CS 188: Artificial Intelligence

Constraint Satisfaction Problems

Instructors: Josh Hug and Adam Janin

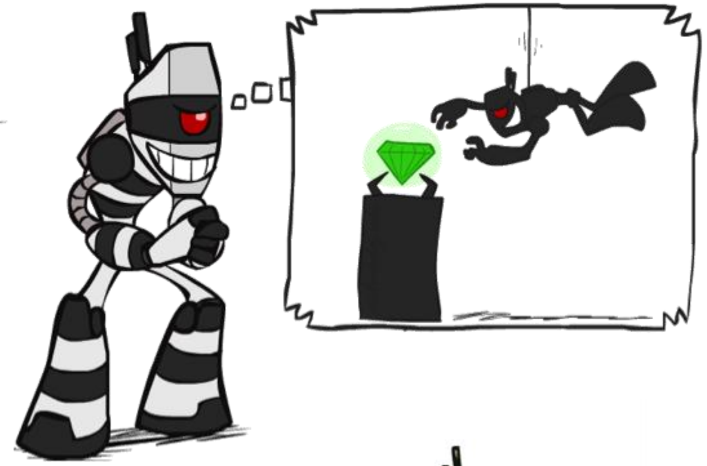
University of California, Berkeley

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

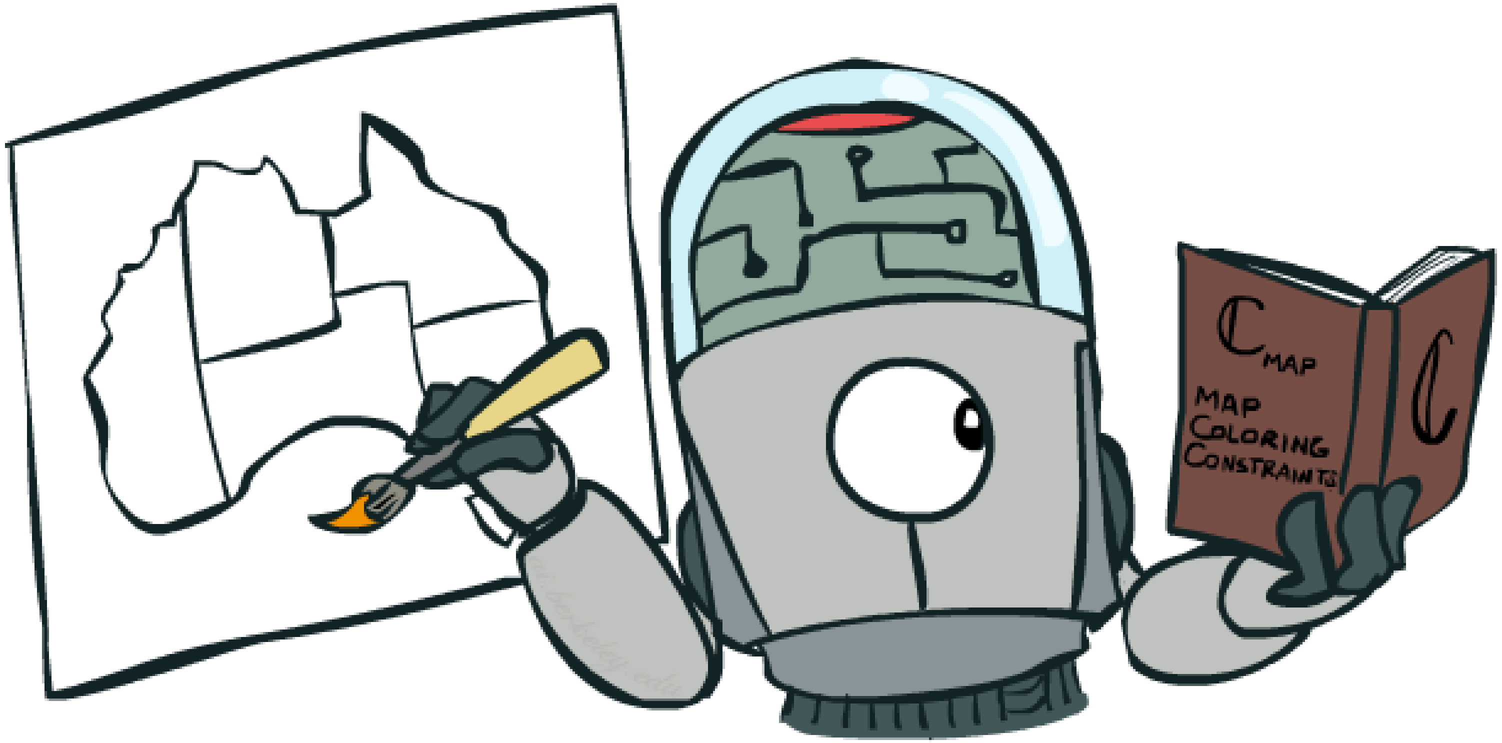
Last week



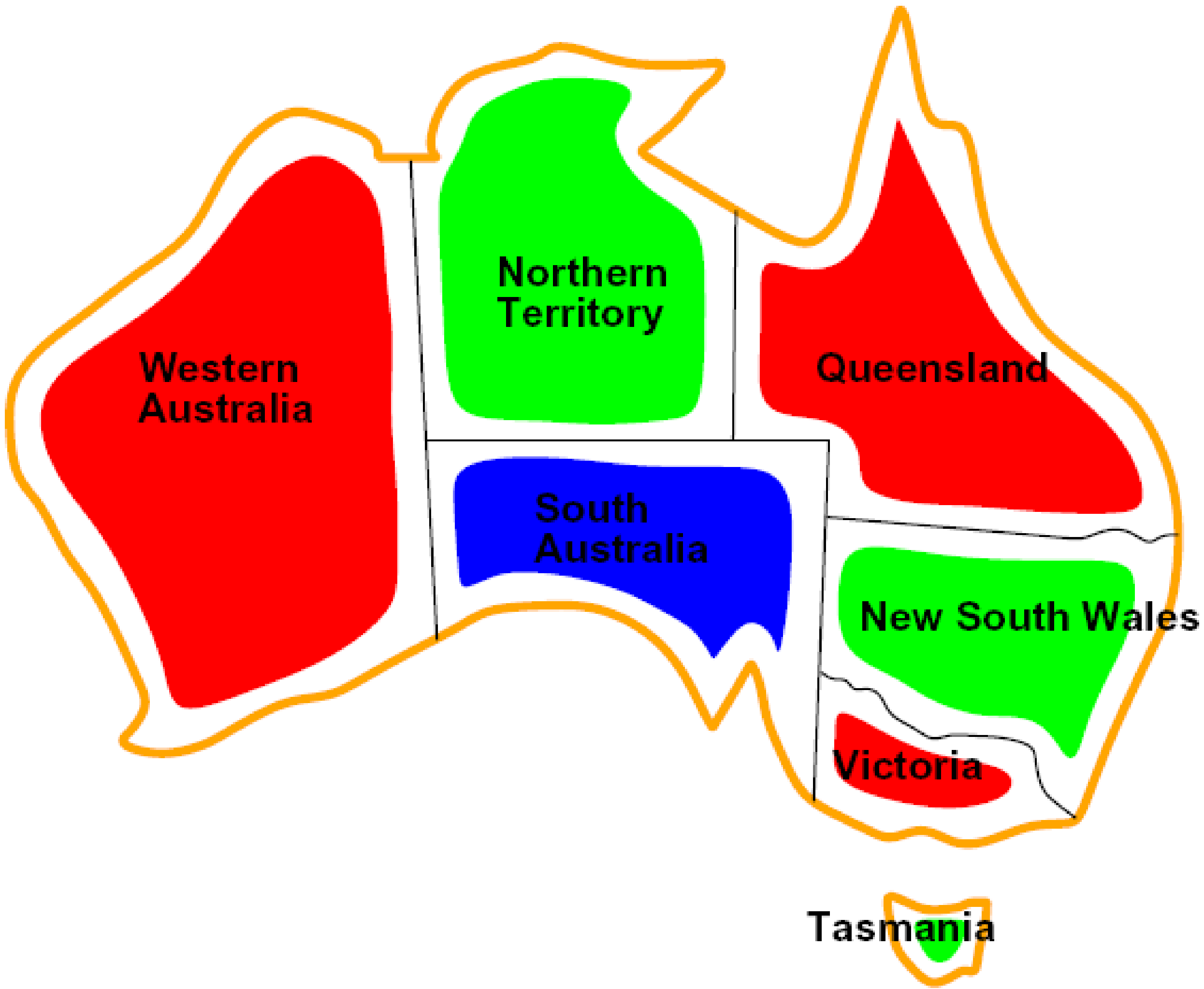
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are a special class of identification problems

This week

Constraint Satisfaction Problems



CSP Examples



Example: Map Coloring

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

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

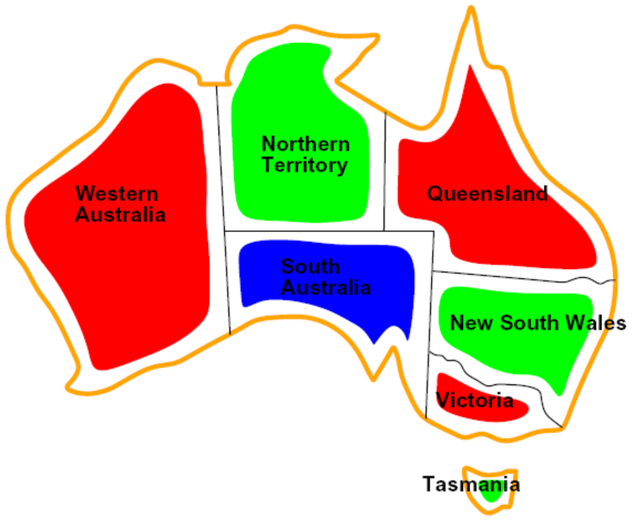
- Constraints: adjacent regions must have different colors

