

# Announcements

---

- ❖ **Project 2: Multi-agent Search**
  - ❖ Due Friday June 19 at 11:59pm.
- ❖ **Homework 4: RL**
  - ❖ Has been released! Due Wed. June 17 at 11:59pm.
- ❖ **Midterm Exam**
  - ❖ June 22, 4pm-5:40pm
  - ❖ Covers everything, CSP included.
- ❖ **Recitations**
  - ❖ Next week, answer poll on Canvas

# Ve492: Introduction to Artificial Intelligence

## Constraint Satisfaction Problems I



Paul Weng

UM-SJTU Joint Institute

Slides adapted from <http://ai.berkeley.edu>, AIMA, UM, CMU

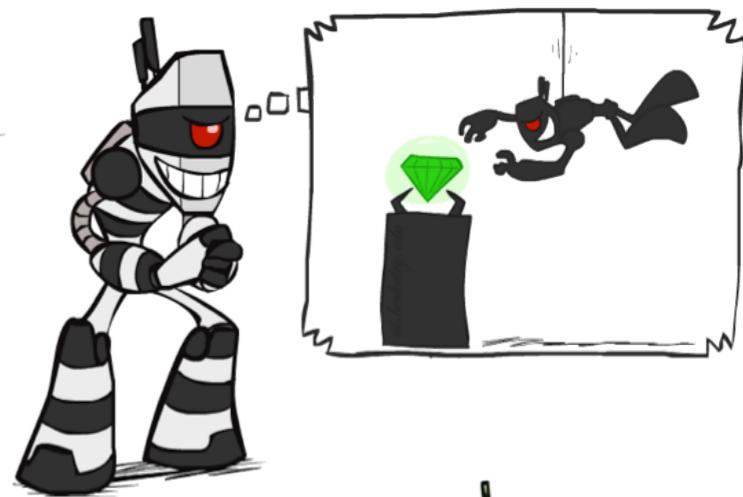
# What have we learned so far?

- ❖ Search and planning
  - ❖ Define a state space, goal test; Find path from start to goal
- ❖ Game trees
  - ❖ Define utilities; Find path from start that maximizes utility
- ❖ Decision theory and game theory
  - ❖ Foundation for MEU; Basic concepts in game theory
- ❖ MDPs
  - ❖ Define rewards, utility = (discounted) sum of rewards
  - ❖ Find policy that maximizes utility
- ❖ Reinforcement learning
  - ❖ Just like MDPs, only T and / or R are not known in advance
- ❖ Today: constraint satisfaction
  - ❖ Find solution that satisfies constraints; Not just for finding a sequential plan

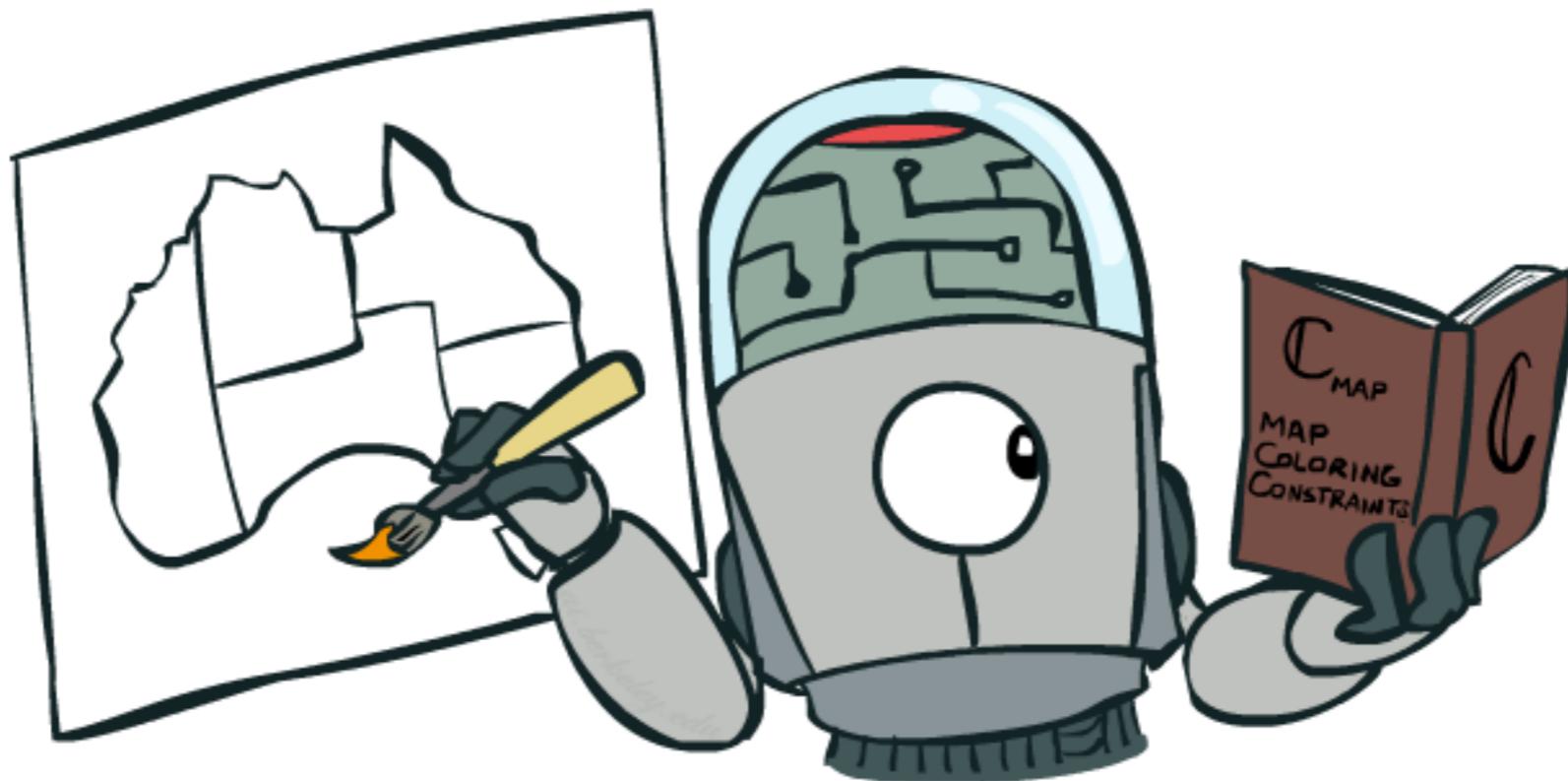


# What is Search For?

- ❖ 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

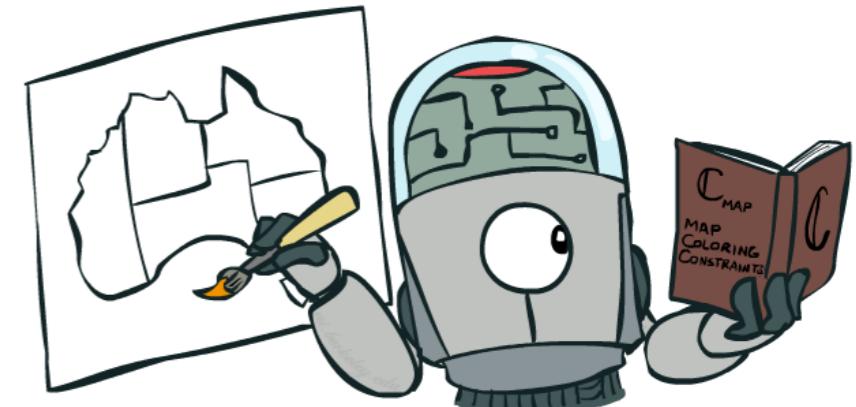
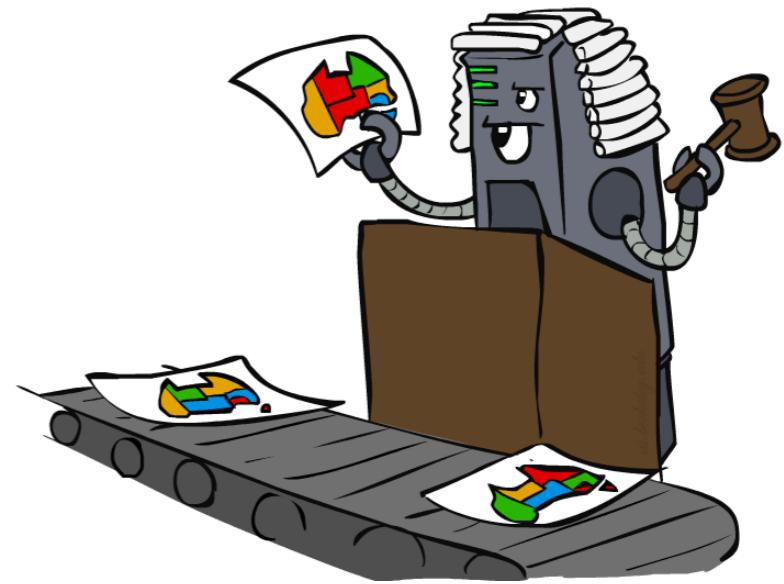


