Lecture 4: Constraint Satisfaction Problems

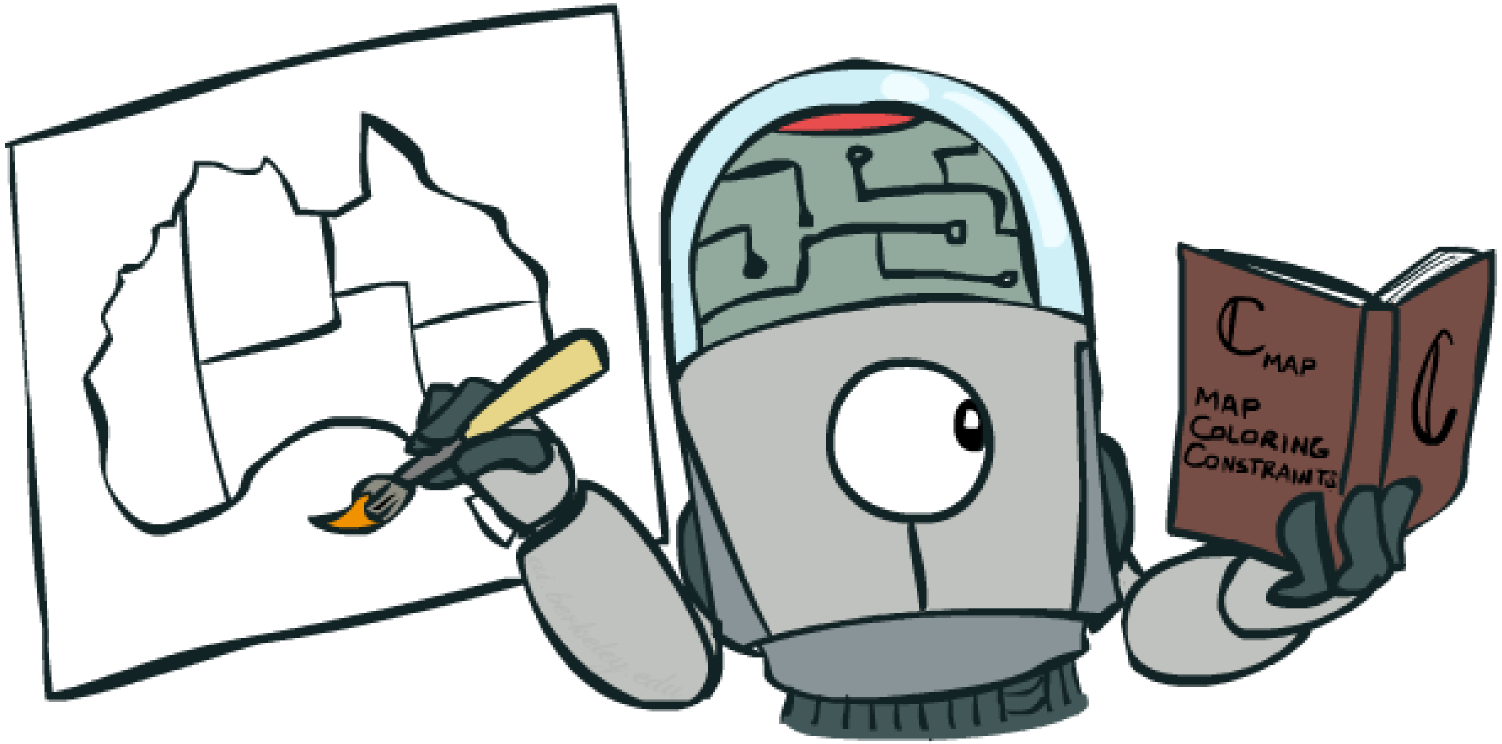
Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

https://shuaili8.github.io/Teaching/CS410/index.html

Part of slide credits: CMU AI & http://ai.berkeley.edu



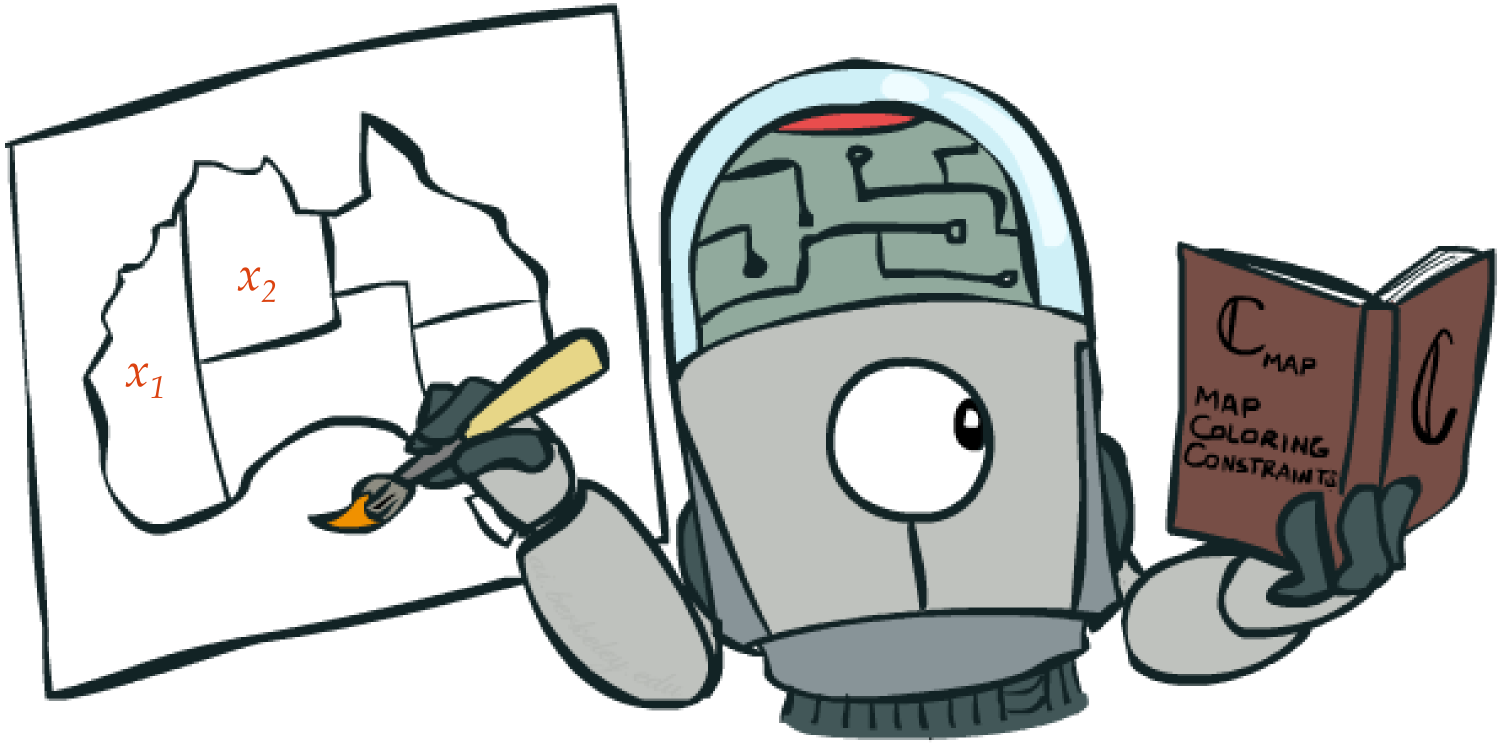
Constraint Satisfaction Problems

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?

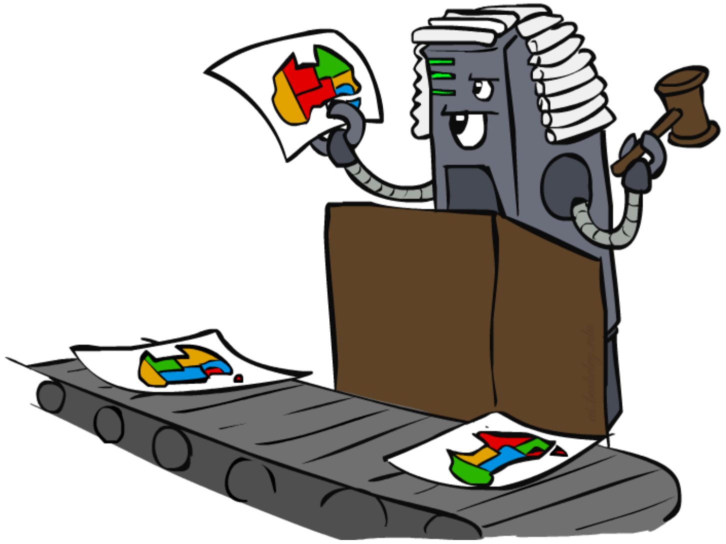
- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

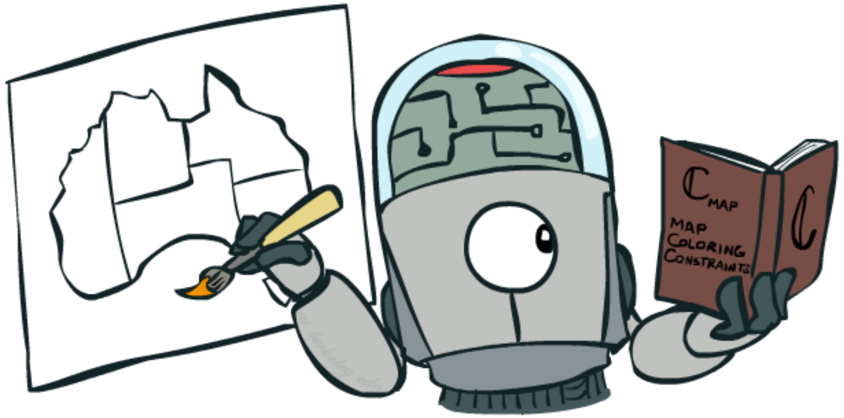
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