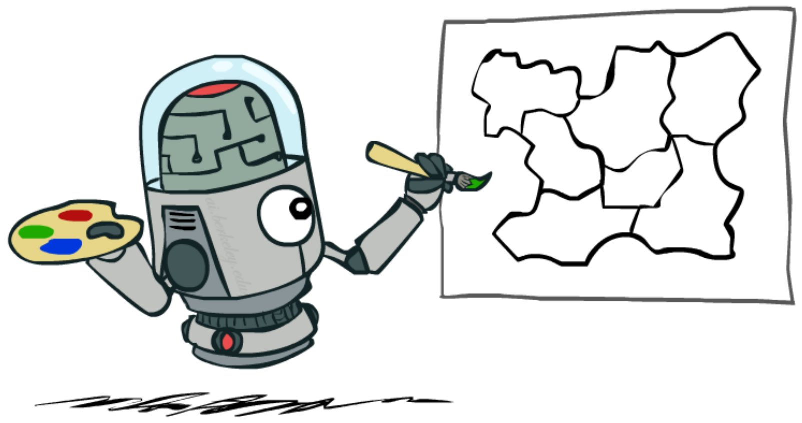
  Implicit: $\text{WA} \neq \text{NT}$

  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}

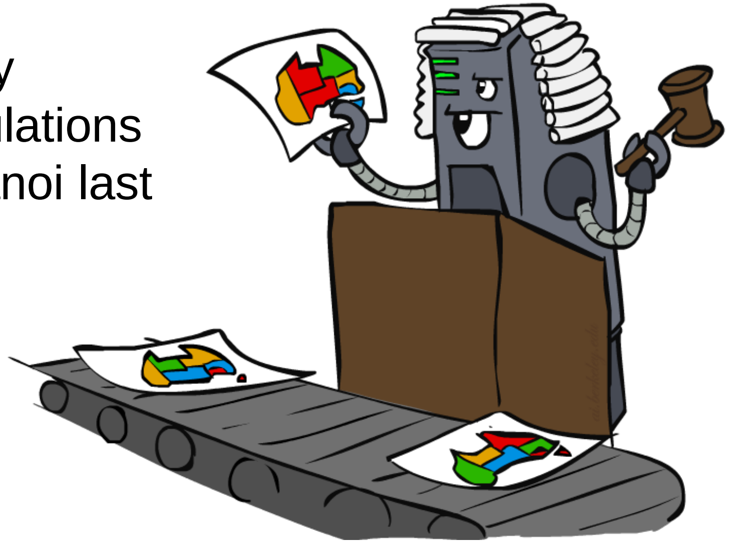




Constraint Satisfaction Problems

- 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

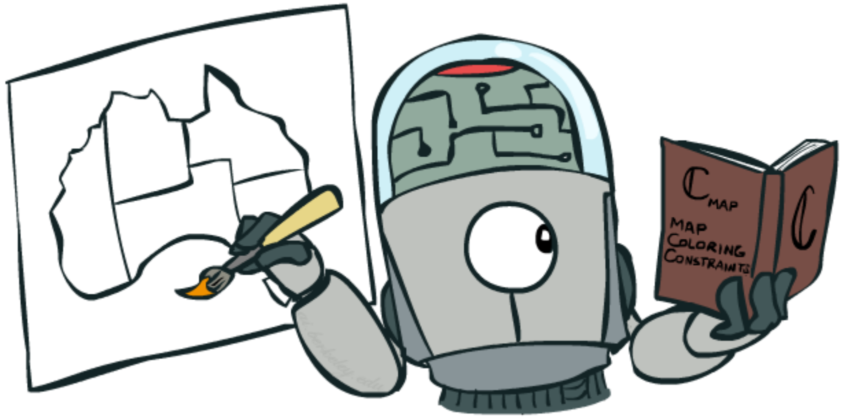
  e.g., saw many possible formulations of Tower of Hanoi last week.



- 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

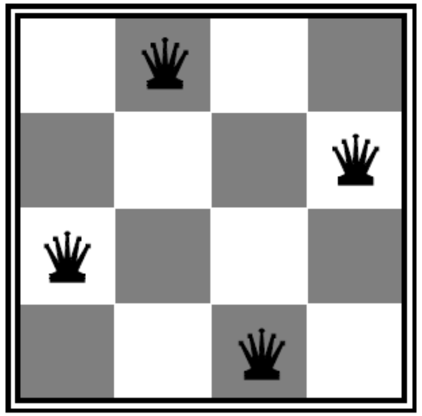
- Simple example of a *formal representation language*

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

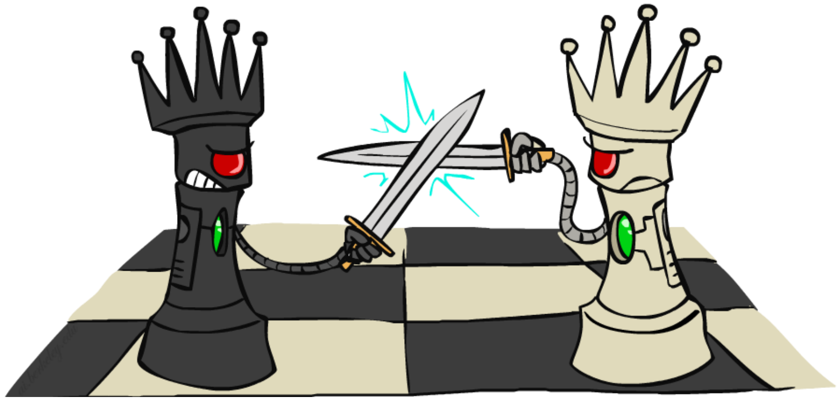


Example: N-Queens

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



16 variables for 4x4 case

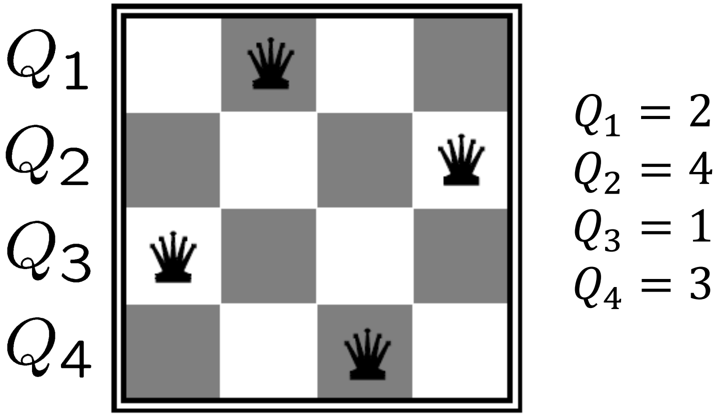


$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

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

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



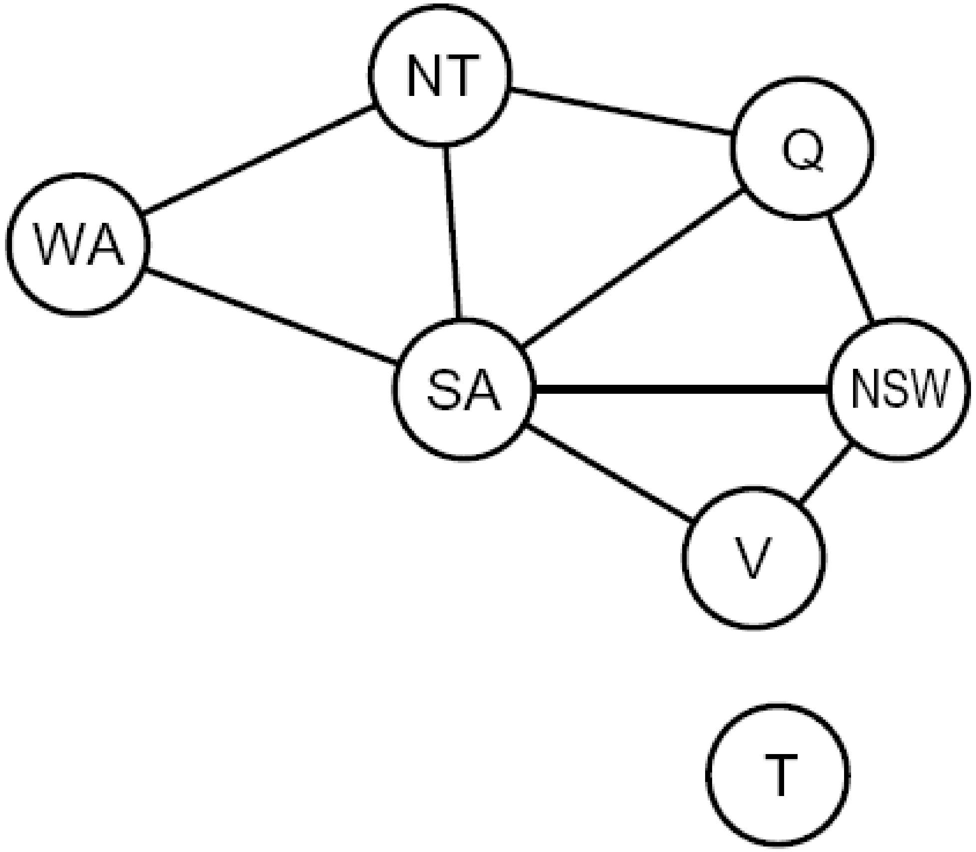
4 variables for 4x4 case

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

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$ (6 possibilities)