# Constraint Satisfaction Problems

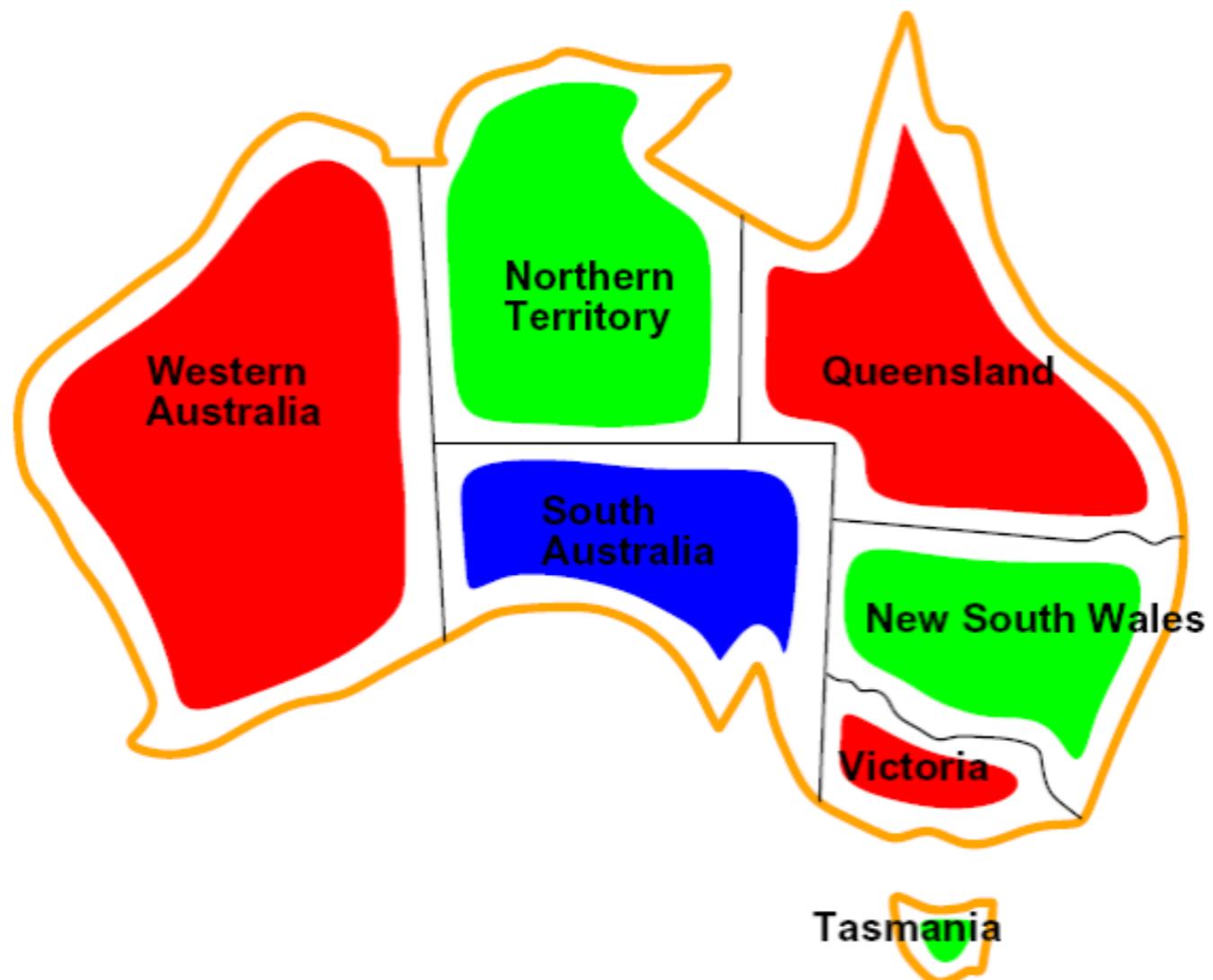


# Constraint Satisfaction Problems

- ❖ Standard search problems:
  - ❖ State is a “black box”: arbitrary data structure
  - ❖ Goal test can be any function over states
  - ❖ Successor function can also be anything
- ❖ Constraint satisfaction problems (CSPs):
  - ❖ A special subset of search problems
  - ❖ State is defined by variables  $X_i$  with values from a domain  $D$  (sometimes  $D$  depends on  $i$ )
  - ❖ 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



# CSP Examples



# Example: Map Coloring

- ❖ Variables: WA, NT, Q, NSW, V, SA, T

- ❖ Domains:  $D = \{\text{red, green, blue}\}$

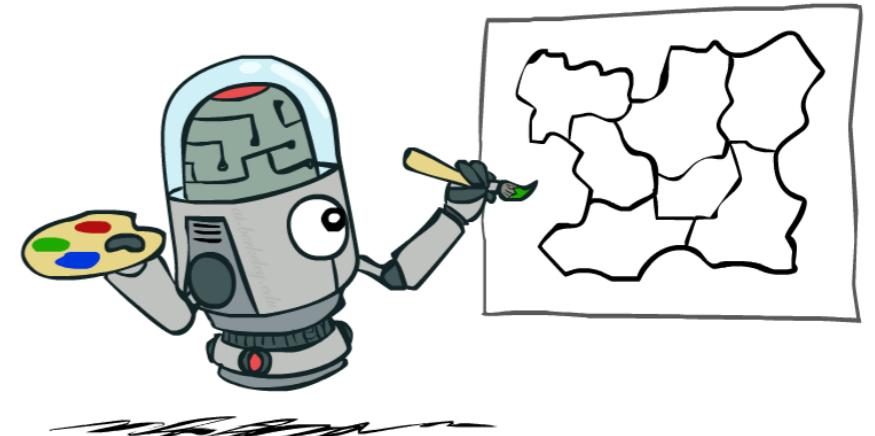
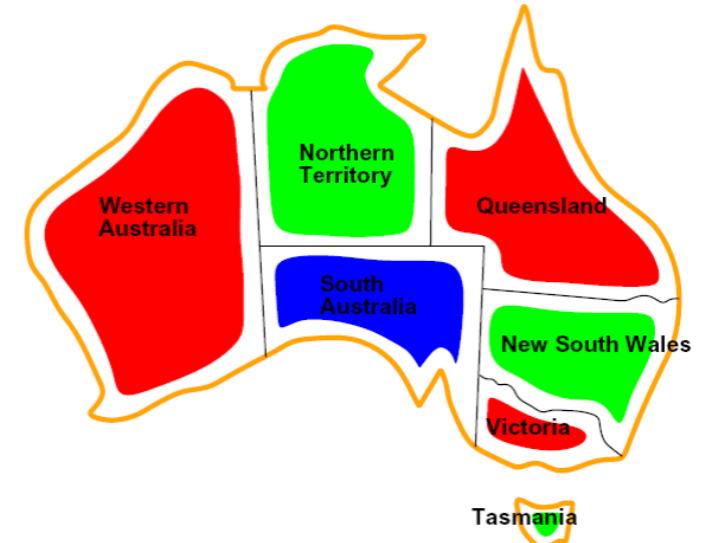
- ❖ Constraints: adjacent regions must have different colors

Implicit:  $\text{WA} \neq \text{NT}$

Explicit:  $(\text{WA}, \text{NT}) \in \{(\text{red, green}), (\text{red, blue}), \dots\}$

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

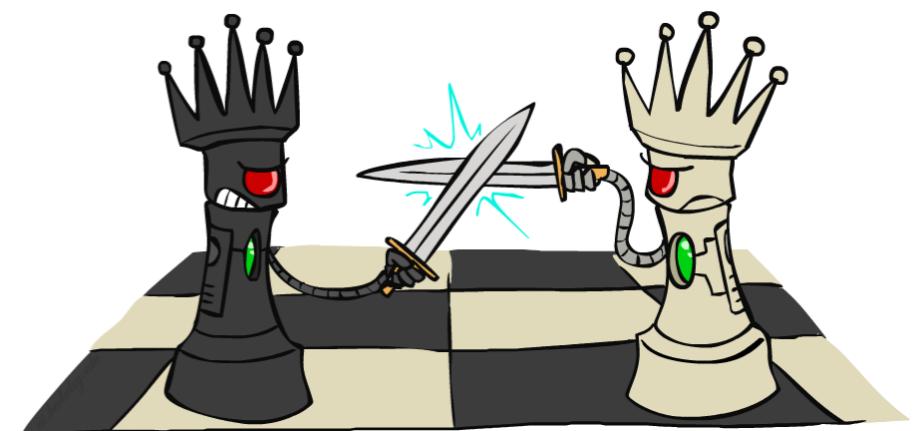
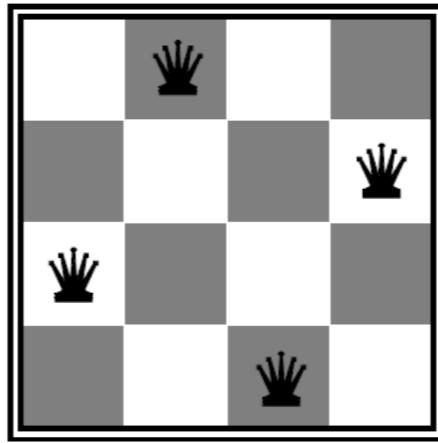
$\{\text{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}\}$



# 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

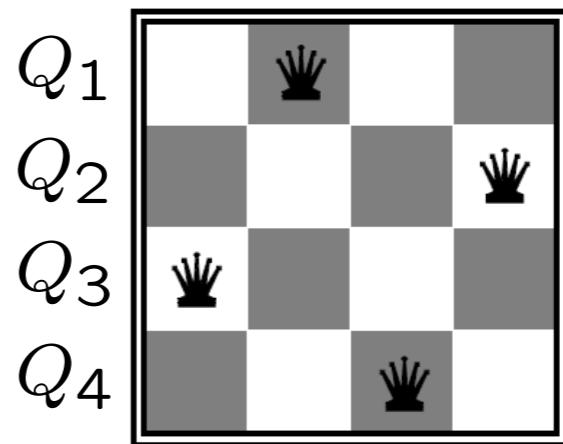
- ❖ Formulation 2:

- ❖ Variables:  $Q_k$
- ❖ Domains:  $\{1, 2, 3, \dots, N\}$
- ❖ Constraints:

