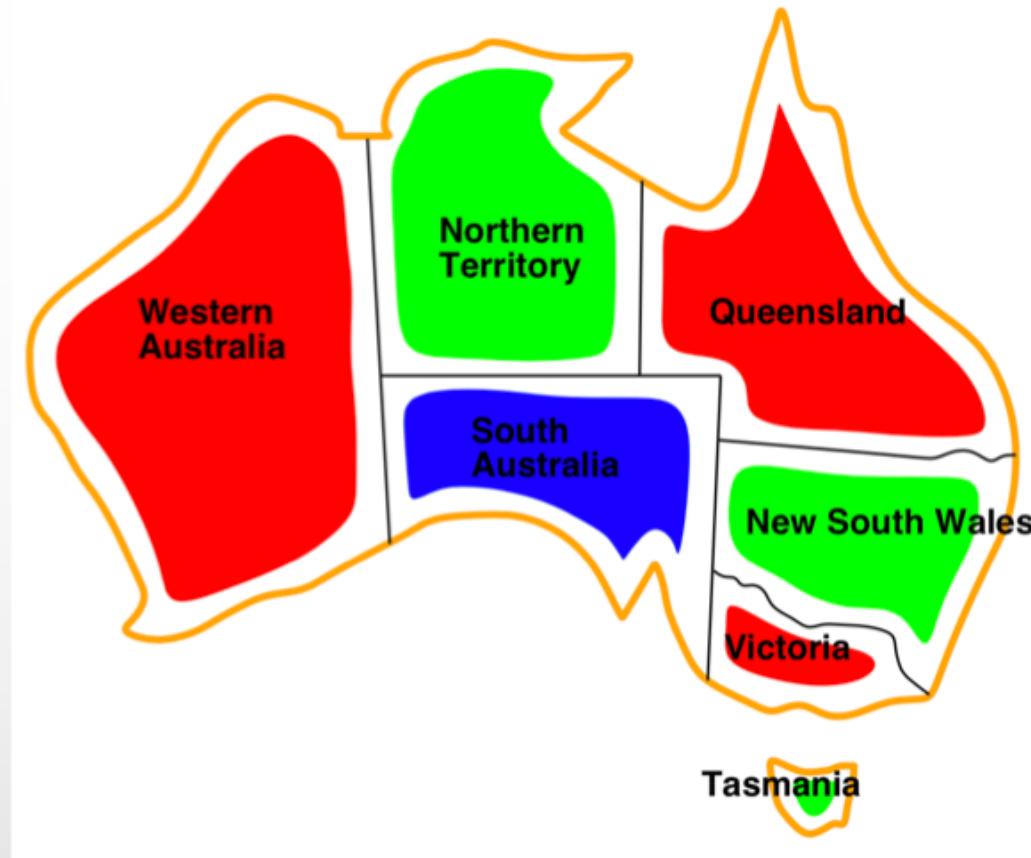


# Example: Map-Coloring



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

## Example: Map-Coloring (cont.)

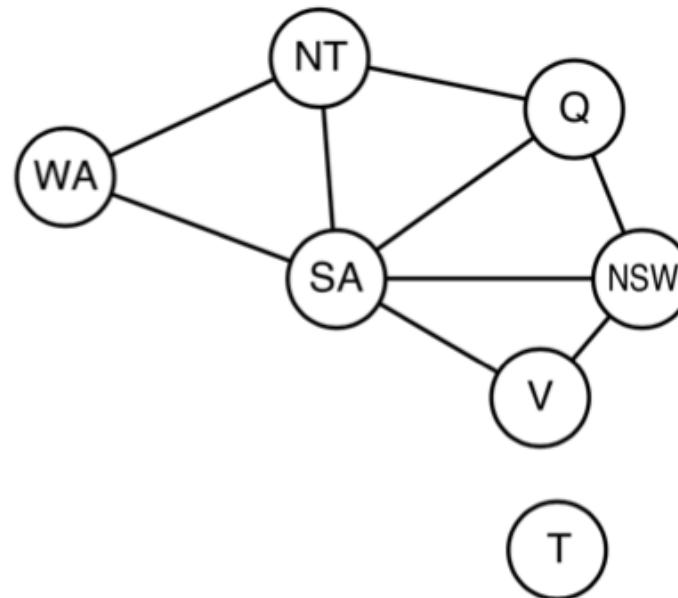


Solutions are assignments satisfying all constraints, e.g.,

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

# Constraint graph

- **Binary CSP:** each constraint relates at most two variables
- **Constraint graph:** nodes are variables, arcs show constraints

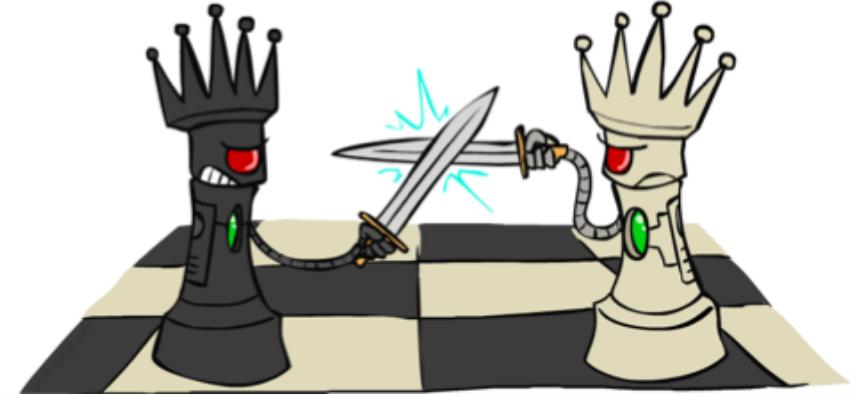
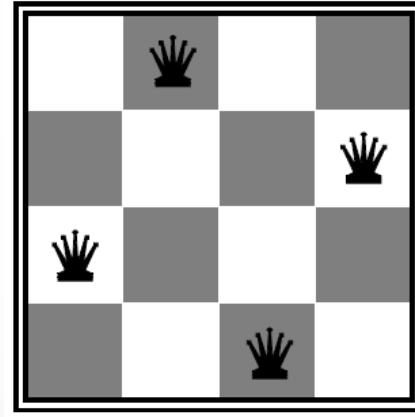


- General-purpose CSP algorithms use the graph structure to speed up search. e.g., Tasmania is an independent subproblem!

# Example: 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: Cryptarithmetic

- Variables:

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

- Domains:

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

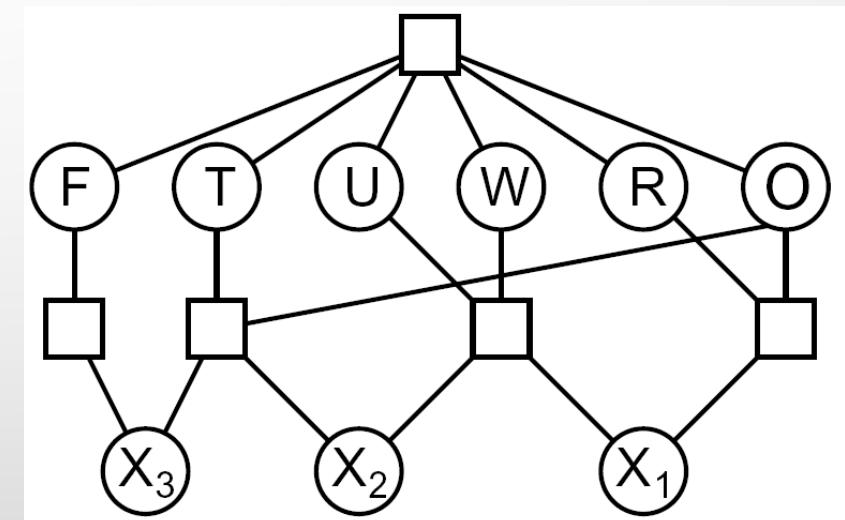
- Constraints:

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

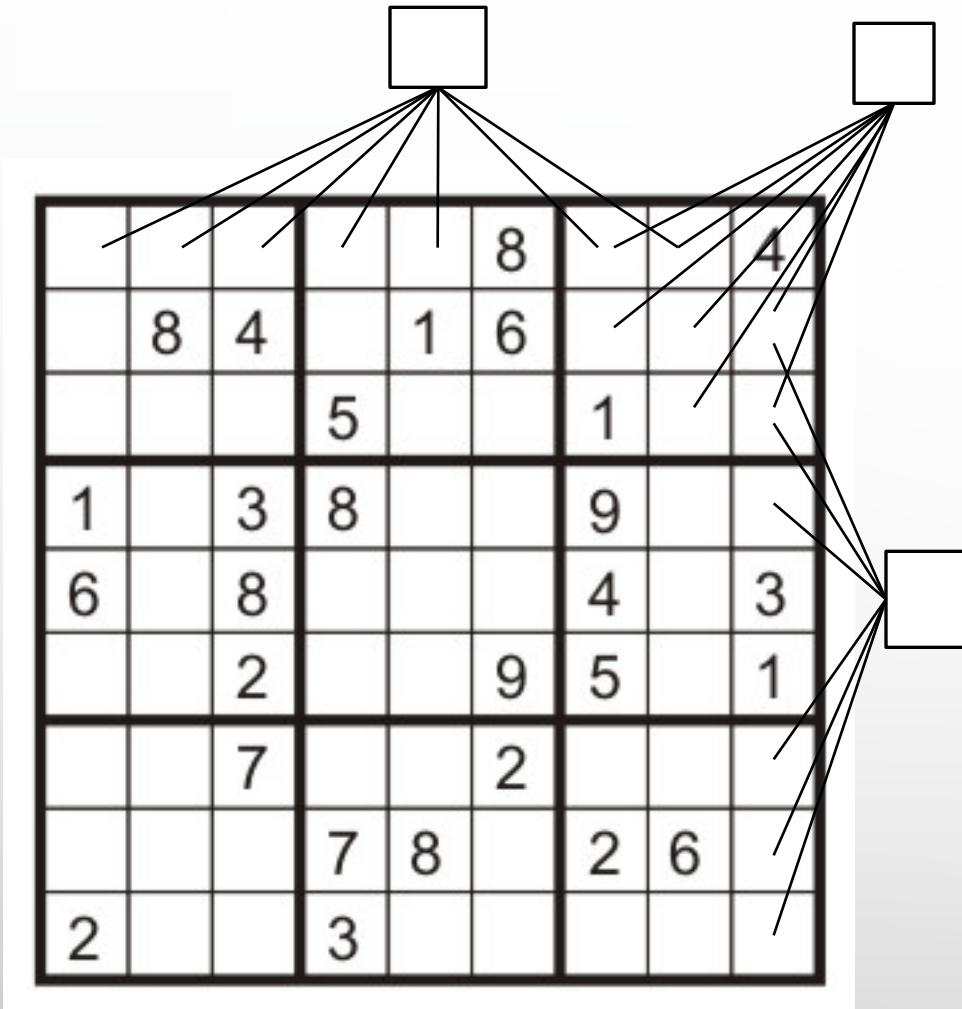
$$O + O = R + 10 \cdot X_1$$

...

$$\begin{array}{r} \text{T} \ \text{W} \ \text{O} \\ + \ \text{T} \ \text{W} \ \text{O} \\ \hline \text{F} \ \text{O} \ \text{U} \ \text{R} \end{array}$$



# Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - $\{1,2,\dots,9\}$
- 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)

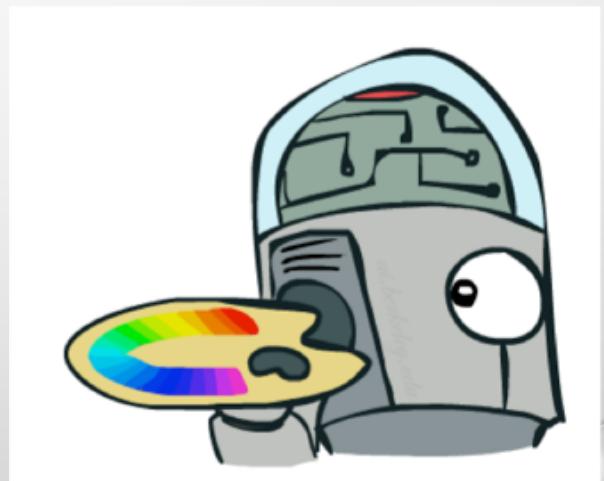
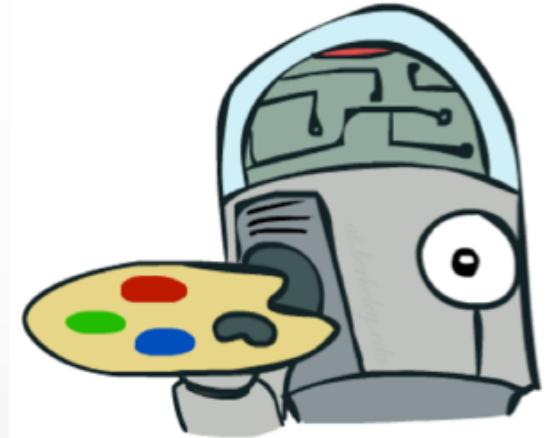
# Varieties of CSPs

- Discrete variables

- finite domains; size  $d \Rightarrow O(d^n)$  complete assignments
  - e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
- infinite domains (integers, strings, etc.)
  - e.g., job scheduling, variables are start/end days for each job
  - need a **constraint language**, e.g.,  $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$
  - linear constraints solvable, nonlinear undecidable

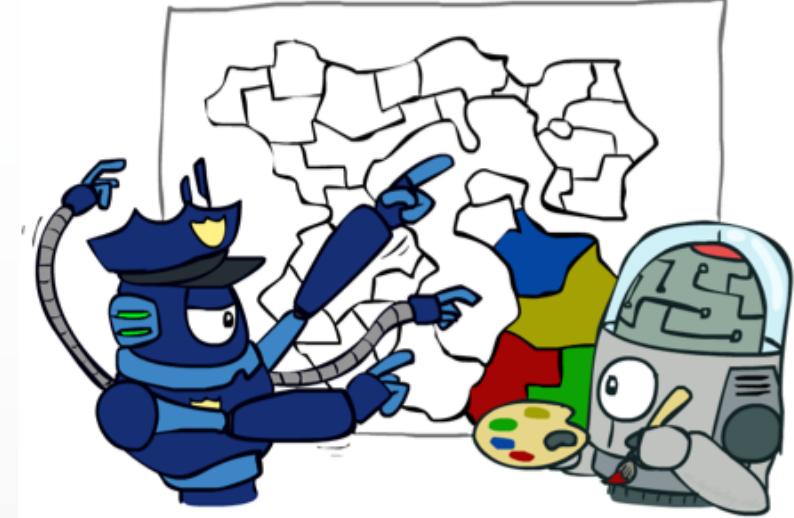
- Continuous variables

- e.g., start/end times for Hubble Telescope observations
- linear constraints solvable in poly time by LP methods



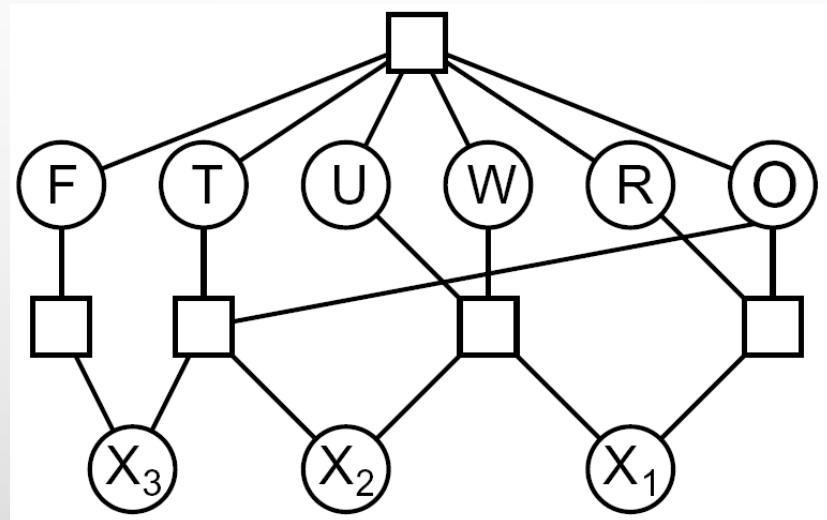
# Varieties of constraints

- **Unary** constraints involve a single variable,
  - e.g., SA  $\neq$  green
- **Binary** constraints involve pairs of variables,
  - e.g., SA  $\neq$  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  
→ constrained optimization problems



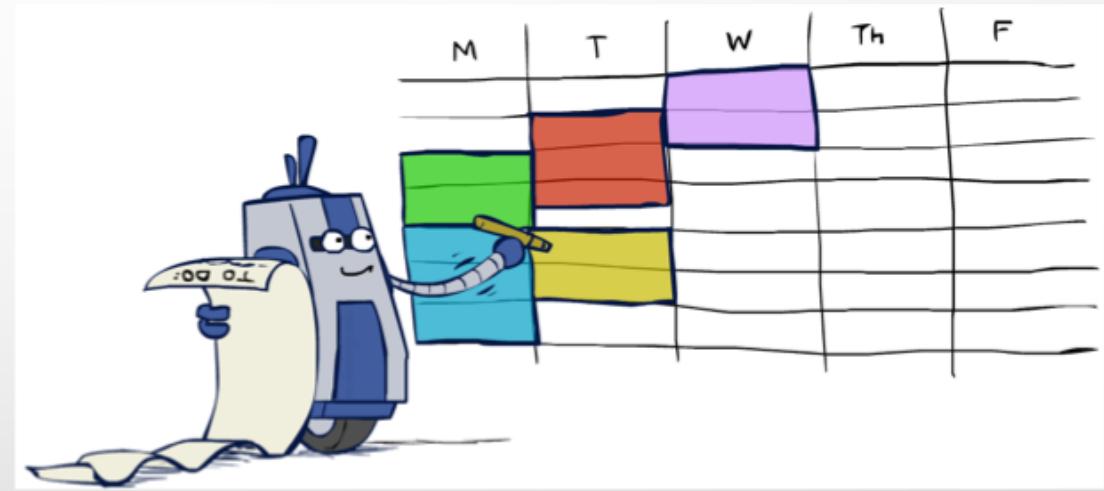
# Converting n-ary CSP to a binary CSP

- Is this possible?



