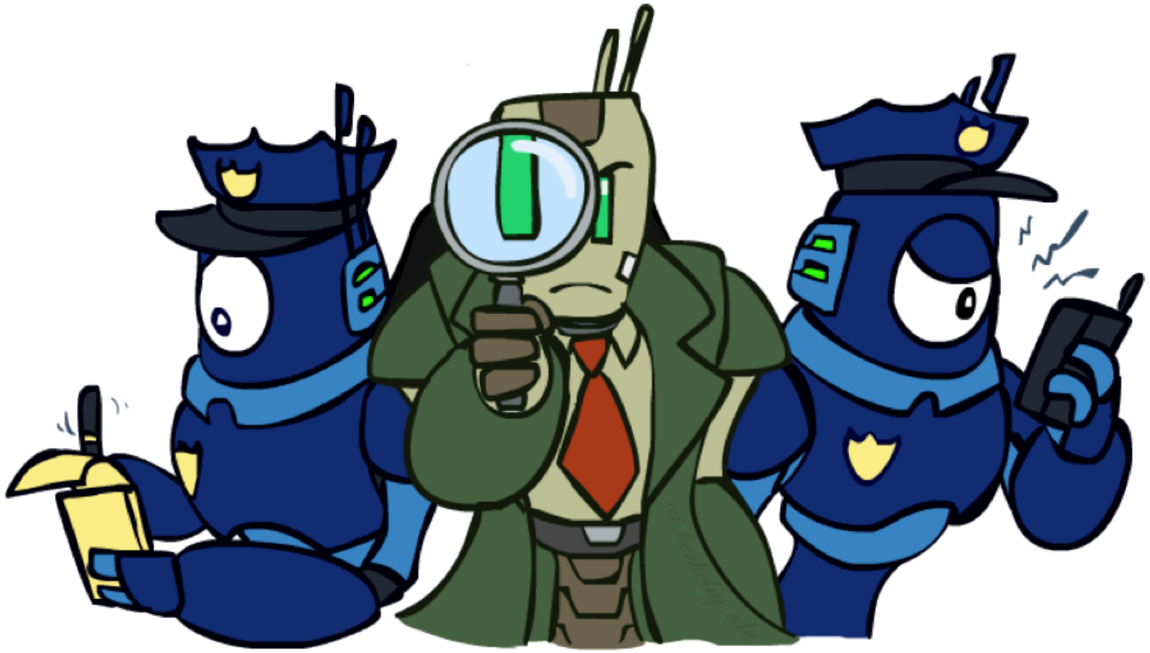
CS 188: Artificial Intelligence

Constraint Satisfaction Problems



Instructor: Anca Dragan

University of California, Berkeley

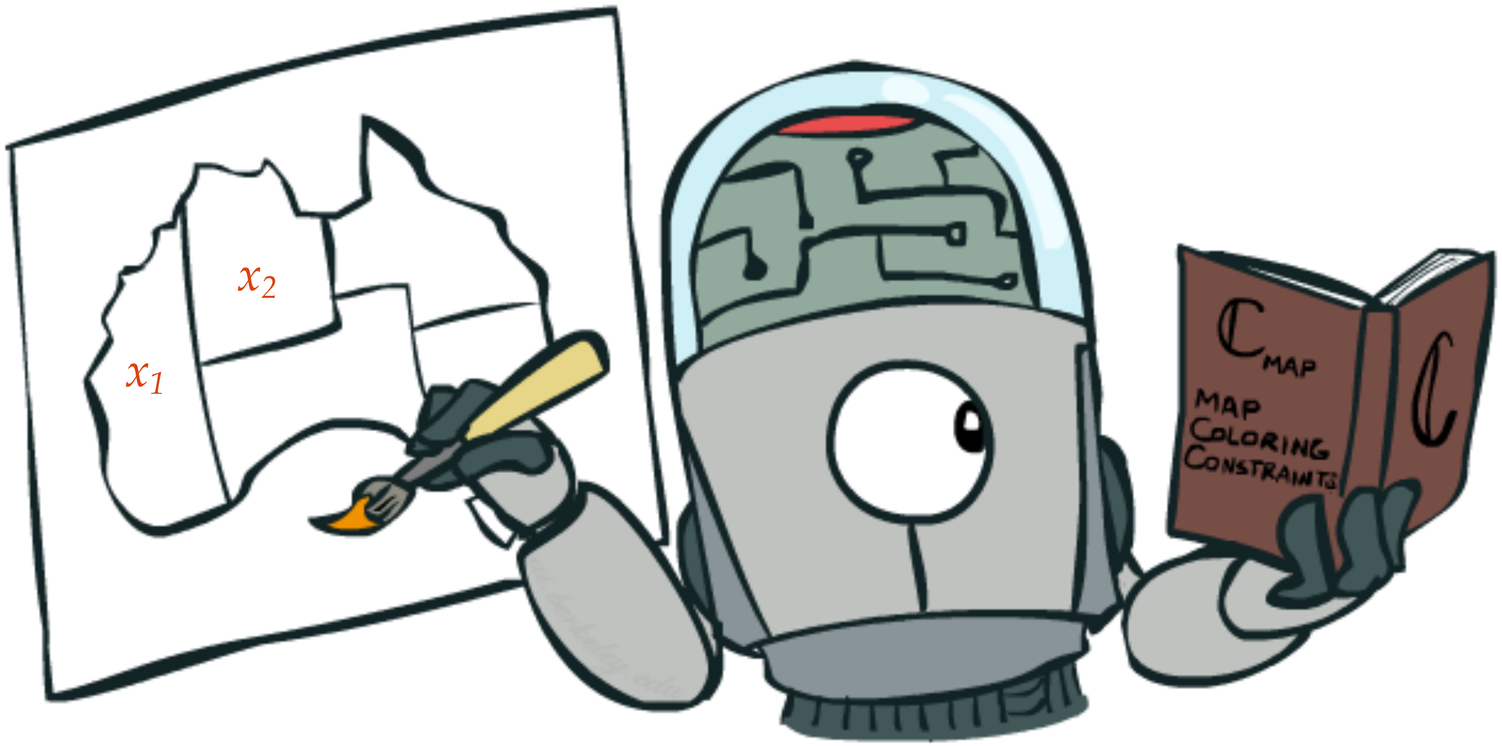
[These slides adapted from Dan Klein and Pieter Abbeel]