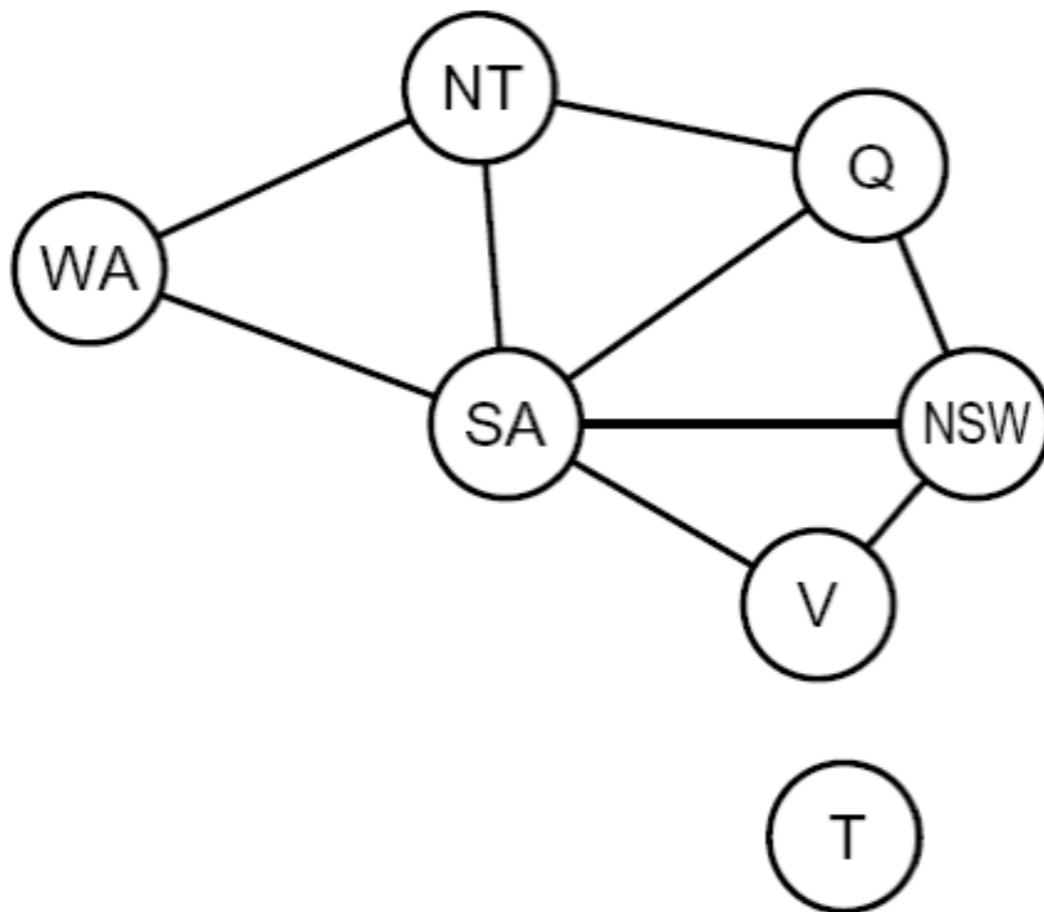
Implicit:  $\forall i, j \text{ non-threatening}(Q_i, Q_j)$

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

• • •

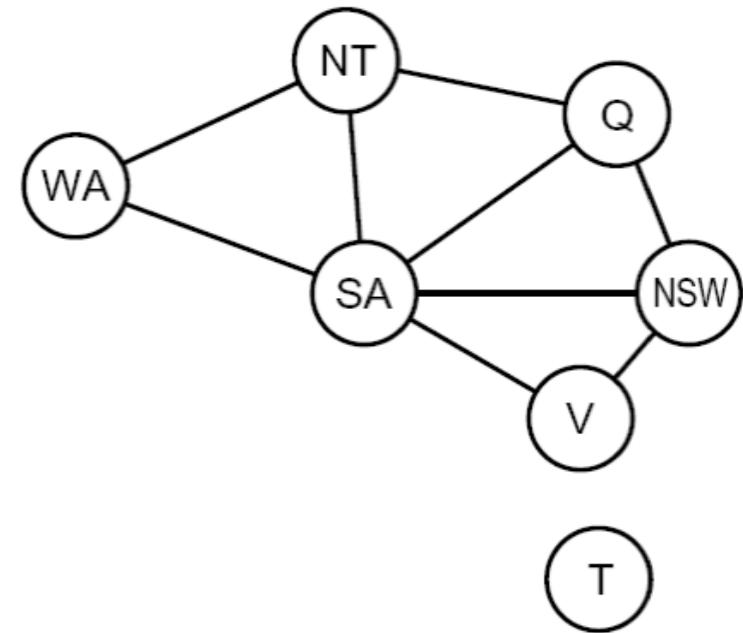


# Constraint Graphs



# 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: [CSP solver](#)



# Example: Cryptarithmetic

- ❖ **Variables:**

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

- ❖ **Domains:**

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

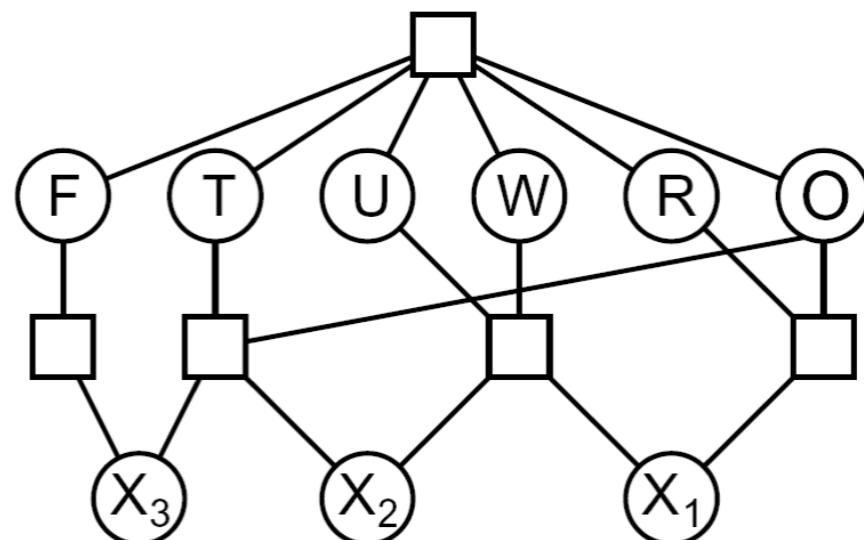
- ❖ **Constraints:**

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

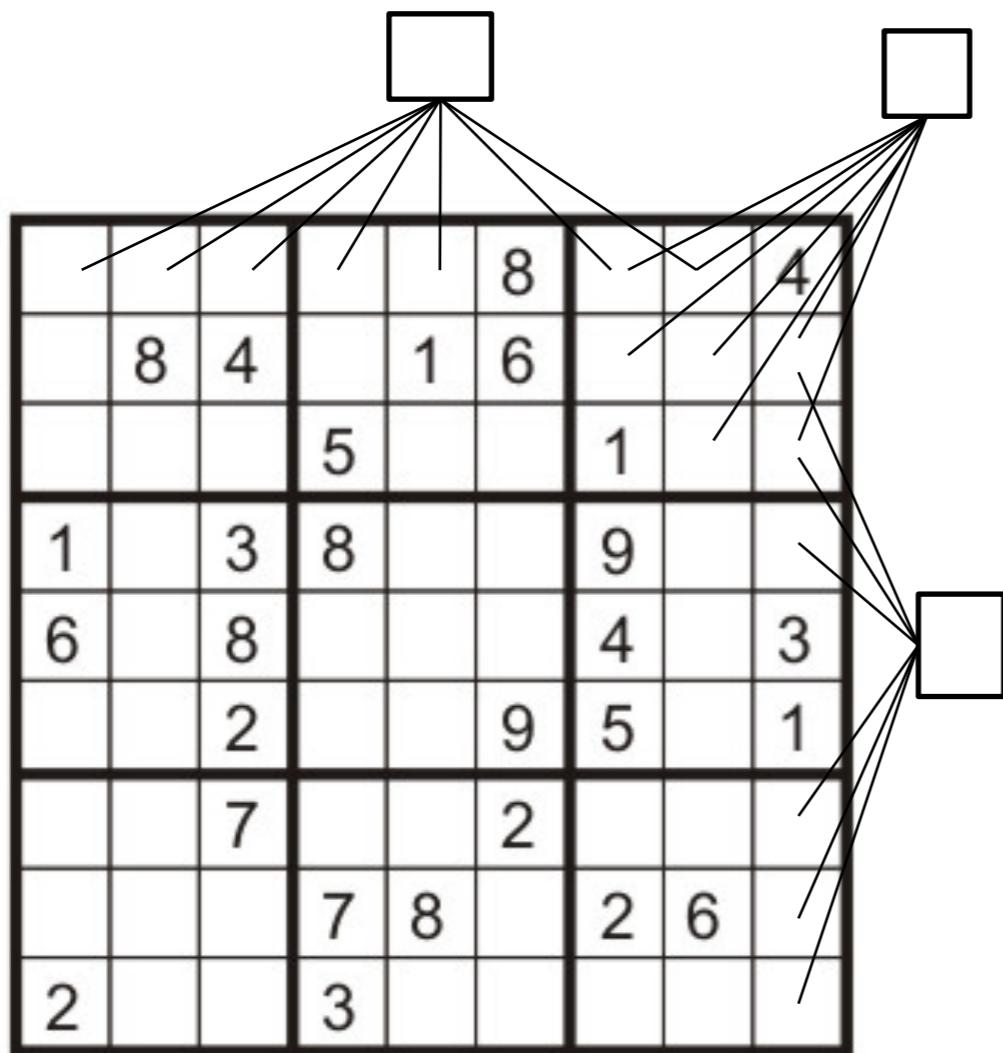
$$O + O = R + 10 \cdot X_1$$

...

