Constraint Satisfaction Problems

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

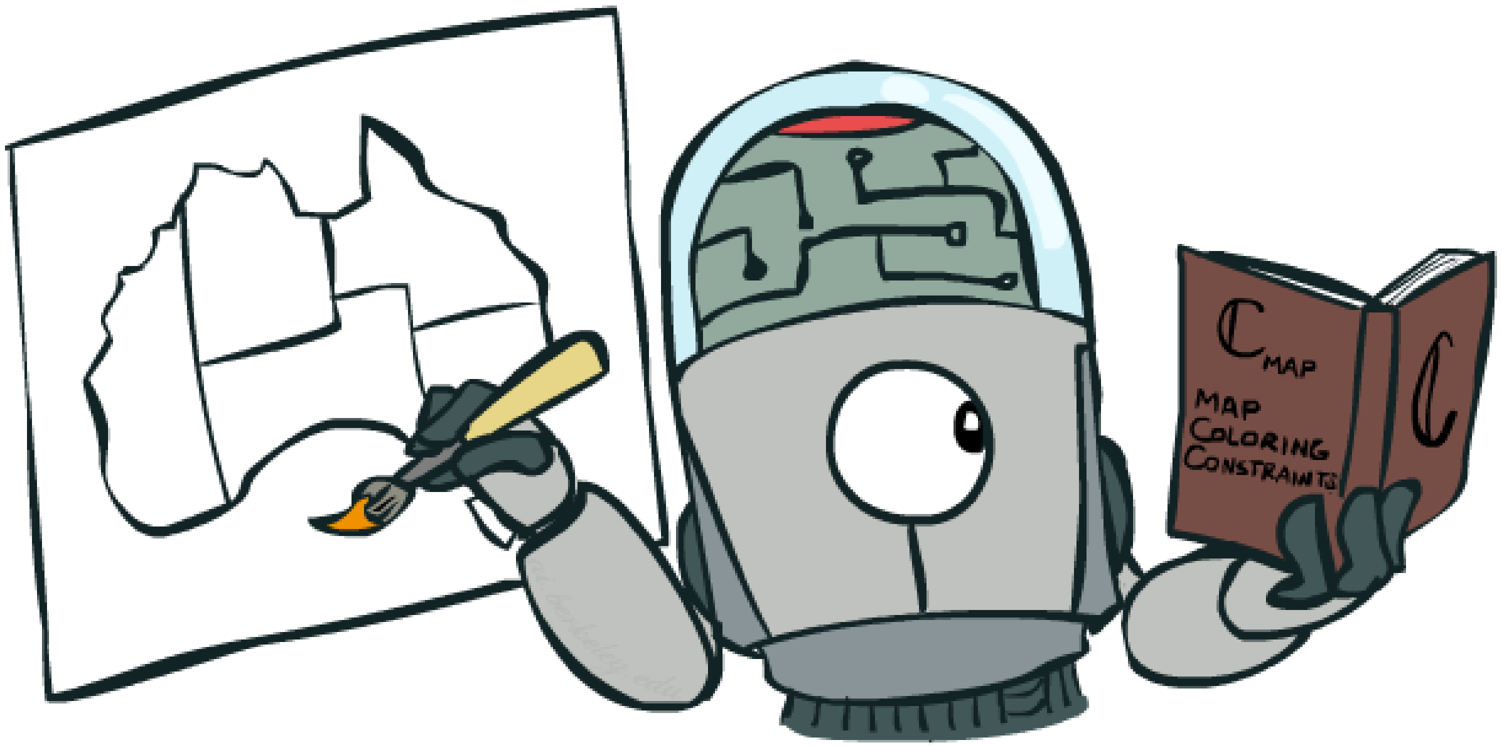
What is Search For?

- 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



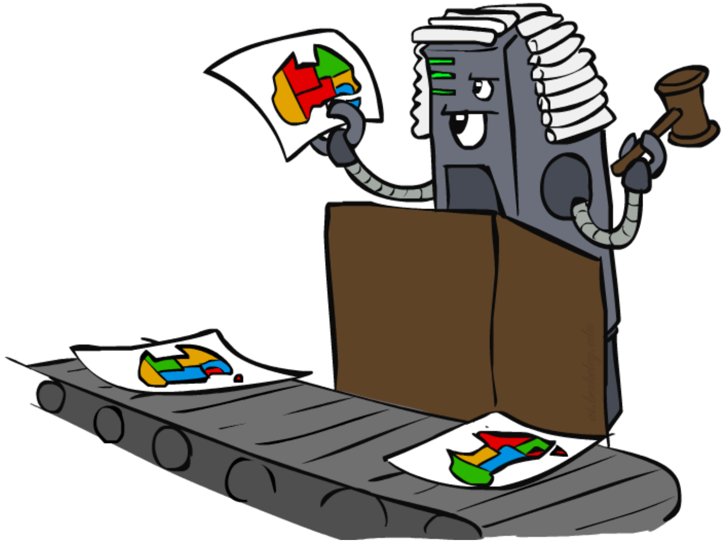
Constraint Satisfaction Problems

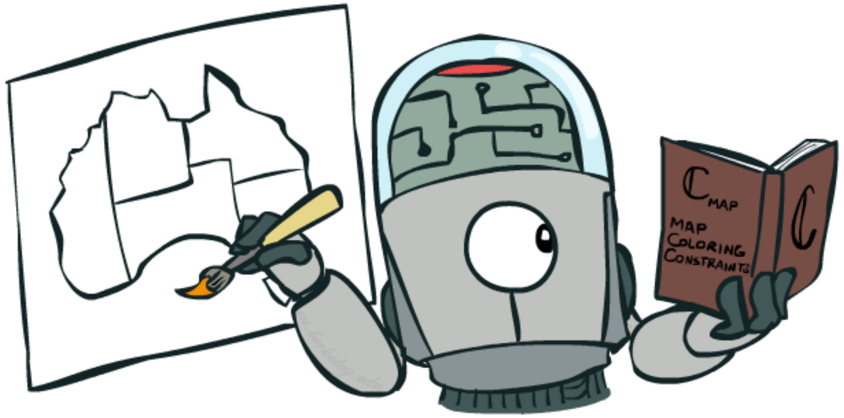
- Standard search problems:
  - State is a "black box": arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

- 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

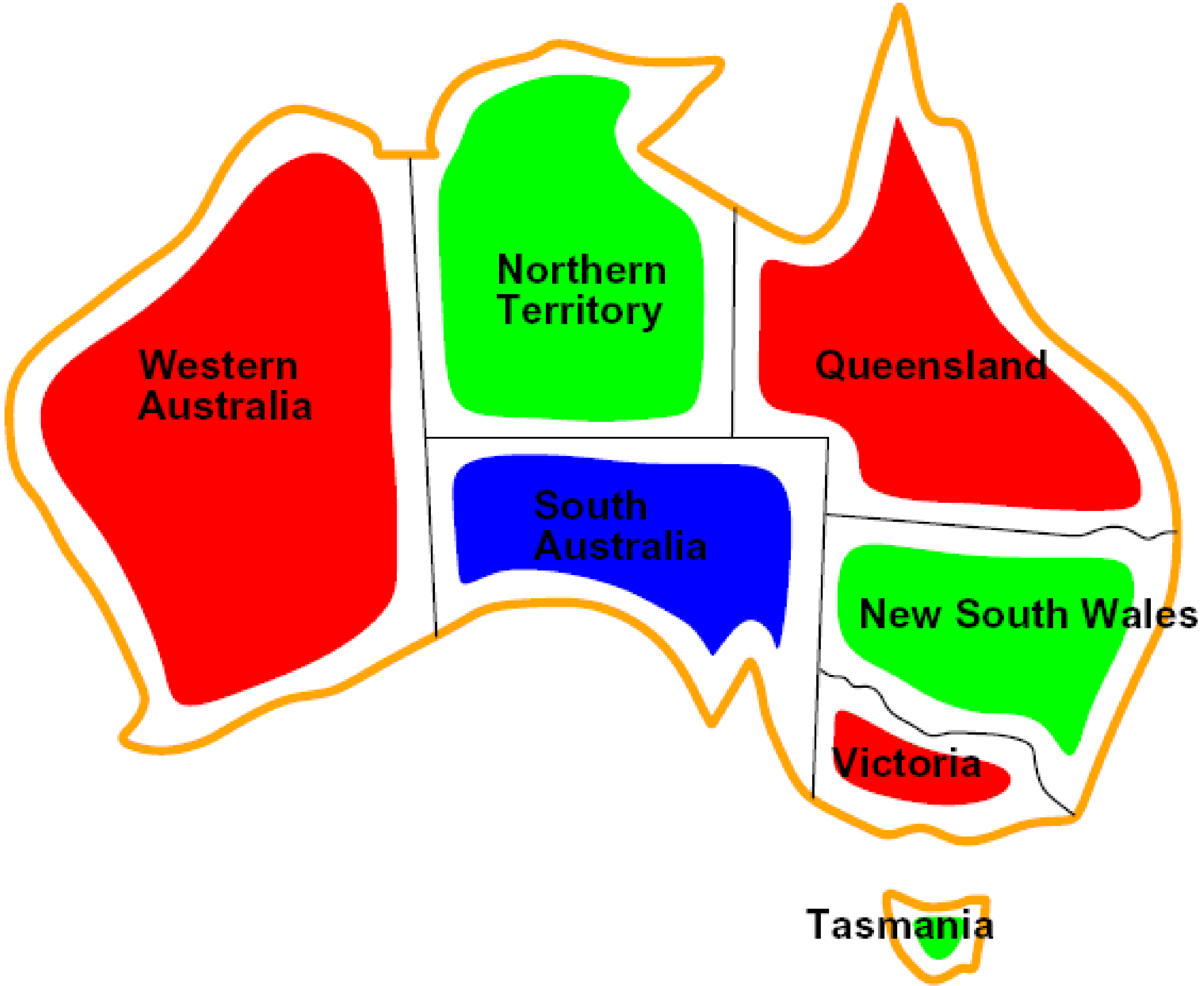
- Simple example of a *formal representation language*

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





CSP Examples



Example: Map Coloring

- Variables: $\mathsf{WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T}$

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

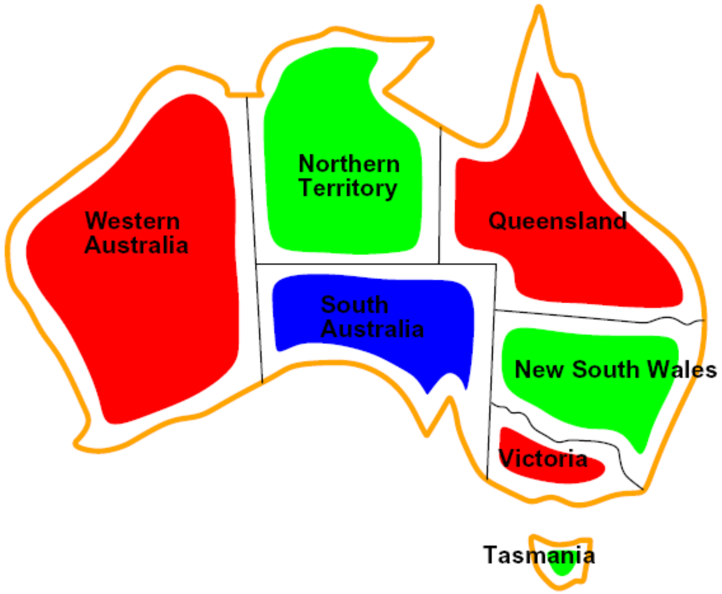
- Constraints: adjacent regions must have different colors

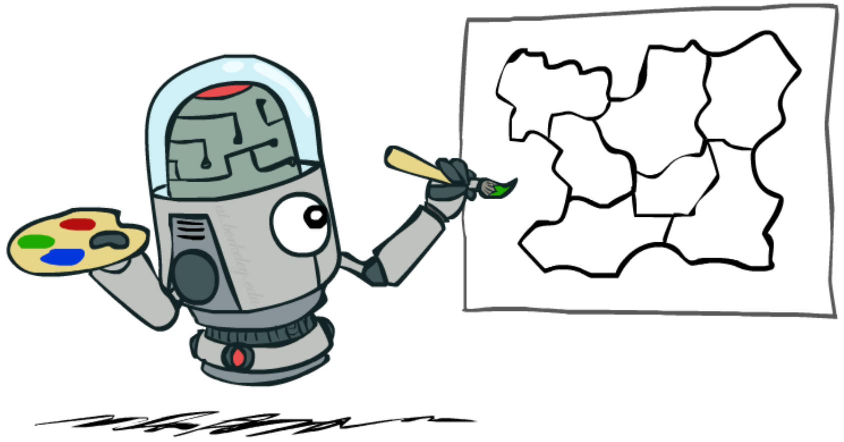
  Implicit: $\mathsf{WA} \neq \mathsf{NT}$

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

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

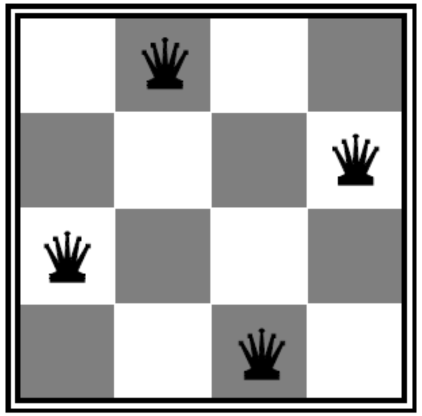
  $\{\mathsf{WA}=\text{red},\ \mathsf{NT}=\text{green},\ \mathsf{Q}=\text{red},\ \mathsf{NSW}=\text{green},\ \mathsf{V}=\text{red},\ \mathsf{SA}=\text{blue},\ \mathsf{T}=\text{green}\}$

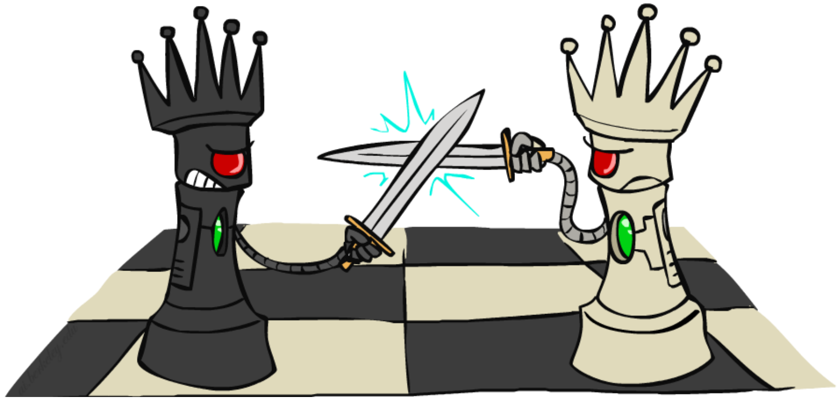




Example: N-Queens

- Formulation 1:
  - Variables: X_{ij} (i = row, j = column)
  - Domains: $\{0, 1\}$
  - Constraints





$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$ (Same row, different columns)

$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$ (Different row, same column)

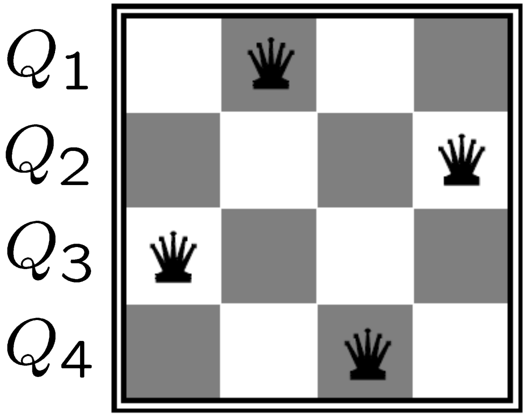
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$ (Diagonal, down and right)

$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$ (Diagonal, down and left)

$$\sum_{i,j} X_{ij} = N$$ (N queens on the board)

Example: N-Queens

- Formulation 2:
  - Variables: Q_k (where queen is in row k)
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:

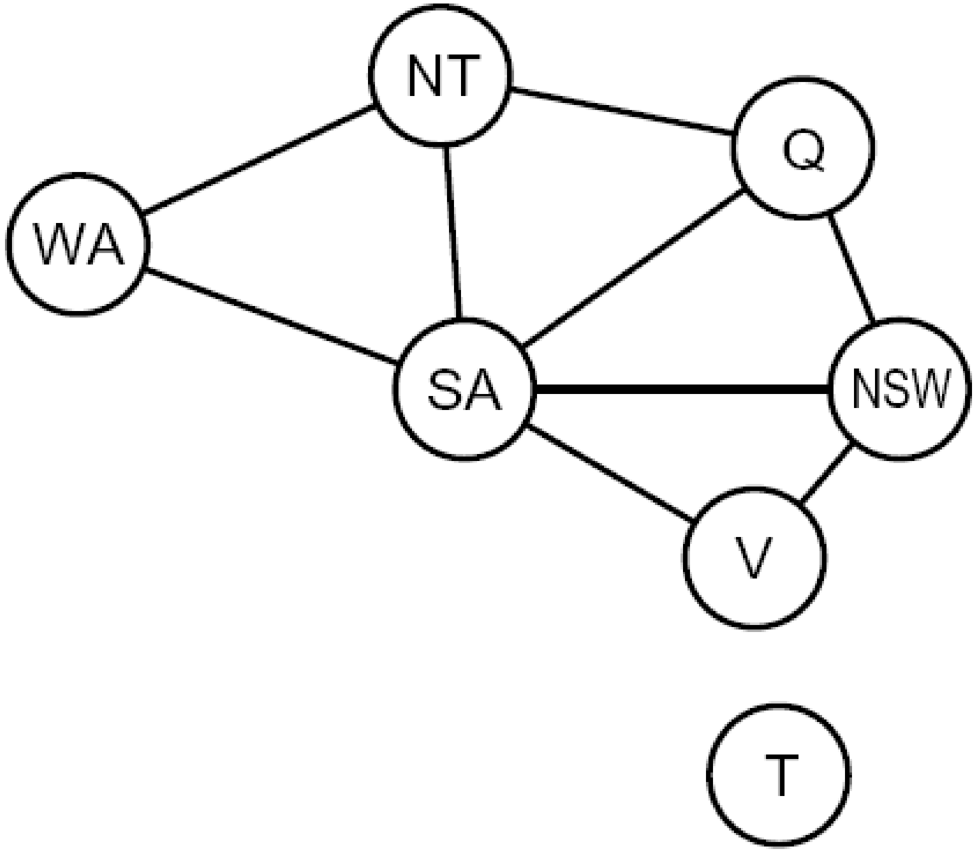


Implicit: $\forall i, j \;\; \text{non-threatening}(Q_i, Q_j)$

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

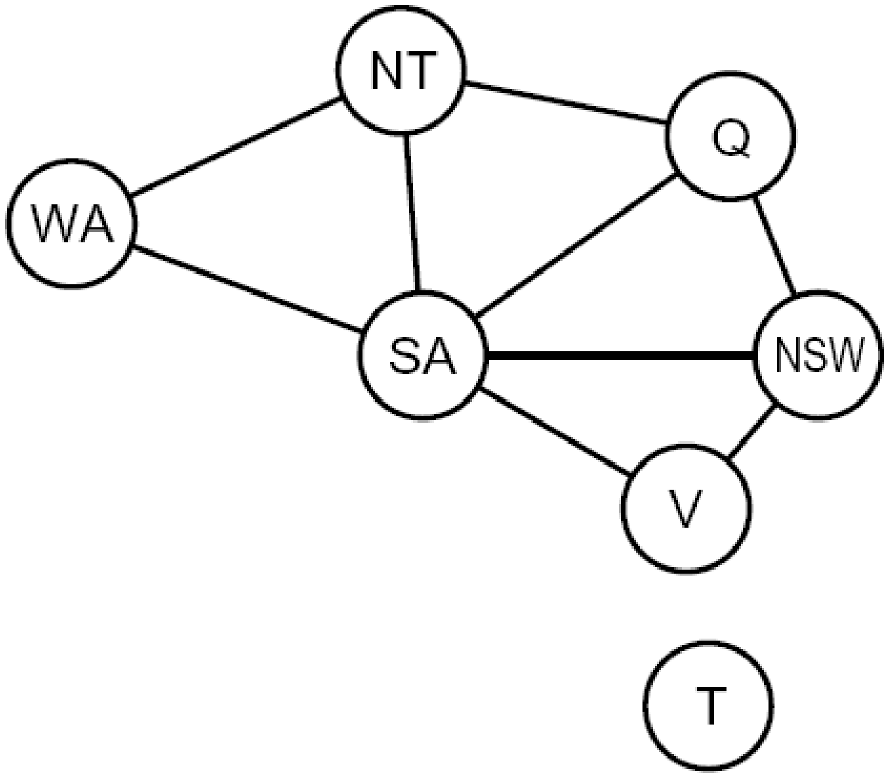
$\cdots$

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:

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

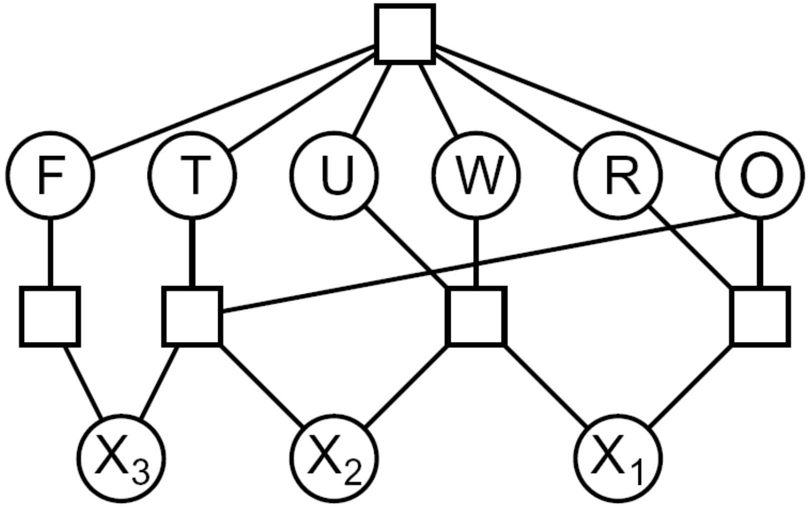
- Constraints:

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

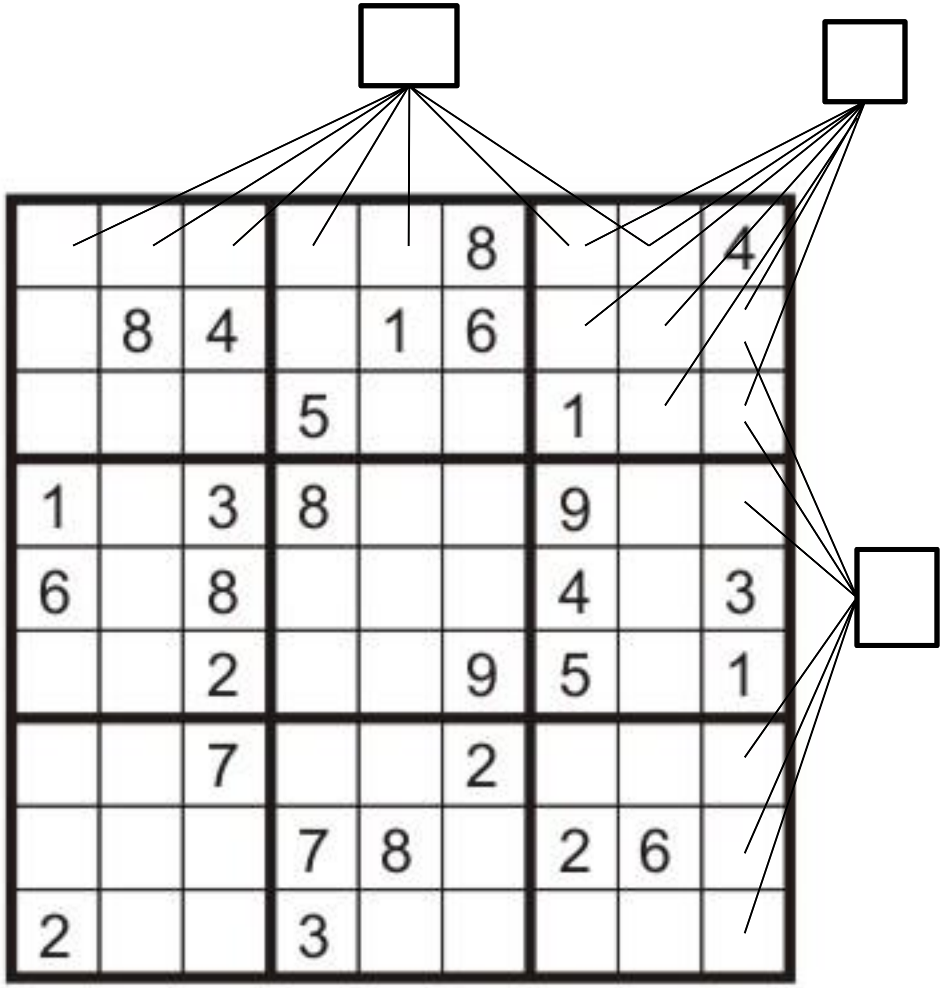
  $O + O = R + 10 \cdot X_1$

  $\cdots$

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



Example: Sudoku



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

9-way alldiff for each column

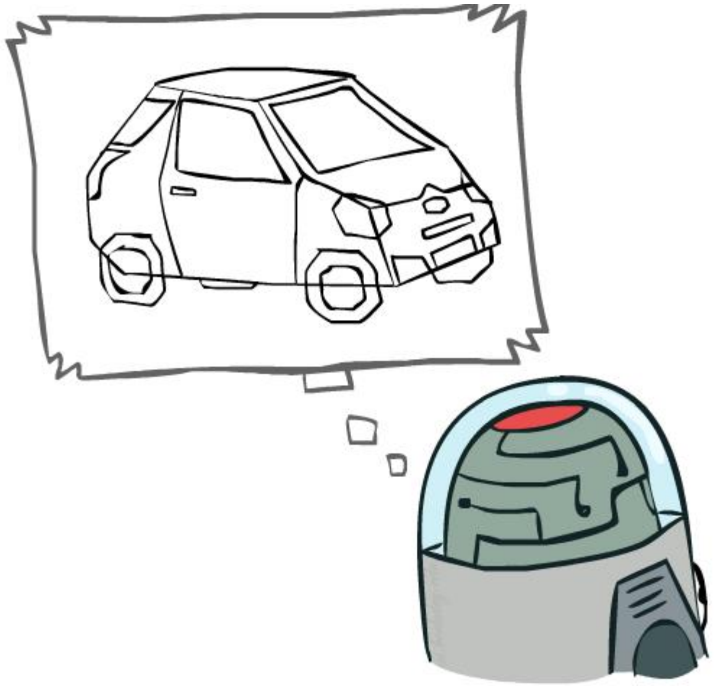
9-way alldiff for each row

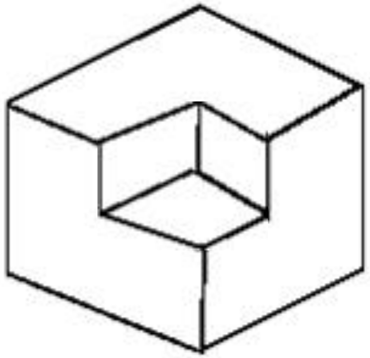
9-way alldiff for each region

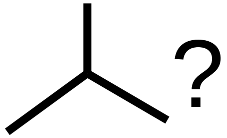
(or can have a bunch of pairwise inequality constraints)

Example: The Waltz Algorithm

- The Waltz algorithm is for interpreting line drawings of solid polyhedra as 3D objects
- An early example of an AI computation posed as a CSP





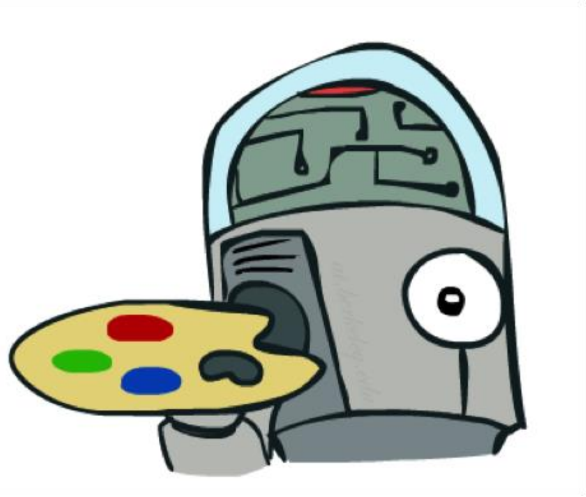


- Approach:
  - Each intersection is a variable
  - Adjacent intersections impose constraints on each other
  - Solutions are physically realizable 3D interpretations

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

Varieties of Constraints

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

$$SA \neq green$$

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

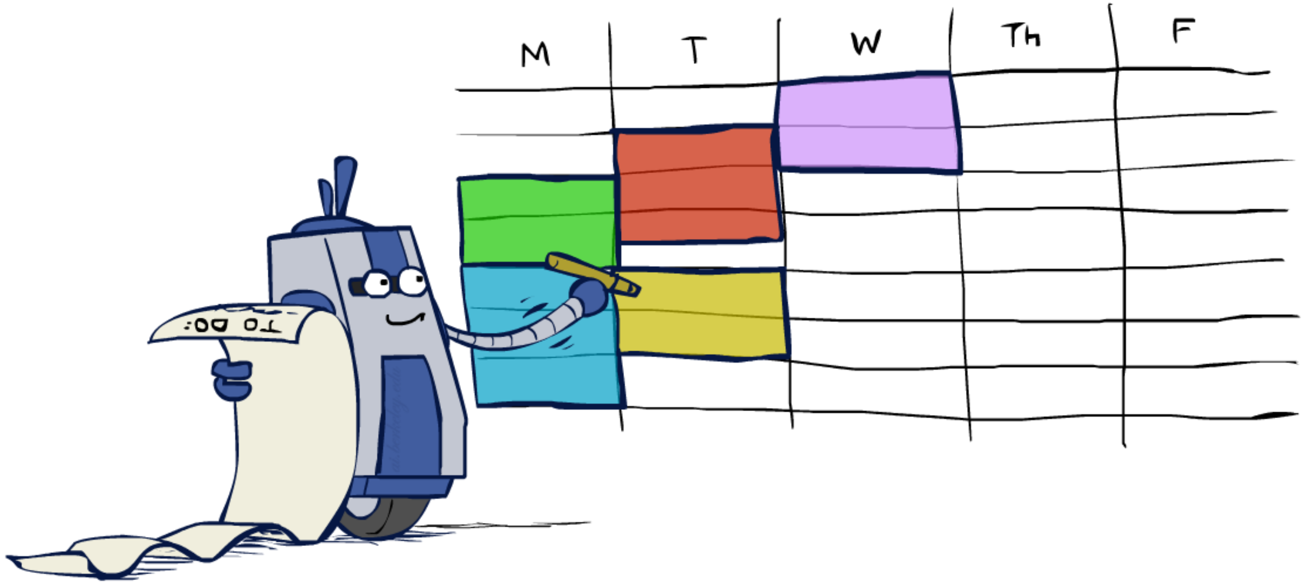
$$SA \neq WA$$

  - 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
  - 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!

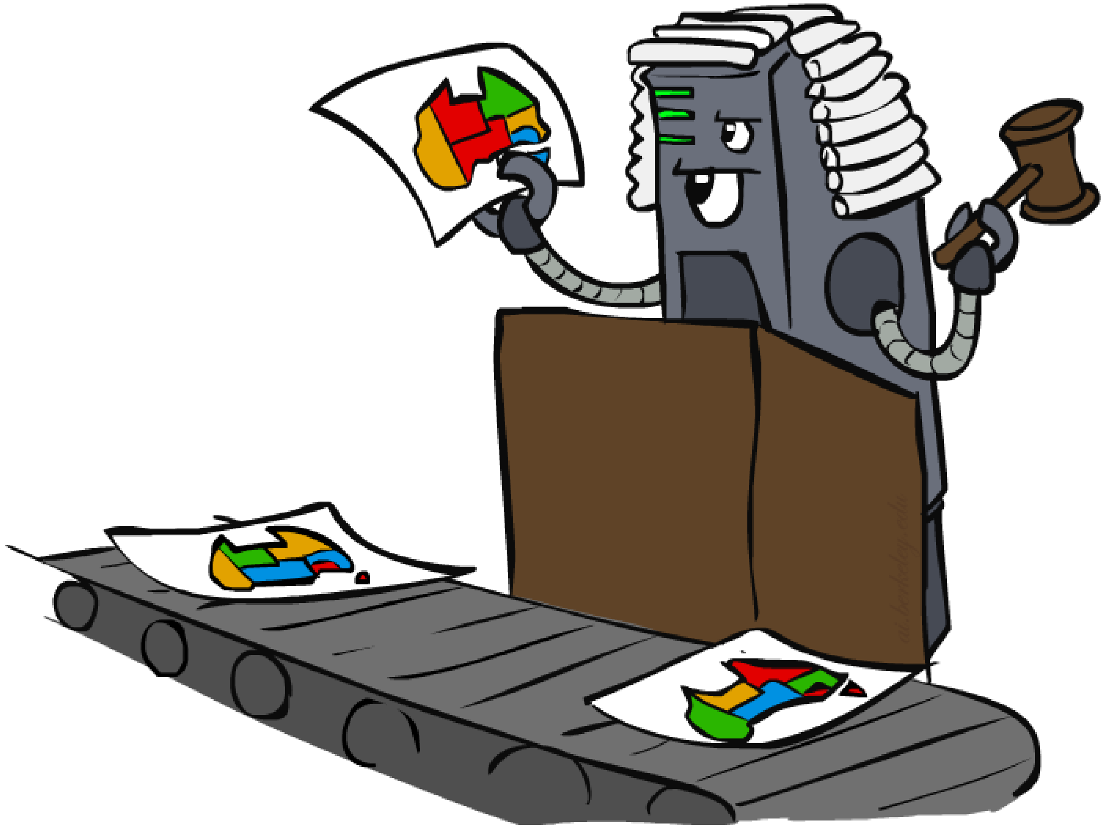


- Many real-world problems involve real-valued variables…

Solving CSPs

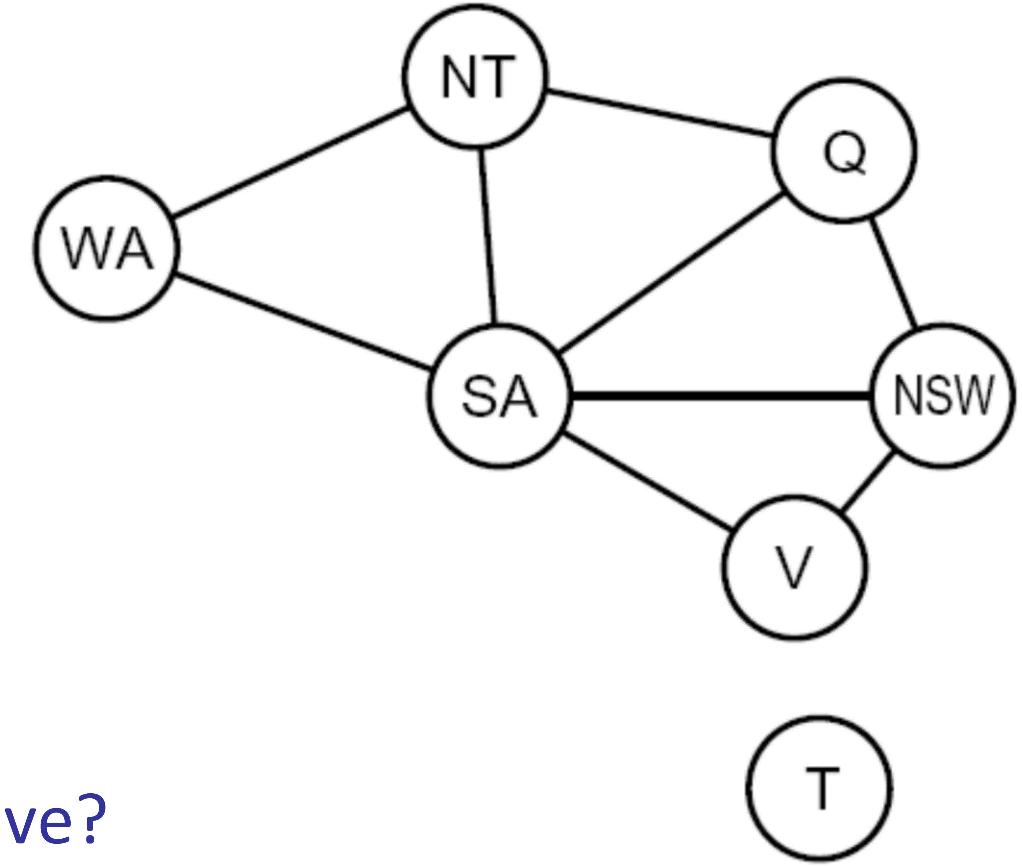
Standard Search Formulation

- 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



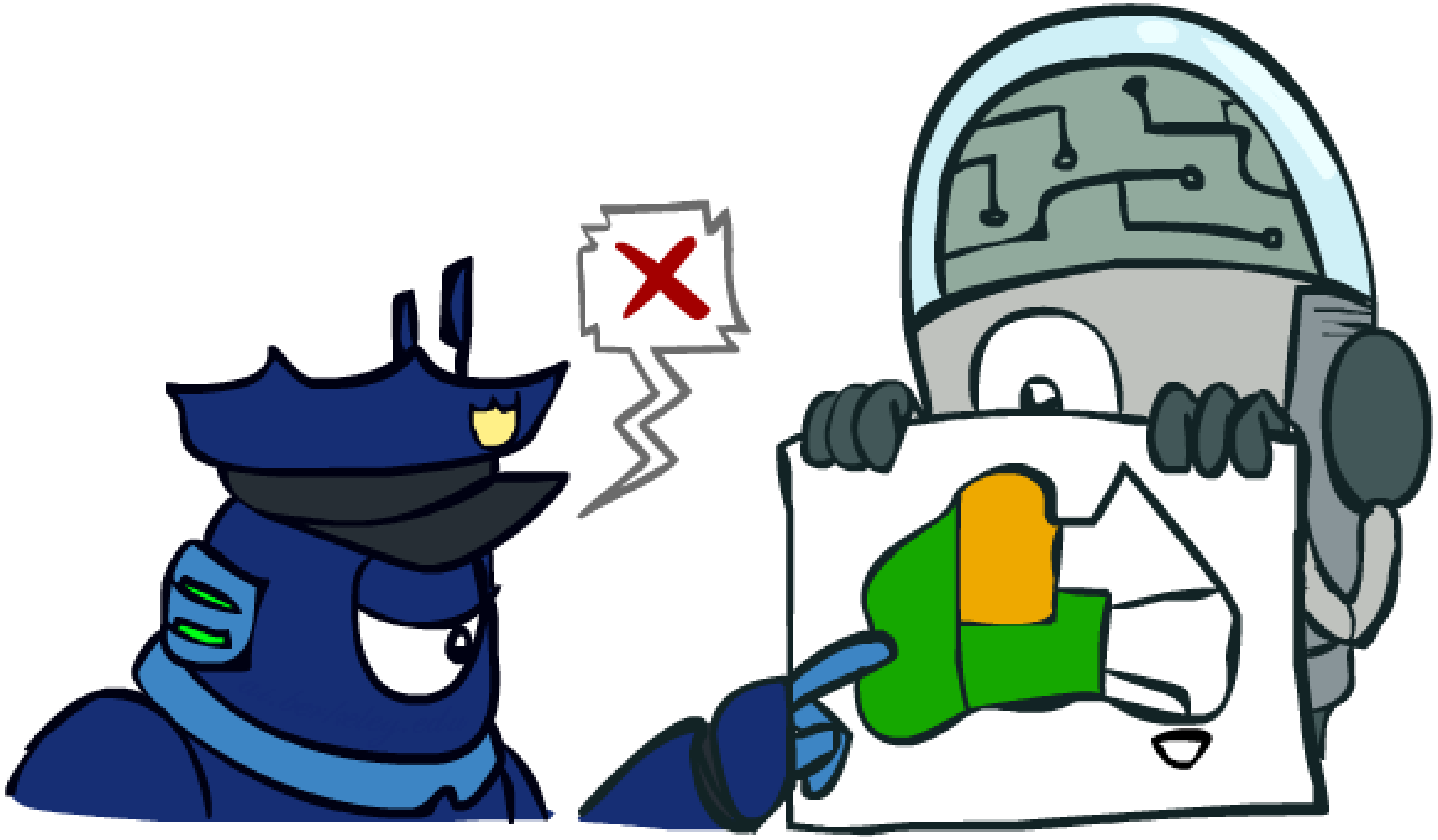
Search Methods

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



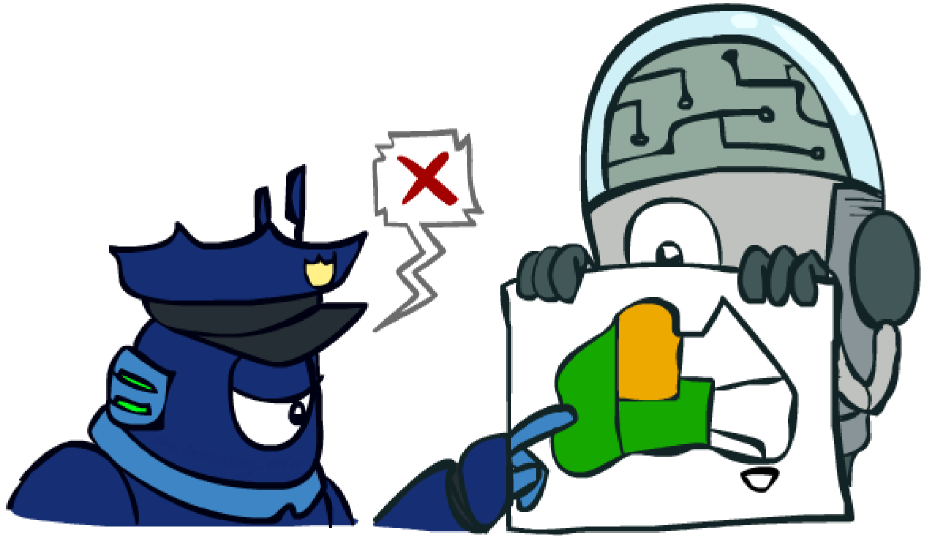
[Demo: coloring -- dfs]

Backtracking Search

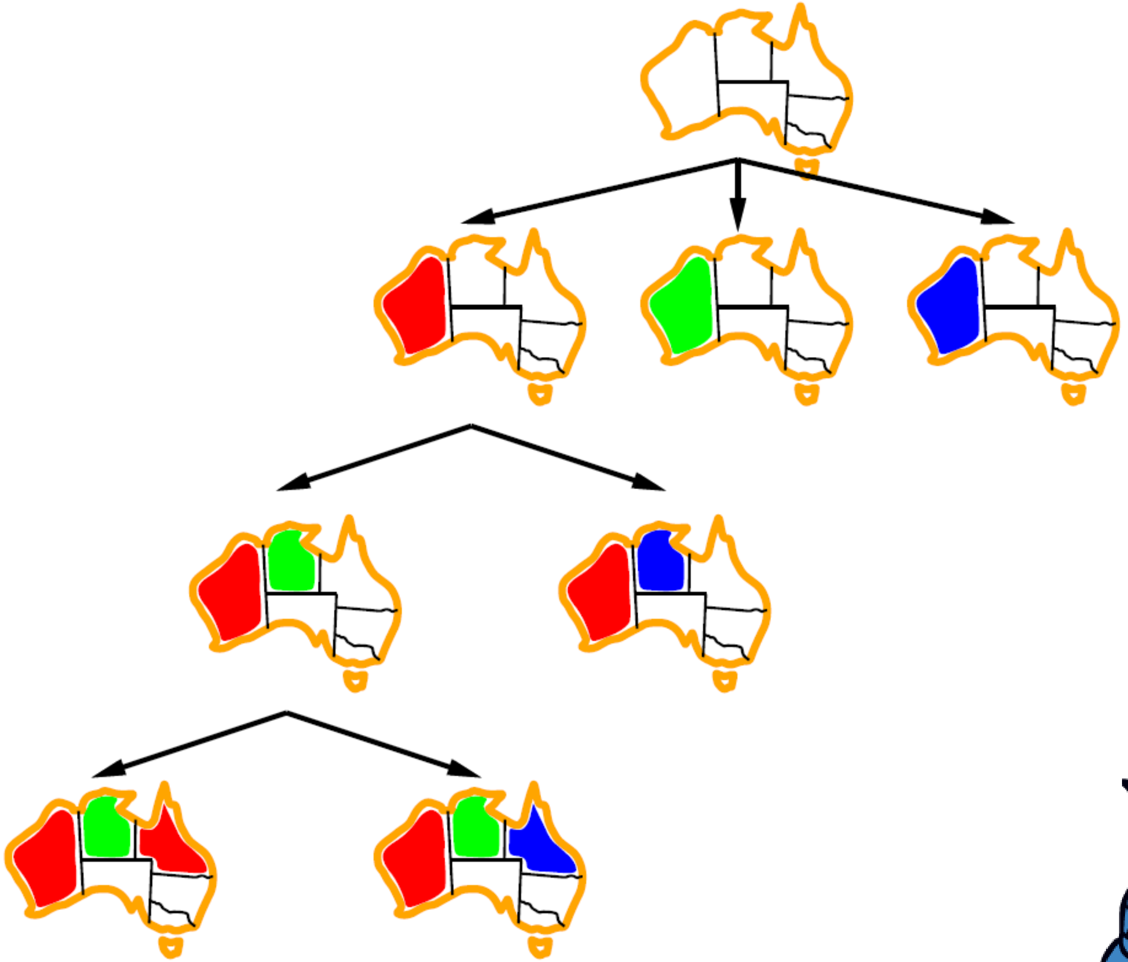


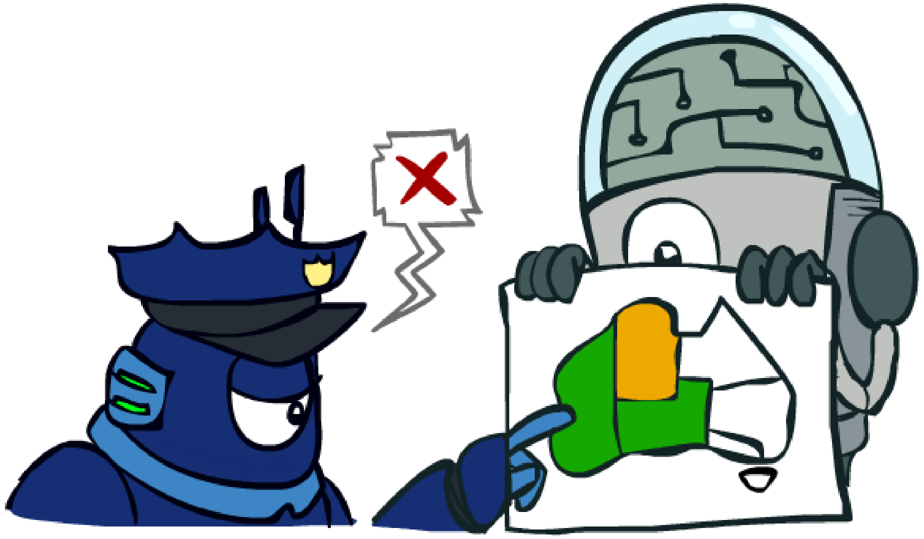
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - 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 constraint
  - "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?

[Demo: coloring -- backtracking]

Next time: 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?