$\cdots$

Constraint Graphs

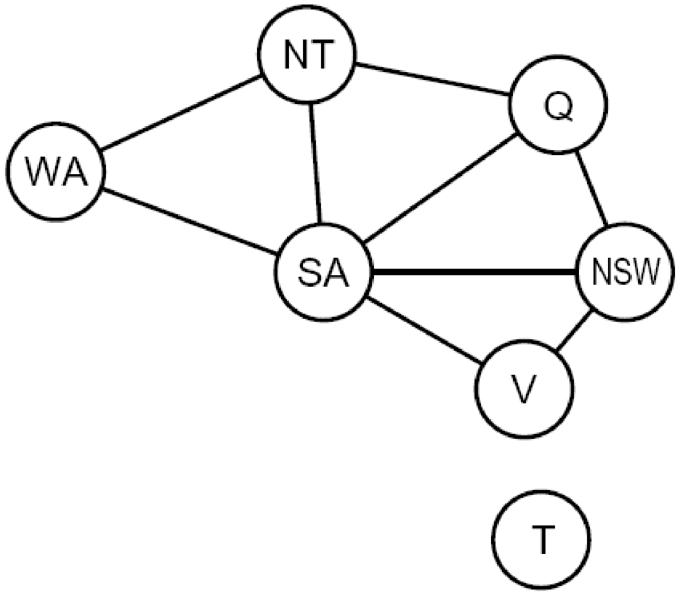


Constraint Graphs

- 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!



Non-Binary CSPs, Example: Cryptarithmetic

- Non-Binary CSP: constraints may relate more than two variables.

- 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:

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

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

  $\cdots$

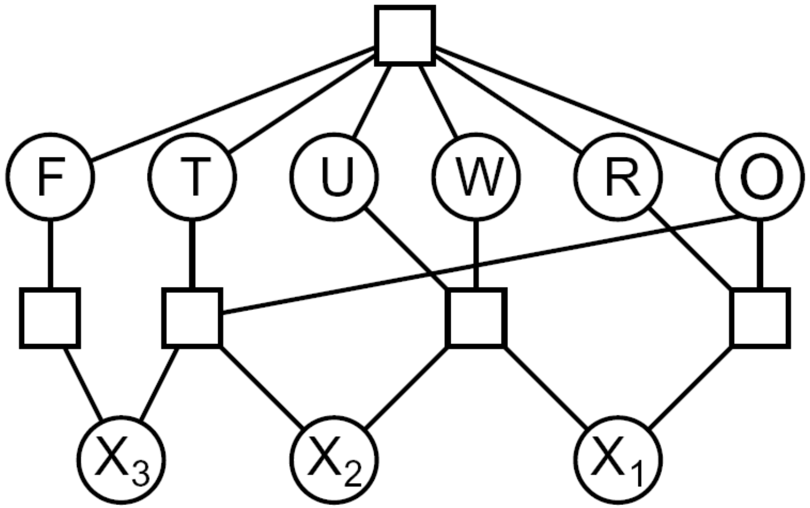
```
   T W O
+  T W O
 -------
 F O U R
```

```
   7 3 4
+  7 3 4
 -------
 1 4 6 8
```

- In Cryptarithmetic game:
  - All letters represent a unique integer
  - Goal is to find letters that are consistent with addition

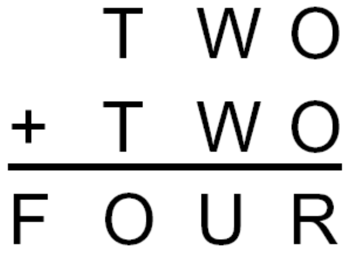
Non-Binary CSPs

- Non-Binary CSP: may relate more than two variables.
- Non-binary constraint graph: nodes may be either variables, or special constraint nodes
  - Example: alldiff(F, T, U, W, R, O) is represented by node at top

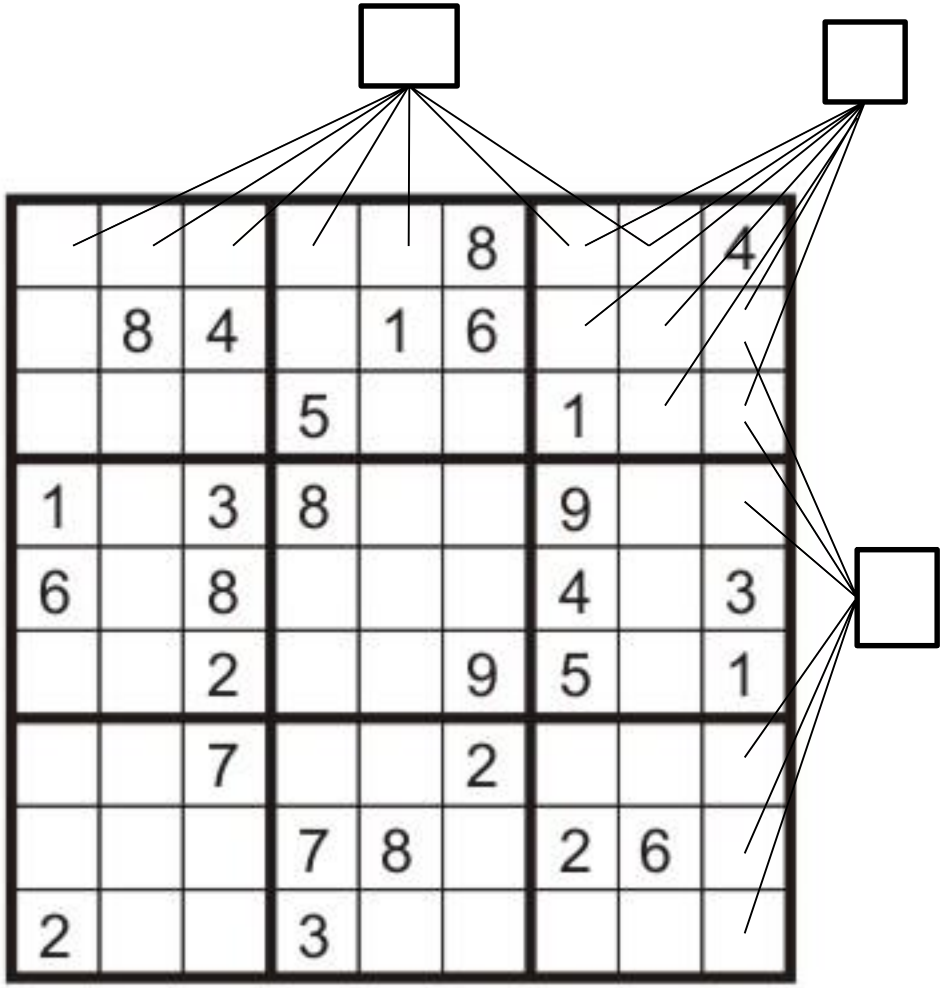


- Constraints:

$$\text{alldiff}(F, T, U, W, R, O)$$
$$O + O = R + 10 \cdot X_1$$
$$\cdots$$



Example: Sudoku



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

9-way alldiff for each column

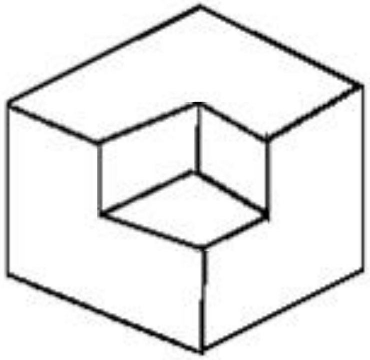
9-way alldiff for each row

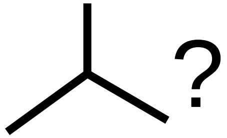
9-way alldiff for each region

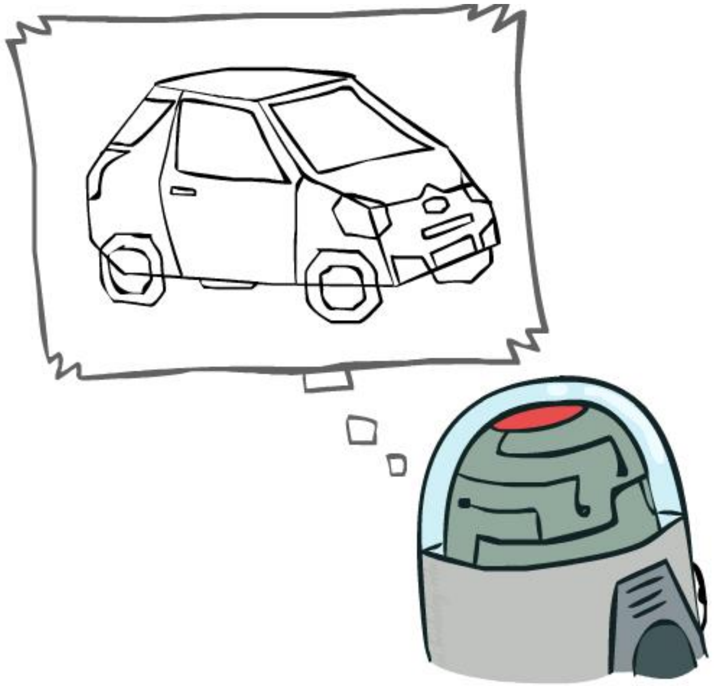
(other specifications are possible, e.g. can have a **bunch** of pairwise inequality constraints)