# Real-World CSPs

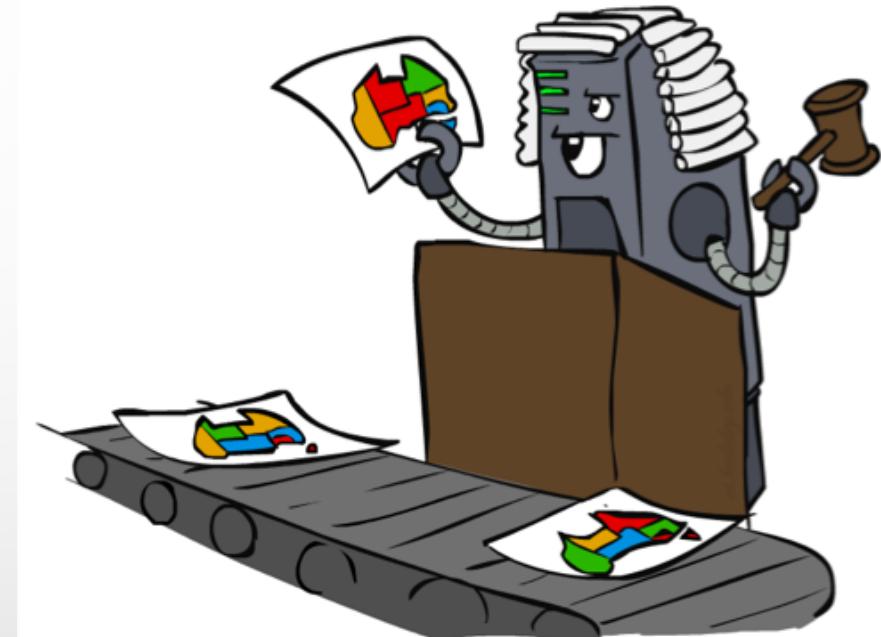
- Assignment problems: e.g., Who teaches what class
- Timetabling problems: e.g., Which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- ... Lots more!



- Many real-world problems involve real-valued variables...

# Standard search formulation (incremental)

- Let's start with the straightforward, dumb approach, then fix it.
- States are defined by the values assigned so far
  - **Initial state:** the empty assignment, { }
  - **Successor function:** assign a value to an unassigned variable that does not conflict with current assignment.  
⇒ fail if no legal assignments (not fixable!)
  - **Goal test:** the current assignment is complete

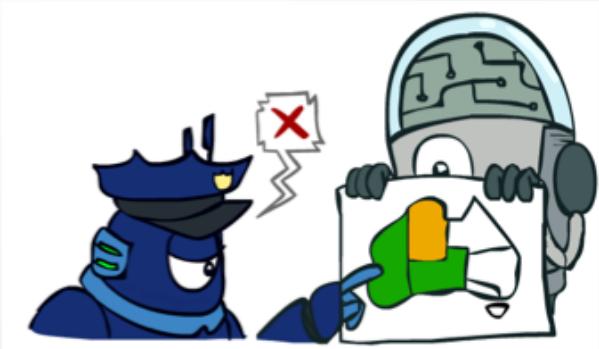


## Standard search formulation (incremental) (cont.)

- This is the same for all CSPs!
- Every solution appears at depth  $n$  with  $n$  variables  $\Rightarrow$  use depth-first search
- Path is irrelevant, so can also use complete-state formulation.
- $b = (n - l)d$  at depth  $l$ , hence  $n! d^n$  leaves!!!

# Backtracking search

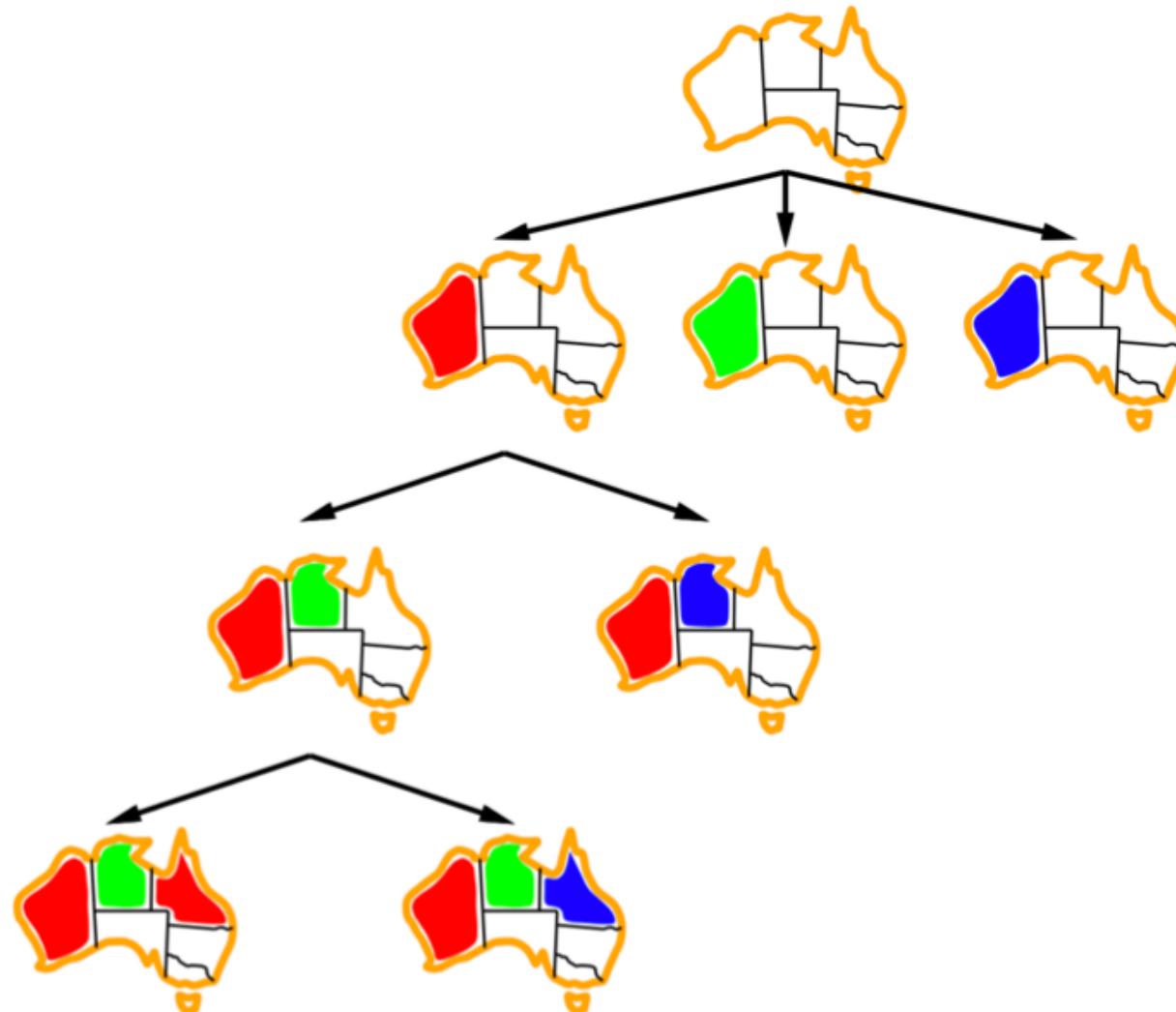
- Variable assignments are commutative, i.e.,  
[WA=red then NT =green] same as [NT =green then WA=red]
- Only need to consider assignments to a single variable at each node  
 $\Rightarrow b=d$  and there are  $d^n$  leaves
- Depth-first search for CSPs with single-variable assignments is called **backtracking** search.
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for  $n \approx 25$



# Backtracking search

```
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  $\leftarrow$  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  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
            if result  $\neq$  failure then return result
            remove {var = value} from assignment
    return failure
```

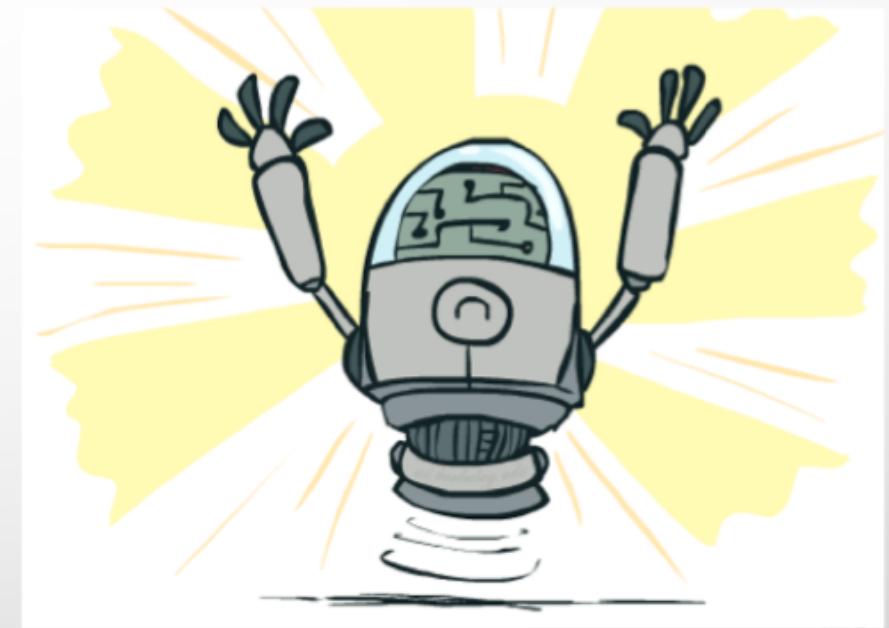
# Backtracking search (cont.)