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

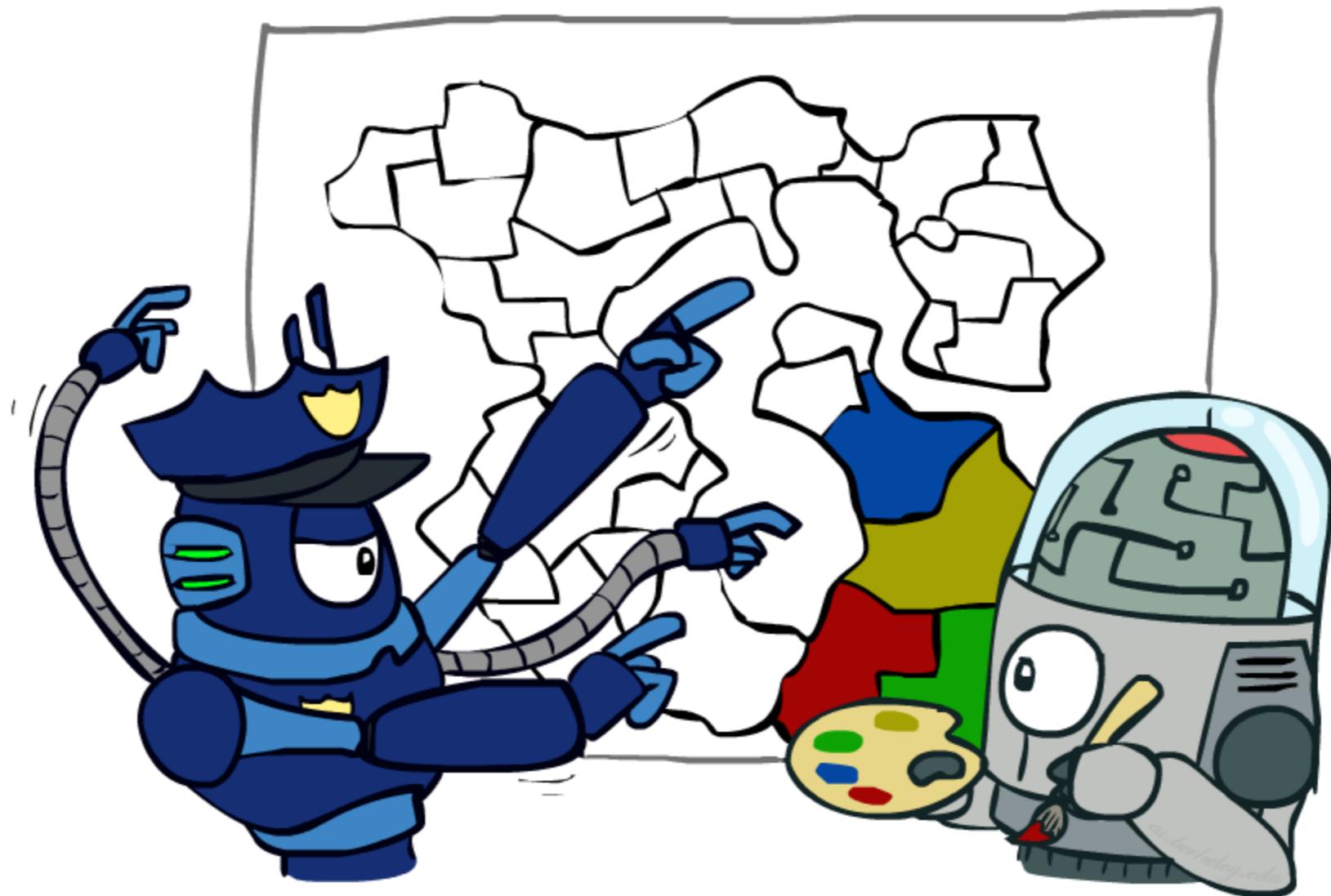


# Example: Sudoku



- ❖ **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)

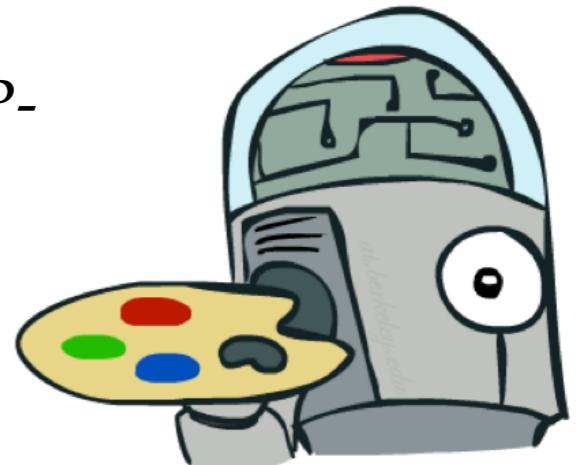
# Varieties of CSPs and Constraints



# 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



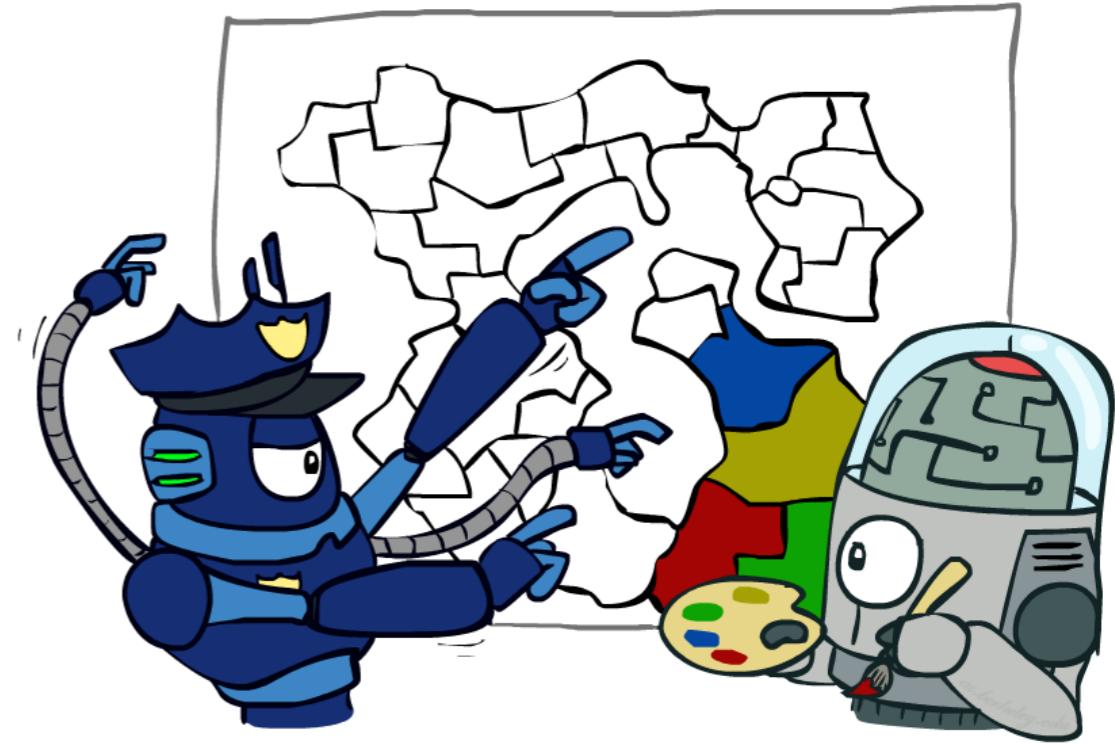
## ❖ Continuous variables

- ❖ E.g., start/end times for Hubble Telescope observations
- ❖ Linear constraints solvable in polynomial time by LP methods (see Ve555 for a bit of this theory)



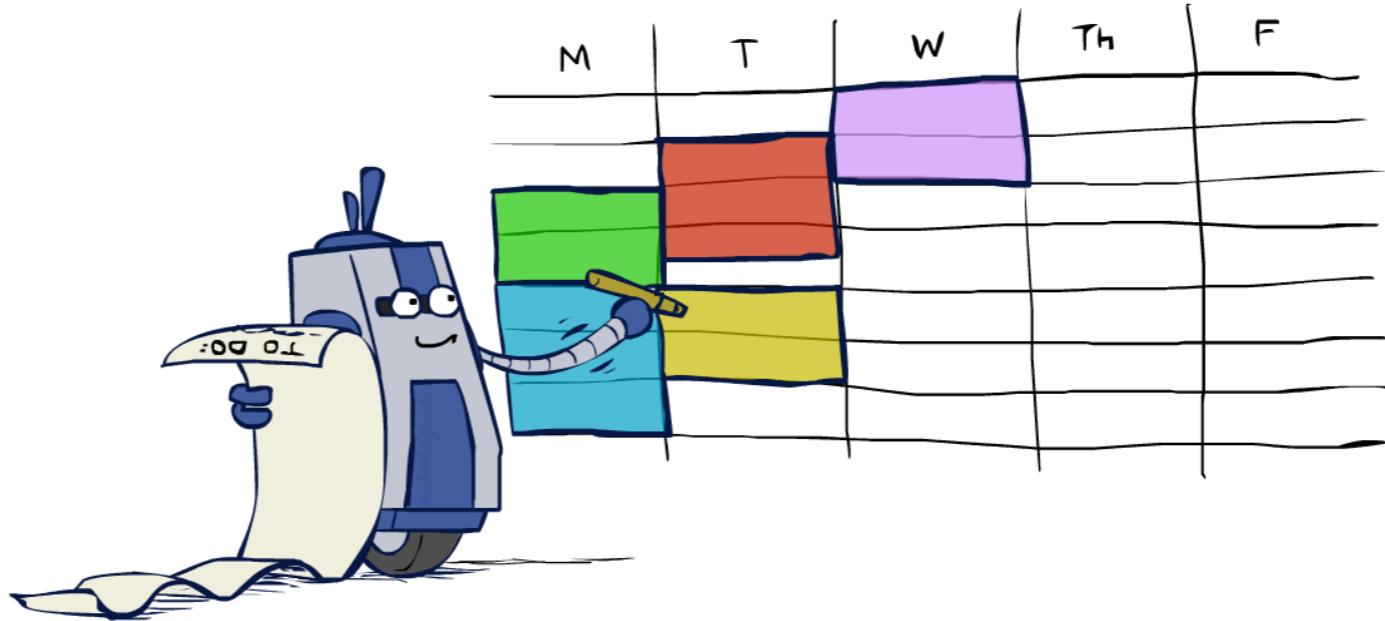
# Varieties of Constraints

- ❖ Varieties of Constraints
  - ❖ Unary constraints involve a single variable (equivalent to reducing domains), e.g.:  
 $SA \neq \text{green}$
  - ❖ Binary constraints involve pairs of variables, e.g.:  
 $SA \neq WA$
  - ❖ Higher-order constraints involve 3 or more variables:  
e.g., cryptarithmic 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)