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







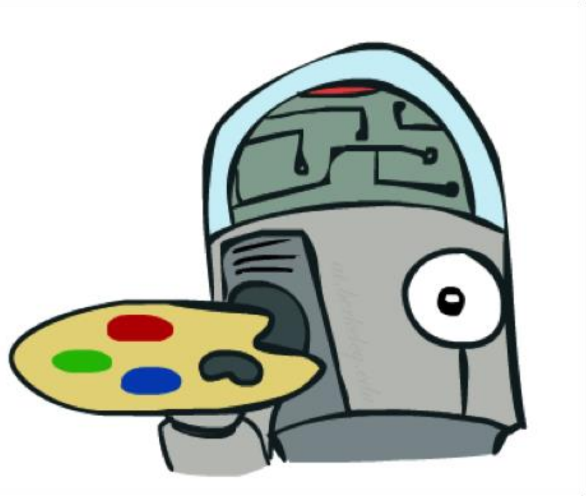
- Approach:
  - Each intersection is a variable
  - Adjacent intersections impose constraints on each other
  - Solutions are physically realizable 3D interpretations

Varieties of CSPs and Constraints

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Domain size d means $O(d^n)$ complete assignments
    - E.g., $d = 2$ includes 3-SAT 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)

won’t discuss in our course



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.:

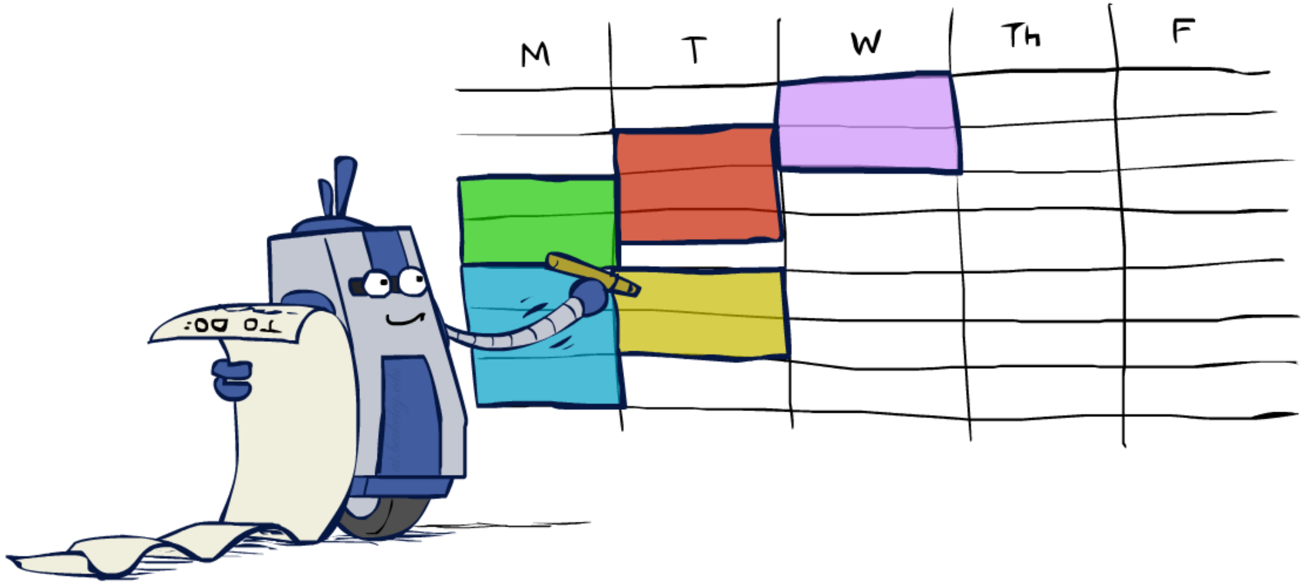
$$SA \neq WA$$

  - 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!

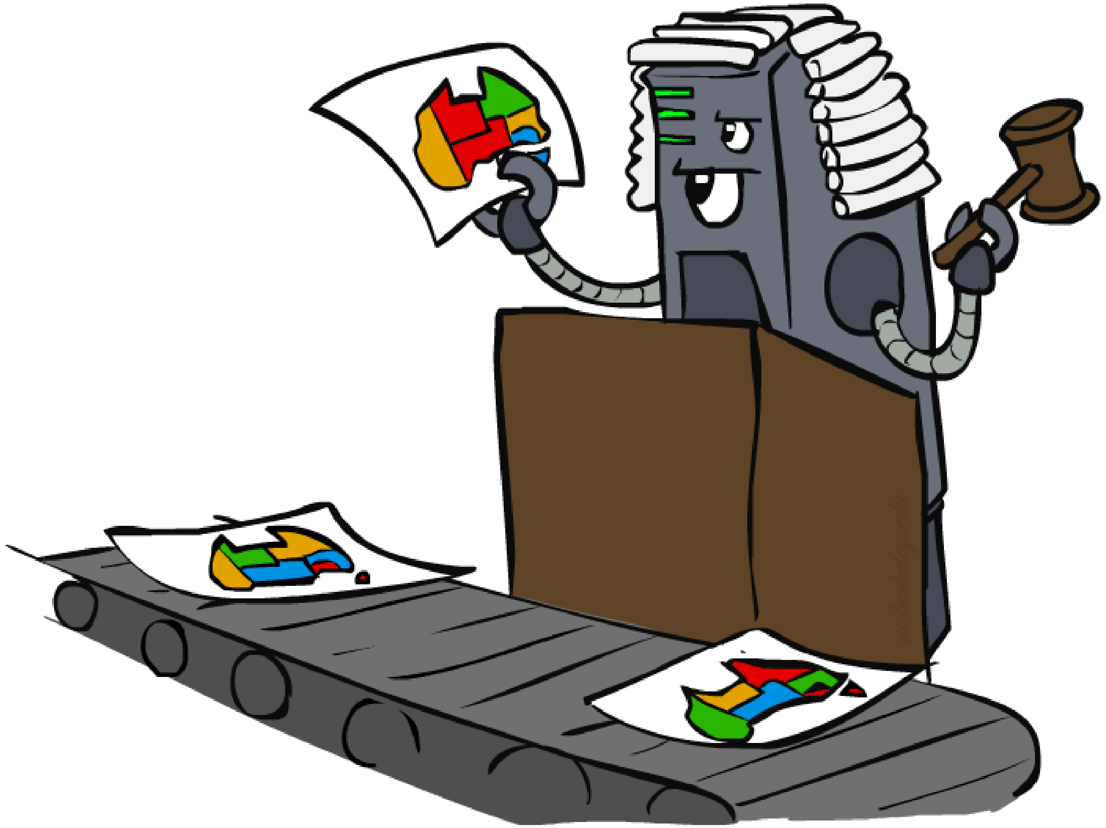


- Many real-world problems involve real-valued variables…

Solving CSPs

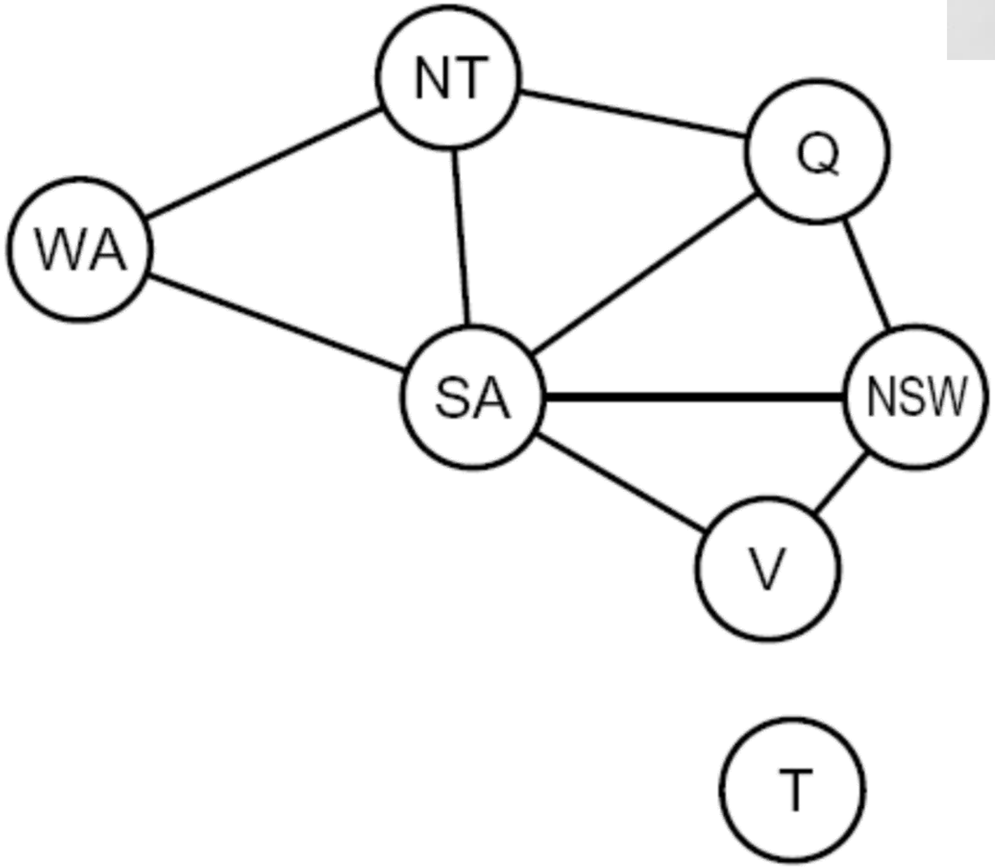
The Naive Formulation

- 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 this straightforward, naïve approach, then improve it



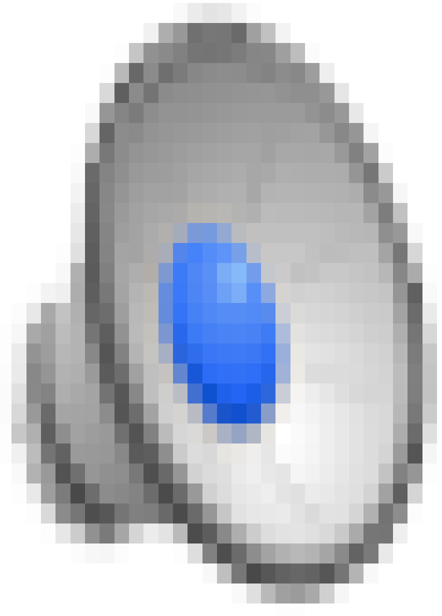
Naive Search, i.e. Using DFS/BFS

- Challenge: Write out the first 3 states that BFS would expand.
- Challenge #2: Write out the first 3 states that DFS would expand.