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



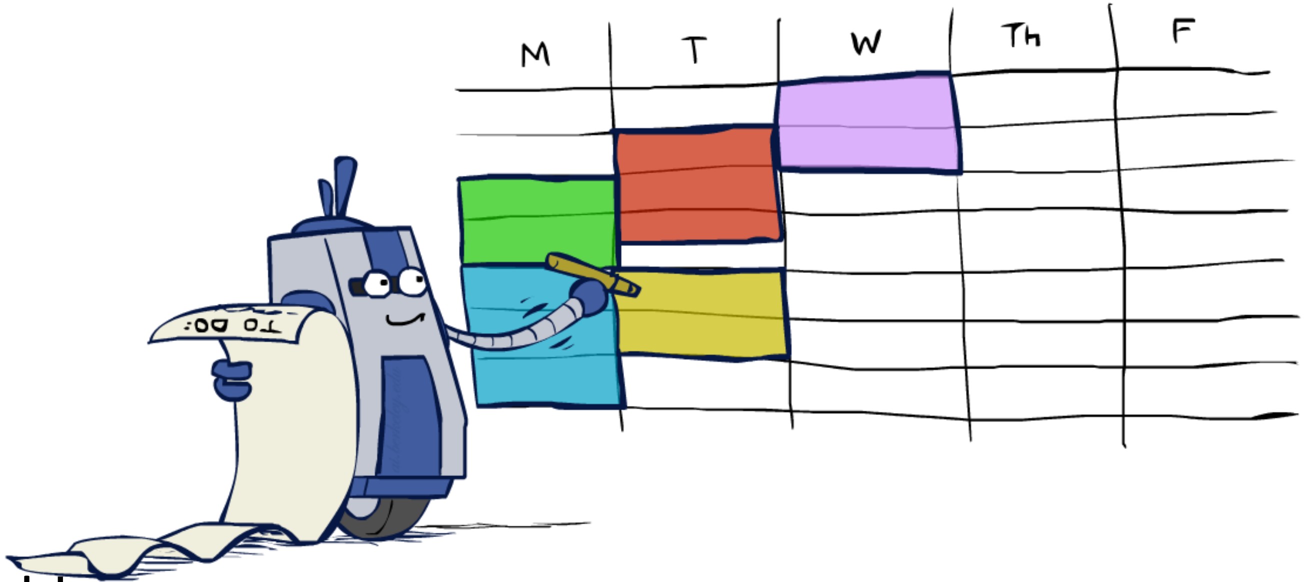
- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables



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

Why study CSPs?

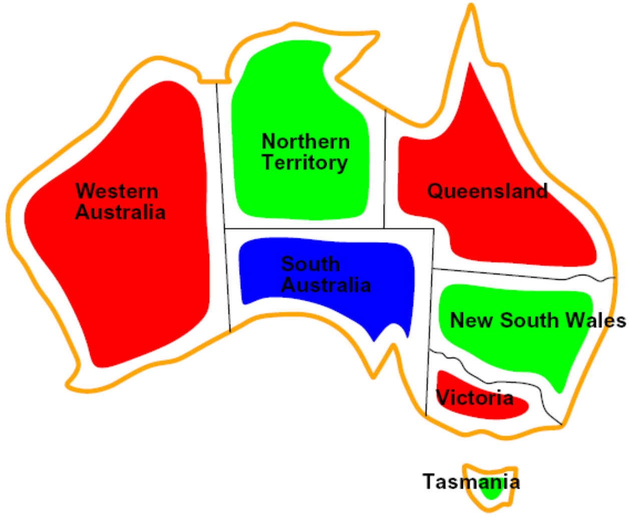
- Many real-world problems can be formulated as CSPs
- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!



- Sometimes involve real-valued variables…

Example: Map Coloring

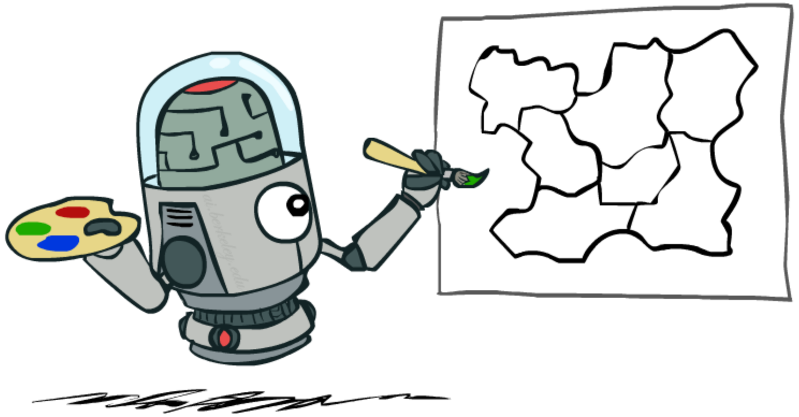
- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: D={red, green, blue}



- Constraints: adjacent regions must have different colors:
  - Implicit: WA≠NT
  - Explicit: (WA,NT)∈{(red, green), (red, blue), …}

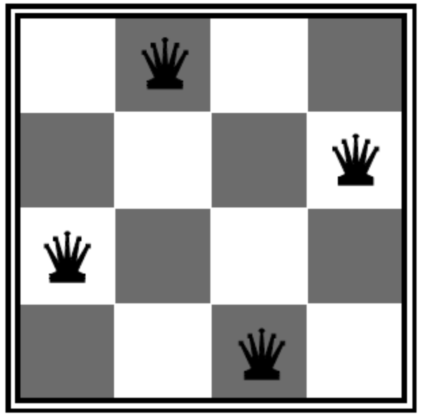
- Solutions are assignments satisfying all constraints, e.g.:

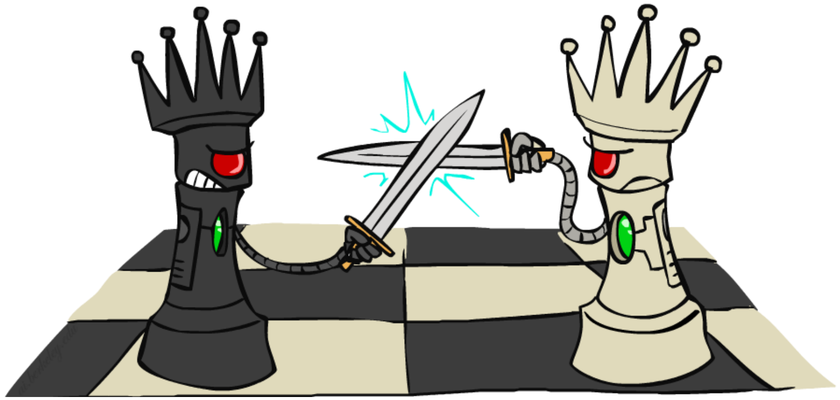
  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}



Example: N-Queens

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

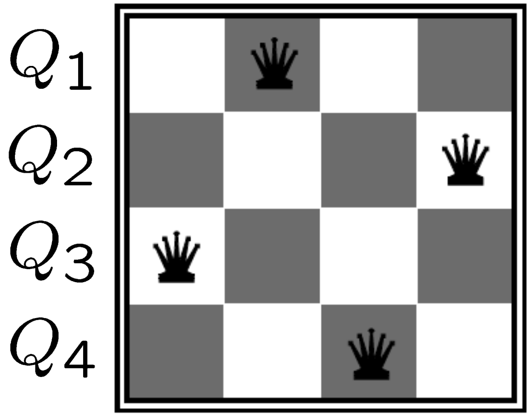




$$\forall i,j,k \;\; (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \;\; (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$

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

Example: N-Queens 2



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

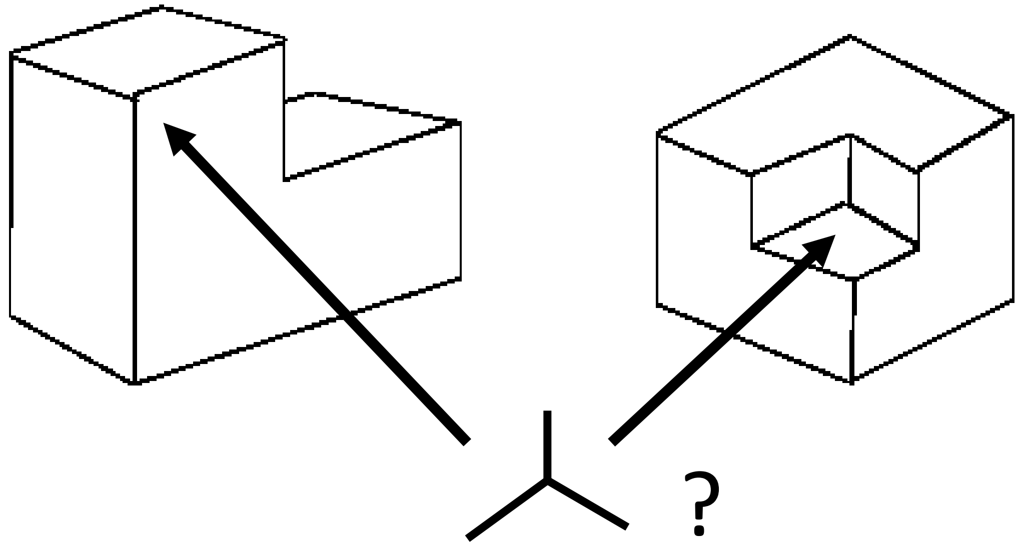
Implicit: $\forall i, j \;\; \text{non-threatening}(Q_i, Q_j)$

