

CS 188: Artificial Intelligence

Constraint Satisfaction Problems



Fall 2024

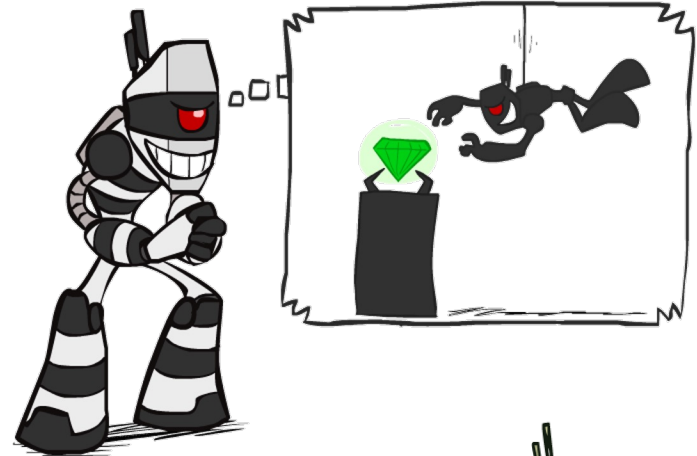
University of California, Berkeley

Announcements

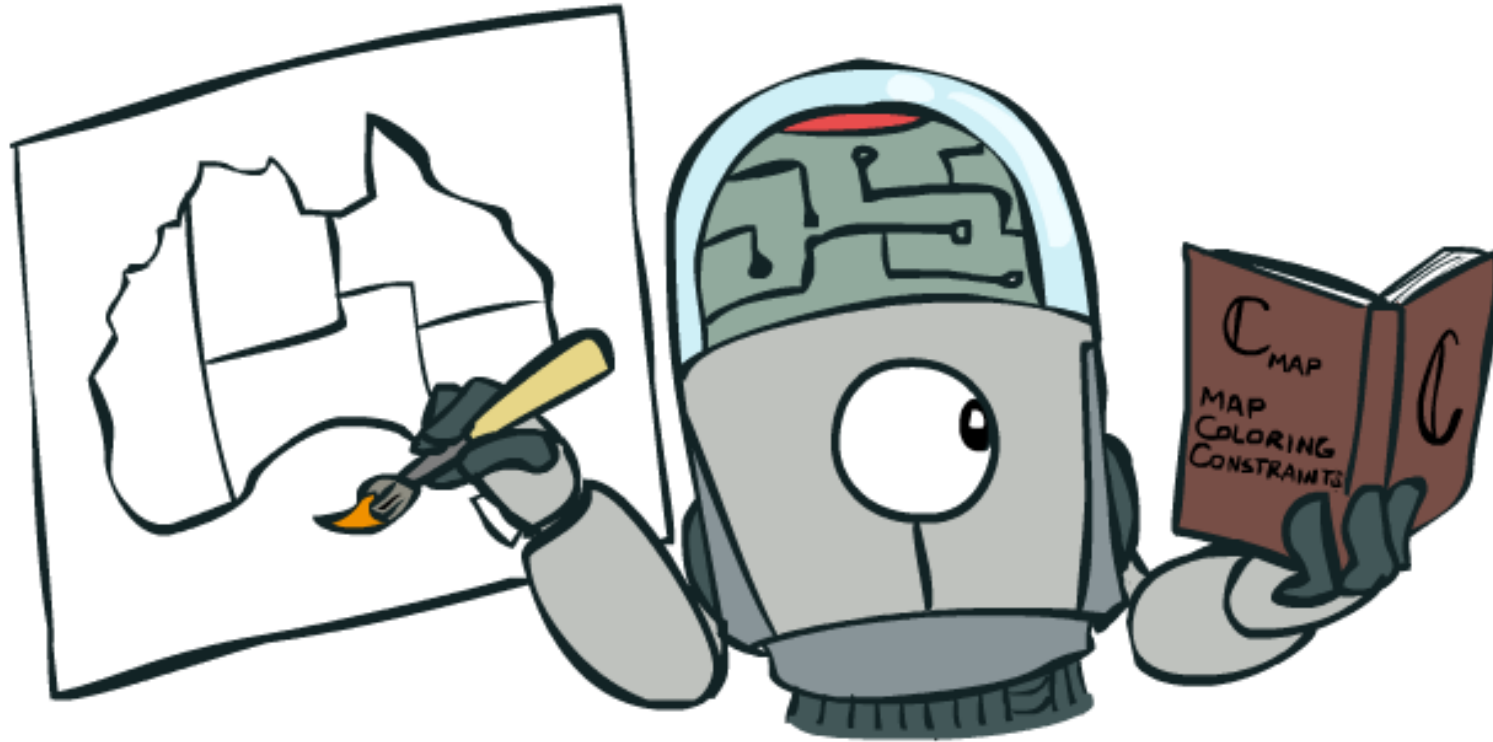
- **Homework 1** due **today** (Tuesday, Sept 10) at 11:59PM PT
- **Project 1** due **Friday, Sept 13** at 5:00PM PT
- **Discussions** start this week (times posted on course calendar)
- **Project Parties:**
 - Thursday, Sept 12 from 6:00PM to 8:00PM PT in Soda 341B
 - Friday, Sept 13 from 9:00AM to 2:00PM PT in Soda 341B
- Please do not use course staff personal emails for questions related to the course, email cs188@Berkeley.edu or post on Ed instead
- When posting (private) questions with code, either upload zip files of your code, or use *code block* functionality on Ed

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 a specialized class of identification problems

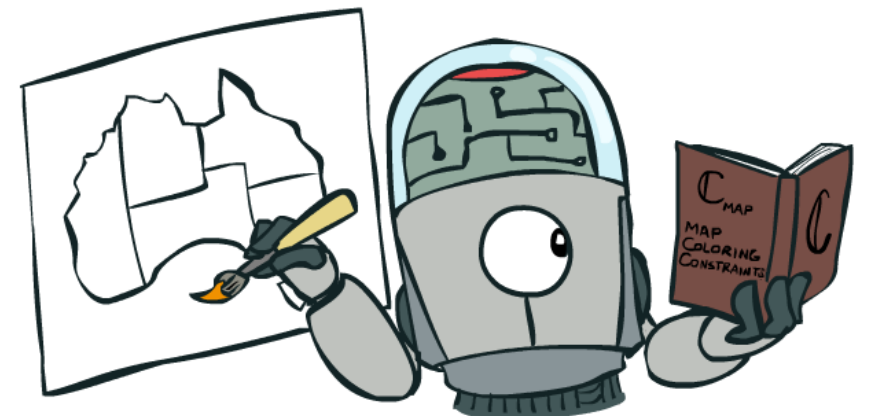
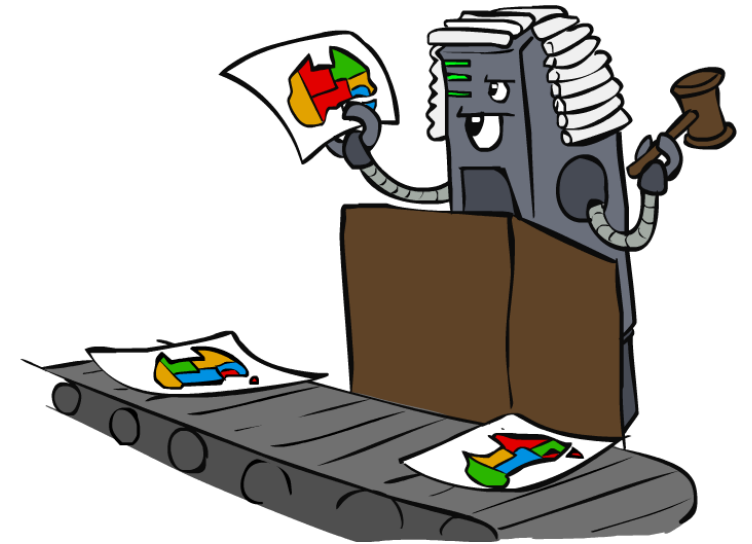