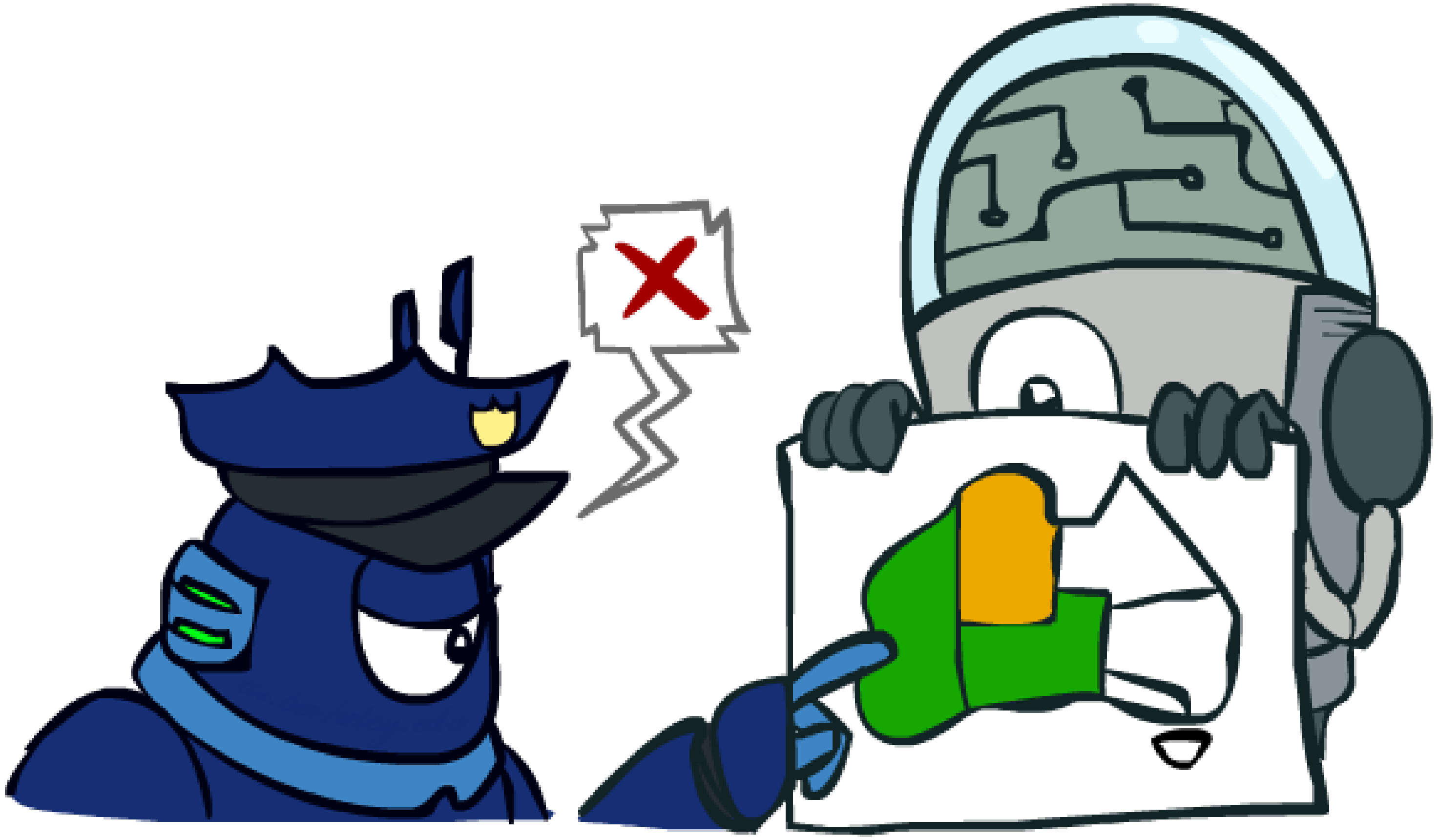
- Extra questions: Does the path matter? Where are the goal states? What are the problems with DFS & BFS?

Video of Demo Coloring -- DFS



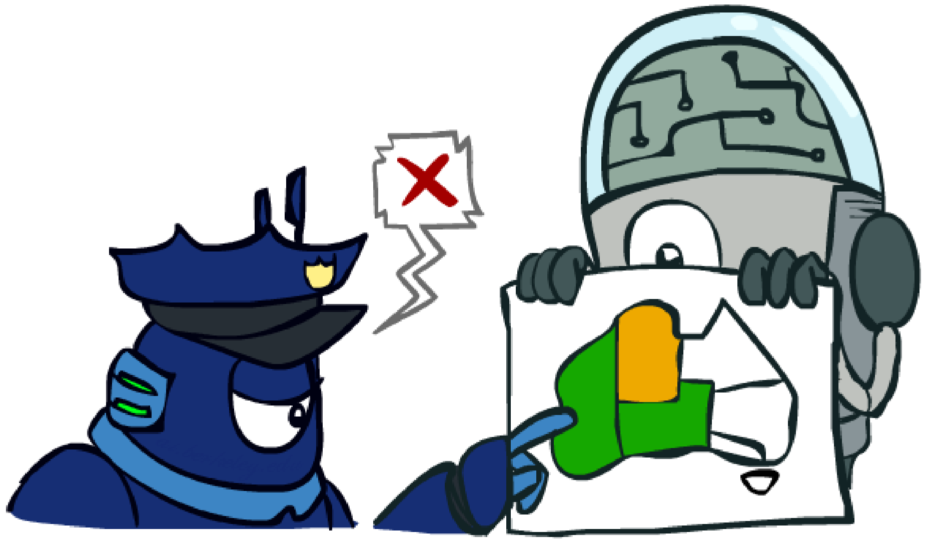
[Demo: CSPs Demo]