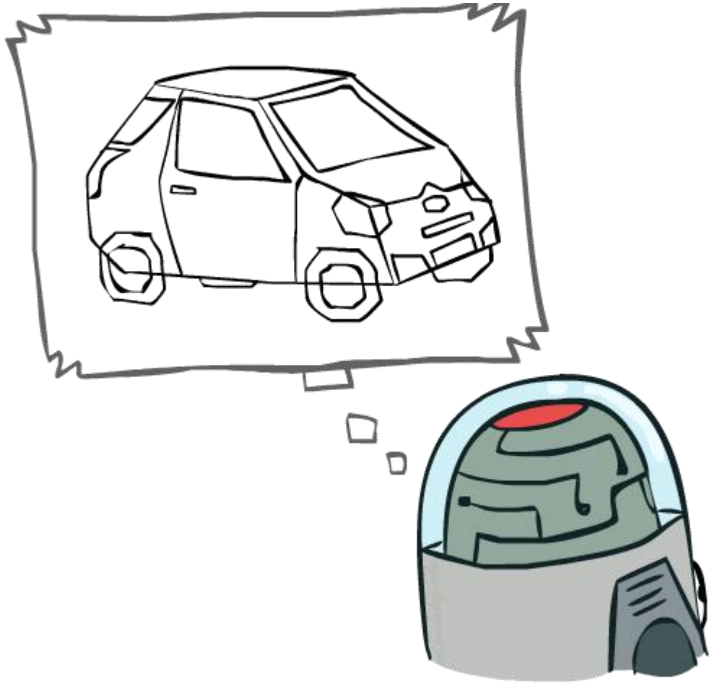
Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

$\cdots$

Example: 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

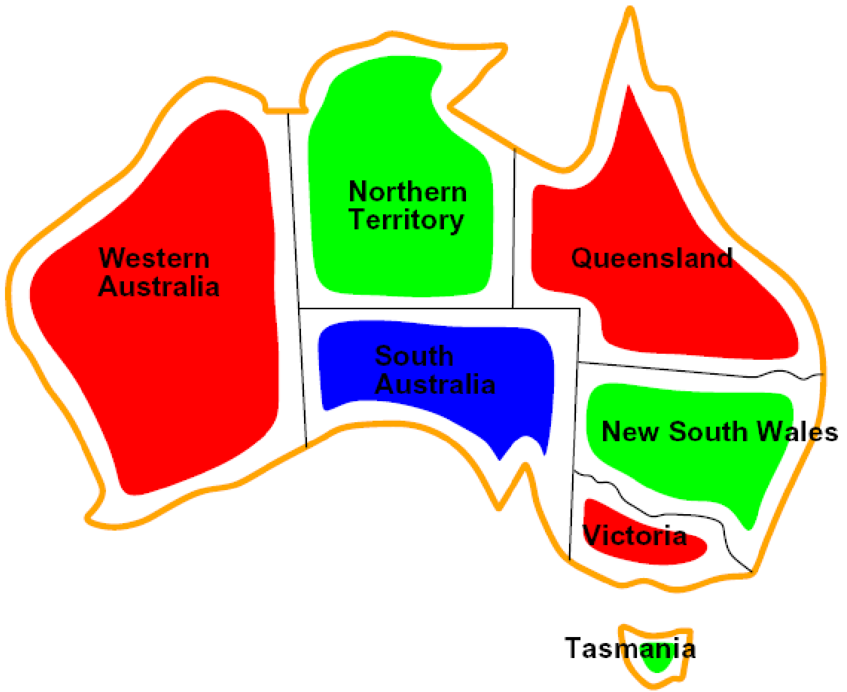


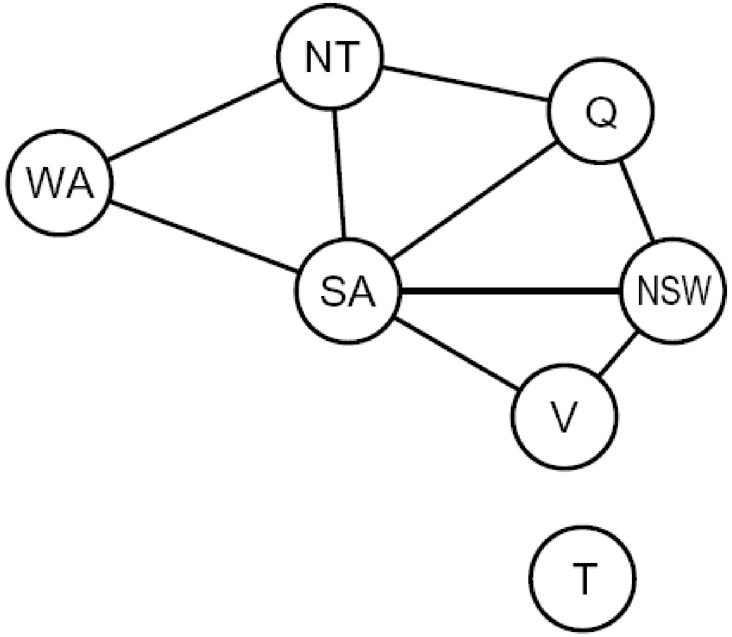


- Approach:
  - Each intersection is a variable
  - Adjacent intersections impose constraints on each other
  - Solutions are physically realizable 3D interpretations

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:

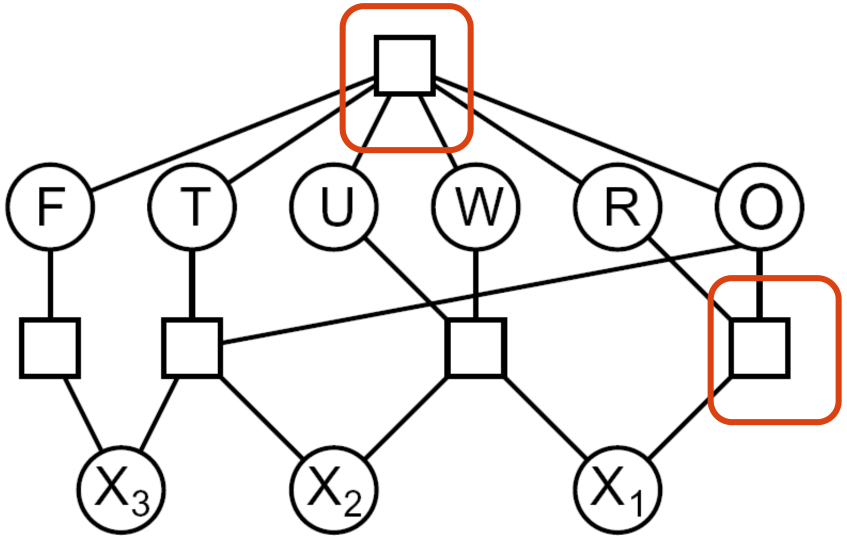
  $\boxed{\text{alldiff}(F,T,U,W,R,O)}$

  $\boxed{O + O = R + 10 \cdot X_1}$

  $\ldots$

X_1

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



Example: Sudoku

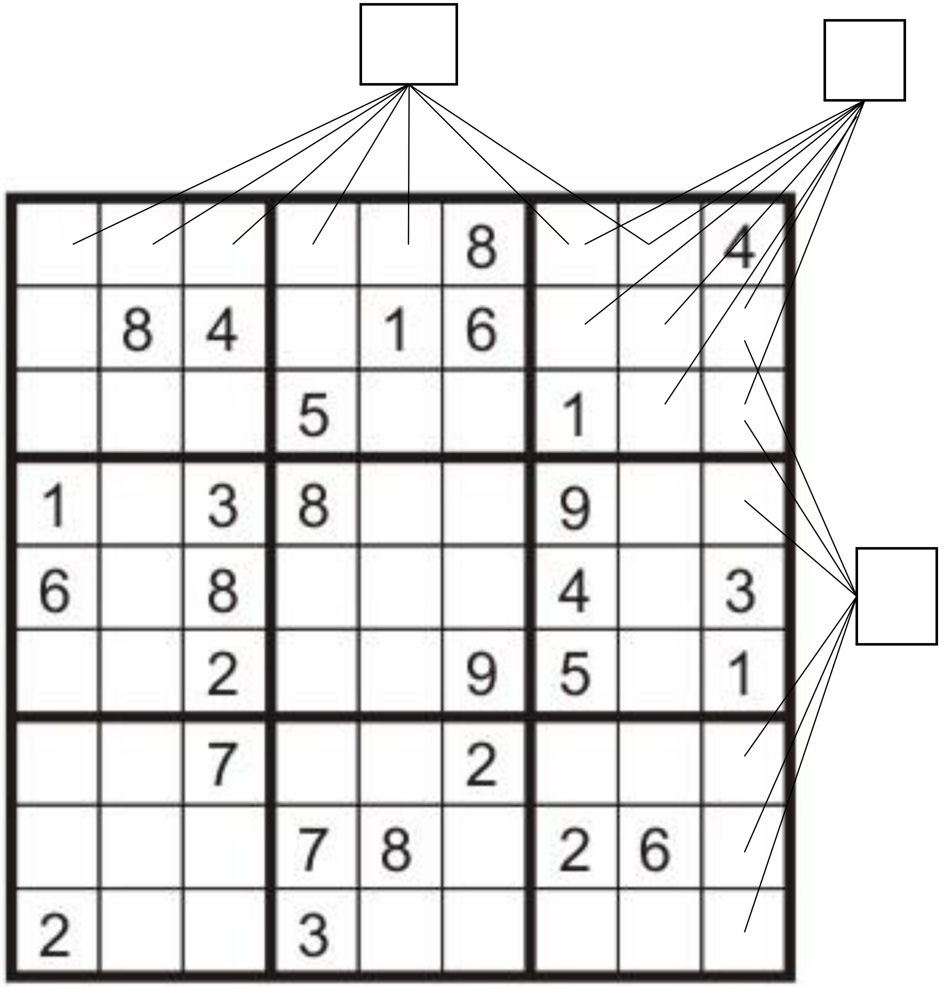
- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

  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)

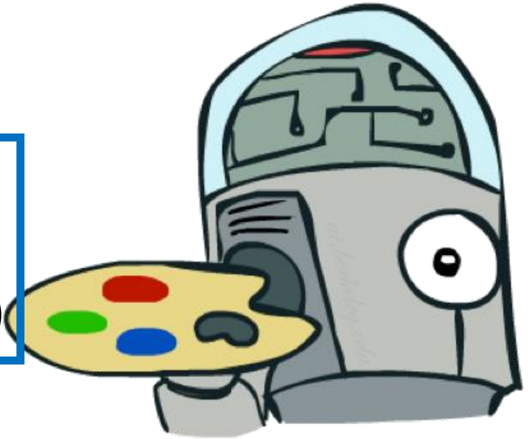


Varieties of CSPs

- Discrete Variables
  We will cover in this lecture
  - 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

Related with linear programming

- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods



Varieties of Constraints 2

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

    $$SA \neq green$$

    Focus of this lecture

  - 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)

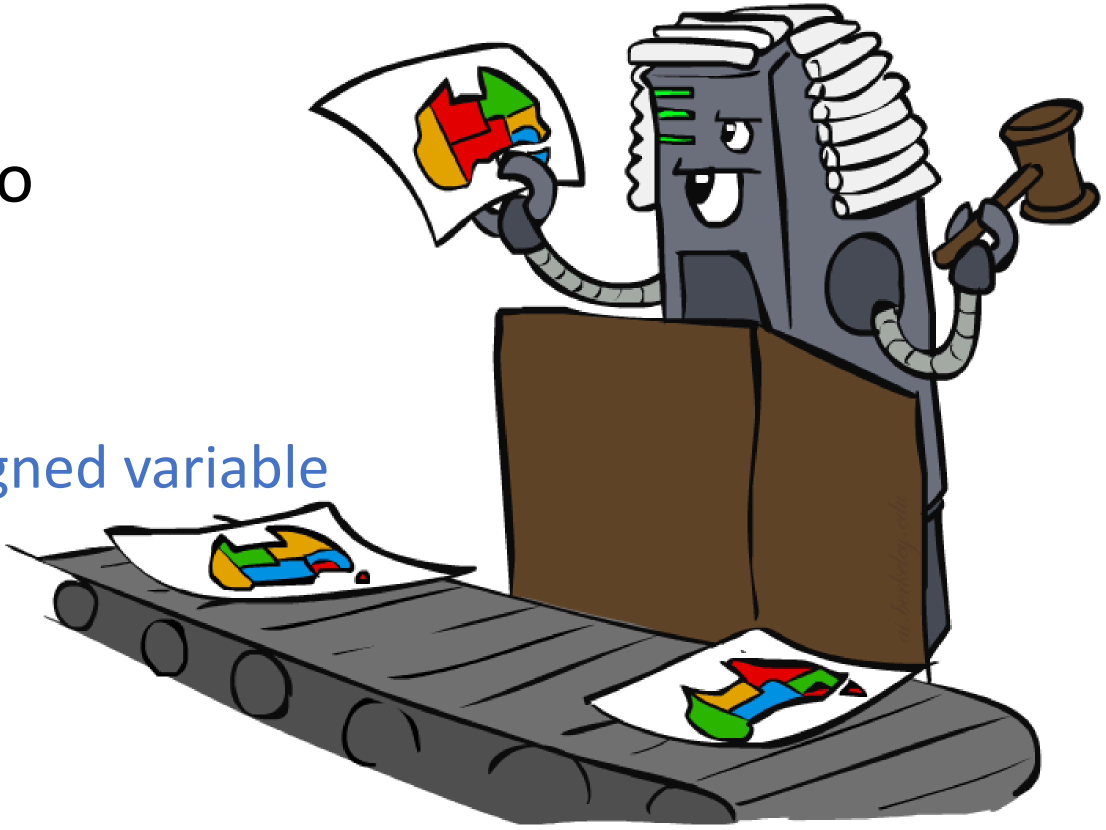
Solving CSPs

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 →Can be any unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

- We'll start with the straightforward, naïve approach, then improve it

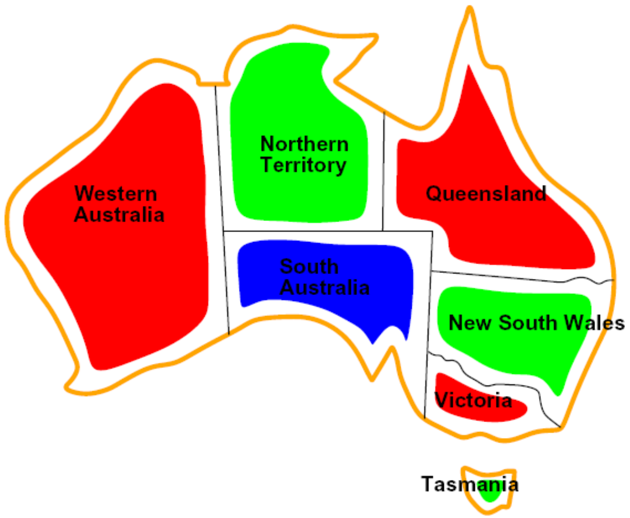


Search Methods: BFS

- What would BFS do?

$\{\}$

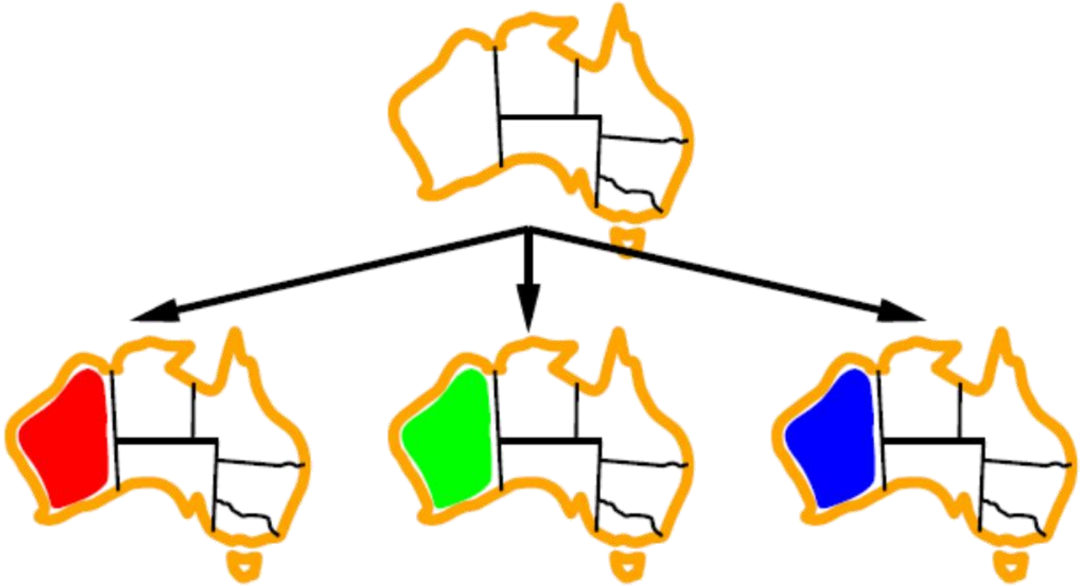
$\{WA{=}g\}$ $\{WA{=}r\}$ … $\{NT{=}g\}$ …



[Demo: coloring -- dfs]

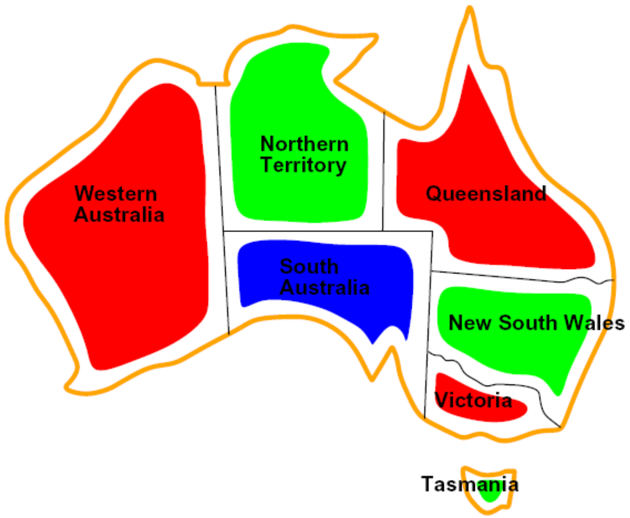
Search Methods: DFS

- At each node, assign a value from the domain to the variable
- Check feasibility (constraints) when the assignment is complete



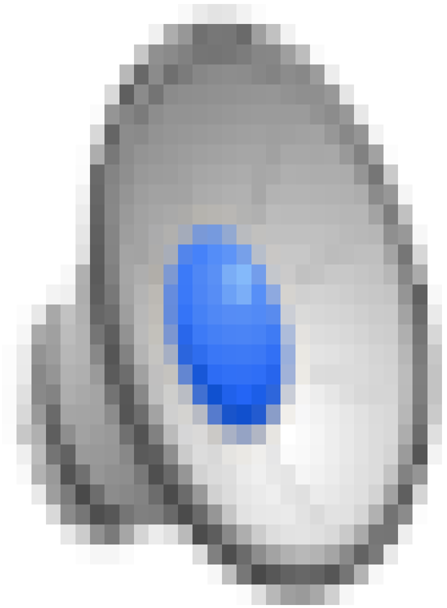
Search Methods

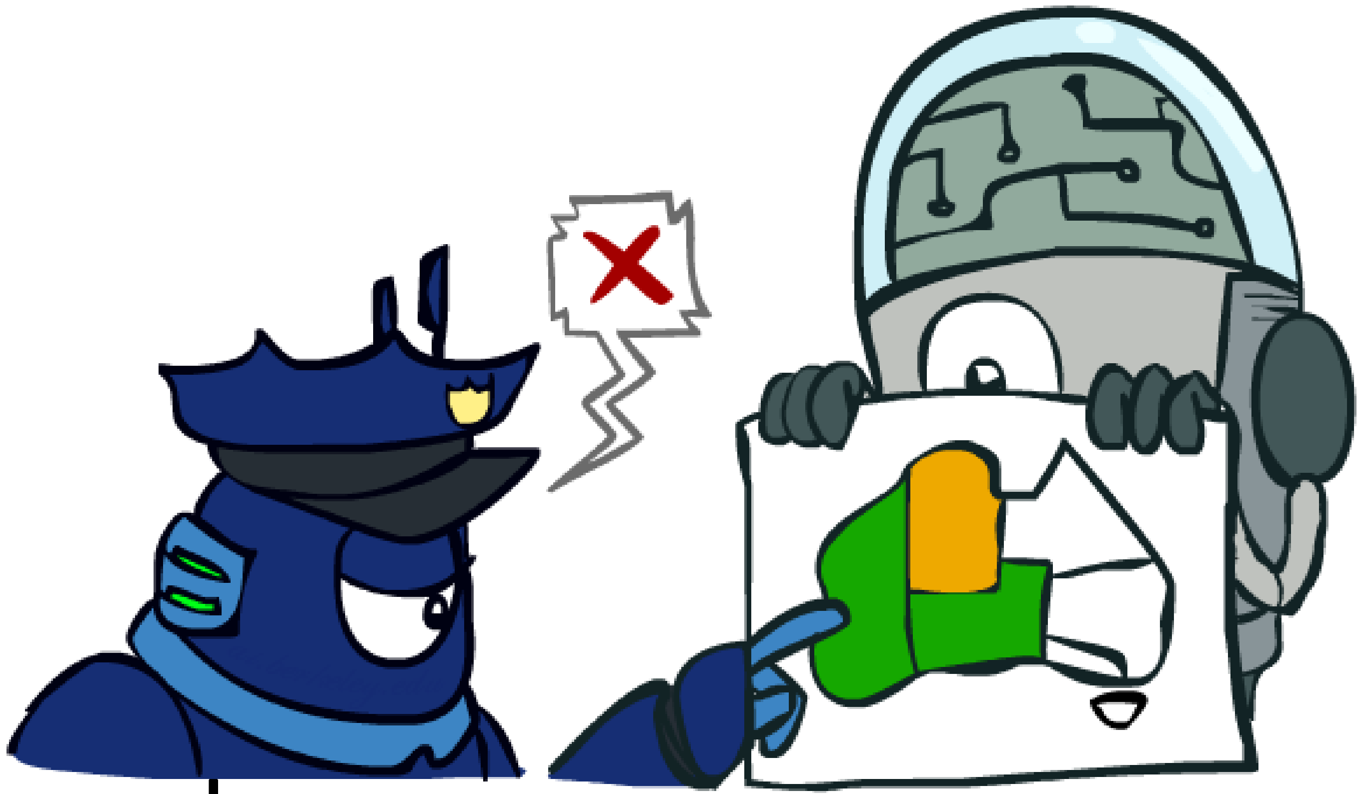
- What would BFS do?
- What would DFS do?
  - let's see!
- What problems does naïve search have?



[Demo: coloring -- dfs]

Video of Demo Coloring -- DFS

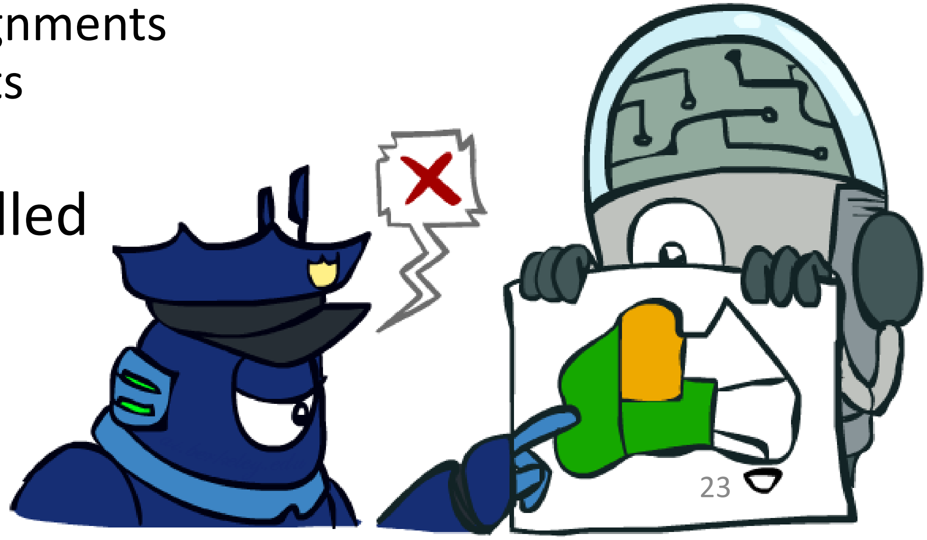




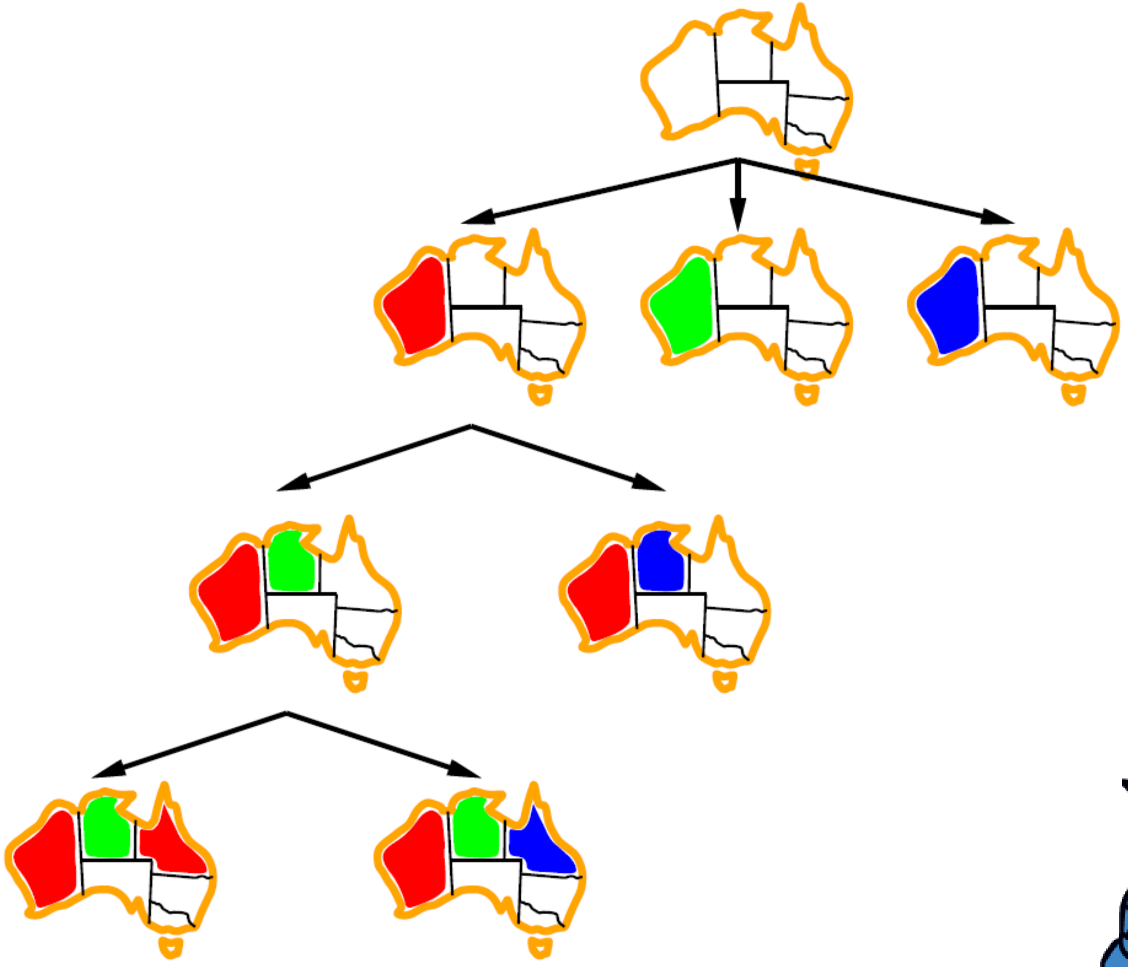
Backtracking Search

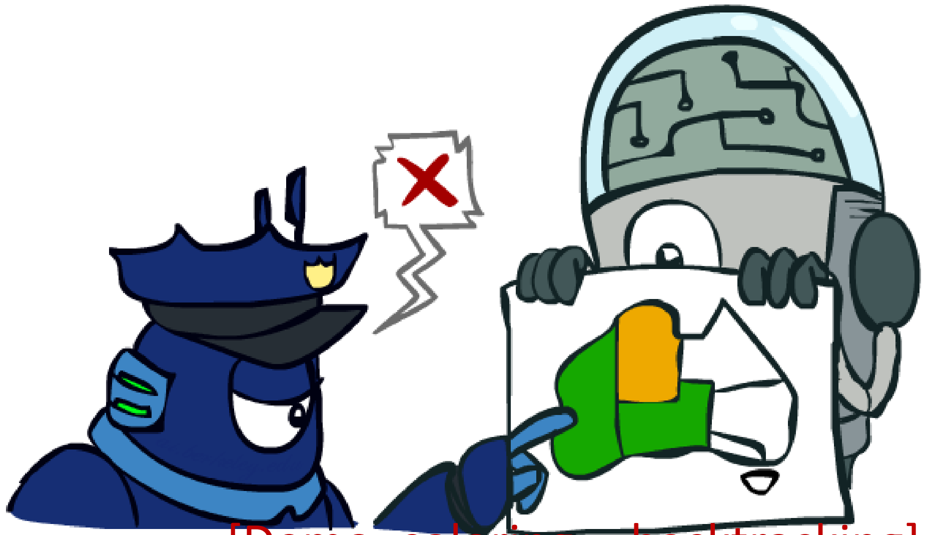
Backtracking Search

- 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 $n \approx 25$



Example





[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
```

No need to check consistency for a complete assignment

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
```

Checks consistency at each assignment

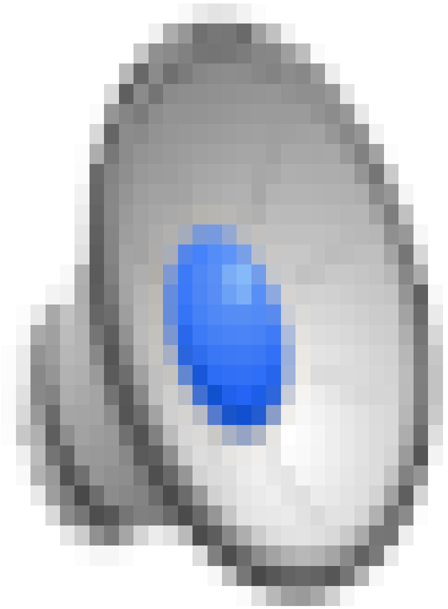
Backtracking Search

```
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?

Video of Demo Coloring – Backtracking

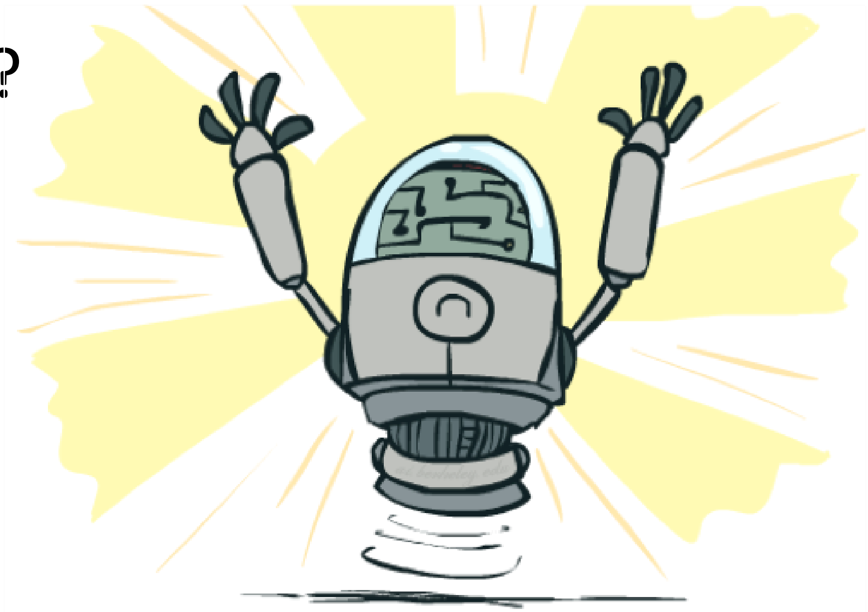


Improving Backtracking

- 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?



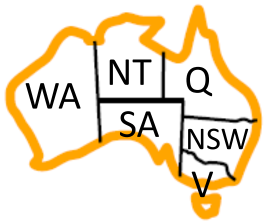
- Structure: Can we exploit the problem structure?

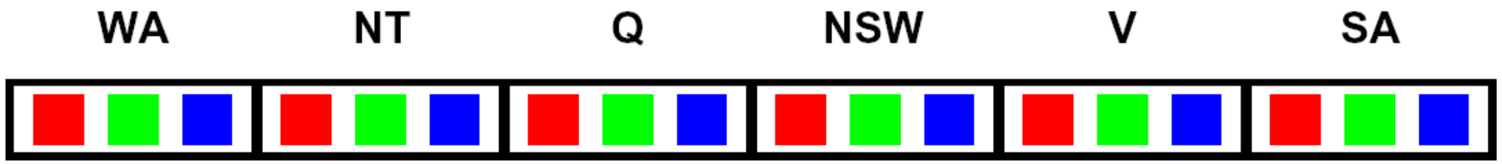
Filtering

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

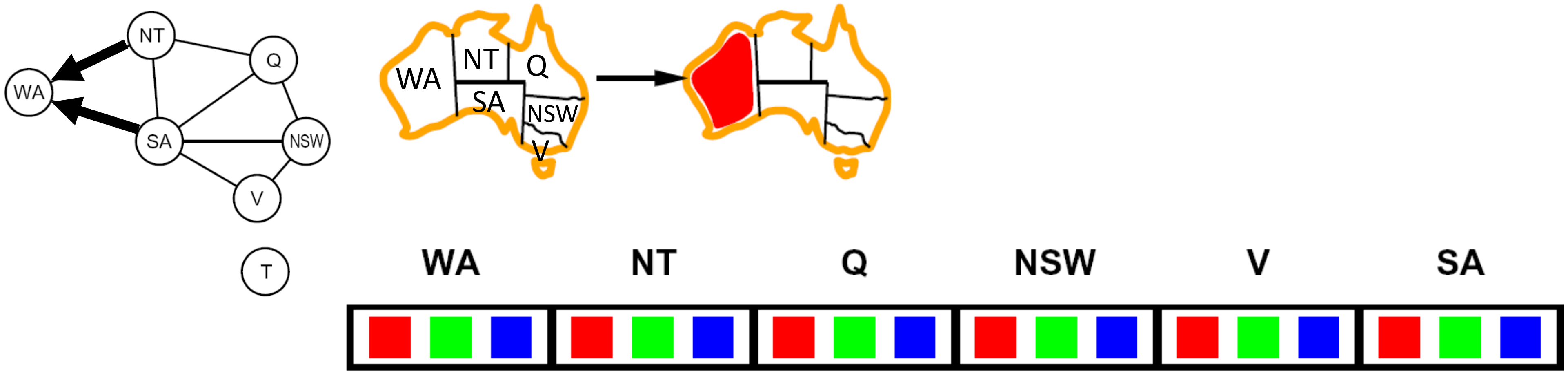




[Demo: coloring -- forward checking]

Filtering: Forward Checking

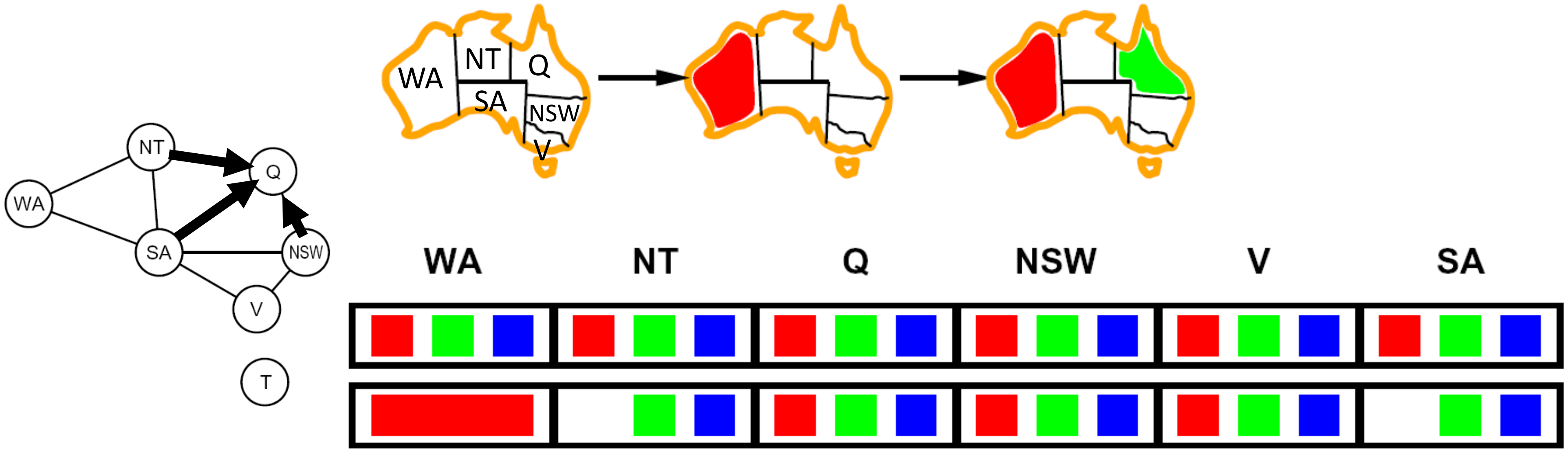
- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment



[Demo: coloring -- forward checking]

Filtering: Forward Checking

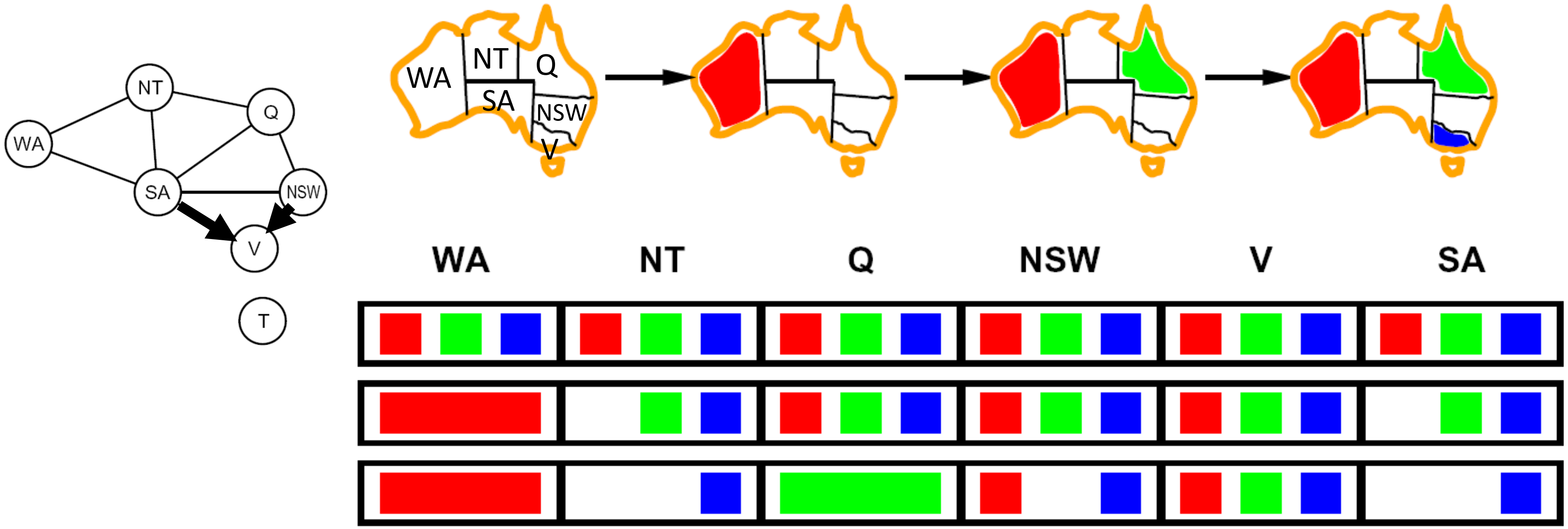
- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment



[Demo: coloring -- forward checking]

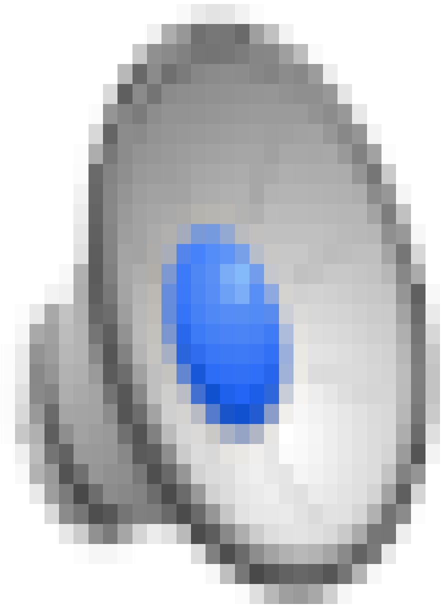
Filtering: Forward Checking

- 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



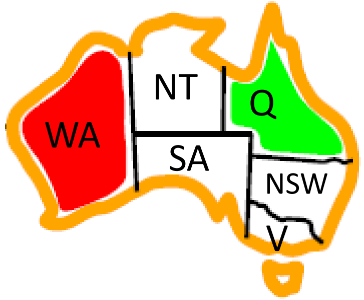
[Demo: coloring -- forward checking]

Video of Demo Coloring – Backtracking with Forward Checking



Filtering: Constraint Propagation

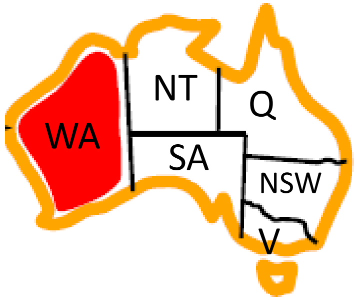
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

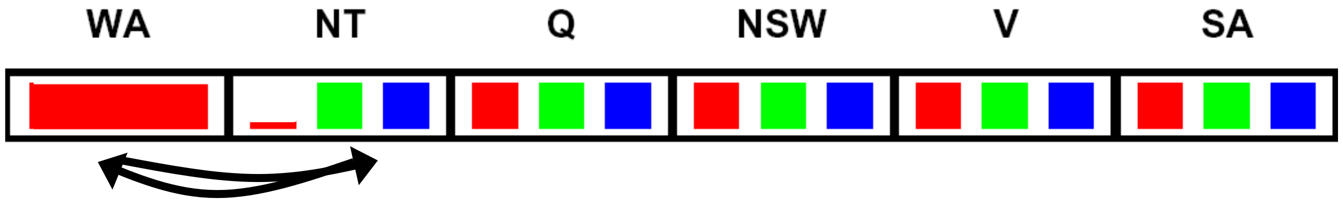


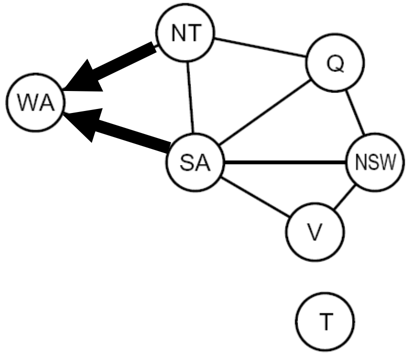
- NT and SA cannot both be blue!
- Why didn’t we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

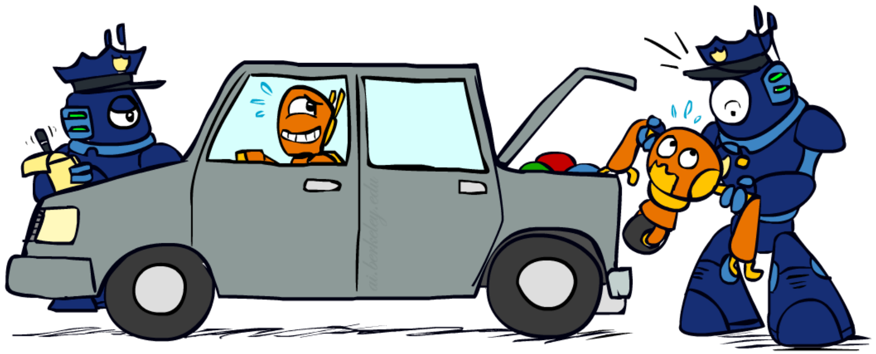
Consistency of A Single Arc

- 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









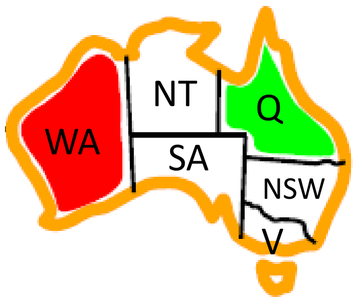
Delete from the tail!

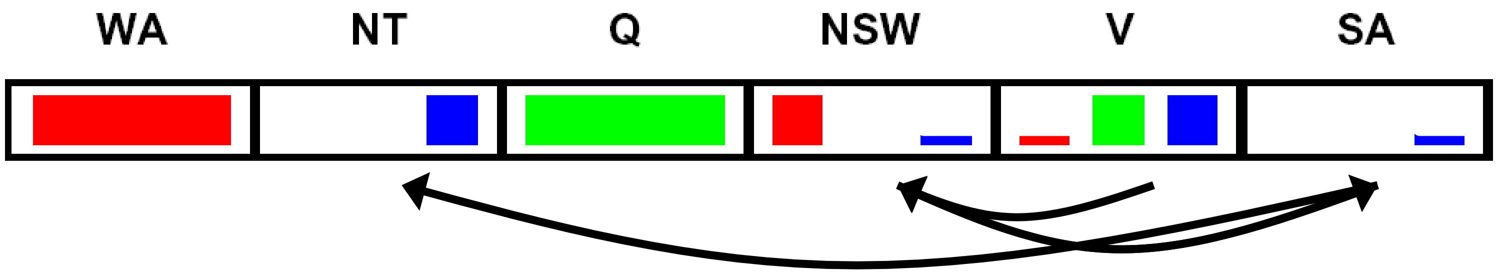
Forward checking?

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:



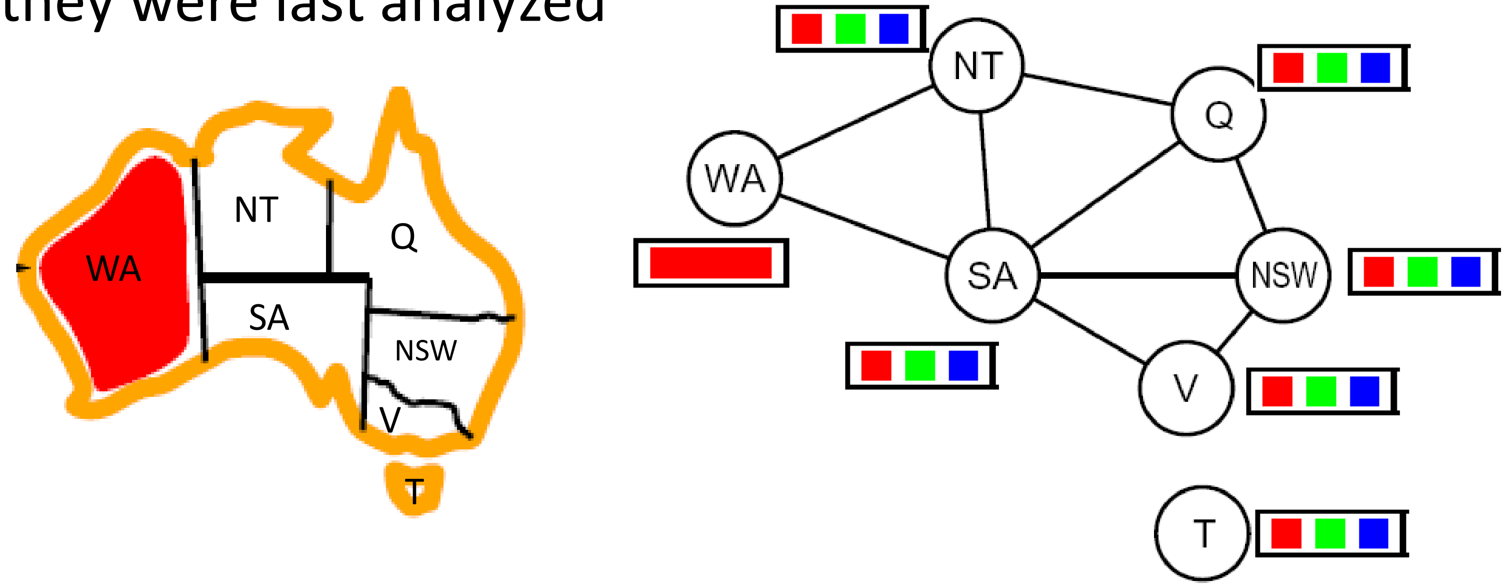


- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

How to Enforce Arc Consistency of Entire CSP

- A simplistic algorithm: Cycle over the pairs of variables, enforcing arc-consistency, repeating the cycle until no domains change for a whole cycle
- AC-3 (short for Arc Consistency Algorithm #3):
  - A more efficient algorithm ignoring constraints that have not been modified since they were last analyzed



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, ..., 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) ← REMOVE-FIRST(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 ← 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 ↔ X_j
            then delete x from DOMAIN[X_i];  removed ← true
    return removed
```

Constraint Propagation!

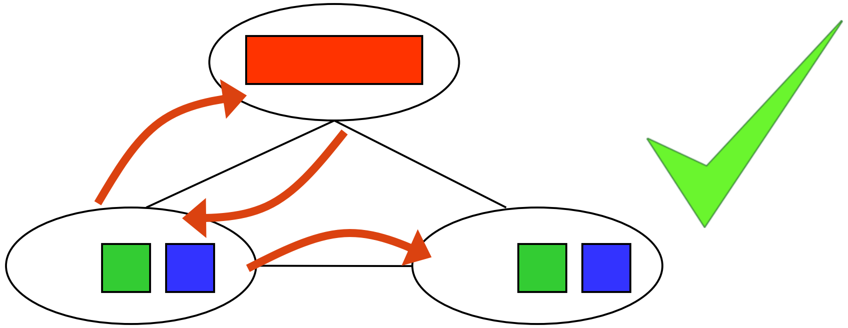
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

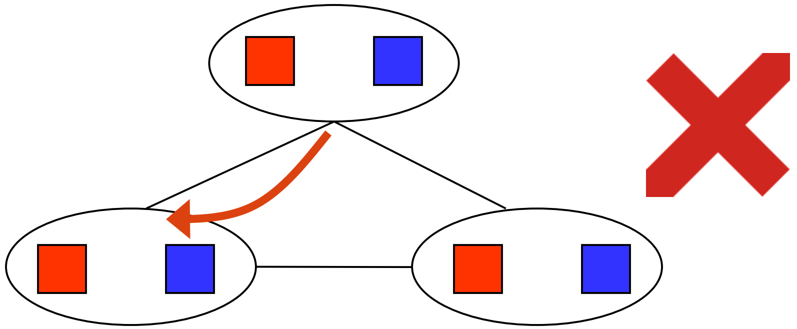
AC-3: Enforce Arc Consistency of Entire CSP

- Examples, next time!

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)

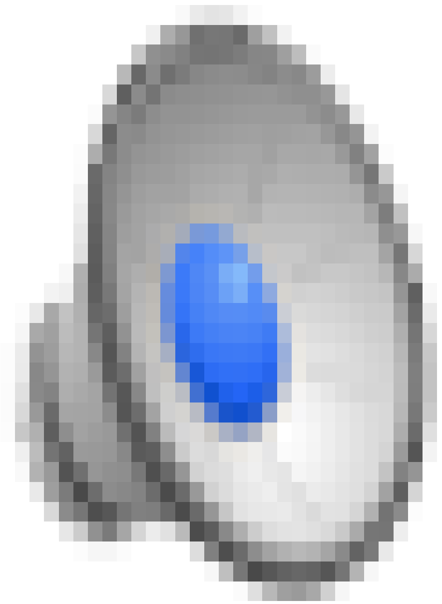




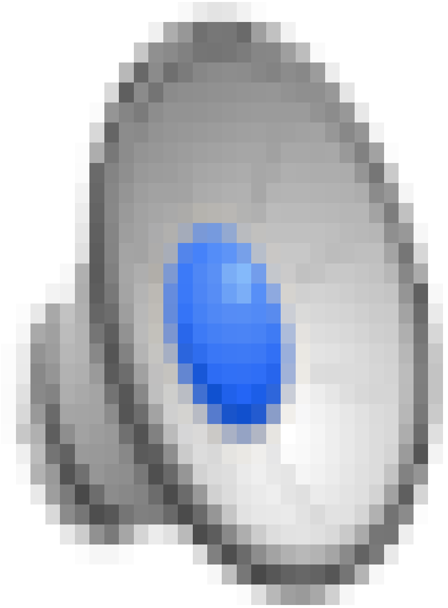
- Arc consistency still runs inside a backtracking search!

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

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

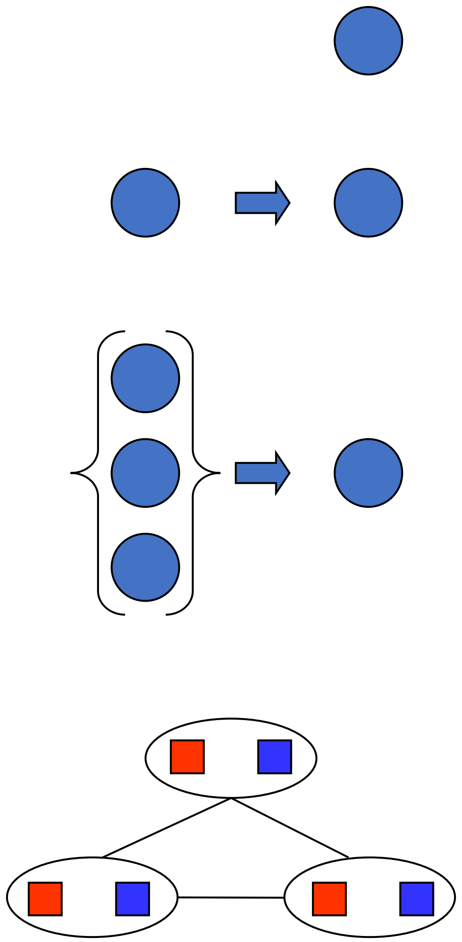


Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph



K-Consistency

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the k^{th} node.
- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)



Strong K-Consistency

- Strong k-consistency: also k-1, k-2, … 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - …

- Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)

Summary

- CSPs

Shuai Li

https://shuaili8.github.io

Questions?