# 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!



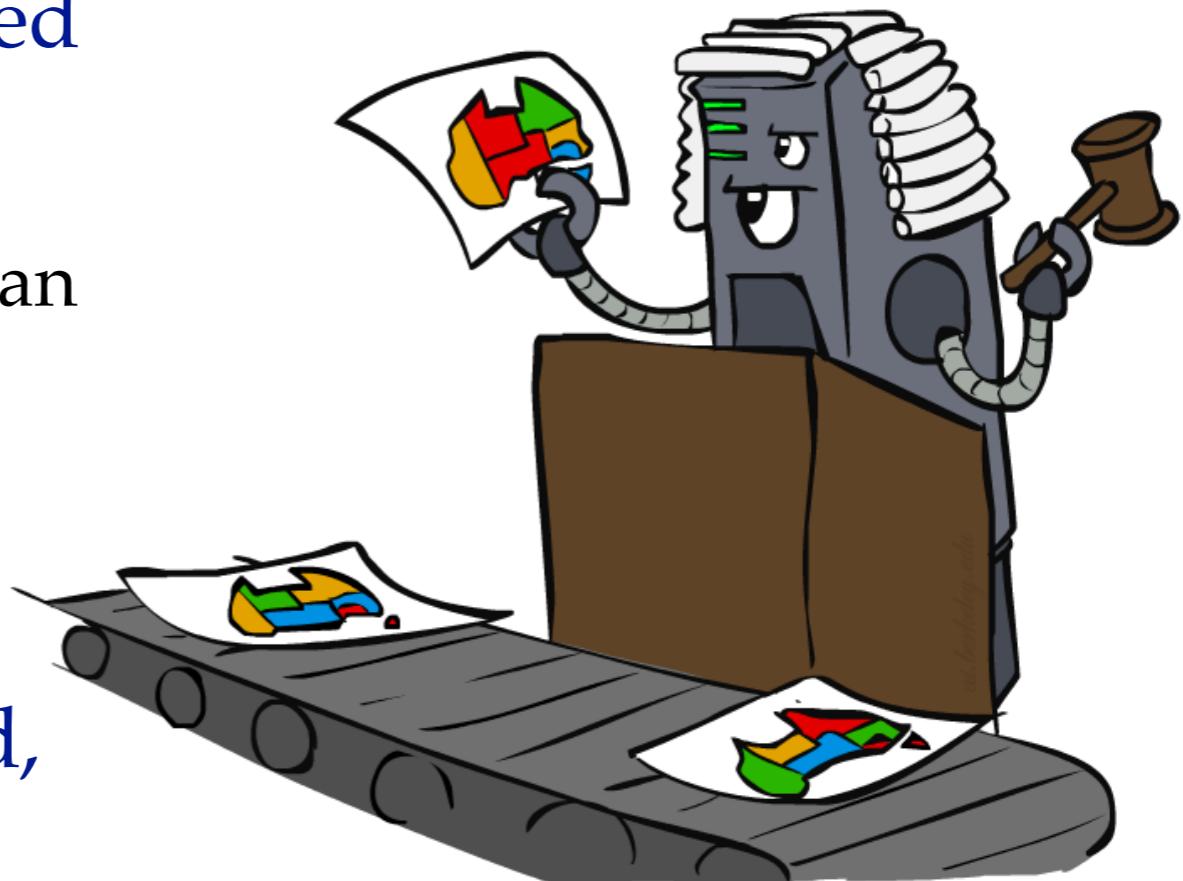
- ❖ Many real-world problems involve real-valued variables...

# Solving CSPs



# 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

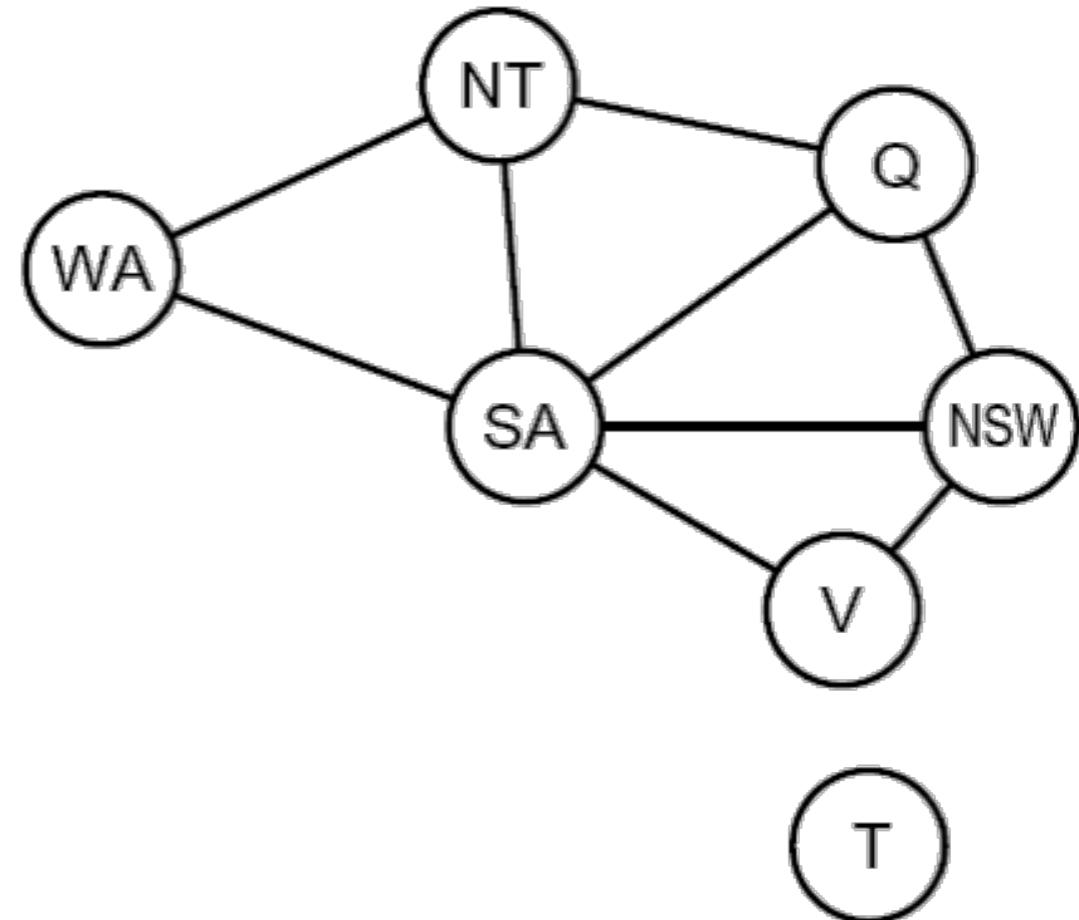


# Search Methods

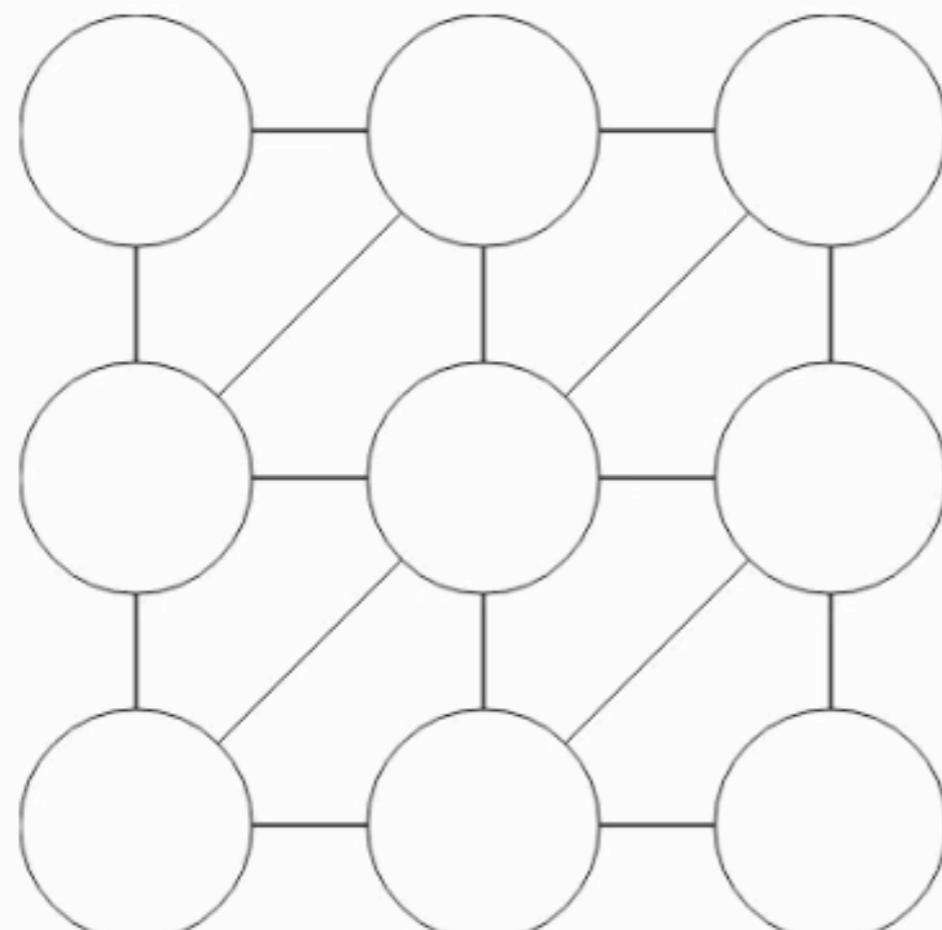
- ❖ What would BFS do?