Backtracking Search

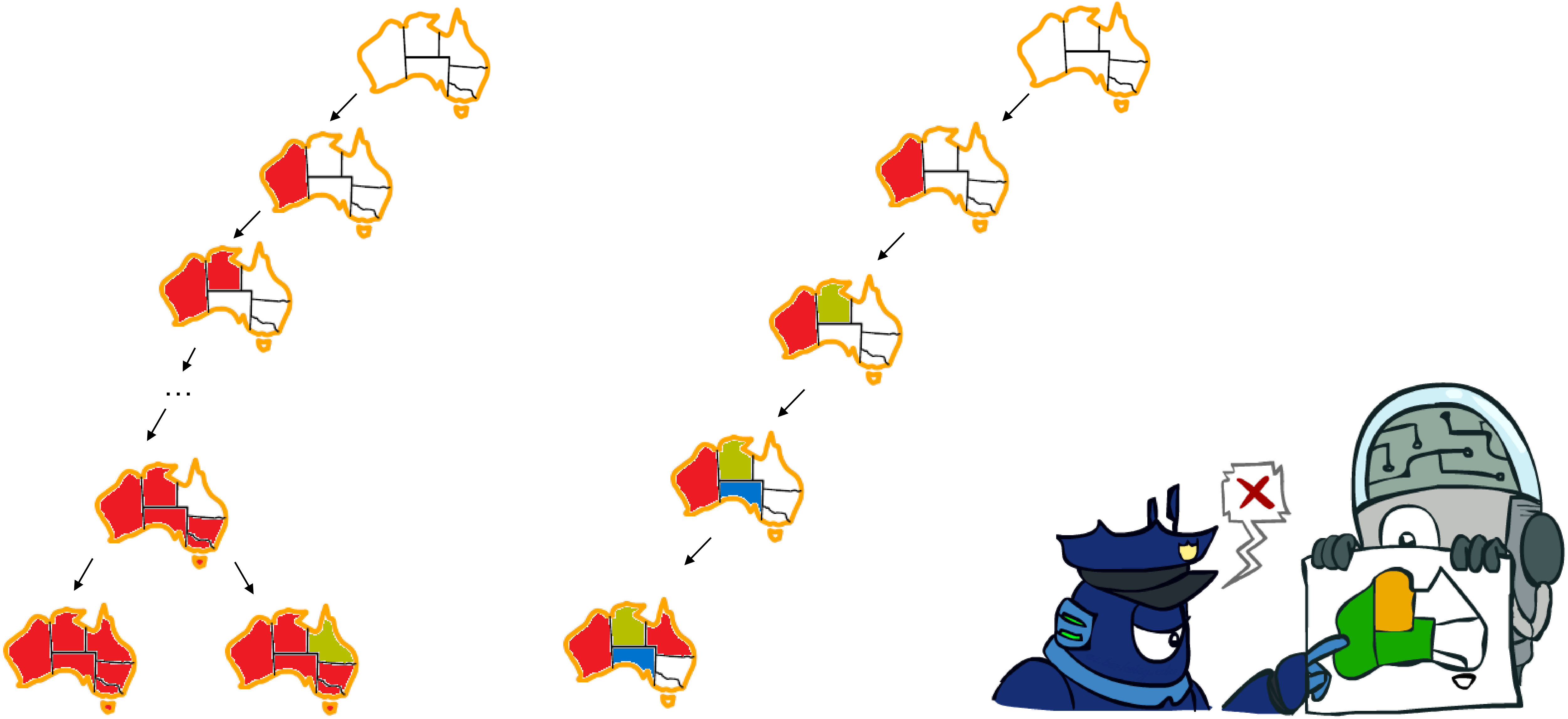


Backtracking Search

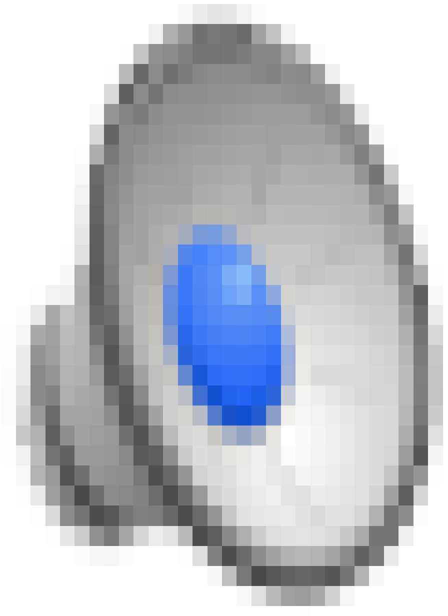
- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each 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)
  - I'll use the term "CSP-backtracking" to avoid confusion.
- Can solve n-queens for $n \approx 25$



DFS (left) vs. CSP-Backtracking (Right)



Video of Demo Coloring – CSP-Backtracking



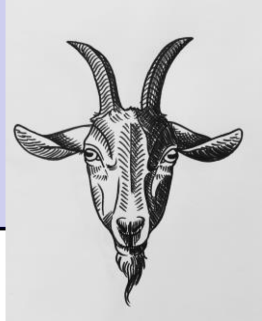
[Demo: CSPs Demo]

CSP-Backtracking Search

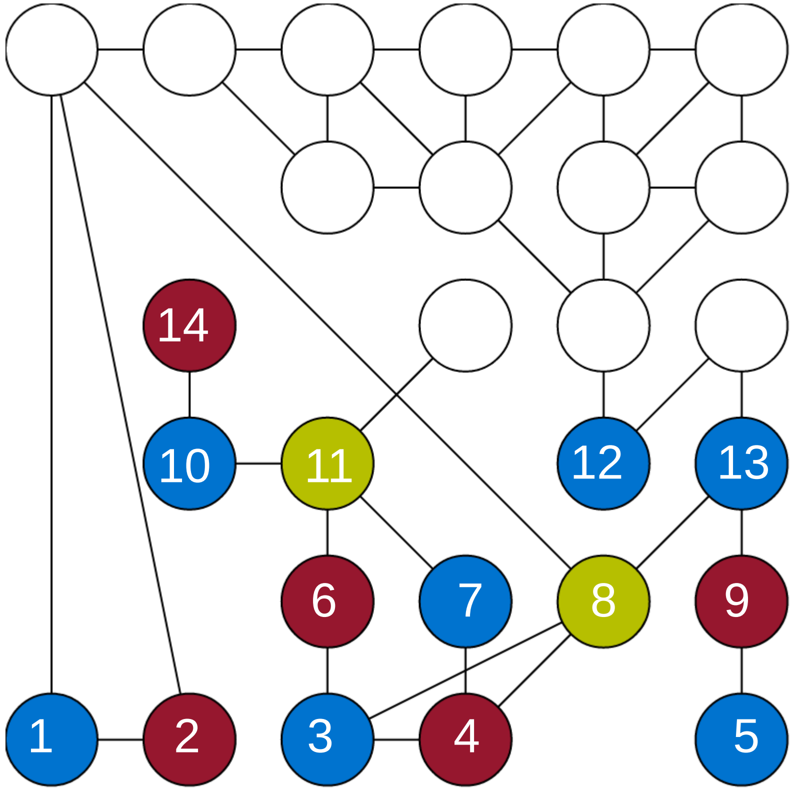
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

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
```

- CSP-Backtracking = DFS + variable-ordering + fail-on-violation
- One optimization possibility: Pick “better” variable orderings and value orderings.

The Return of Question Goat



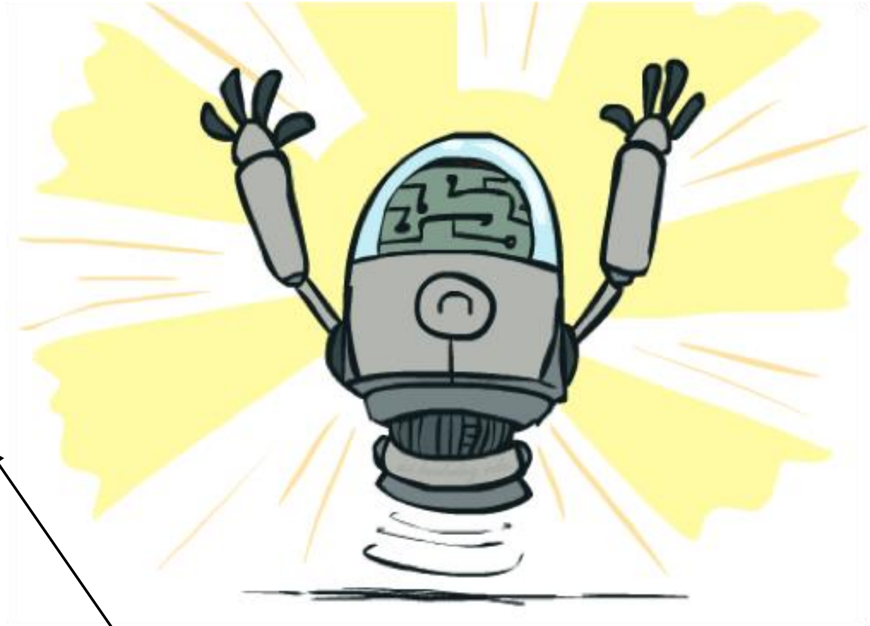
- Consider the partially completed CSP assignment. Assume the decisions about variable assignments are made bottom-up, left-to-right. Let X be the number of the decision that most obviously doomed the current assignment.
  - What is X?
  - Bonus: How many decisions will be made before CSP-Backtracking search realizes its error?



Improving CSP-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?

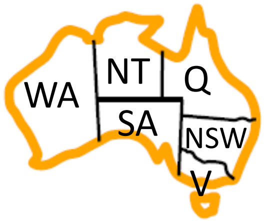


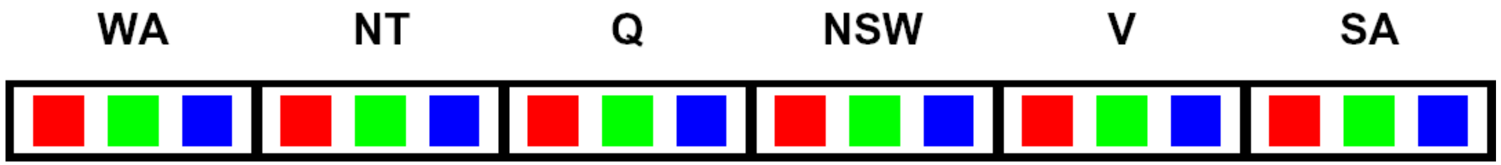
We'll start with filtering.

Filtering

Filtering: Forward Checking (Finding 8)

- 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



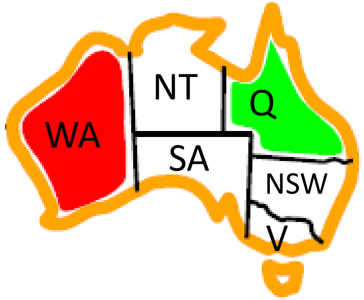


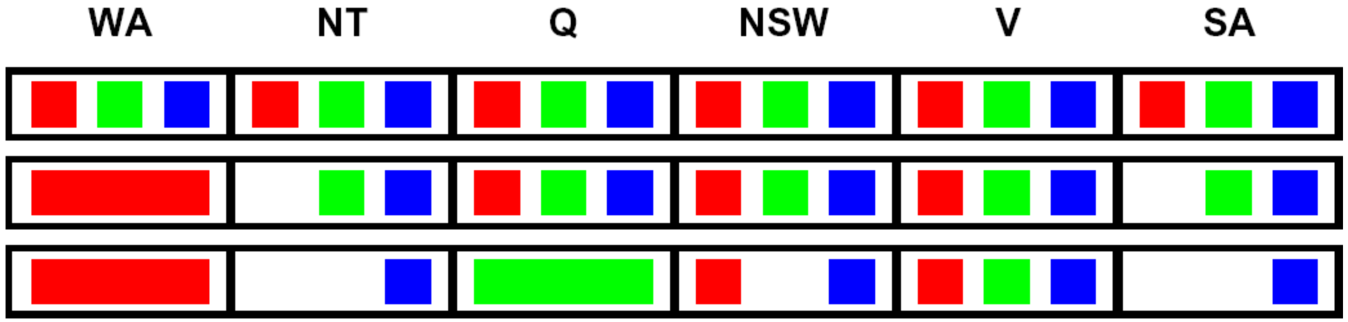
[Demo: coloring -- forward checking]

Coloring Demo – CSP-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:

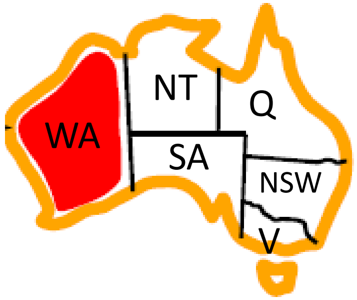


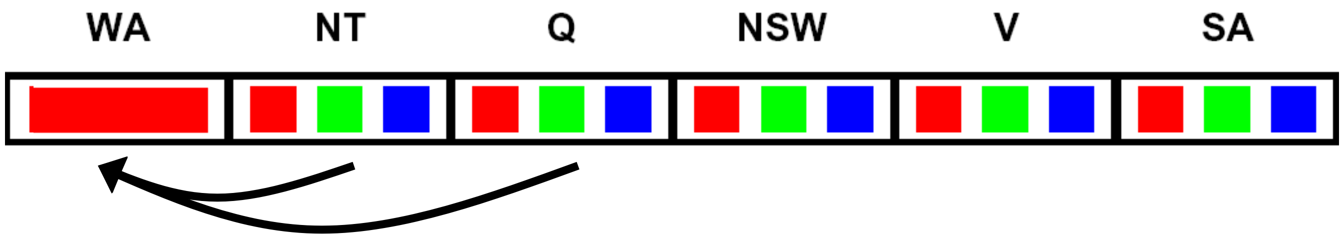


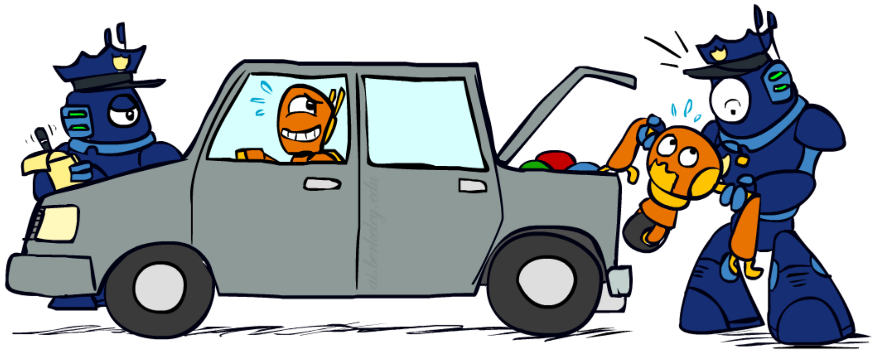
- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

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





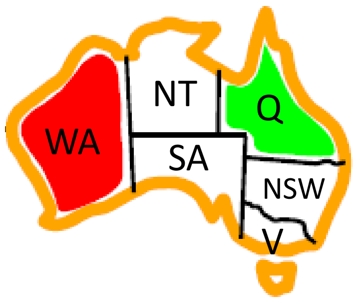


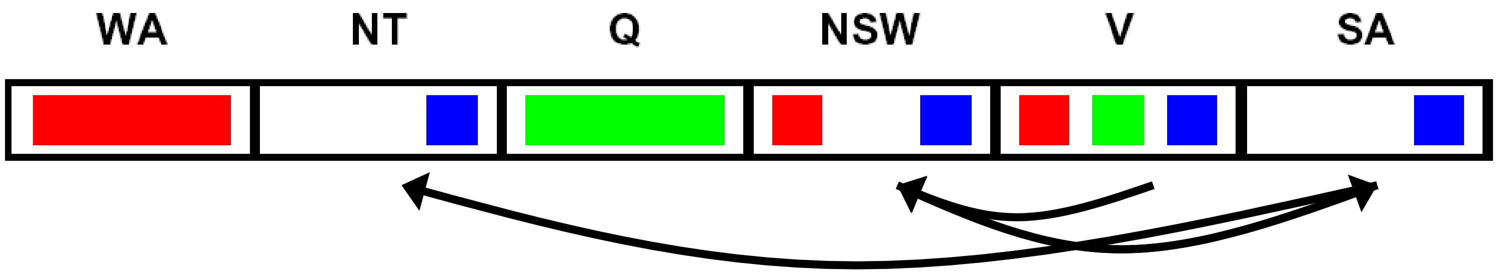
Delete from the tail!

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

Arc Consistency of an Entire CSP (Finding 4)

- 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 (finding 4 vs. 8)
- Can be run as a preprocessor or after each assignment
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

AC-3: Enforcing Arc Consistency in a CSP

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

tail head

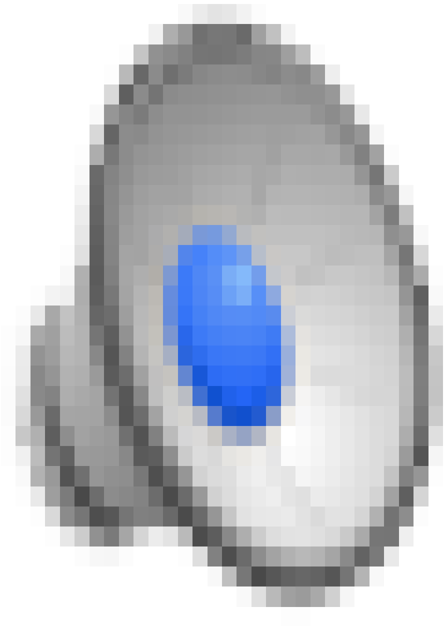
```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi]; removed ← true
   return removed
```

The “delete from trunk” function.