# Improving backtracking efficiency

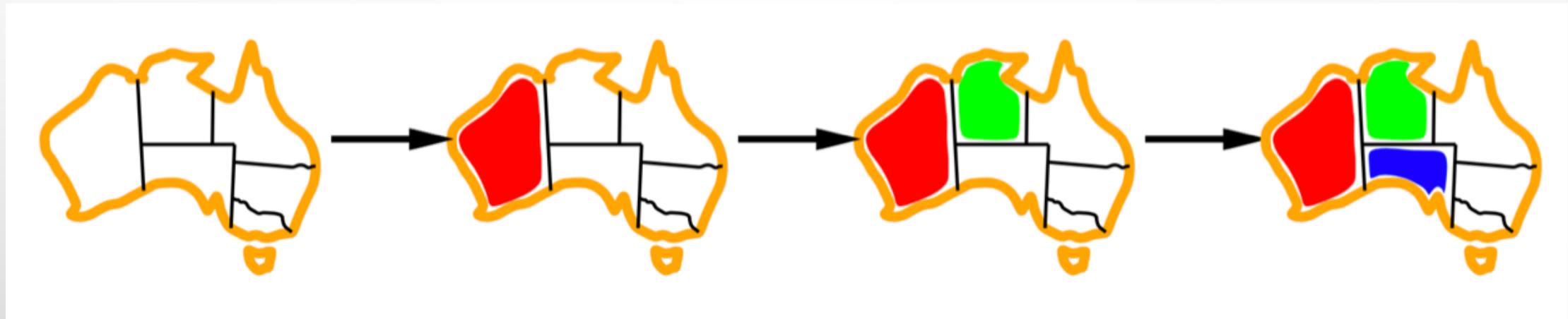
**General-purpose** methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?



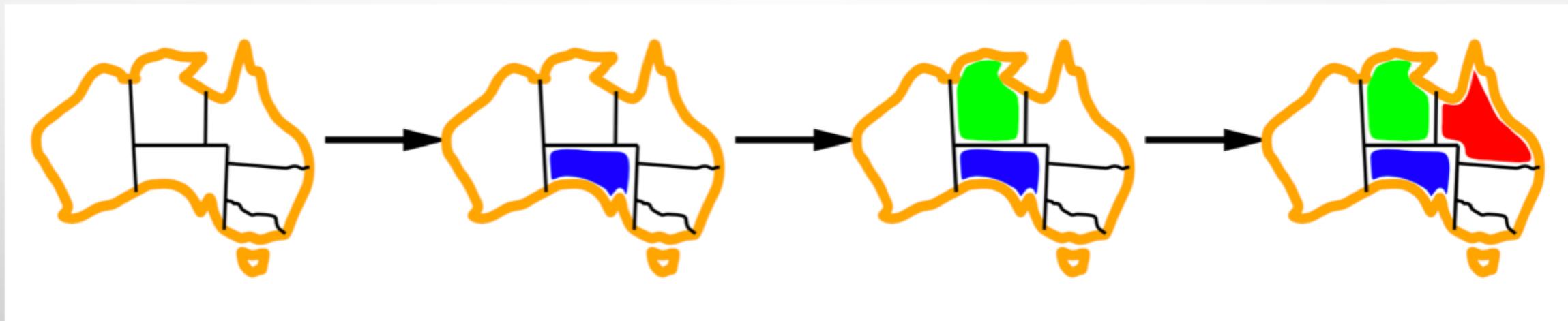
# Minimum remaining values

- Minimum remaining values (MRV):  
choose the variable with the fewest legal values



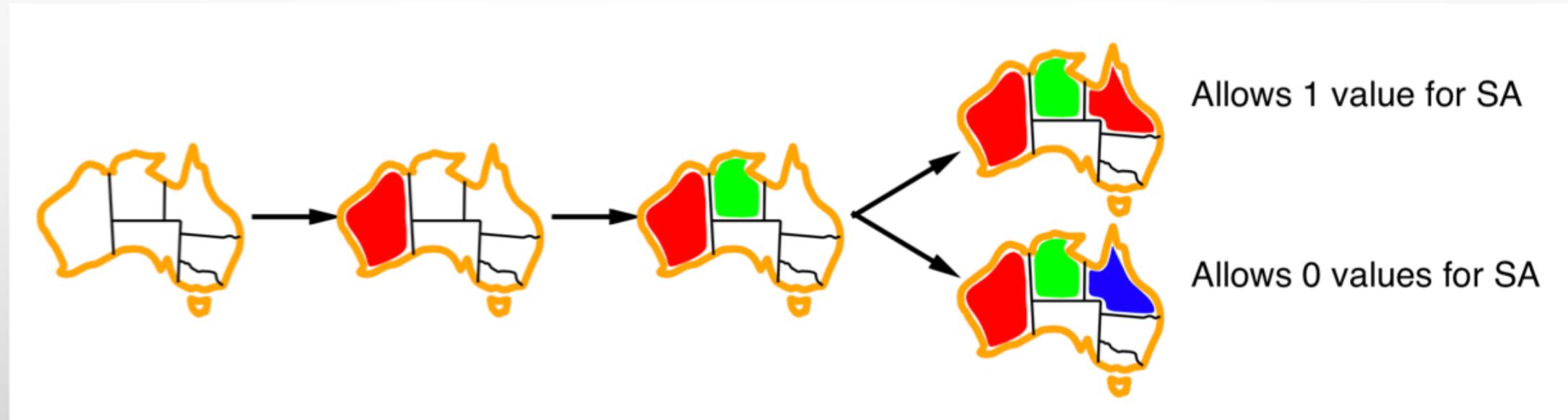
# Degree heuristic

- Tie-breaker among MRV variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables



# Least constraining value

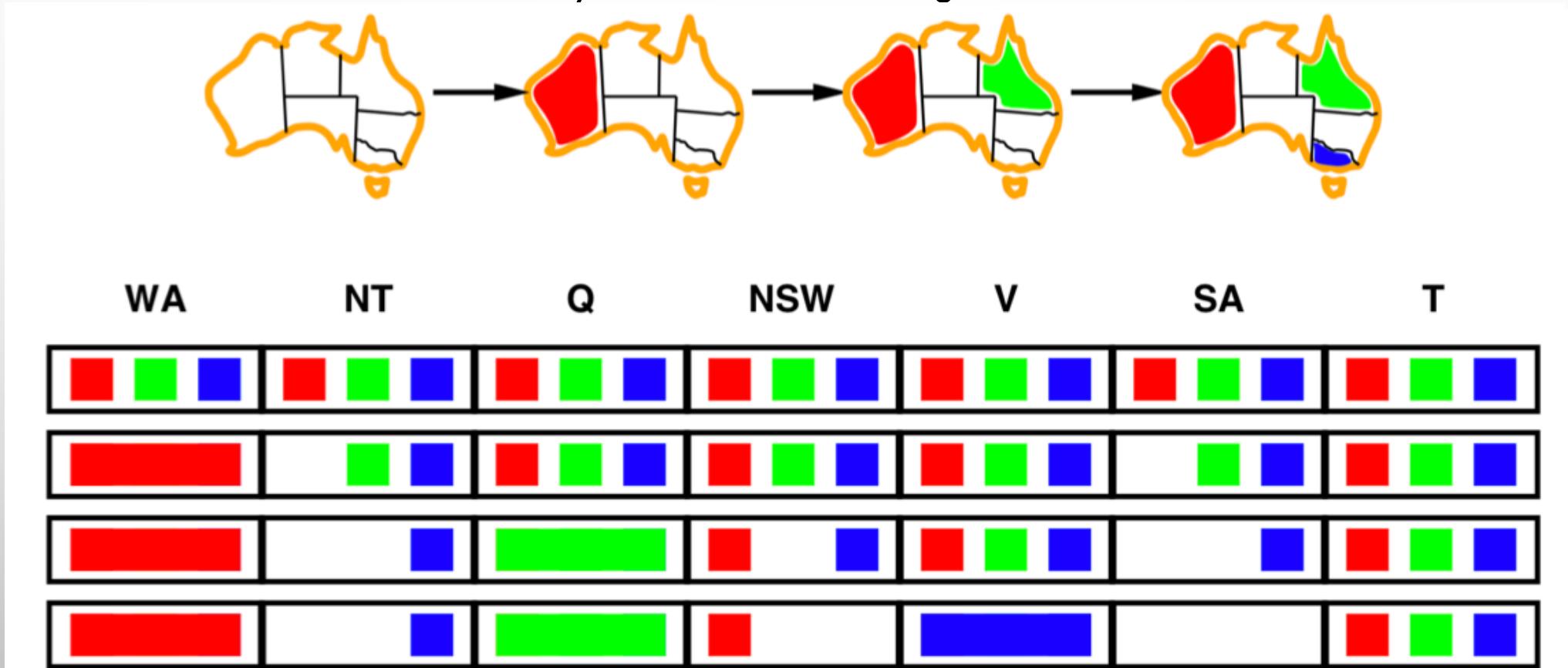
- Given a variable, choose the least constraining value:  
the one that rules out the fewest values in the remaining variables