Constraint Satisfaction Problems

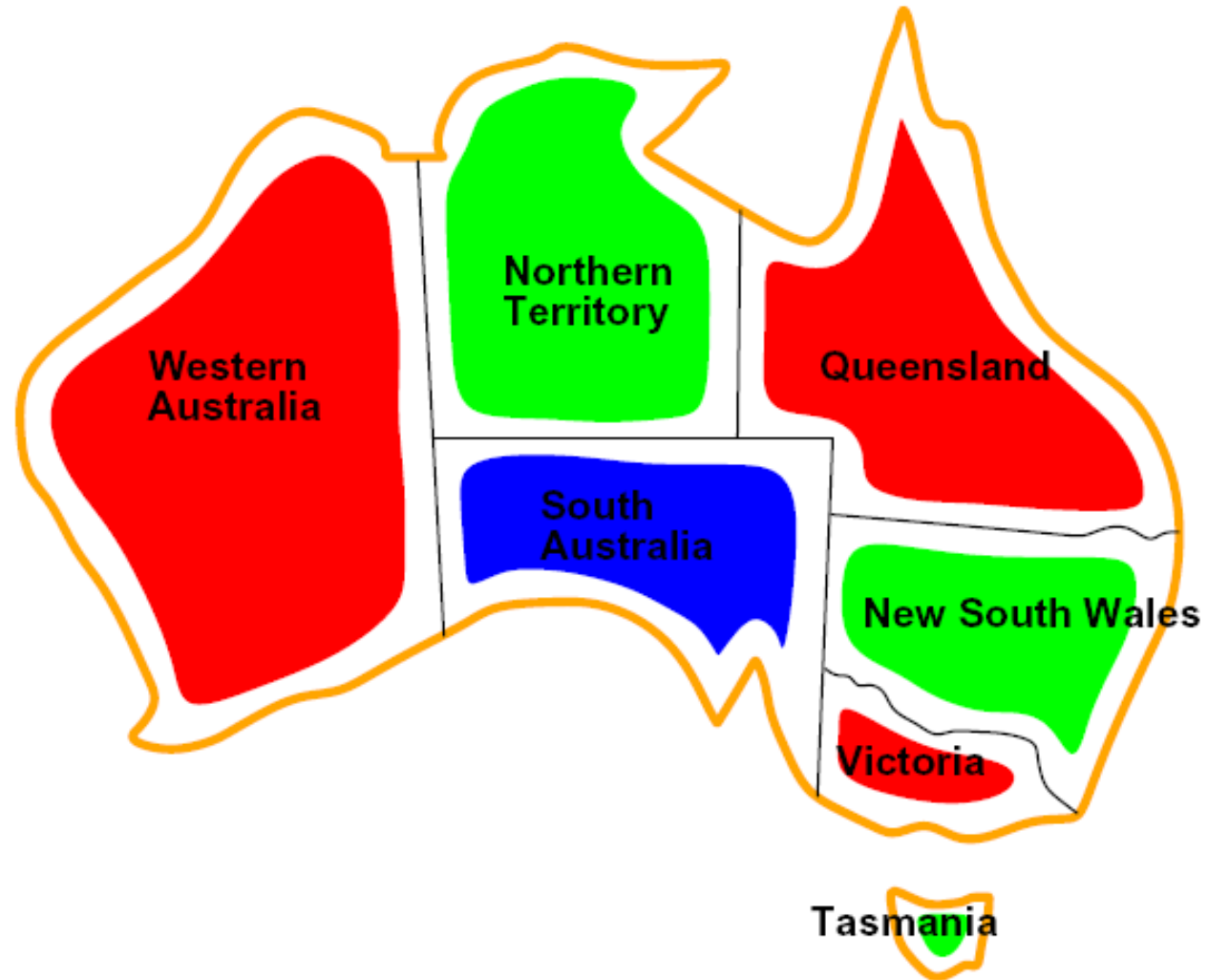


Constraint Satisfaction Problems

- 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

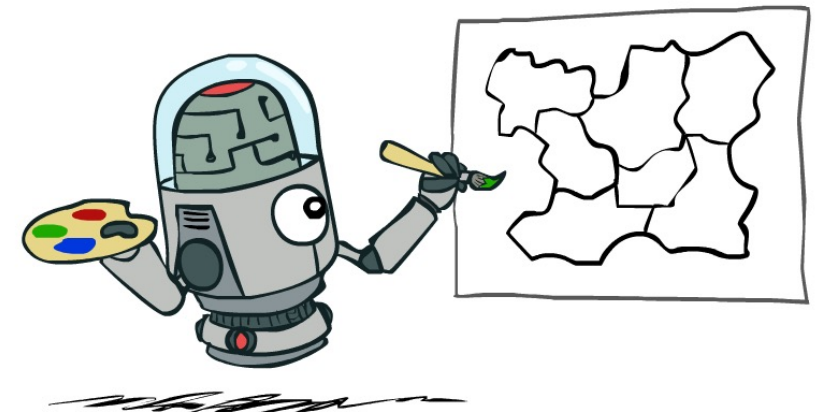
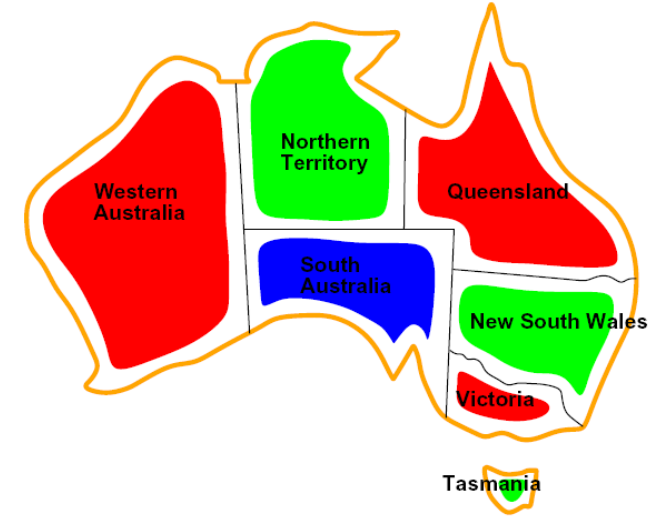
- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D = \{\text{red, green, blue}\}$
- Constraints: adjacent regions must have different colors

Implicit: $WA \neq NT$

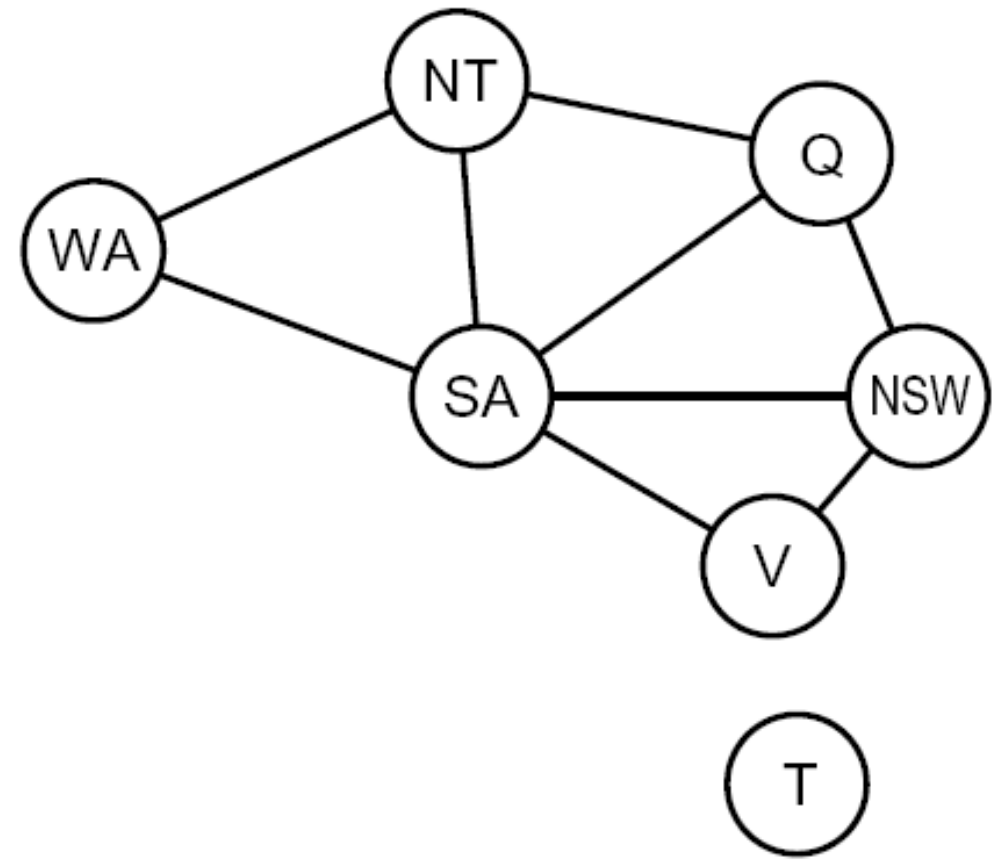
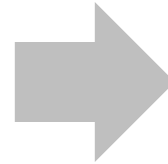
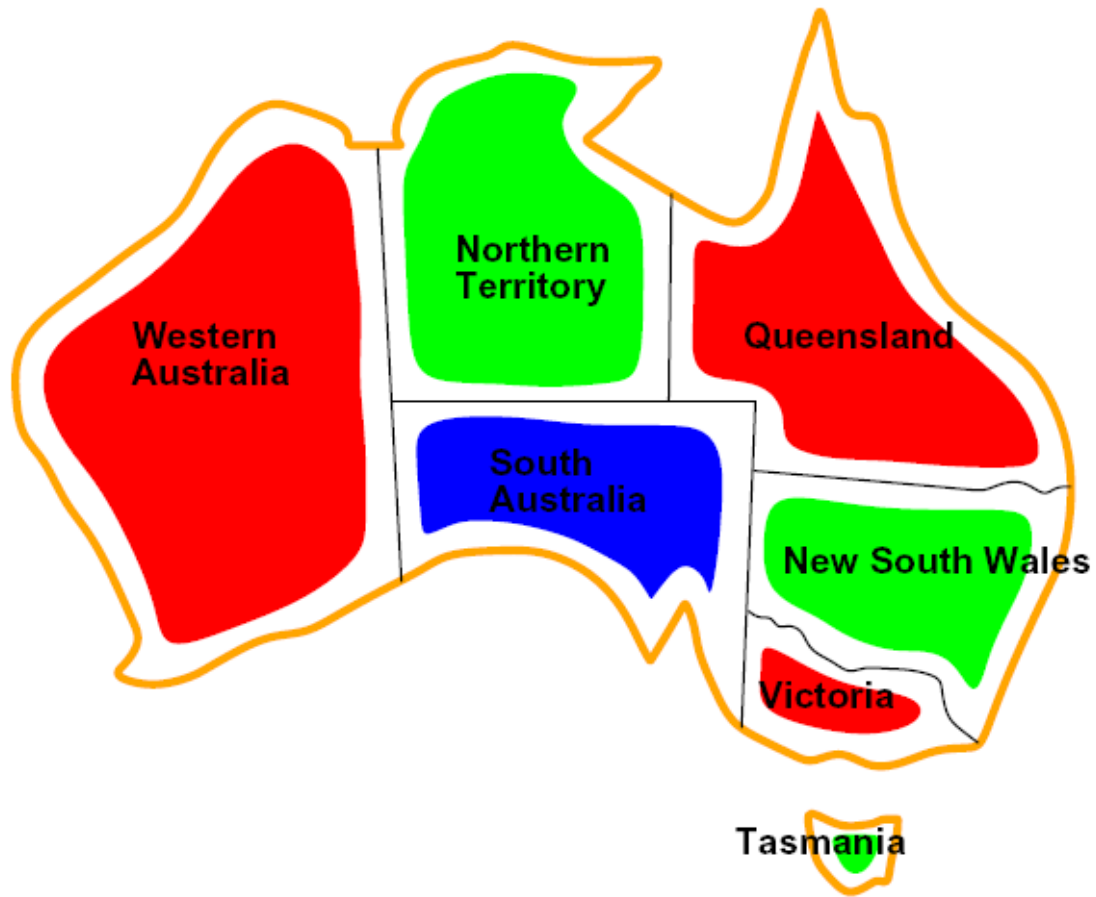
Explicit: $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), \dots\}$