- ❖ What would DFS do?

- ❖ What problems does naïve search have?



# Video of Demo Coloring -- DFS



Reset

Prev

Pause

Next

Play

Faster

## Graph

Simple

## Algorithm

Naive Search

## Ordering

- None
- MRV
- MRV with LCV

## Filtering

- None
- Forward Checking
- Arc Consistency

## Speed

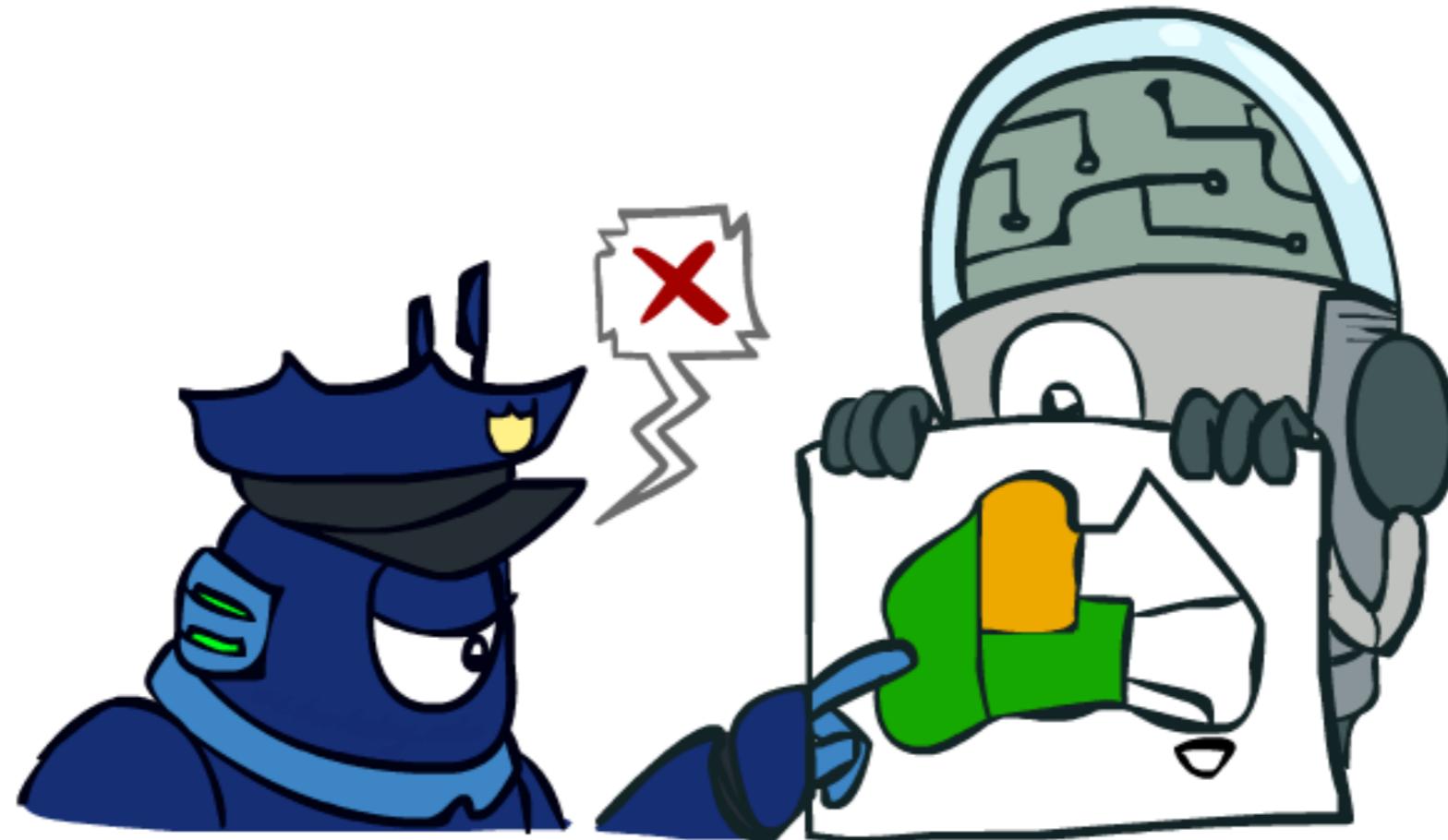
Speedup

1

Frame Delay

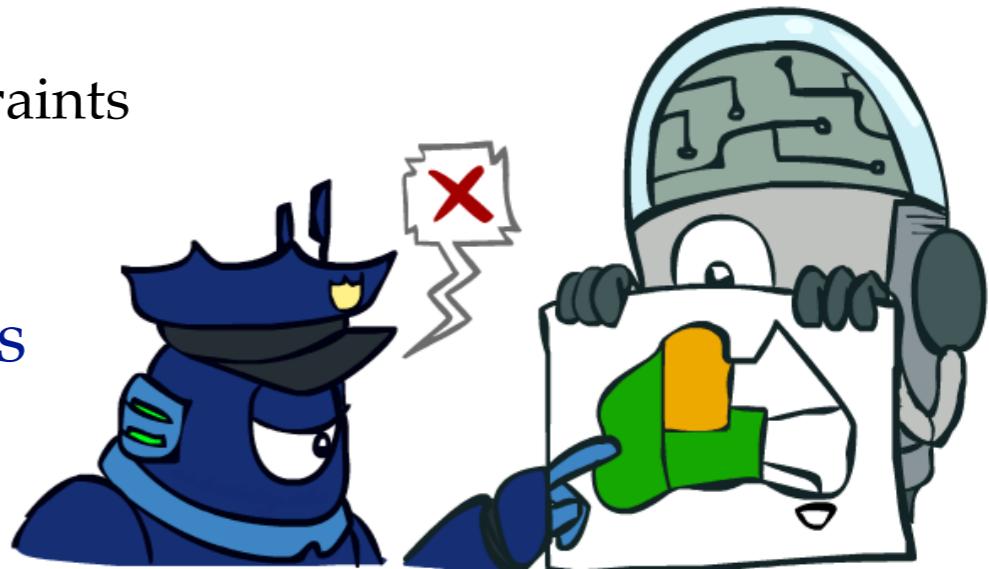
700

# Backtracking Search

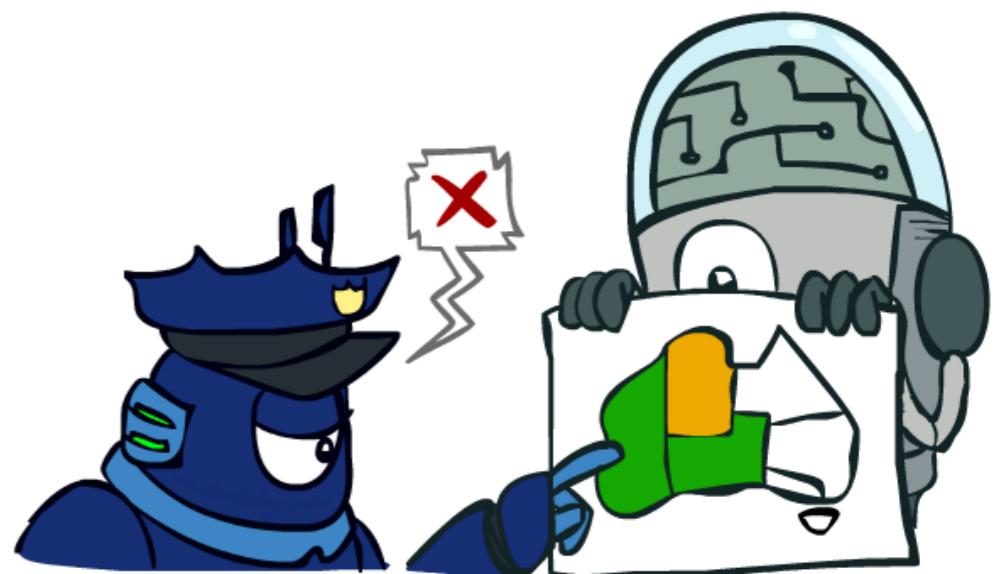
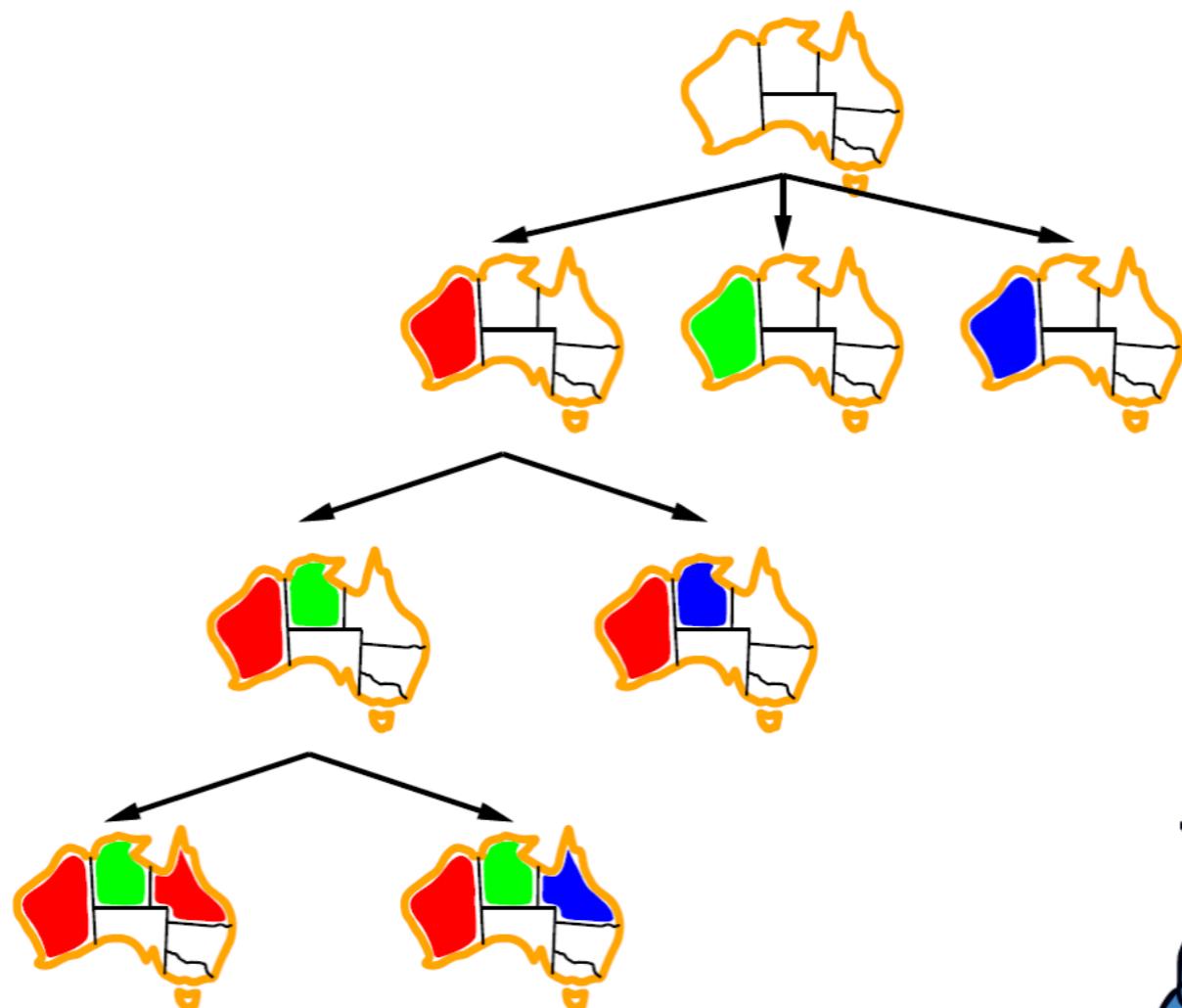