- Combining these heuristics makes 1000 queens feasible

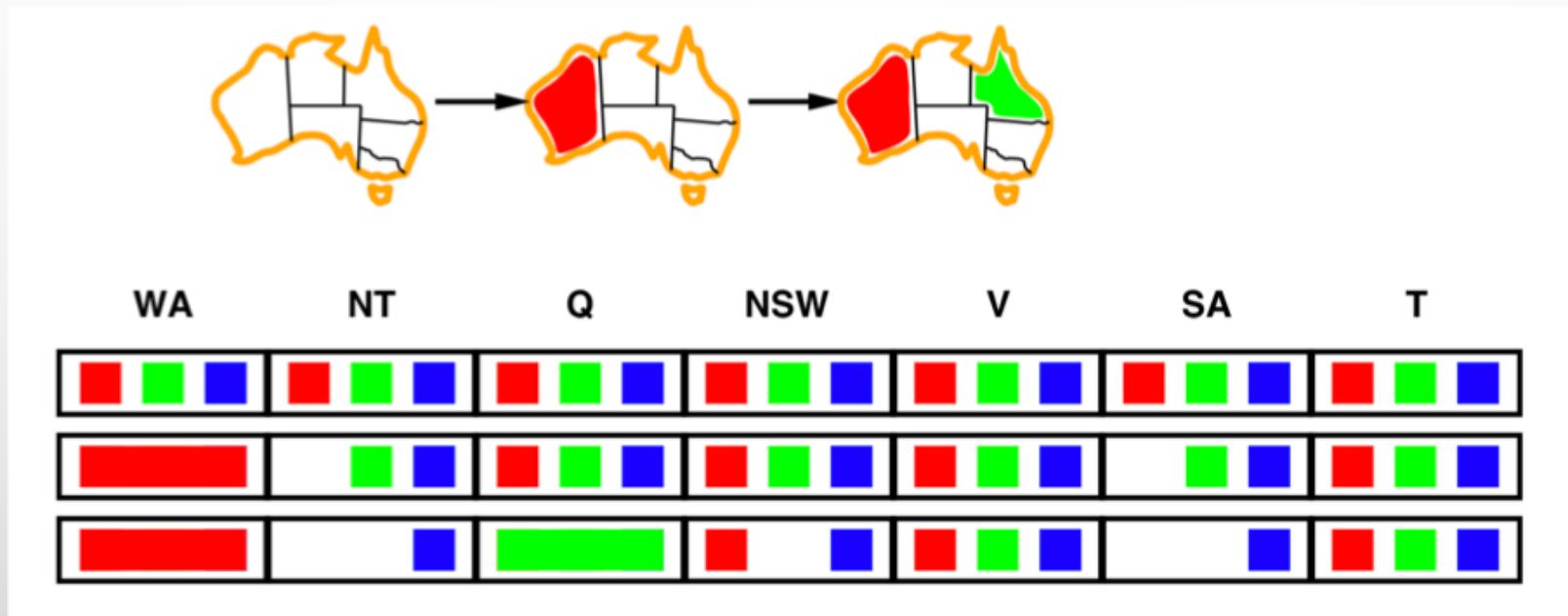
# Forward checking

- Idea: Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



# Constraint propagation

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

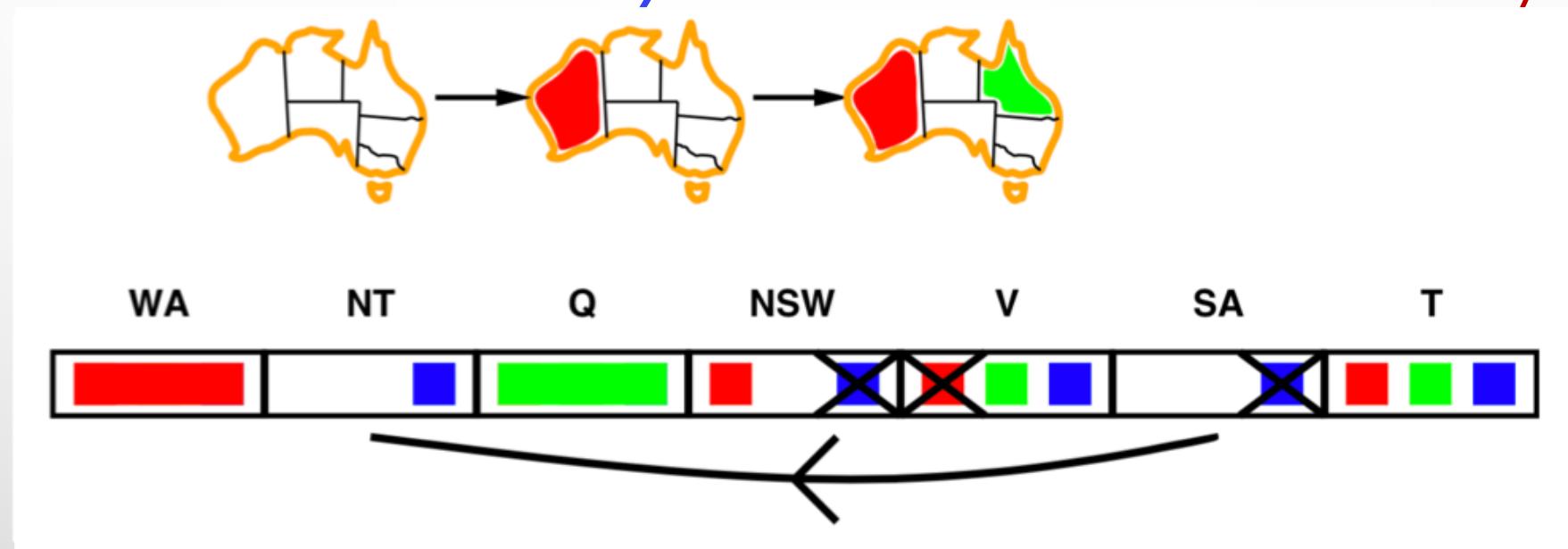


- NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

# Arc consistency

- Simplest form of propagation makes each **arc** consistent
- $X \rightarrow Y$  is consistent iff. for **every** value  $x$  of  $X$  there is **some** allowed  $y$



- 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

# Arc consistency algorithm

**function** AC-3(*csp*) **returns** the CSP, possibly with reduced domains

**inputs:** *csp*, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$

**local variables:** *queue*, a queue of arcs, initially all the arcs in *csp*

**while** *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

**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*  $\leftarrow$  false

**for each**  $x$  **in** DOMAIN[ $X_i$ ] **do**

**if** no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$

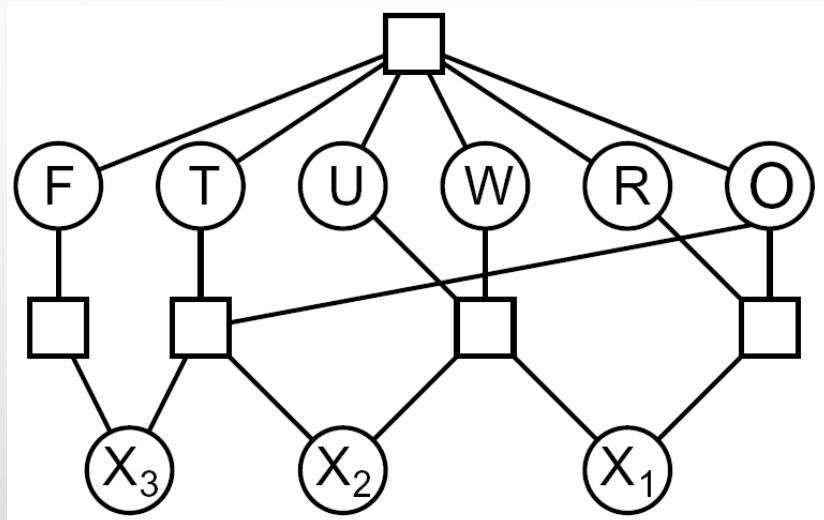
**then** delete  $x$  from DOMAIN[ $X_i$ ]; *removed*  $\leftarrow$  true

**return** *removed*

- $O(n^2d^3)$ , can be reduced to  $O(n^2d^2)$

# Arc consistency for n-ary CSP?

- How to generalize to the n-ary CSP case?