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

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

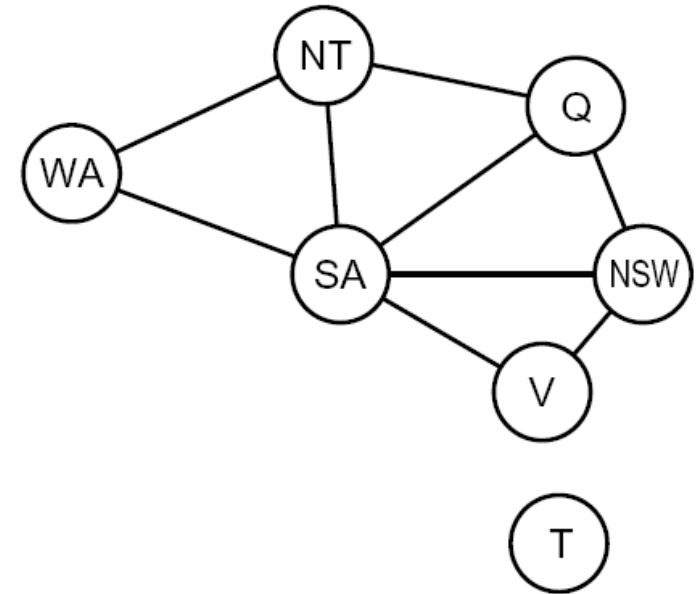


Constraint Graphs



Constraint Graphs

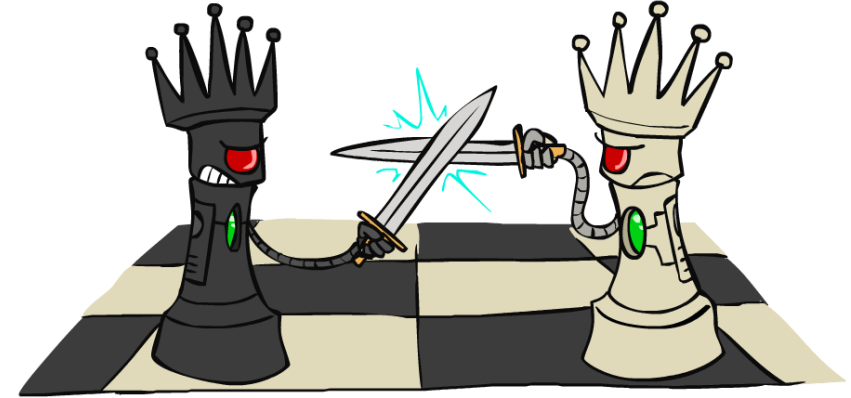
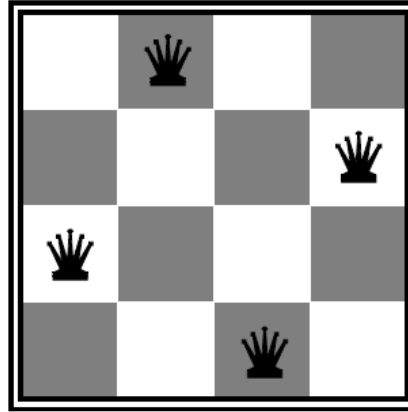
- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



Example: N-Queens

- Formulation 1:

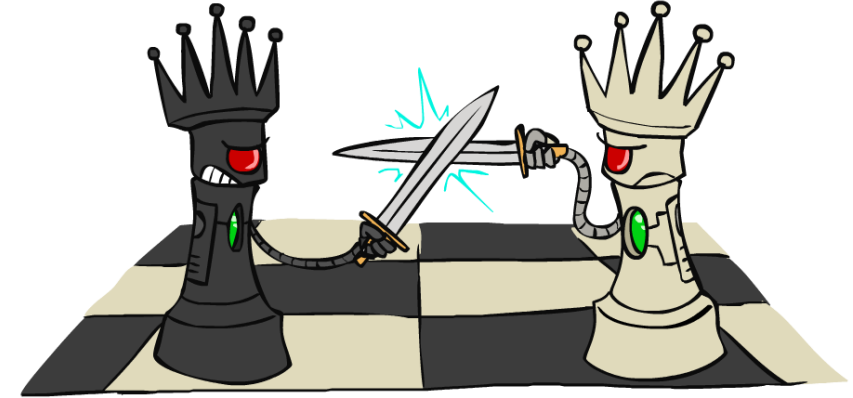
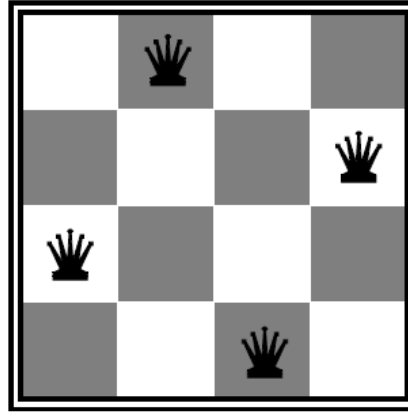
- Variables: X_{ij}
- Domains: $\{0, 1\}$
- Constraints



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

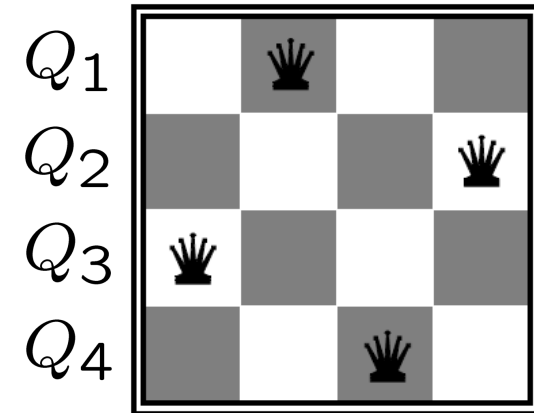
- Domains: $\{1, 2, 3, \dots, N\}$

- Constraints:

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

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

...



Example: Cryptarithmic

- Variables:

$F T U W R O X_1 X_2 X_3$

- Domains:

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

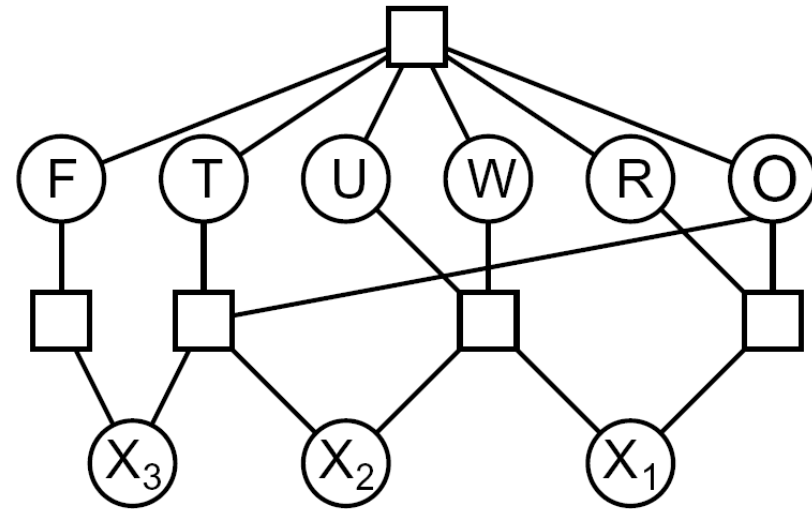
- Constraints:

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

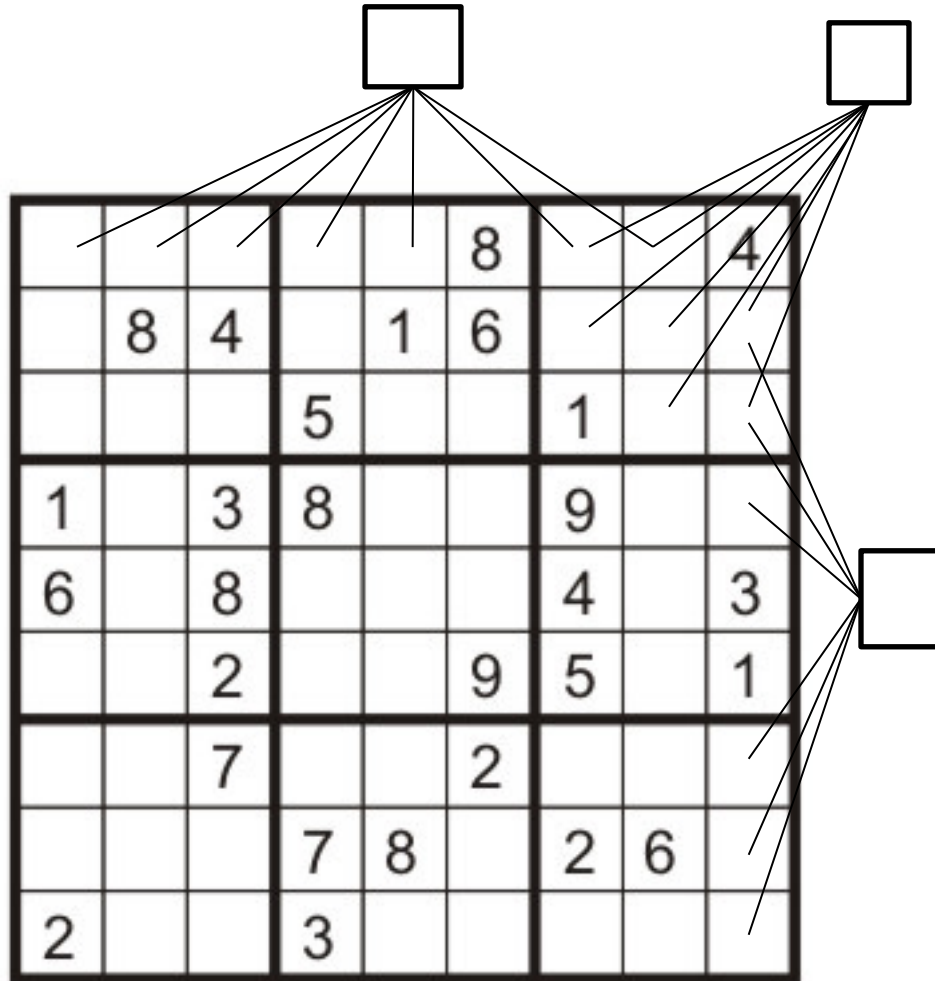
$O + O = R + 10 \cdot X_1$

\dots

	T	W	O
+	T	W	O
<hr/>			
F	O	U	R

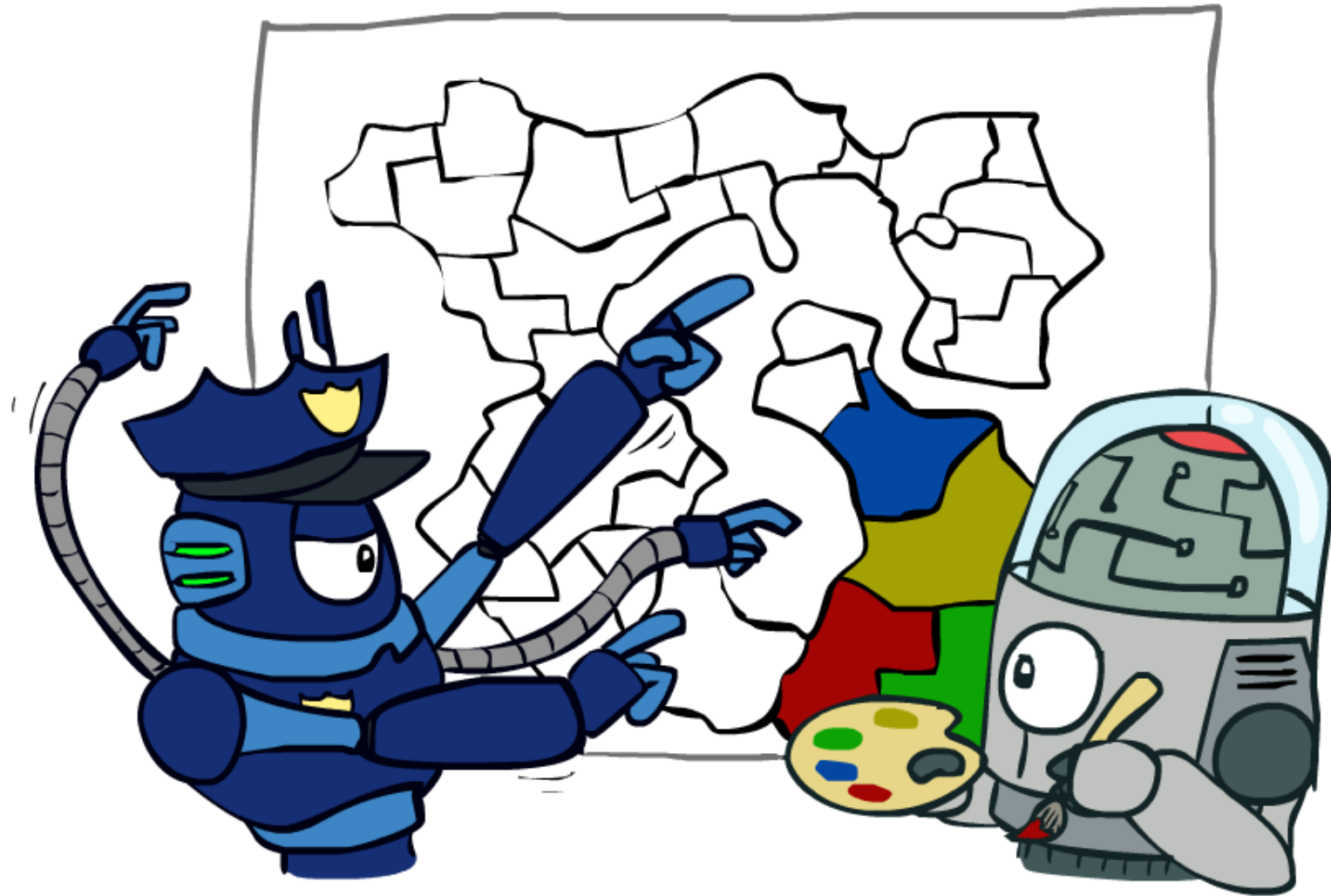


Example: Sudoku



- 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



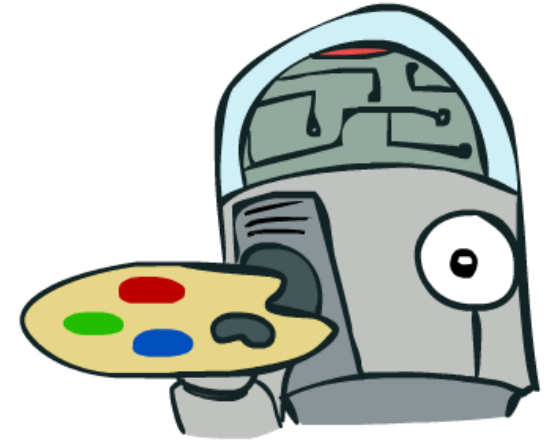
Varieties of CSPs

- 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 cs170 for a bit of this theory)



Varieties of Constraints

- Varieties of Constraints

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

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

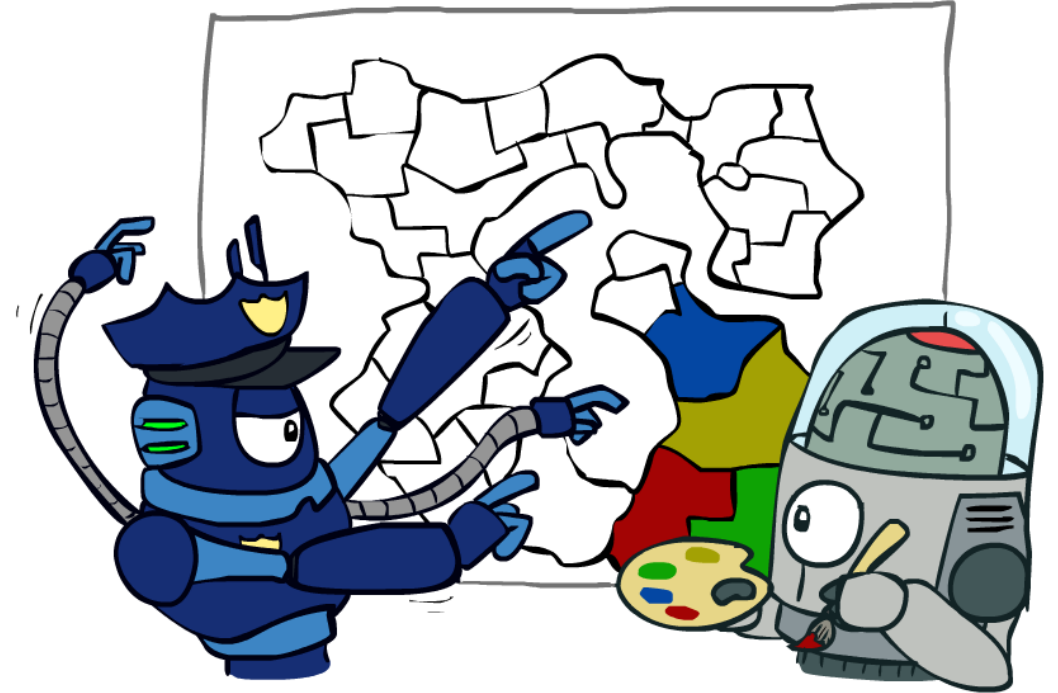
- Binary constraints involve pairs of variables, e.g.:

$$SA \neq WA$$

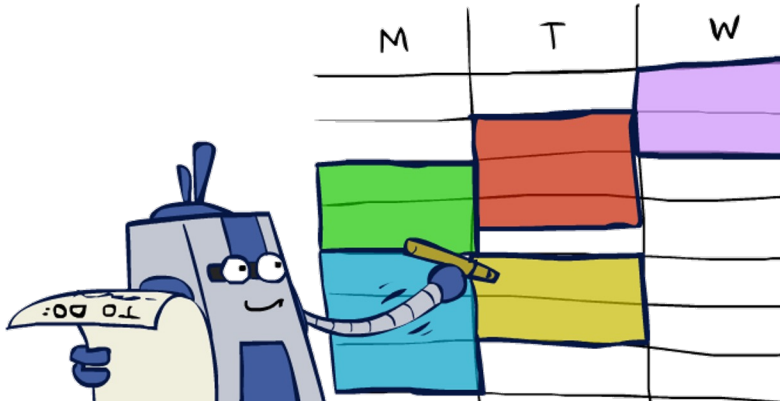
- Higher-order constraints involve 3 or more variables:
e.g., cryptarithmic column constraints

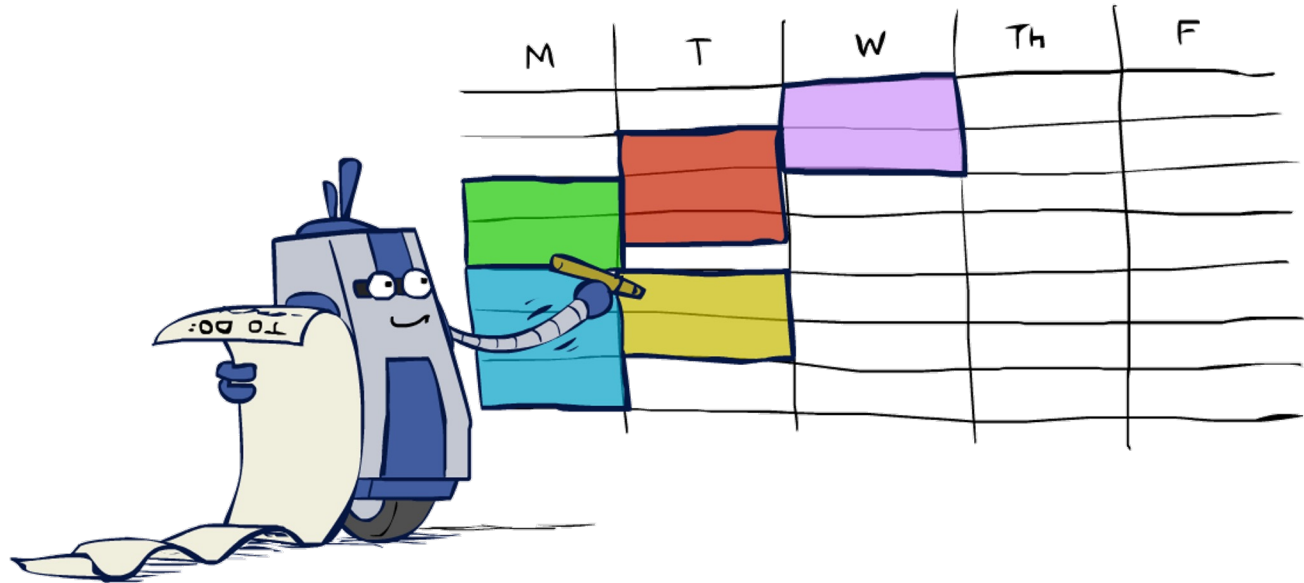
- Preferences (soft constraints):

- 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

- Scheduling problems: e.g., when can we all meet?
 - Timetabling problems: e.g., which class is offered when and where?
 - Assignment problems: e.g., who teaches what class
 - Hardware configuration
 - Transportation scheduling
 - Factory scheduling
 - Circuit layout
 - Fault diagnosis
 - ... lots more!
- 
- A cartoon robot character with a blue body and a yellow head is holding a notepad with a checklist. It is pointing at a grid that represents a timetabling problem. The grid has three columns labeled 'M', 'T', and 'W' (Monday, Tuesday, Wednesday) and several rows. The cells in the grid are colored: the top row is purple, the second row is red, the third row is green, and the fourth row is yellow. The robot is holding a yellow notepad with a checklist that includes 'To Do' and 'Done'.



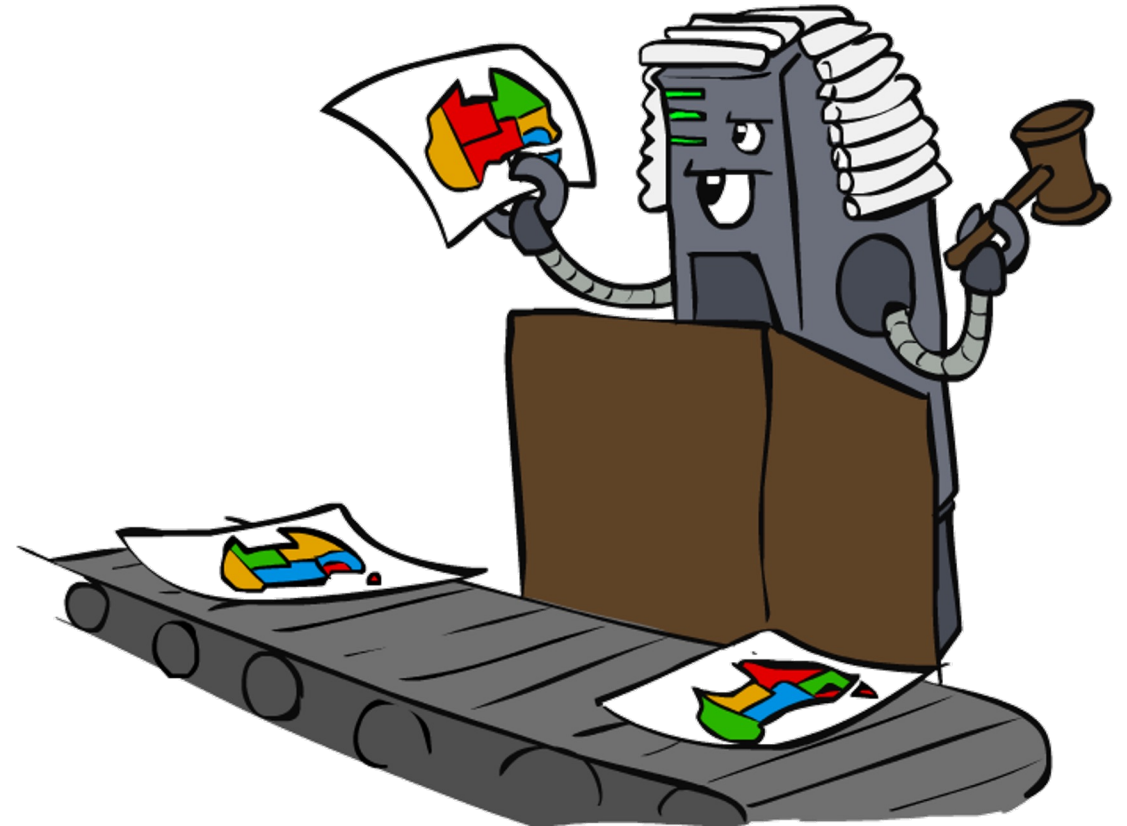
- Many real-world problems involve real-valued variables...