Constraint Satisfaction Problems

N variables

domain D

constraints



states

partial assignment

goal test

complete; satisfies constraints

successor function

assign an unassigned variable

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

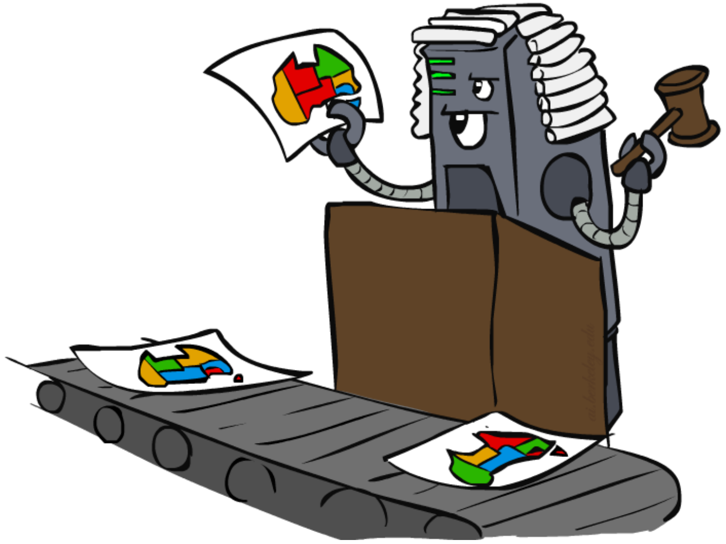
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

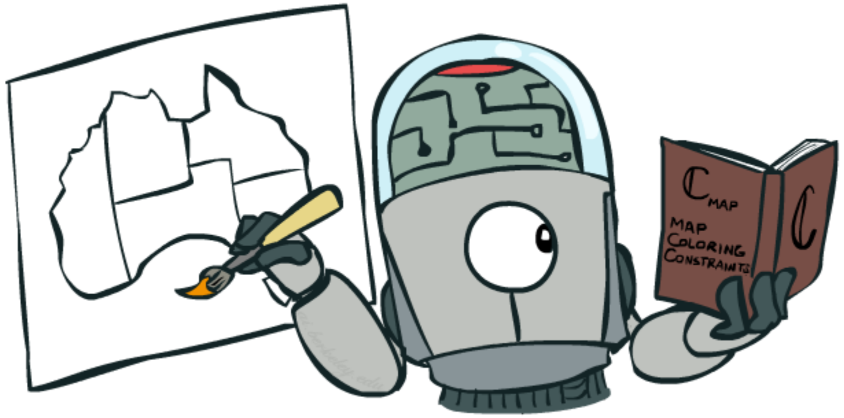
Constraint Satisfaction Problems

- 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

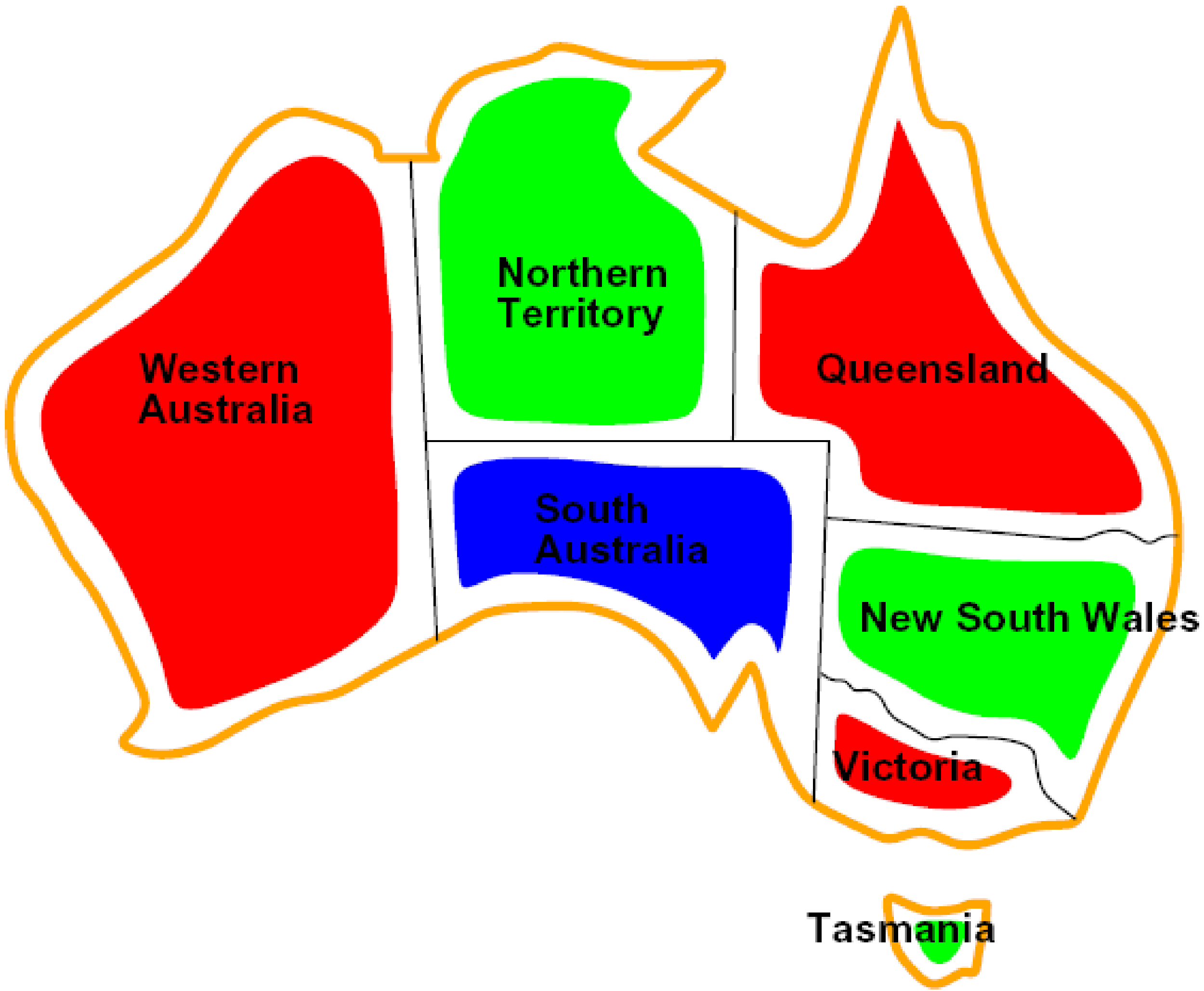
- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

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





CSP Examples



Example: Map Coloring

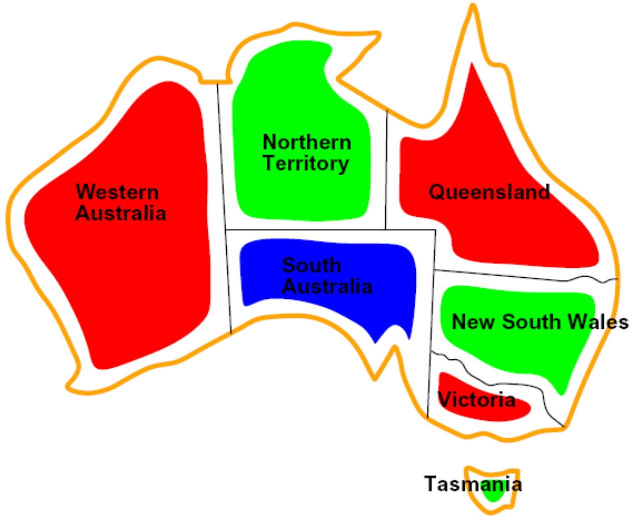
- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D = \{\text{red}, \text{green}, \text{blue}\}$
- Constraints: adjacent regions must have different colors

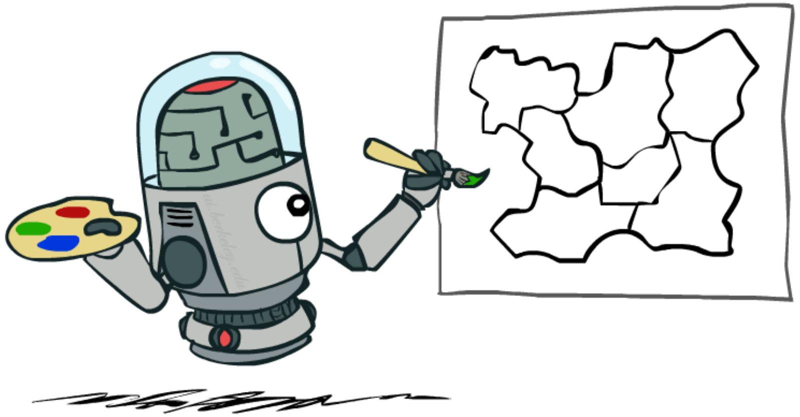
  Implicit: $\text{WA} \neq \text{NT}$

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

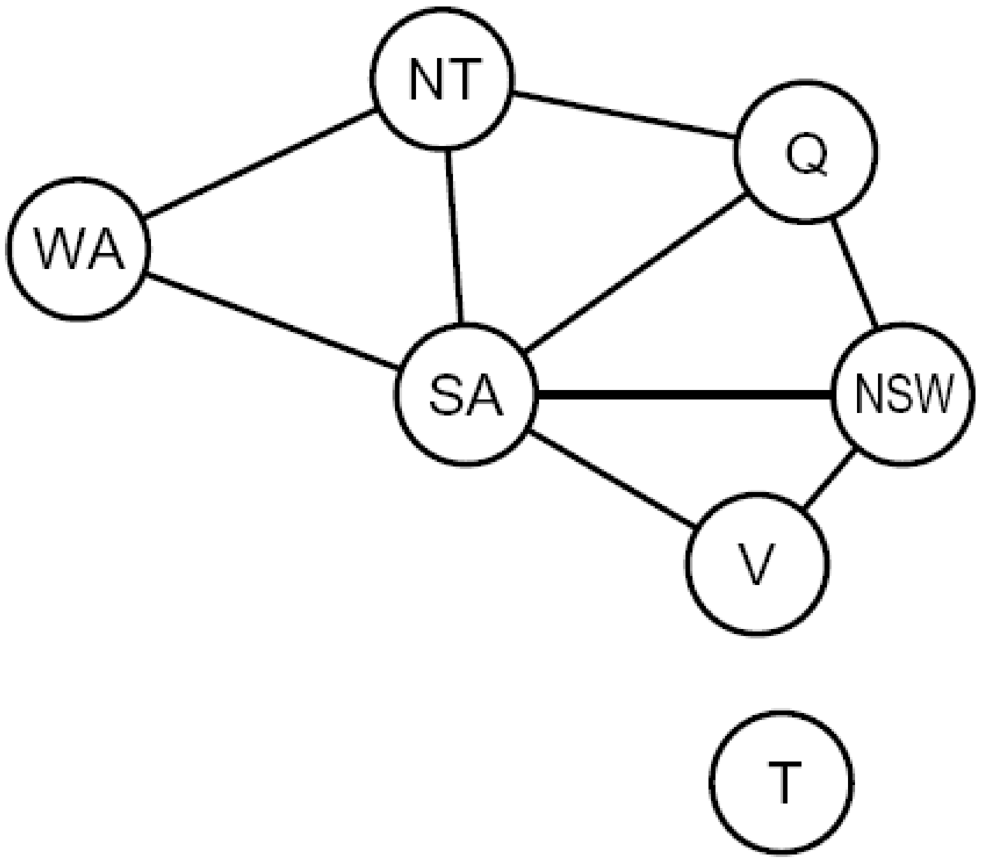
- Solutions are assignments satisfying all constraints, e.g.:

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



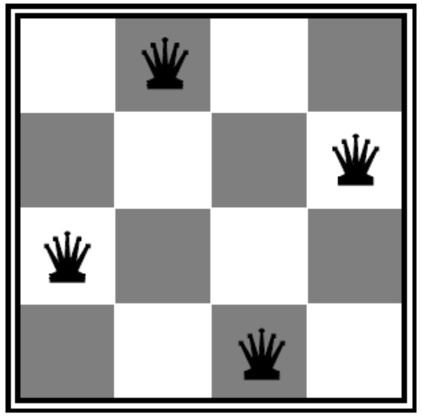


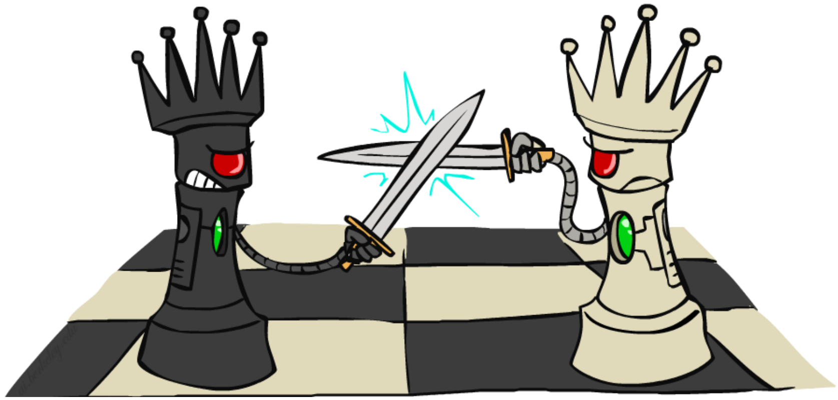
Constraint Graphs



Example: N-Queens

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



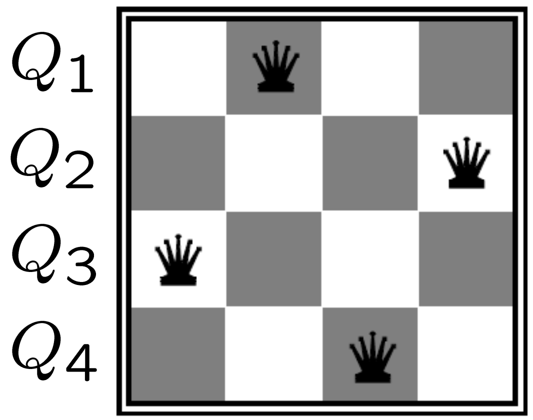


$$\forall i, j, k \;\; (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

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

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

$\cdots$

Example: Cryptarithmetic

- Variables:

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

- Domains:

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

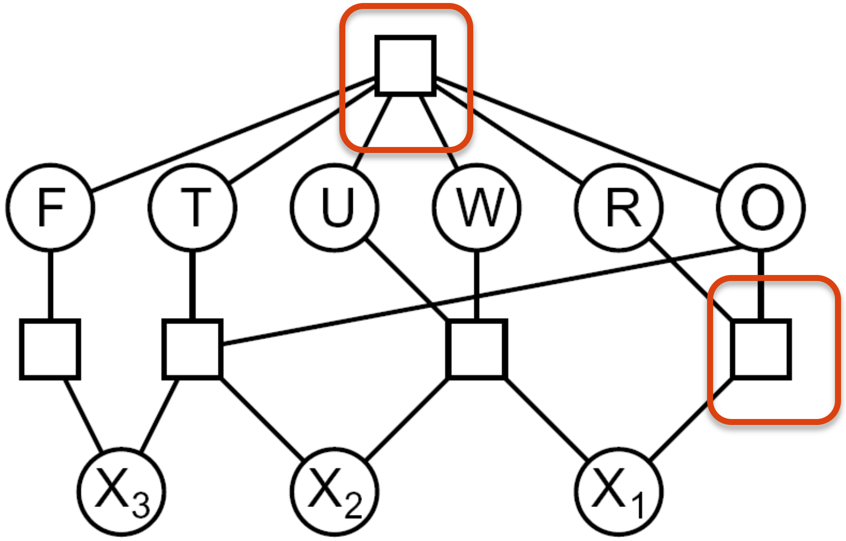
- Constraints:

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

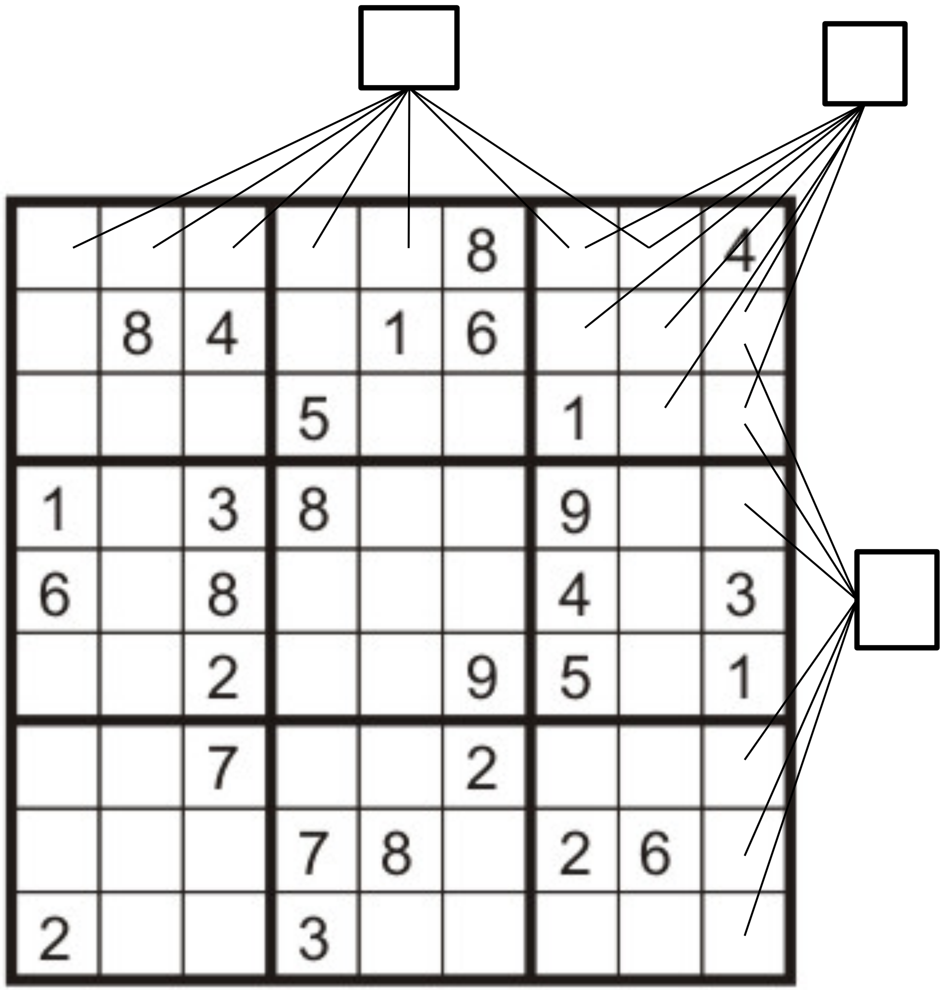
  $O + O = R + 10 \cdot X_1$

  $\cdots$

$$
\begin{array}{cccc}
 & & & X_1 \\
 & T & W & O \\
+ & T & W & O \\
\hline
F & O & U & R
\end{array}
$$



Example: Sudoku



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

9-way alldiff for each column

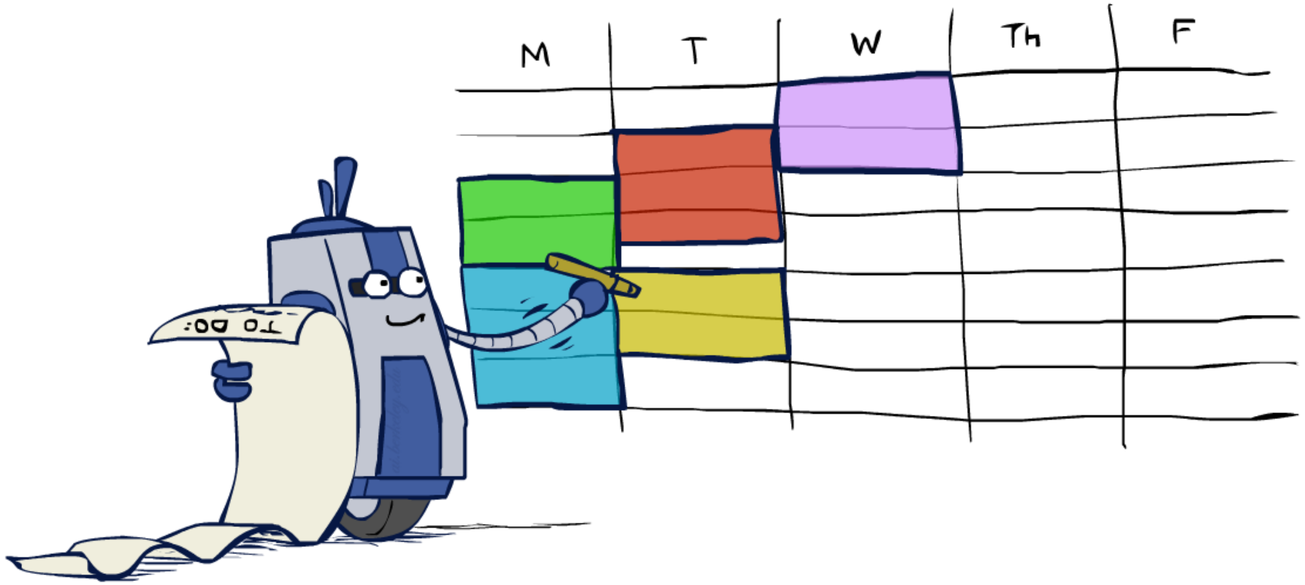
9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

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!

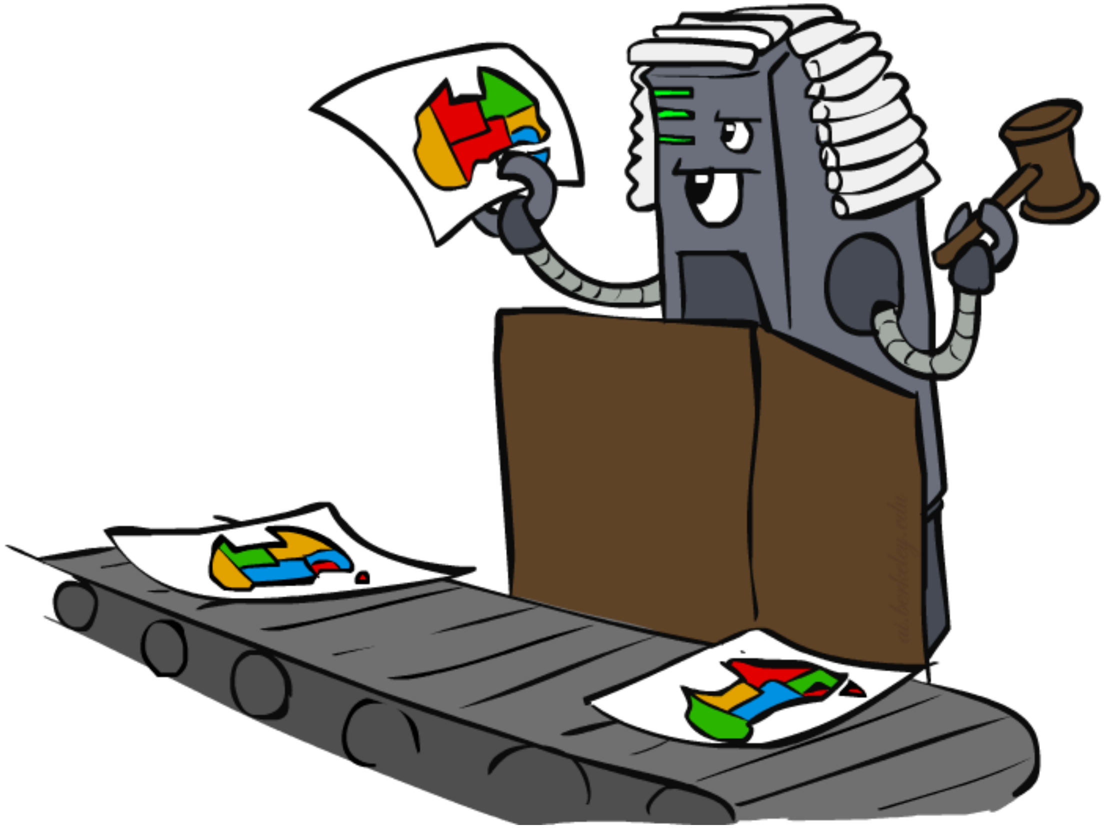


- Many real-world problems involve real-valued variables…

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

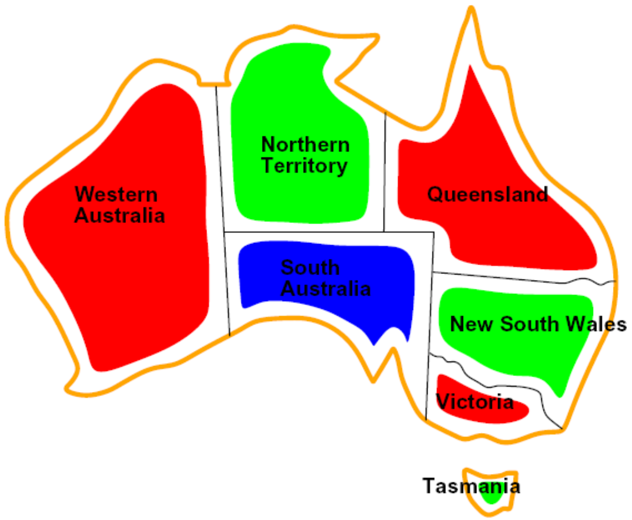


Search Methods

- What would BFS do?

$\{\}$

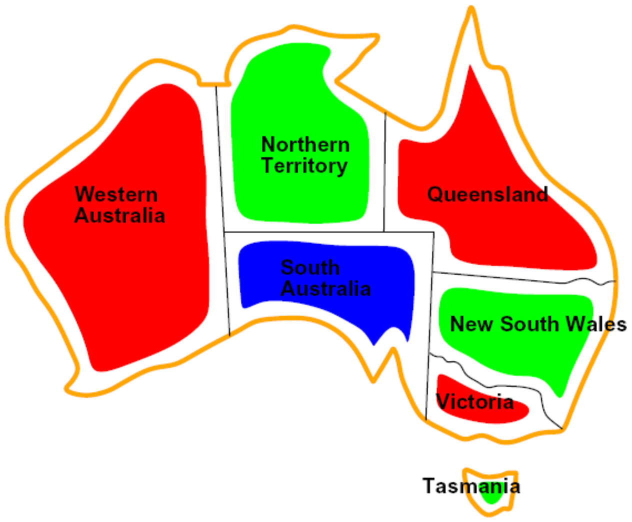
$\{WA=g\}$ $\{WA=r\}$ … $\{NT=g\}$ …



[Demo: coloring -- dfs]

Search Methods

- What would BFS do?



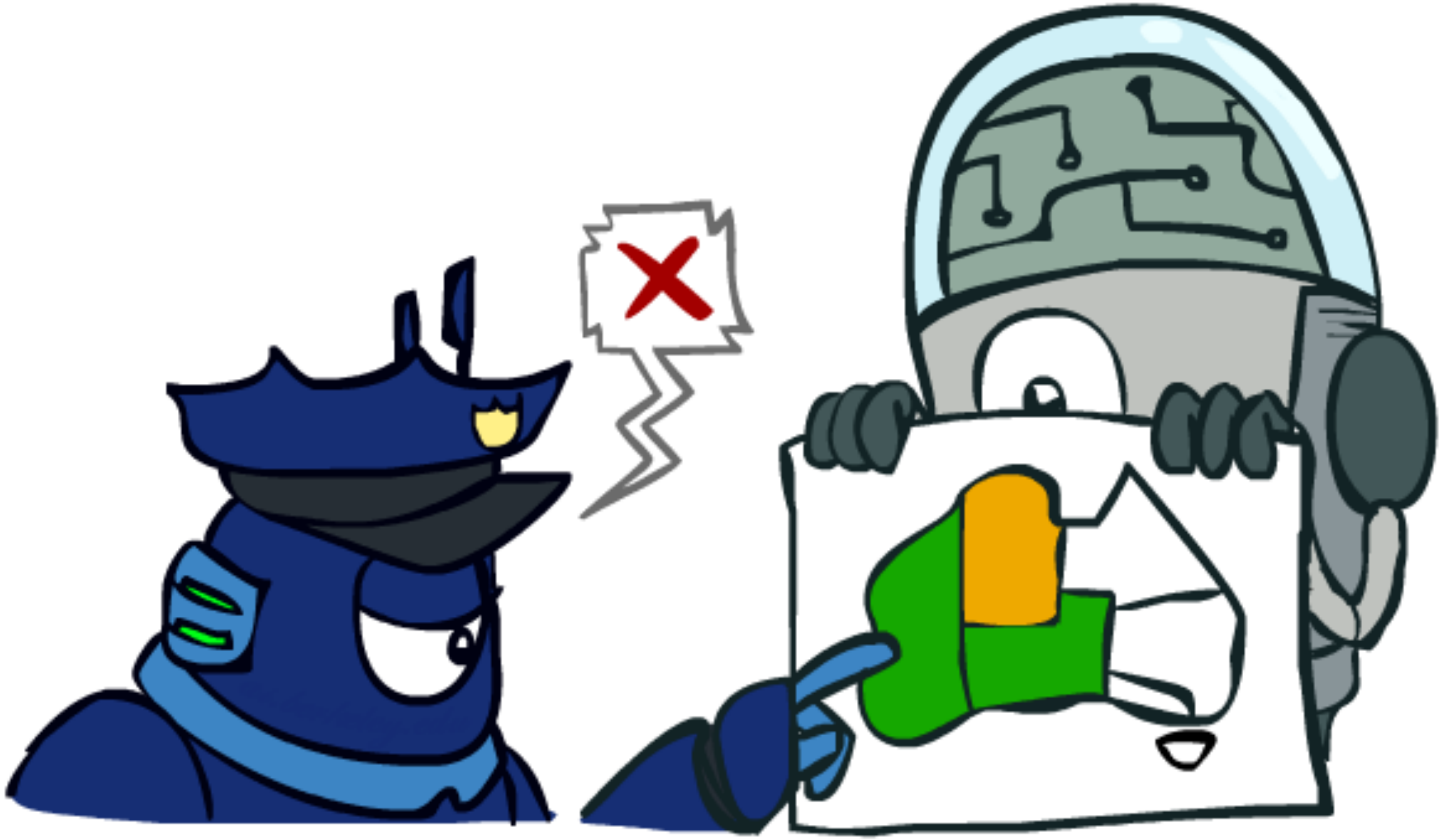
- What would DFS do?
  - let's see!

- What problems does naïve search have?

[Demo: coloring -- dfs]

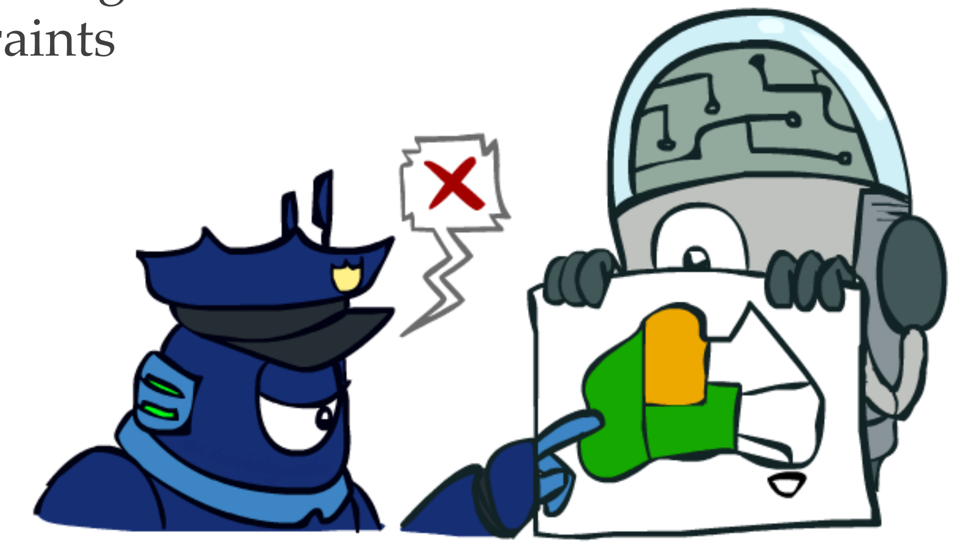
Video of Demo Coloring -- DFS

Backtracking Search

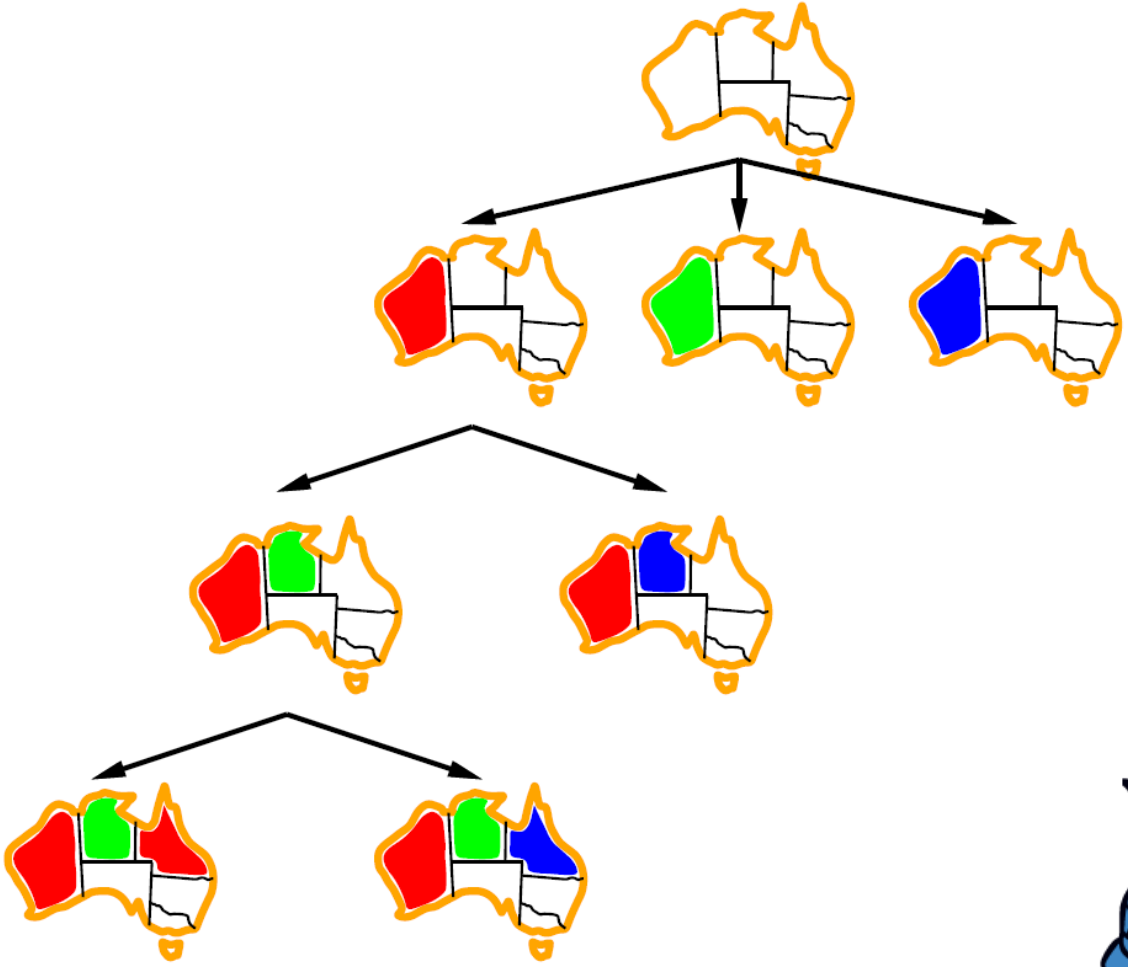


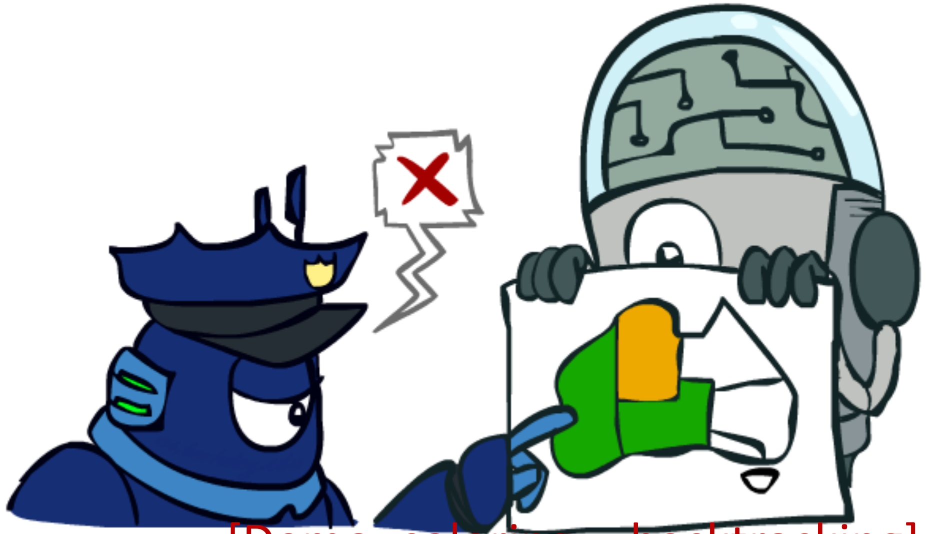
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering -> better branching factor!
  - 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





[Demo: coloring -- backtracking]

Video of Demo Coloring – Backtracking

Backtracking Search

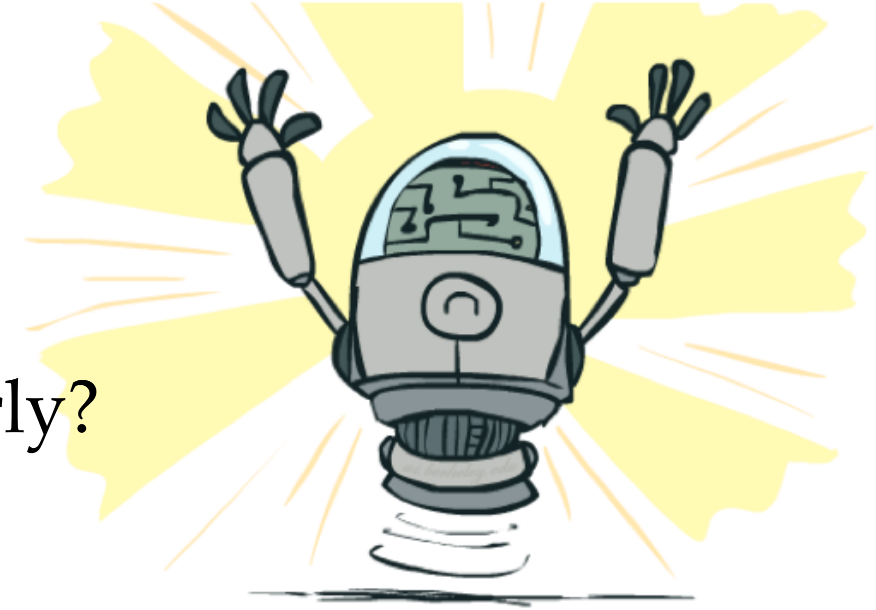
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?

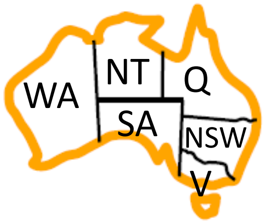


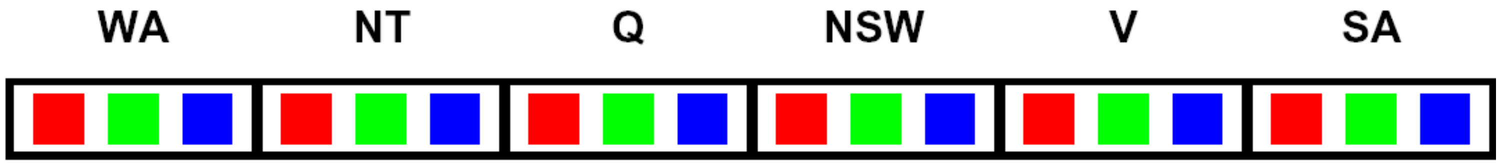
Filtering

Keep track of domains for unassigned variables and cross off bad options

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



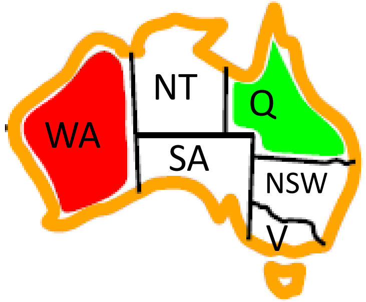


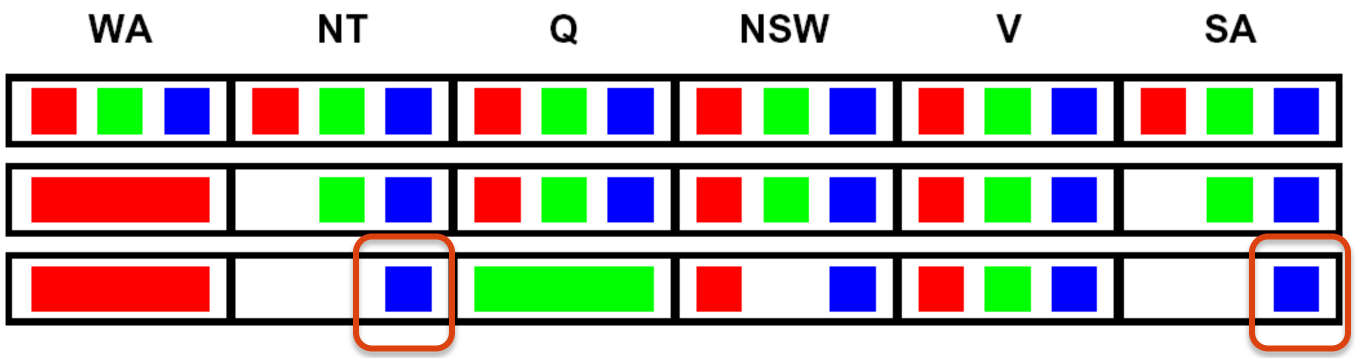
[Demo: coloring -- forward checking]

Video of Demo Coloring – Backtracking with Forward Checking

Filtering: Constraint Propagation

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

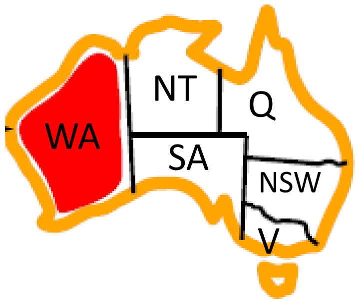


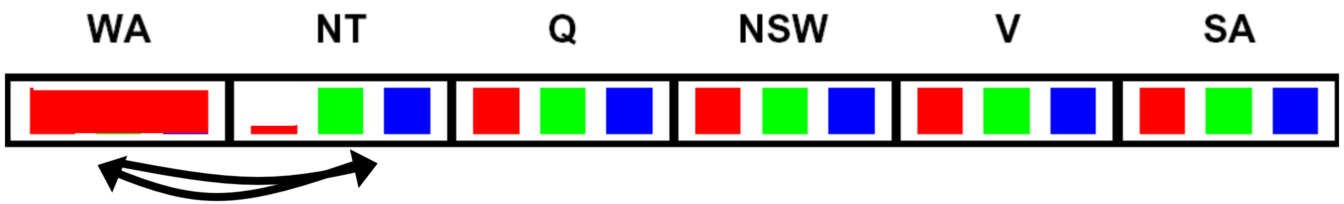


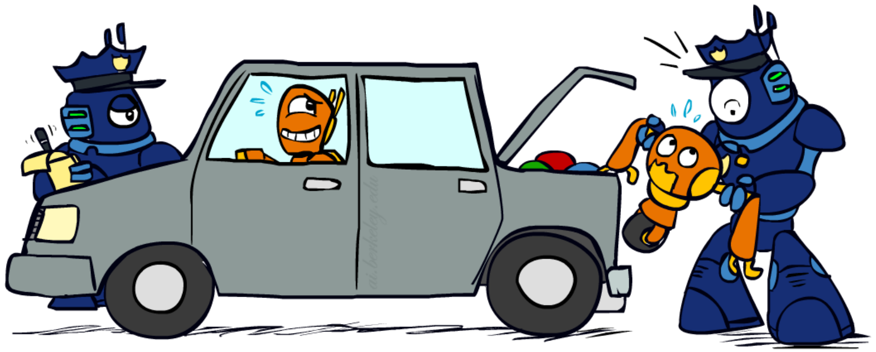
- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

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







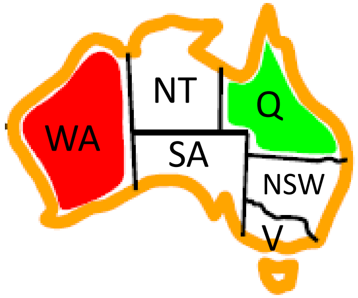
Delete from the tail!

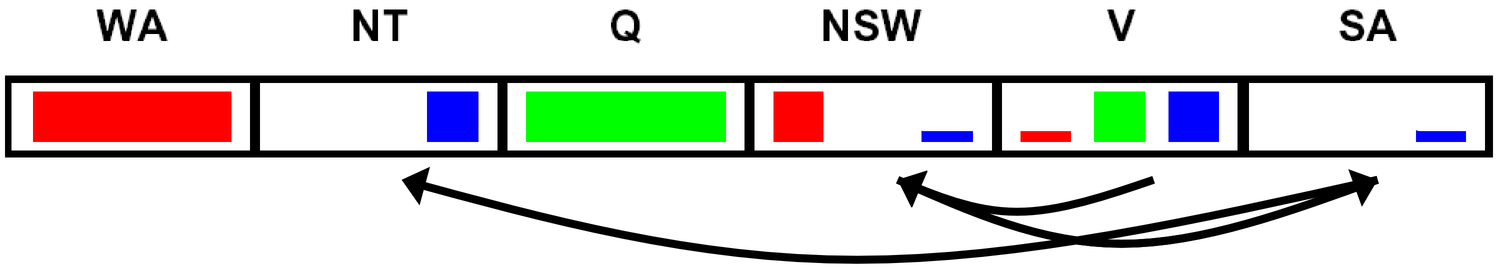
Forward checking?

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

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

---

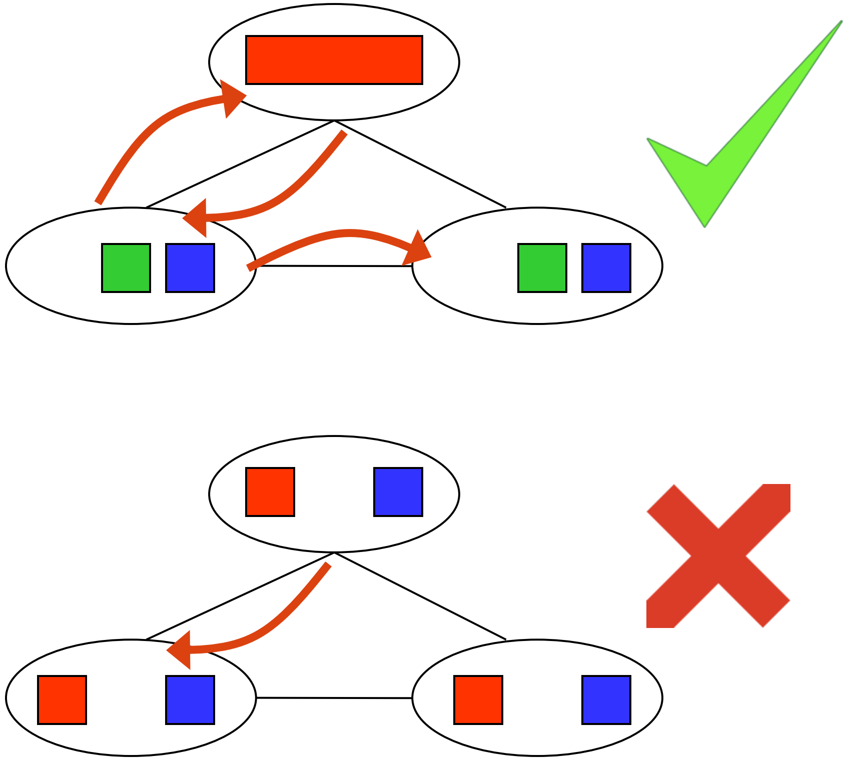
function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds
  $removed \leftarrow false$
  for each x **in** DOMAIN$[X_i]$ **do**
    if no value y in DOMAIN$[X_j]$ allows (x,y) to satisfy the constraint $X_i \leftrightarrow X_j$
      then delete x from DOMAIN$[X_i]$; $removed \leftarrow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



[Demo: coloring -- forward checking]
[Demo: coloring -- arc consistency]

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

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