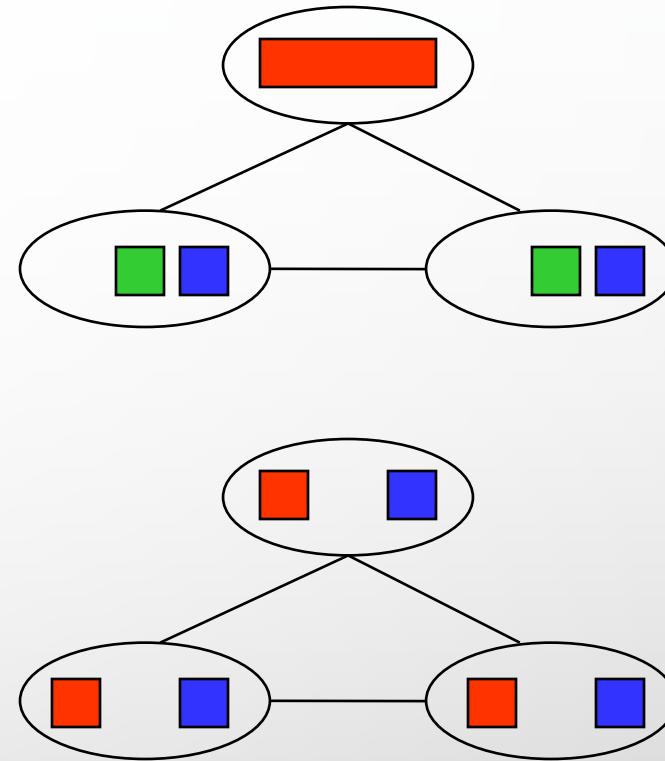


# Bound propagation

- What if the domain for each variable is a large set (i.e. **d** is too large)?
- Lets' say  $F_1$  is an integer in  $[0, 165]$  and  $F_2$  is similarly in  $[0, 385]$ .
- Lets' assume  $F_1 + F_2 = 420$ .
- To make it consistent, the domains should be changed to  $[35, 165]$  and  $[255, 385]$ .
- Def. : A CSP is **bound consistent** if for **every** constraint involving X and Y, and for both the **lower-bound** and **upper-bound** of values of X, there exist some value of Y that satisfies the constraint between X and Y.
- This kind of bounds propagation is widely used in practical constraint problems.

# 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!

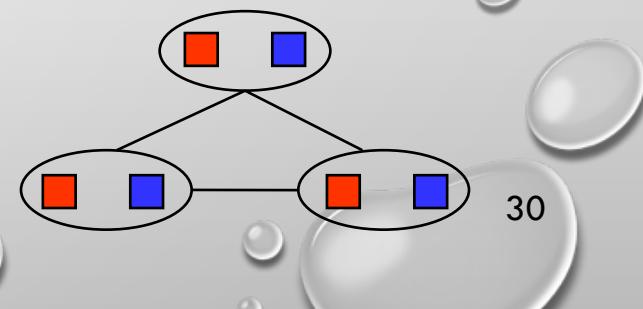
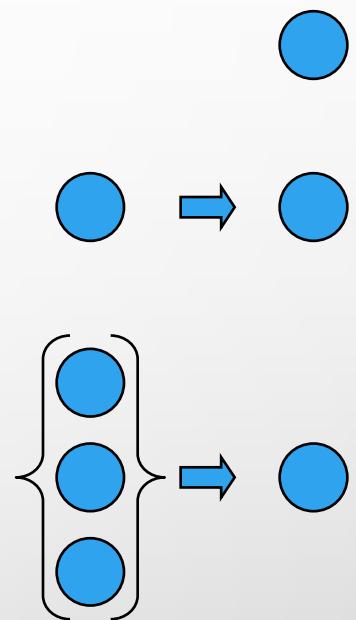


*What went wrong here?*

# k-Consistency

- Increasing degrees of consistency

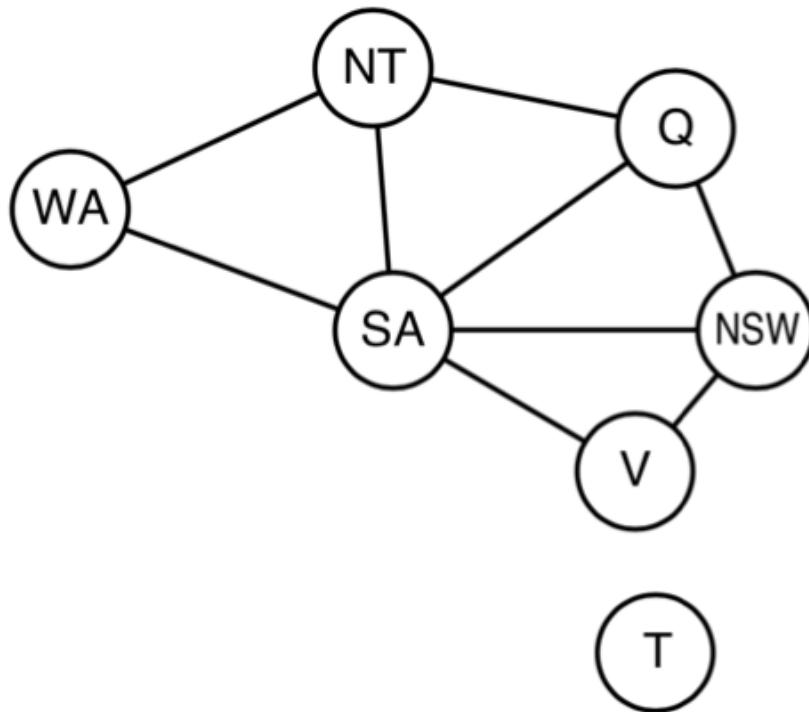
- 1-consistency (node consistency): each single node's domain has a value which meets that node's unary constraints
- 2-consistency (arc consistency): for each pair of nodes, any consistent assignment to one can be extended to the other
- k-consistency: for each k nodes, any consistent assignment to k-1 can be extended to the k<sup>th</sup> node.
- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)



# 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)

# Problem structure

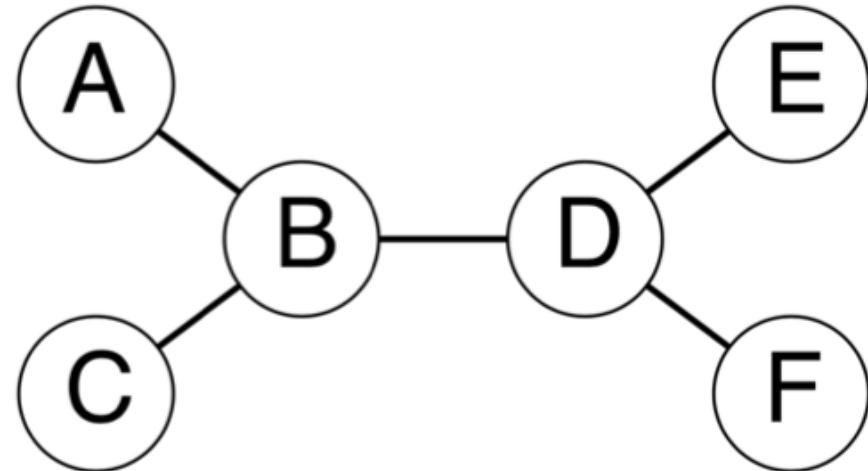


- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph

## Problem structure (cont.)

- Suppose each subproblem has  $c$  variables out of  $n$  total
- Worst-case solution cost is  $n/c \cdot d^c$ , linear in  $n$
- E.g.,  $n=80$ ,  $d=2$ ,  $c=20$ 
  - $2^{80} = 4$  billion years at 10 million nodes/sec
  - $4 \cdot 2^{20} = 0.4$  seconds at 10 million nodes/sec

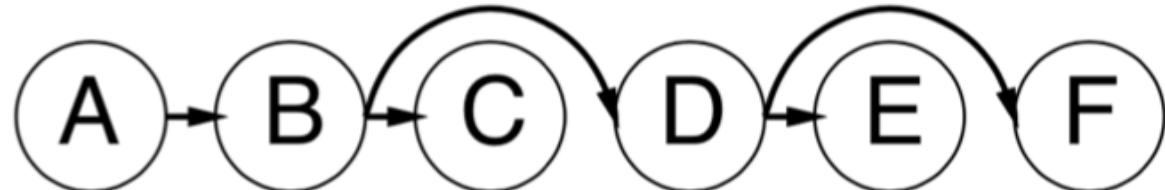
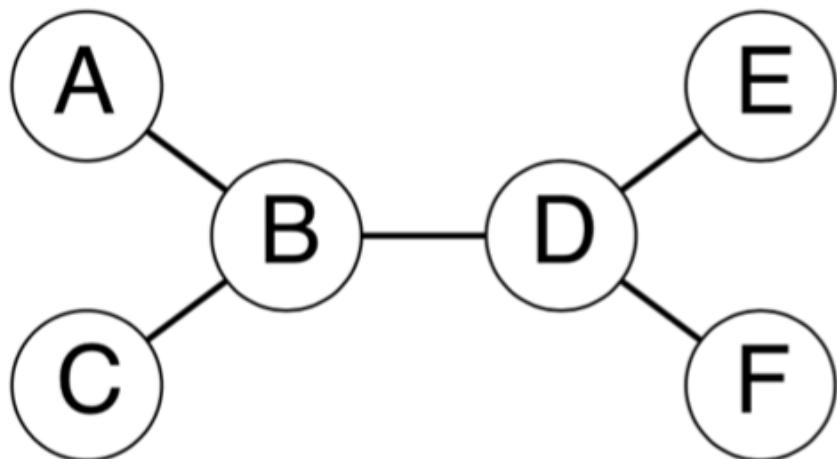
# Tree-structured CSPs