Solving CSPs



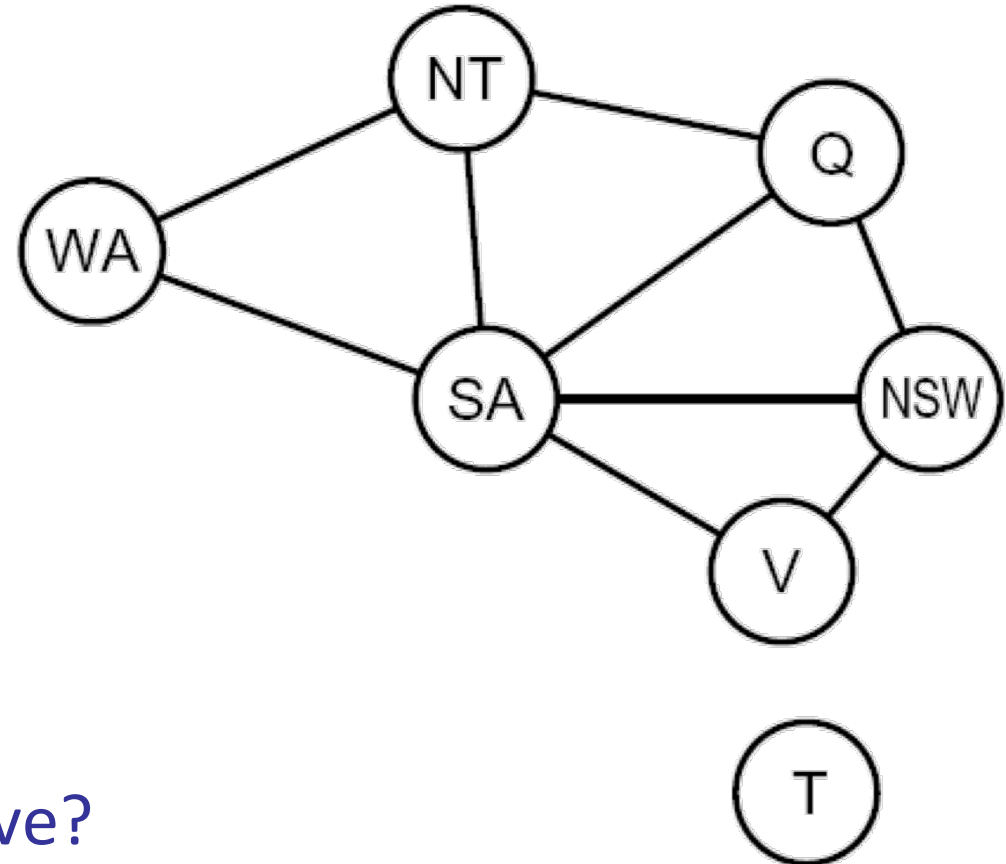
Standard Search Formulation

- 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

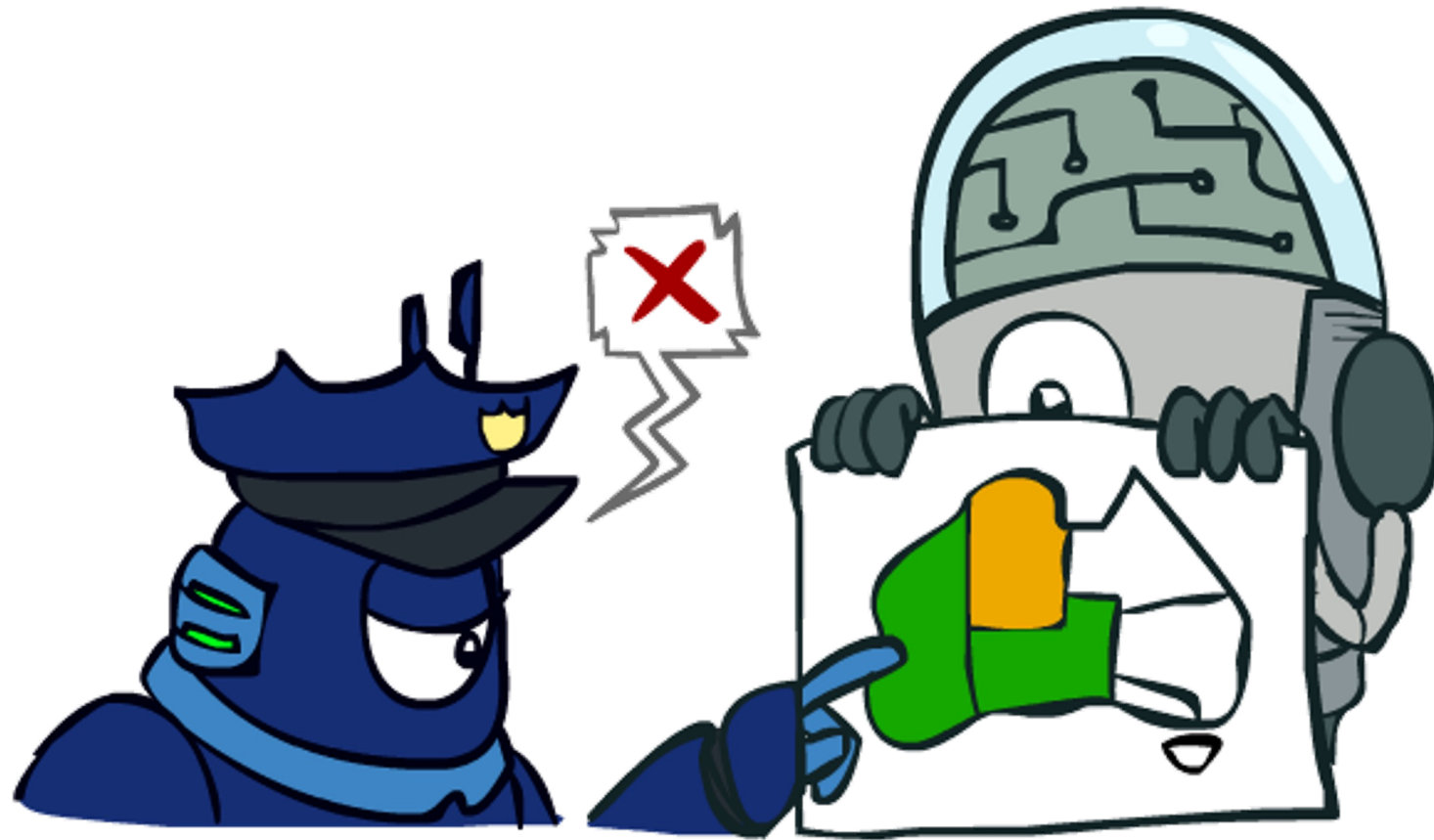


Search Methods

- What would BFS do?
- What would DFS do?
- What problems does naïve search have?