# Backtracking Search

- ❖ 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
  - ❖ I.e. 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

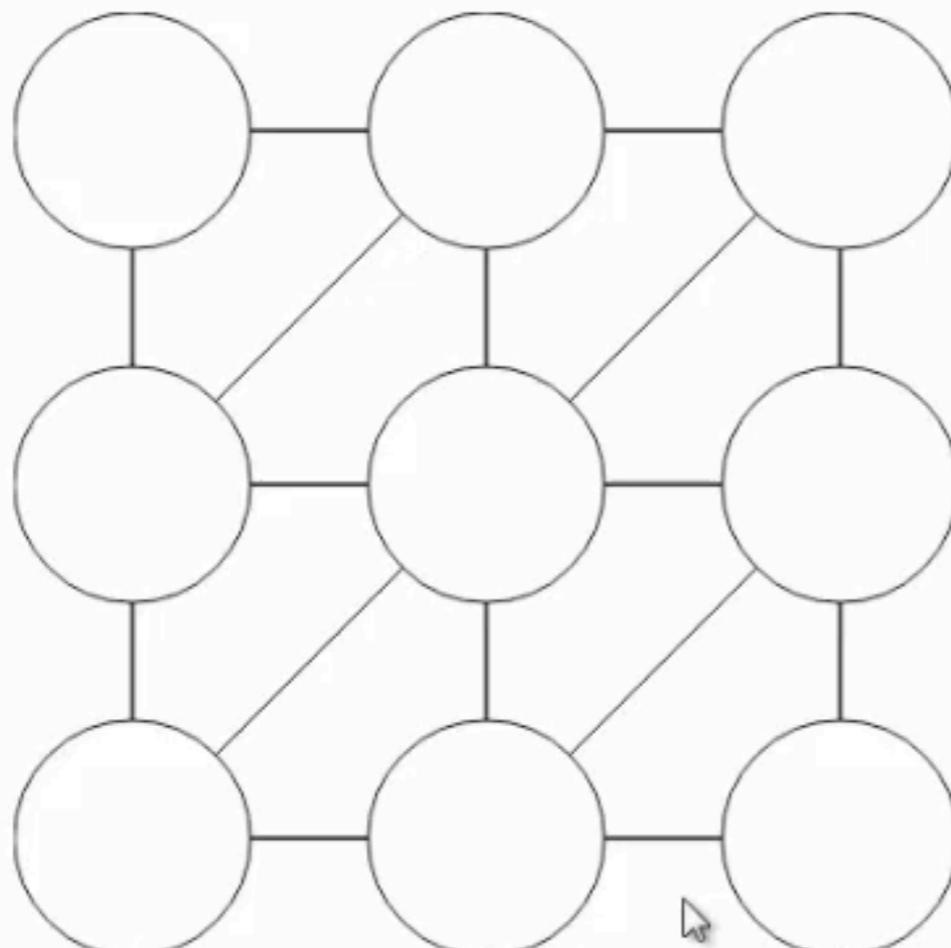


# Backtracking Search

```
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 = DFS + variable-ordering + fail-on-violation
- ❖ What are the choice points?

# Video of Demo Coloring – Backtracking



Reset Prev Pause Next Play Faster

## Graph

Simple ▾

## Algorithm

Backtracking ▾

## Ordering

- None
- MRV
- MRV with LCV

## Filtering

- None
- Forward Checking
- Arc Consistency

## Speed

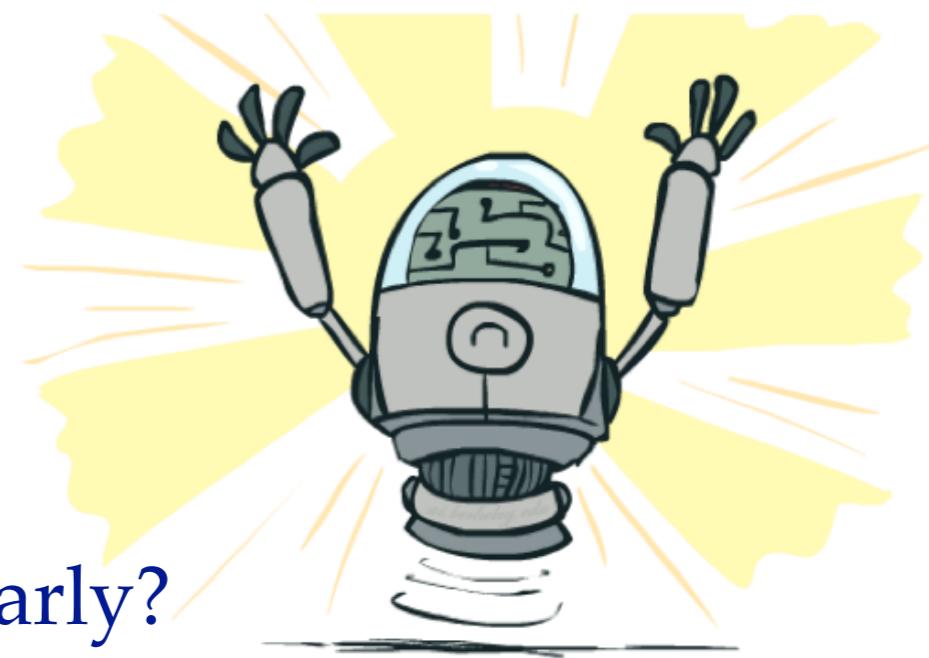
Speedup Frame Delay

1 ▾ x

700

# 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?

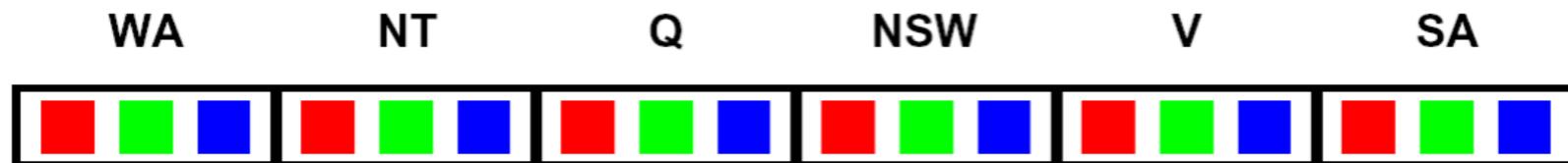


# Filtering

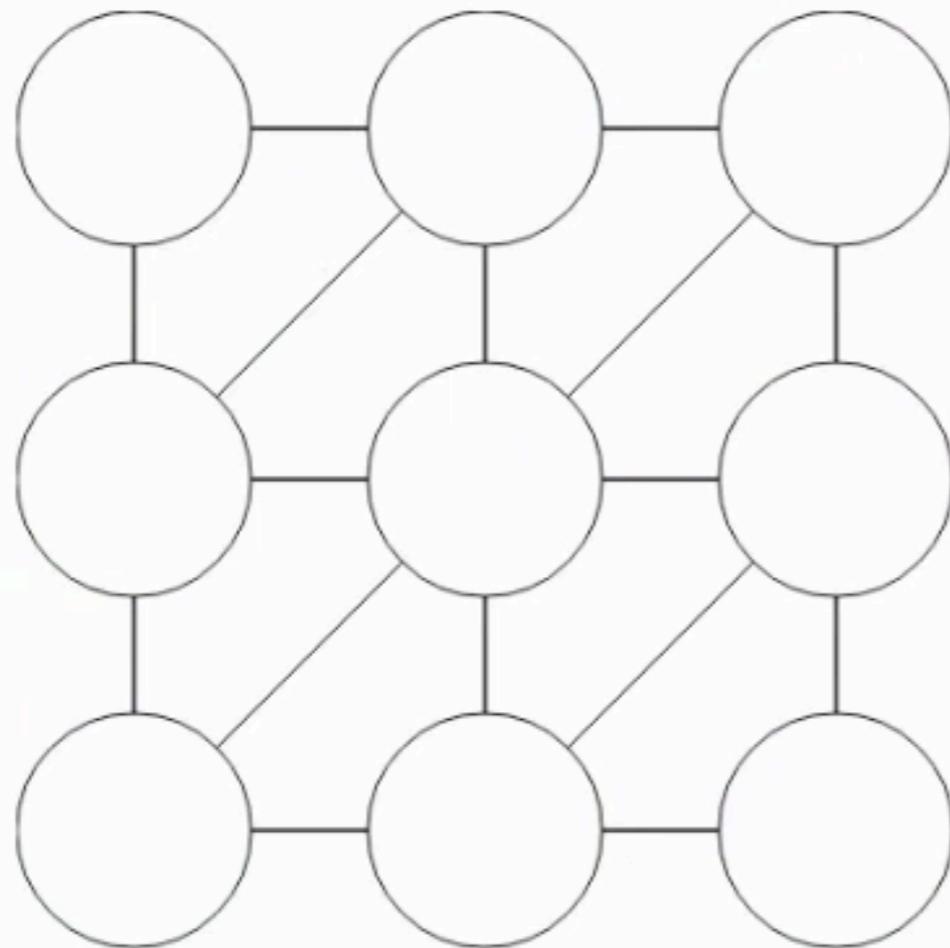


# Filtering: Forward Checking

- ❖ 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



# Video of Demo Coloring – Backtracking with Forward Checking



Reset Prev Pause Next Play Faster

## Graph

Simple

## Algorithm

Backtracking

## Ordering

- None
- MRV
- MRV with LCV

## Filtering

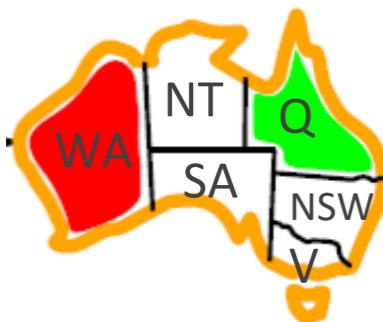
- None
- Forward Checking
- Arc Consistency

## Speed

Speedup 1 x Frame Delay 700

# Filtering: Constraint Propagation

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

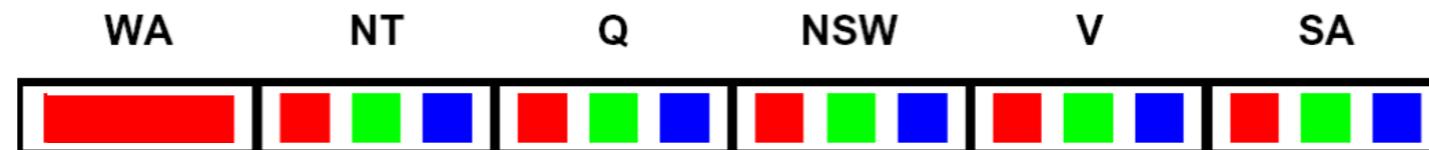


WA	NT	Q	NSW	V	SA
Red	Green	Blue	Red	Green	Blue
Red		Green	Blue	Red	Green
Red		Blue	Green	Red	Blue

- 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

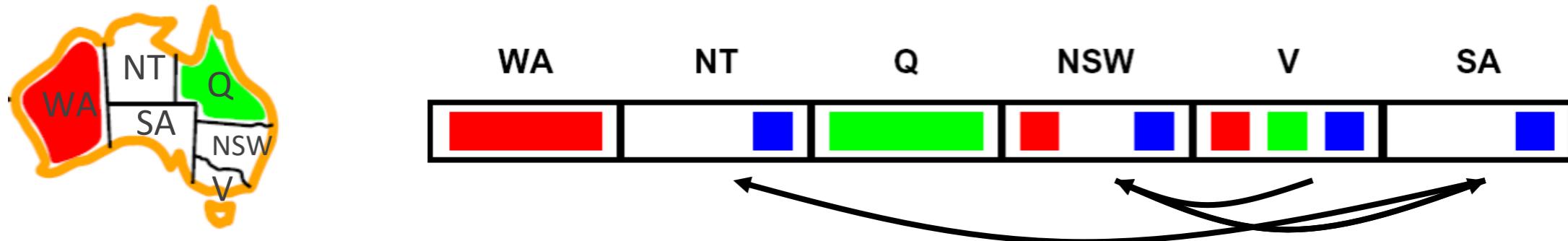


*Delete from the tail!*

- 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  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp

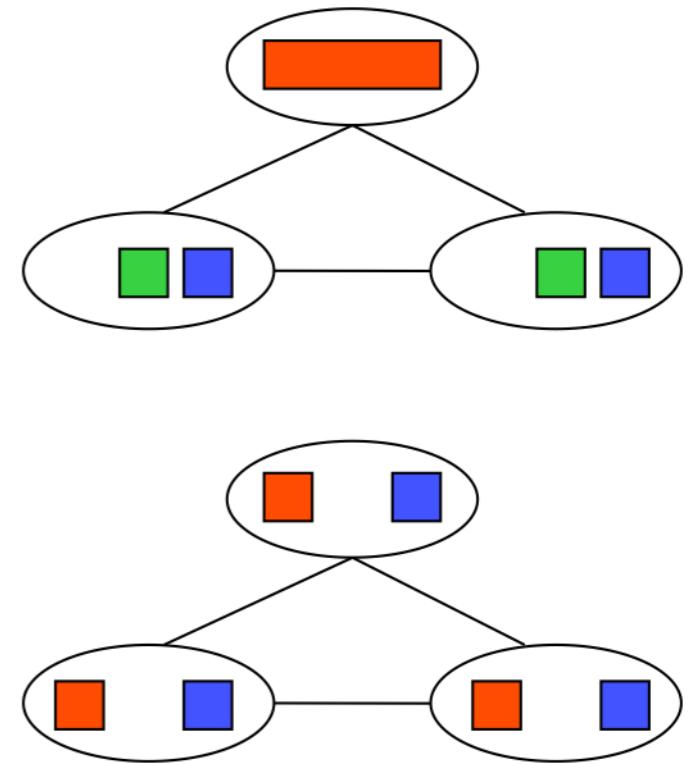
  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
        add  $(X_k, X_i)$  to queue

  function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
    removed  $\leftarrow \text{false}$ 
    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  $\leftarrow \text{true}$ 
    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?*

# Quiz: What is the color of NSW?

- ❖ Run arc consistency on this example:



- ❖ Choose a color
  - ❖ Red
  - ❖ Green
  - ❖ Blue

# Video of Demo Coloring – Backtracking with Forward Checking – Complex Graph

Graph: Complex

Algorithm: Backtracking

Ordering:

- None
- MRV
- MRV with LCV

Filtering:

- None
- Forward Checking
- Arc Consistency

Speed:

- Speedup: 1 x
- Frame Delay: 700

Buttons: Reset, Prev, Pause, **Next**, Play, Faster

# Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph

A complex graph for a 3-coloring problem. The graph consists of 20 nodes arranged in four columns of five nodes each. Each node is a circle containing three colored dots (blue, red, green) representing its current state. Edges connect adjacent nodes in a zig-zag pattern across columns. Some nodes have additional edges connecting them to nodes in other columns, creating a complex web of connections.

**Graph**: Complex

**Algorithm**: Backtracking

**Ordering**:

- None
- MRV
- MRV with LCV

**Filtering**:

- None
- Forward Checking
- Arc Consistency

**Speed**:

Speedup:  x      Frame Delay:

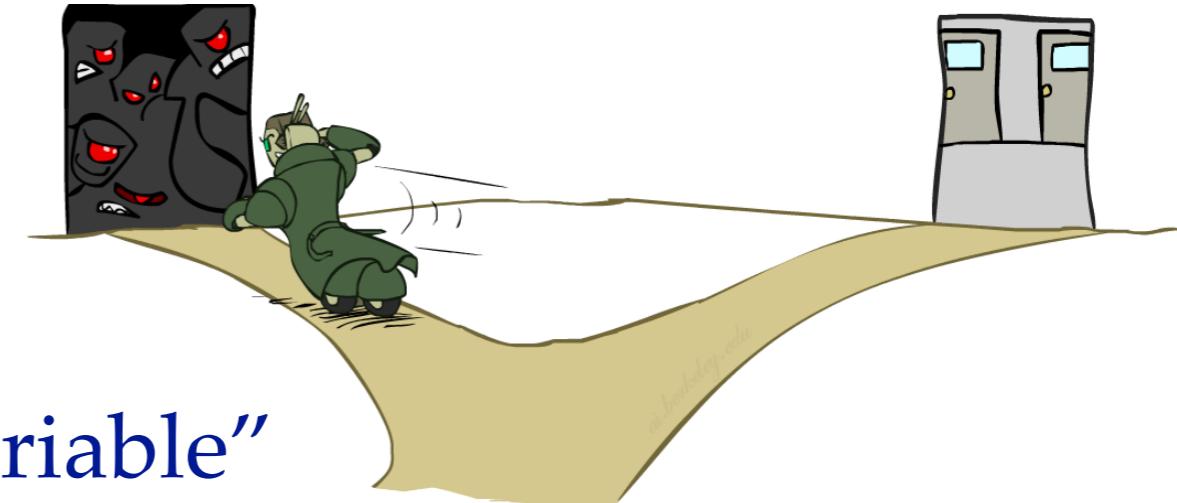
Buttons at the bottom: Reset, Prev, Pause, Next, Play, Faster

# Ordering



# Ordering: Minimum Remaining Values

- ❖ 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

# Ordering: Least Constraining Value

- ❖ 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