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

# Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering

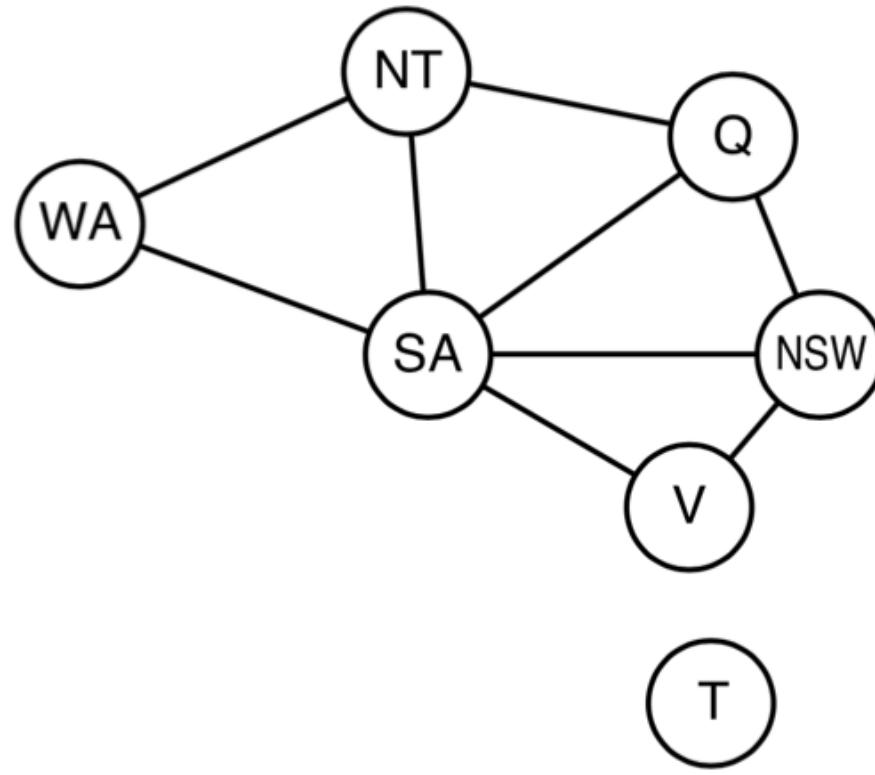


2. For  $j$  from  $n$  down to 2, apply RemoveInconsistent( $\text{Parent}(X_j)$ ,  $X_j$ )
3. For  $j$  from 1 to  $n$ , assign  $X_j$  consistently with  $\text{Parent}(X_j)$ .

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

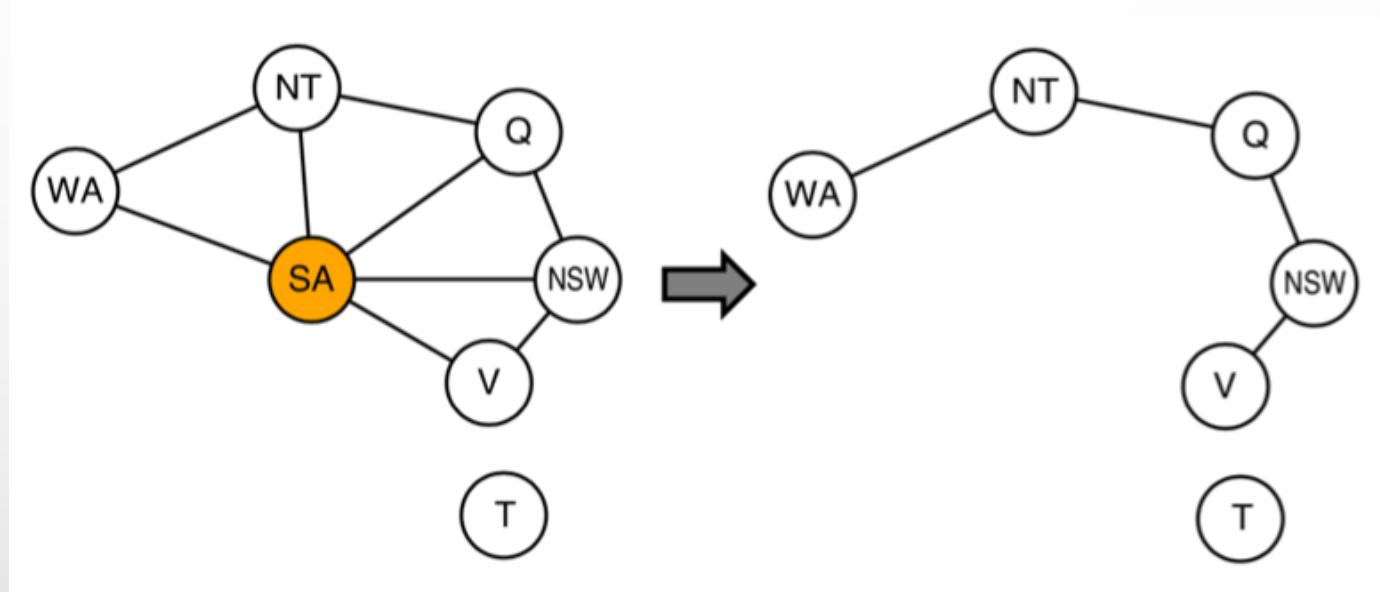
# Nearly tree-structured CSPs

- How to solve the CSP corresponding to this constraint graph using tree structured CSP?



# Nearly tree-structured CSPs (cont.)

- **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  $c \Rightarrow$  runtime  $O(d^c \cdot (n - c)d^2)$ , very fast for small  $c$ .

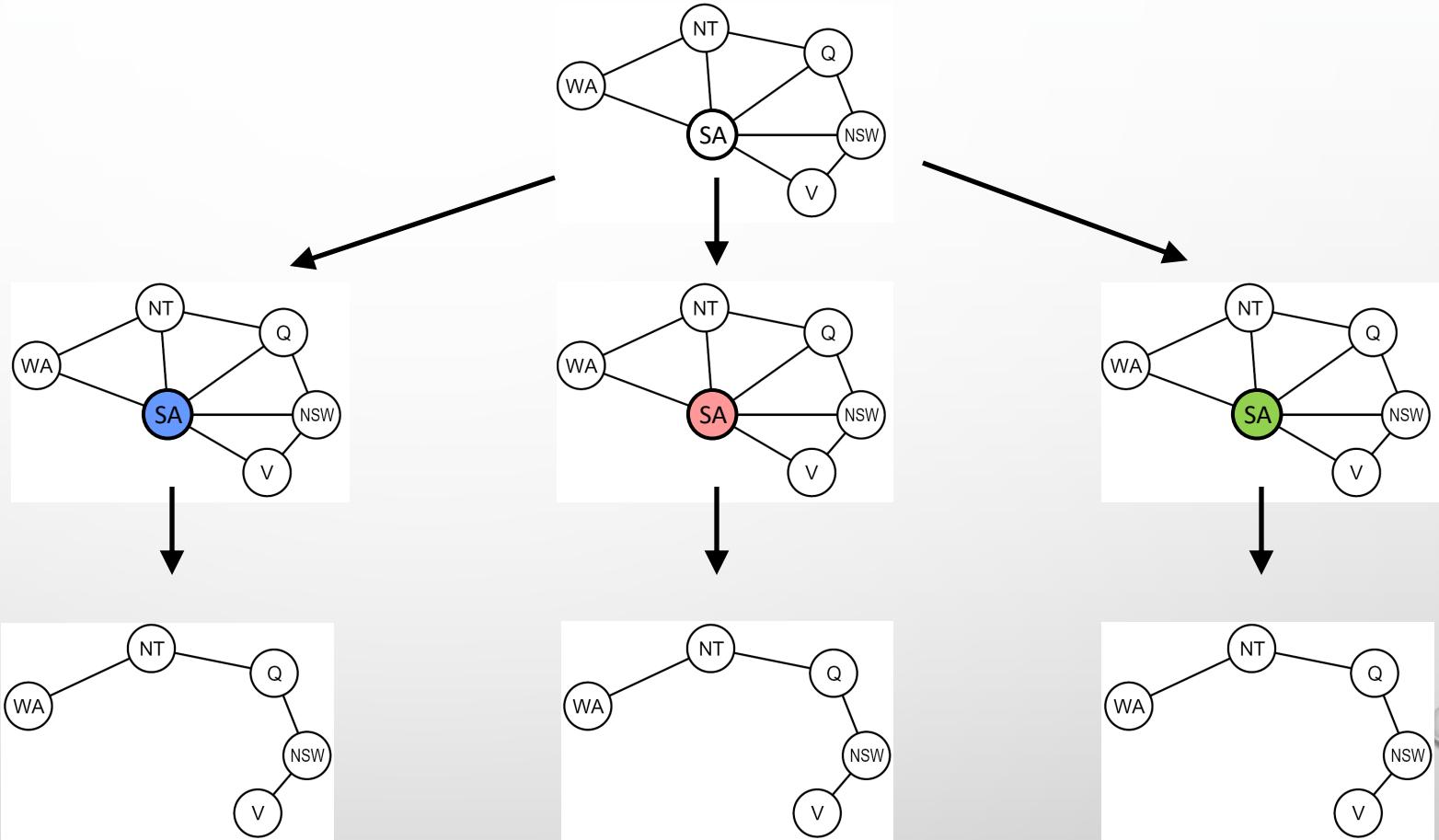
# Cutset Conditioning

Choose a cutset

Instantiate the cutset  
(all possible ways)