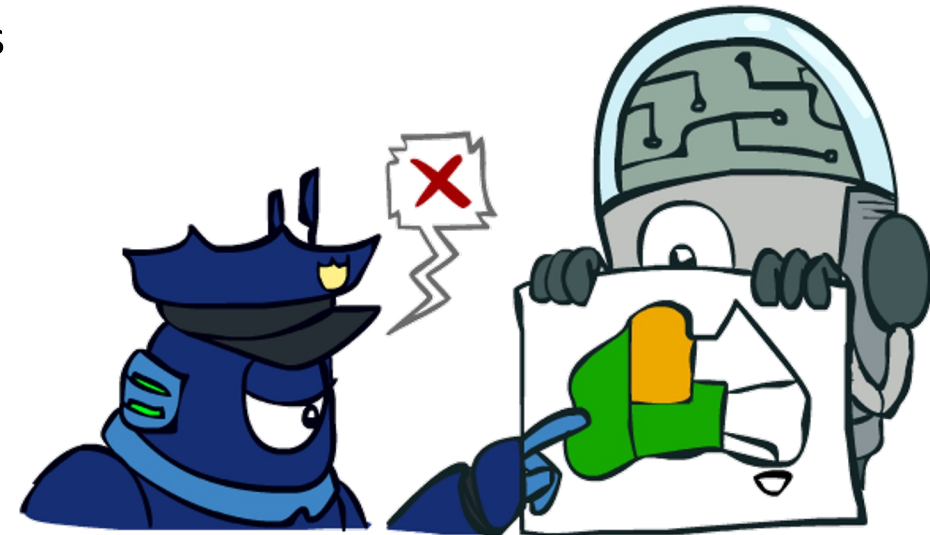


Backtracking Search

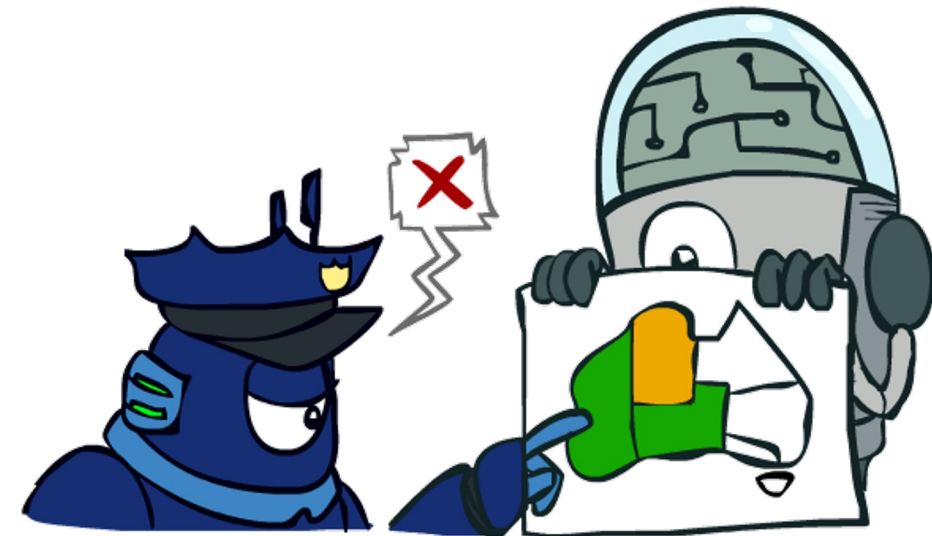
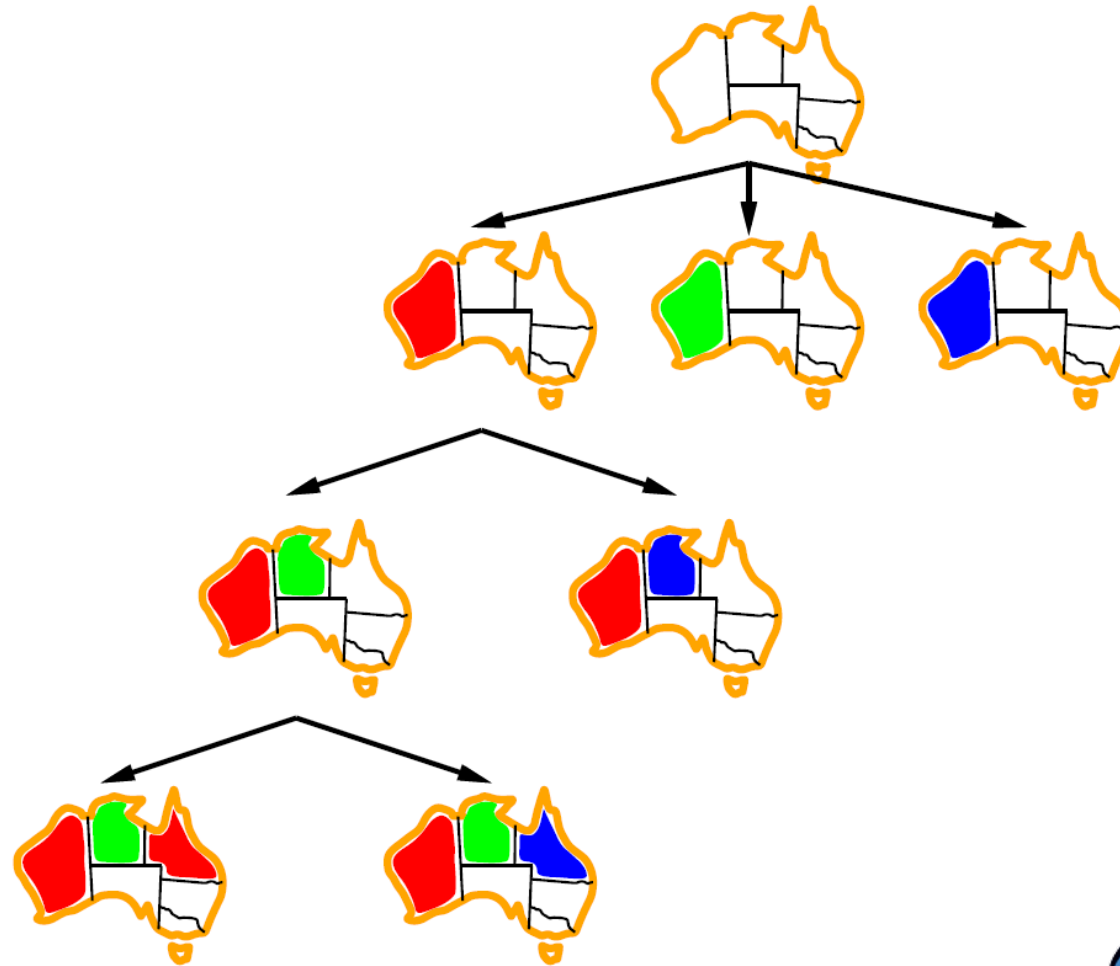


Backtracking Search

- 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 with 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

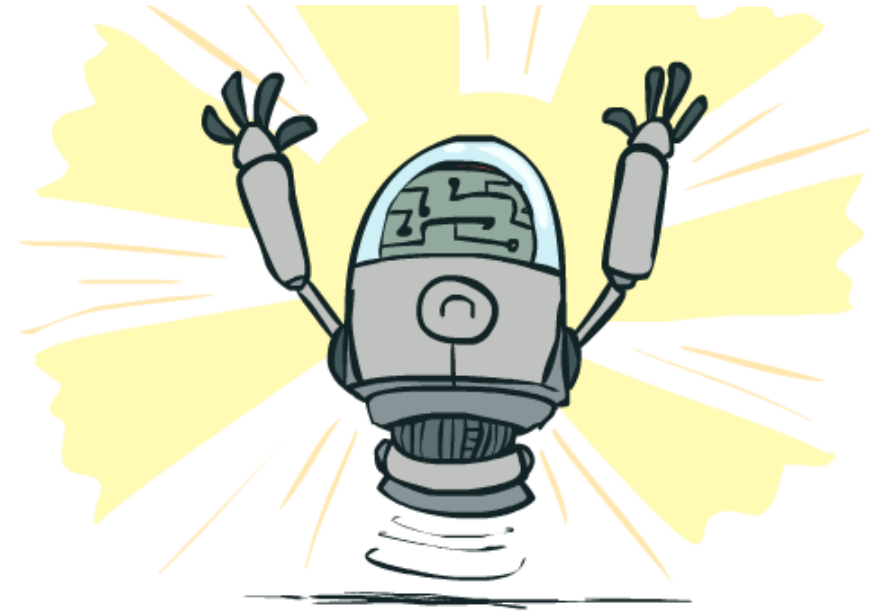
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← 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 ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

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

Improving Backtracking

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

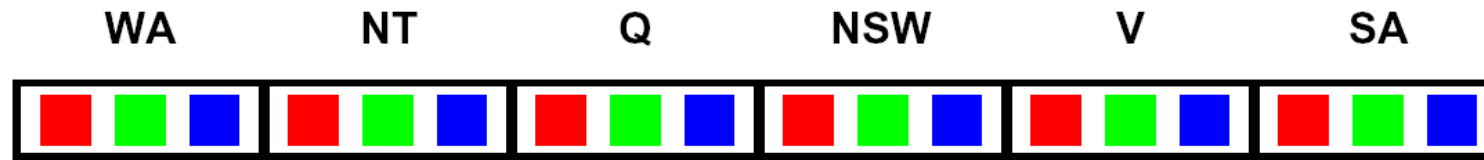


Filtering



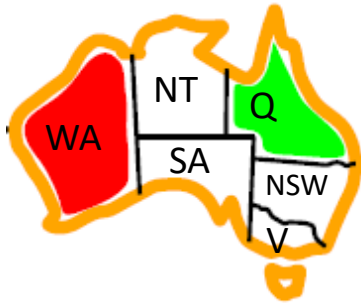
Filtering: Forward Checking

- 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



Filtering: Constraint Propagation

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

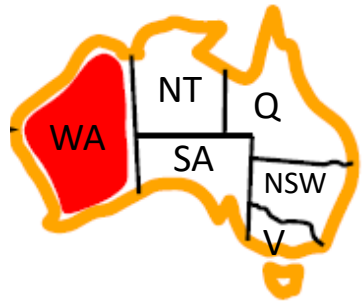


WA	NT	Q	NSW	V	SA
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

- 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

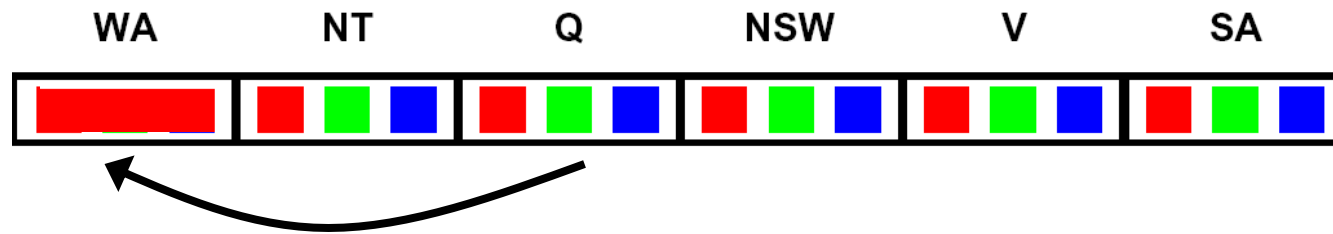
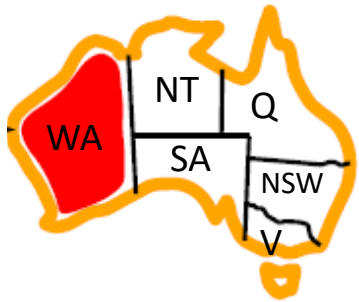
- An arc $X \rightarrow Y$ is **consistent** iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint



- Tail = NT, head = WA
 - If NT = blue: we could assign WA = red
 - If NT = green: we could assign WA = red
 - If NT = red: there is no remaining assignment to WA that we can use
 - Deleting NT = red from the tail makes this arc consistent

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

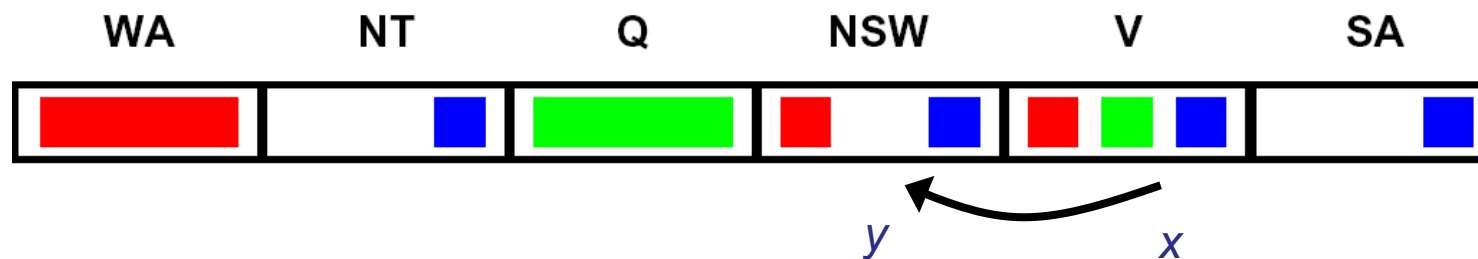
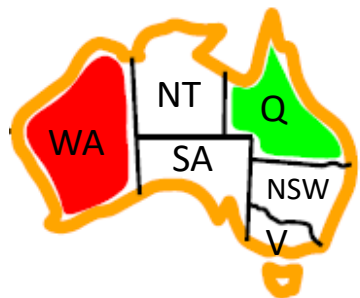


*Remember:
Delete from
the tail!*

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

Arc Consistency of an Entire CSP (1/6)

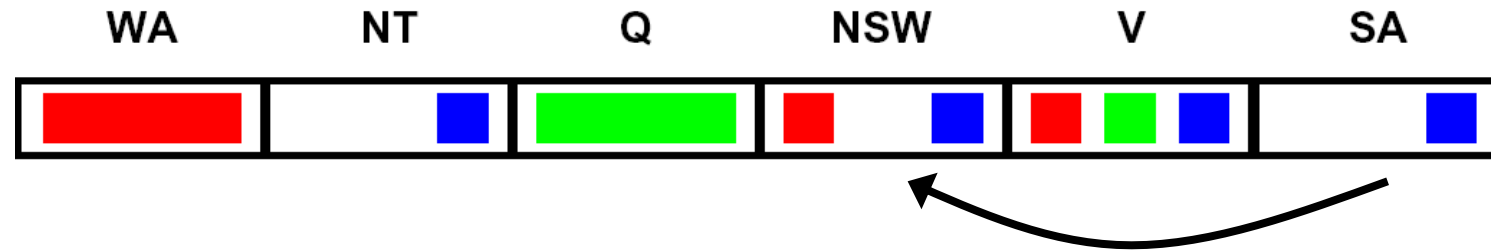
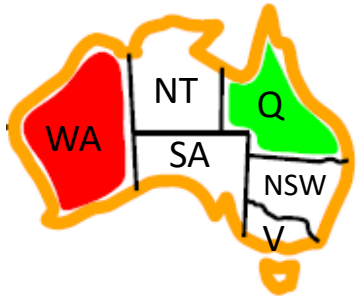
- A simple form of propagation makes sure **all** arcs are consistent:



- Arc V to NSW is consistent: for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint

Arc Consistency of an Entire CSP (2/6)

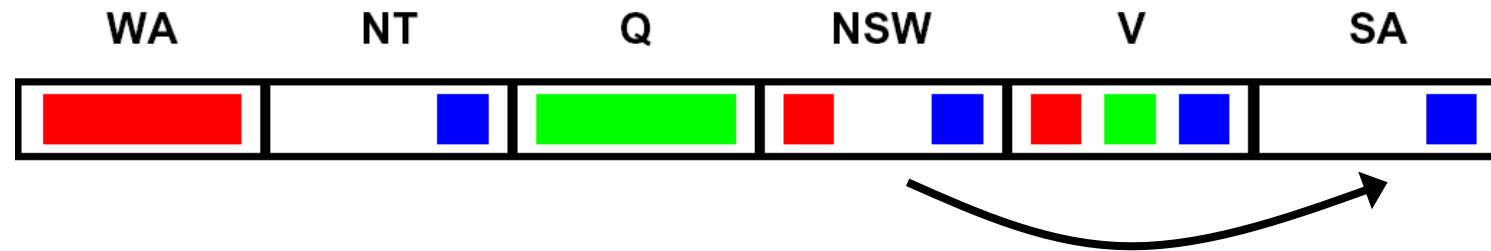
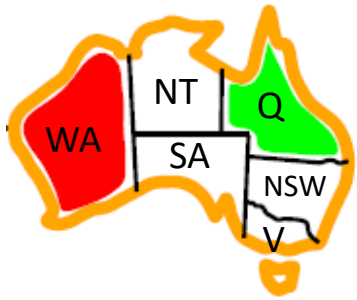
- A simple form of propagation makes sure **all** arcs are consistent:



- Arc SA to NSW is consistent: for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint

Arc Consistency of an Entire CSP (3/6)

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

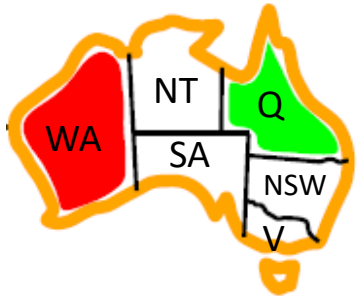


- Arc NSW to SA is not consistent: if we assign NSW = blue, there is no valid assignment left for SA
- To make this arc consistent, we delete NSW = blue from the tail

*Remember:
Delete from
the tail!*

Arc Consistency of an Entire CSP (4/6)

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

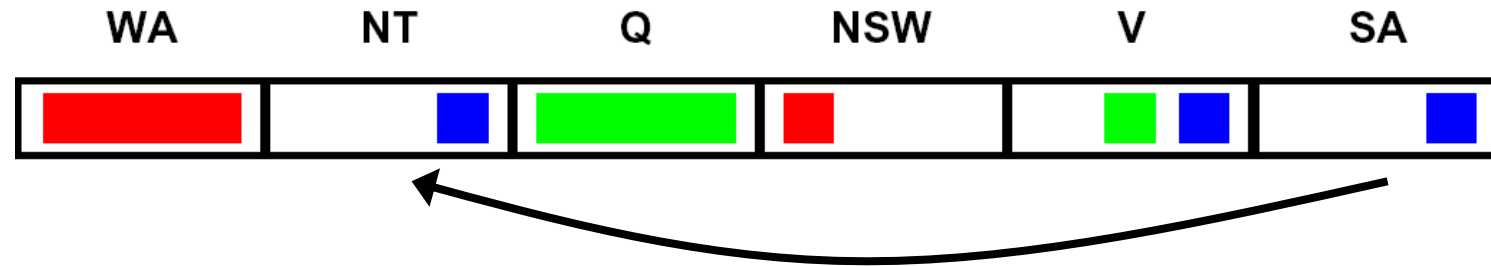
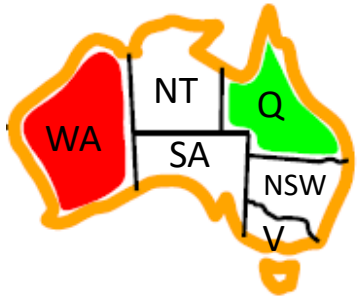


- Remember that arc V to NSW was consistent, when NSW had red and blue in its domain
- After removing blue from NSW, this arc might not be consistent anymore! We need to recheck this arc.
- Important: If X loses a value, neighbors of X need to be rechecked!

*Remember:
Delete from
the tail!*

Arc Consistency of an Entire CSP (5/6)

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

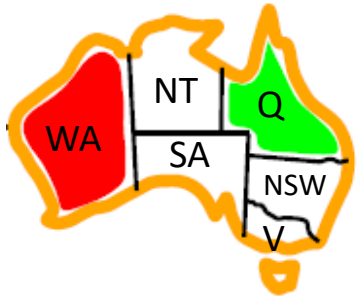


- Arc SA to NT is inconsistent. We make it consistent by deleting from the tail (SA = blue).

*Remember:
Delete from
the tail!*

Arc Consistency of an Entire CSP (6/6)

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



- SA has an empty domain, so we detect failure. There is no way to solve this CSP with WA = red and Q = green, so we backtrack.
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

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



---

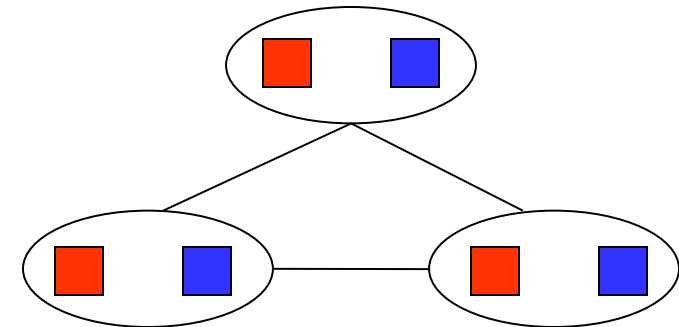
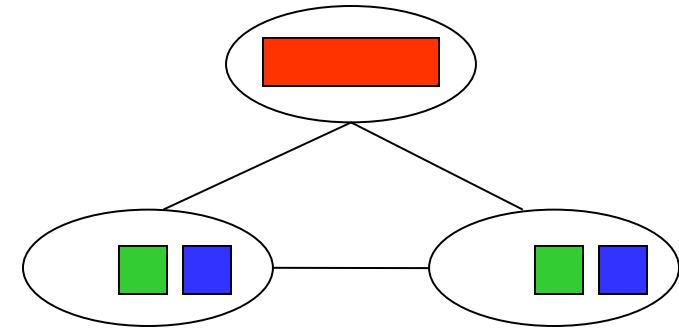


function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
    removed  $\leftarrow$  false
    for each  $x$  in DOMAIN[ $X_i$ ] do
        if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
            then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow$  true
    return removed
```

- 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?*

[Demo: coloring -- forward checking]

[Demo: coloring -- arc consistency]

Can AI solve Sudoku? *

- For small sizes yes: 9x9 puzzle CSPs are solved with DFS
- LLMs solve 50% of 5x5 puzzles
- LLMs + Search solve 80% of 5x5 puzzles
 - ...and 14% of 9x9 puzzles 😞
- Still looking for best ways to combine LLMs + Search!

For more detail see:

- [Large Language Model Guided Tree-of-Thought](#), Long, 2023
- [LLM Prompting and Classical AI: Budding Romance or a Tango with Two Left Feet?](#) Arkoudas, 2023