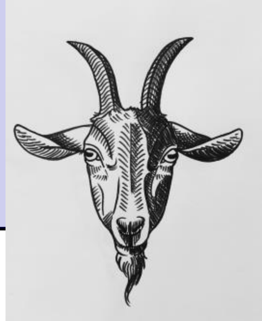
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

[Demo: CSP applet (made available by aispace.org) -- n-queens]

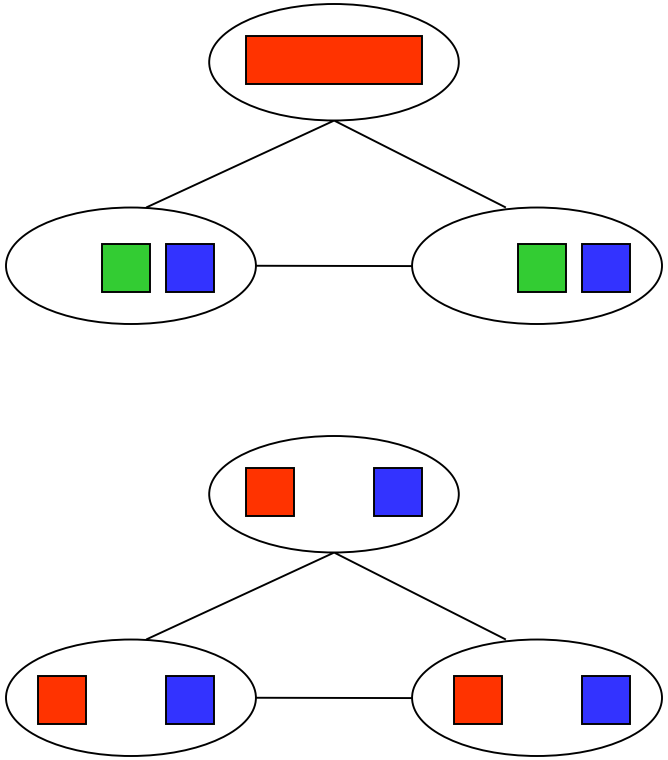
Video of Demo Arc Consistency – CSP Applet – n Queens



Limitations of Arc Consistency



- For the graphs to the right:
  - Are they arc consistent?
  - How many solutions are there to the CSP?



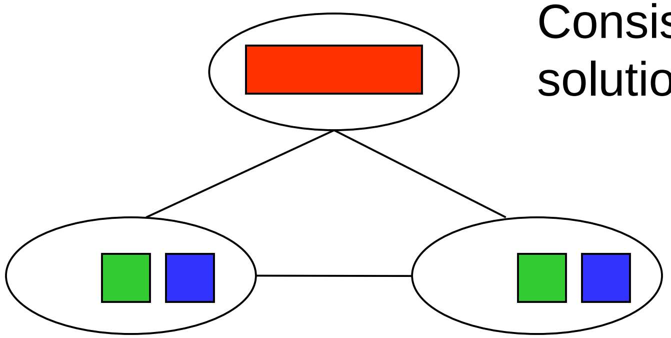
What went wrong here?

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

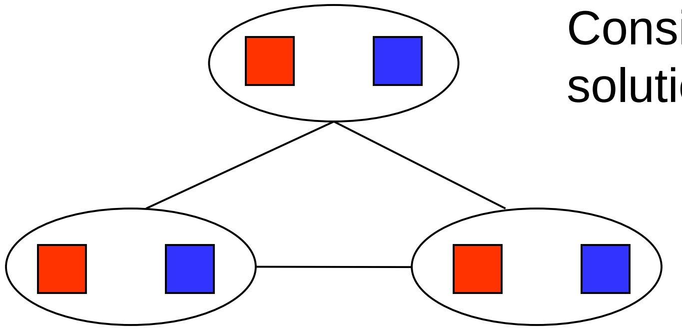
Limitations of Arc Consistency

- 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 CSP-backtracking search!



Consistent, 2 solutions

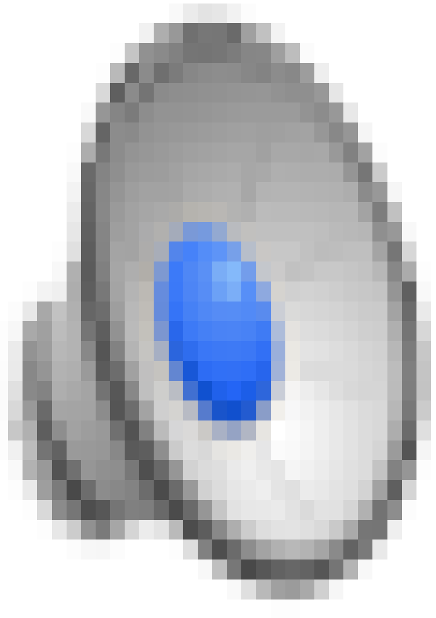


Consistent, no solutions

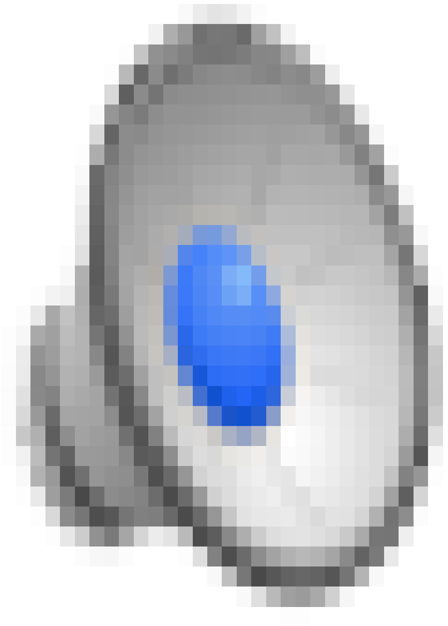
What went wrong here?

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

Coloring Demo Video – CSP-Backtracking with Forward Checking on a Complex Graph



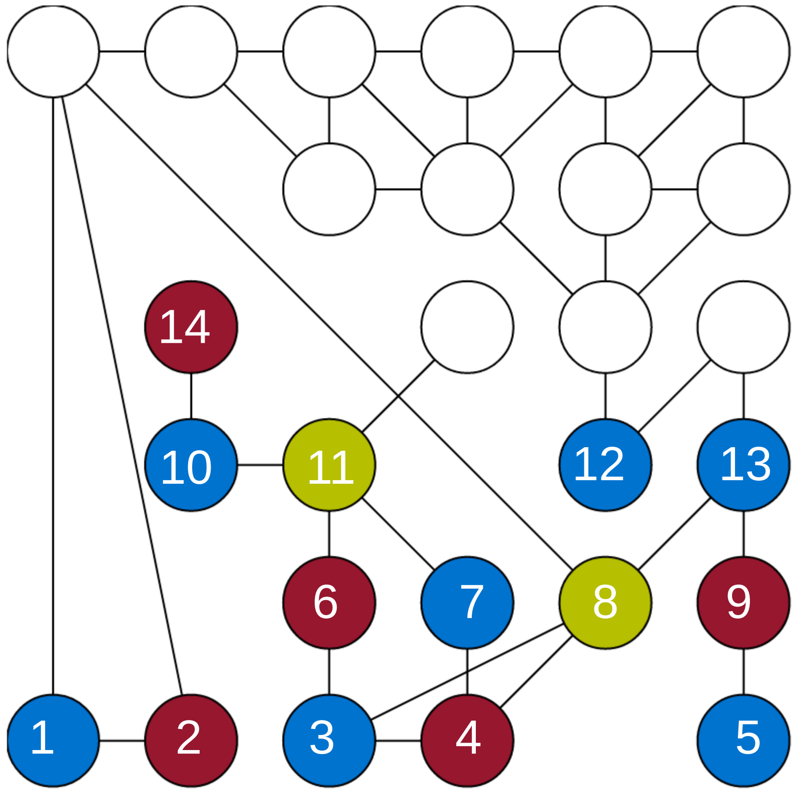
Coloring Demo Video – CSP-Backtracking with Arc Consistency – Complex Graph



Improving CSP-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?
  - Yes! Use forward checking or arc consistency (e.g. AC-3).
  - AC-3 more expensive, but catches problems earlier.
- Structure: Can we exploit the problem structure?

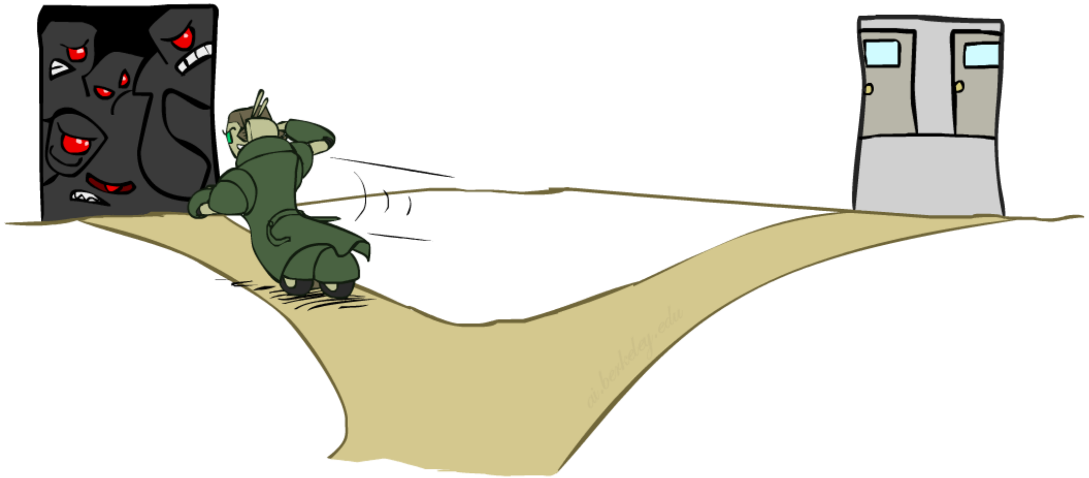


Ordering

Ordering: Minimum Remaining Values

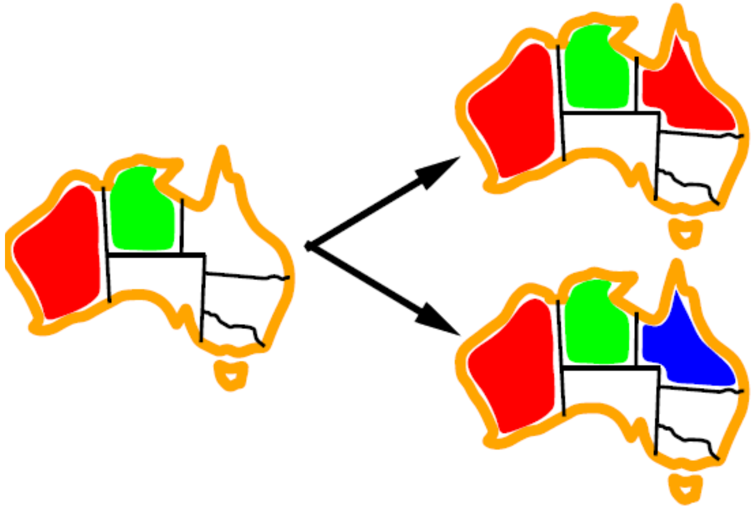
- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values remaining in its domain

- Why min rather than max?
- Also called "most constrained variable"
- "Fail-fast" ordering

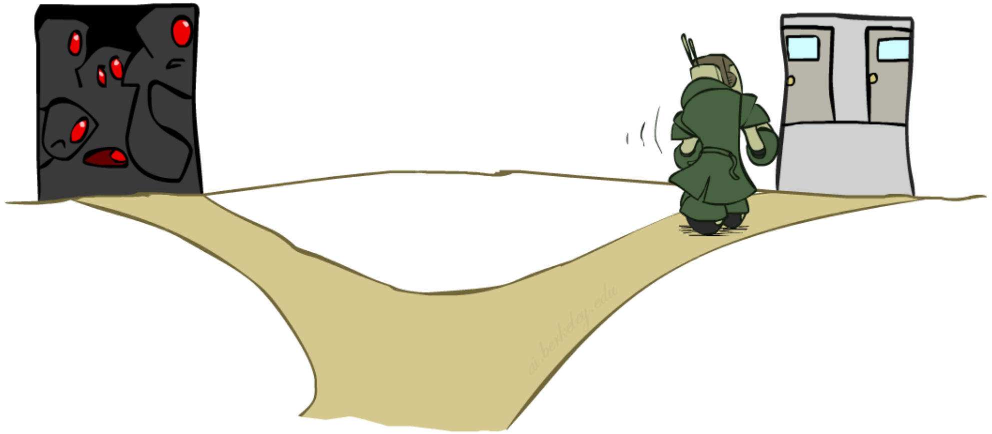


Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - After picking a variable, which value should we pick? 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?
  - Don’t have to try every value! Use the one most likely to work.



- Combining these ordering ideas makes 1000 queens feasible

Demo: Coloring – CSP-Backtracking + Forward Checking + Ordering