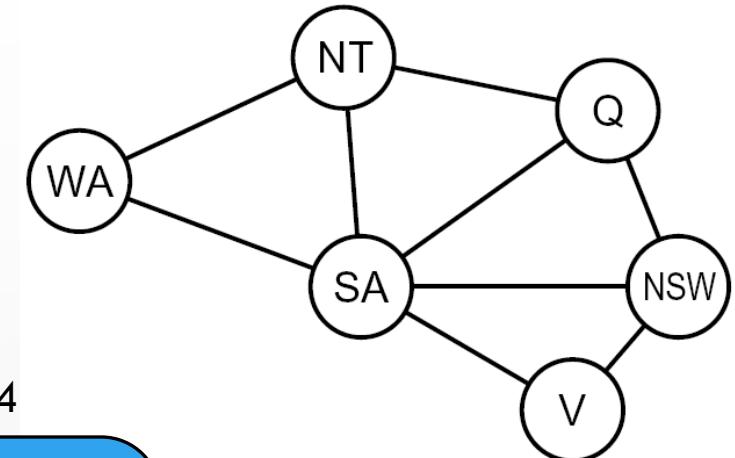
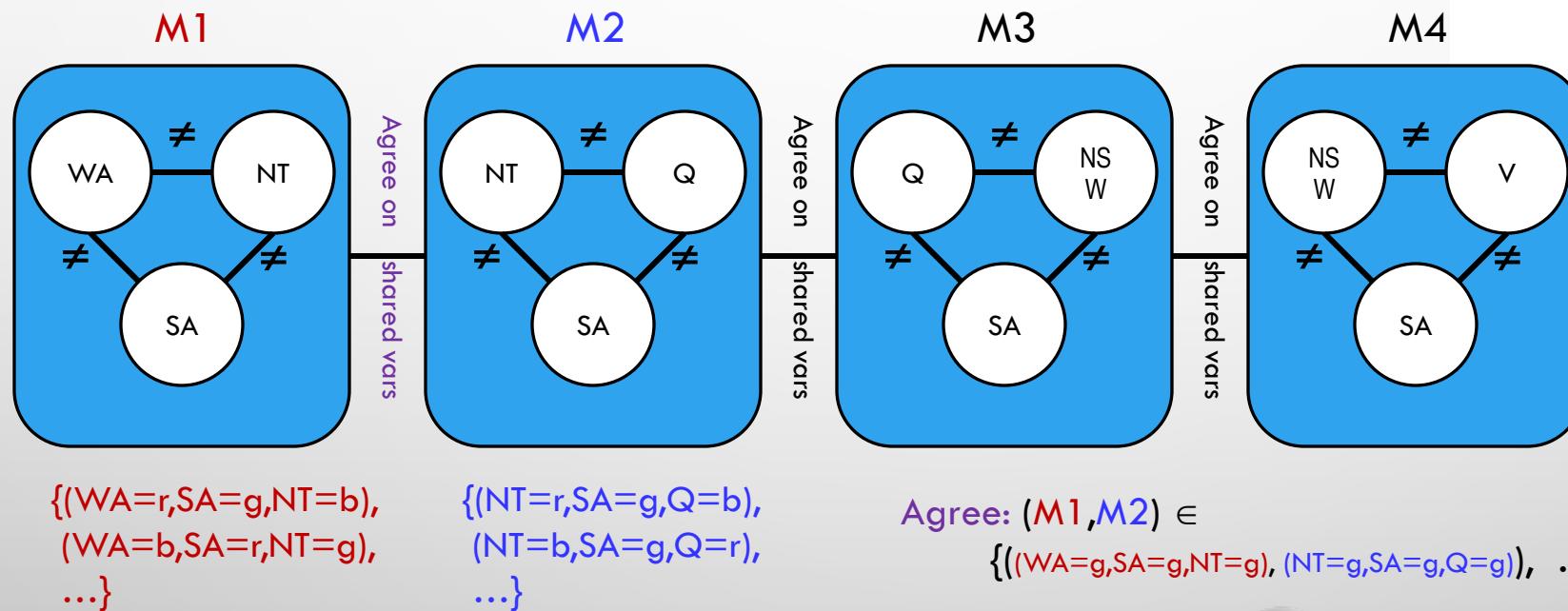
Compute residual CSP  
for each assignment

Solve the residual CSPs  
(tree structured)



# 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



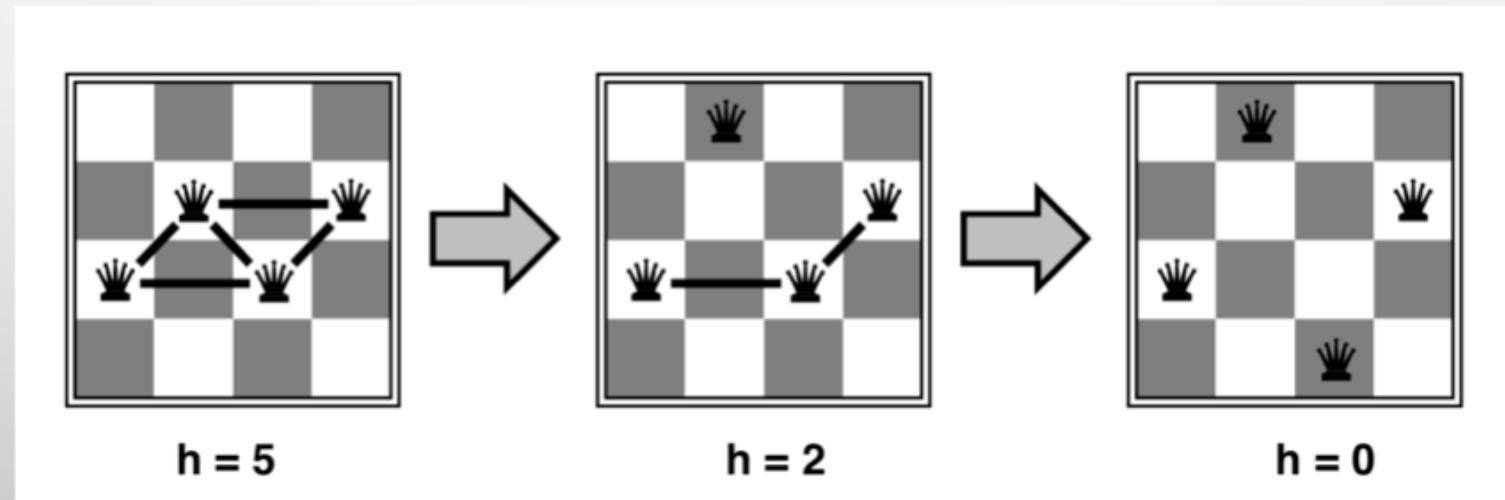
# Iterative algorithms for CSPs

- Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned.
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - Operators: **reassign** variable values
- Variable selection: randomly select any conflicted variable
- Value selection by **min-conflicts** heuristic:  
choose value that **violates the fewest constraints**  
i.e., hill-climb with  $h(n) = \text{total number of violated constraints}$



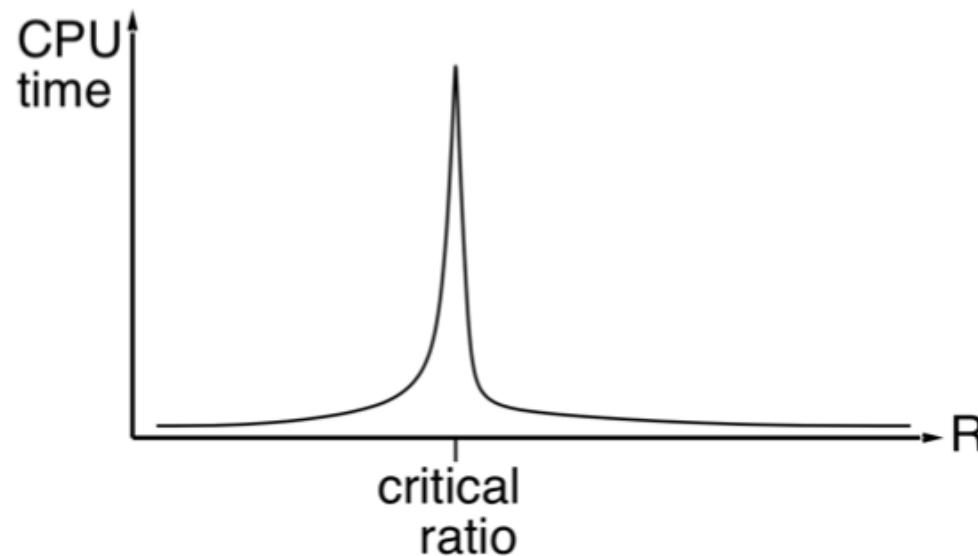
# Example: 4-Queens

- **States:** 4 queens in 4 columns ( $4^4 = 256$  states)
- **Operators:** move queen in column
- **Goal test:** no attacks
- **Evaluation:**  $h(n) = \text{number of attacks}$



# Performance of min-conflicts

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



# Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by **constraints** on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure

## Summary (cont.)

- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- The CSP representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice