Artificial Intelligence: Basics & Applications

Constraint Satisfaction Problems

Instructor: *Mahdi Javanmardi*

Amirkabir University of Technology
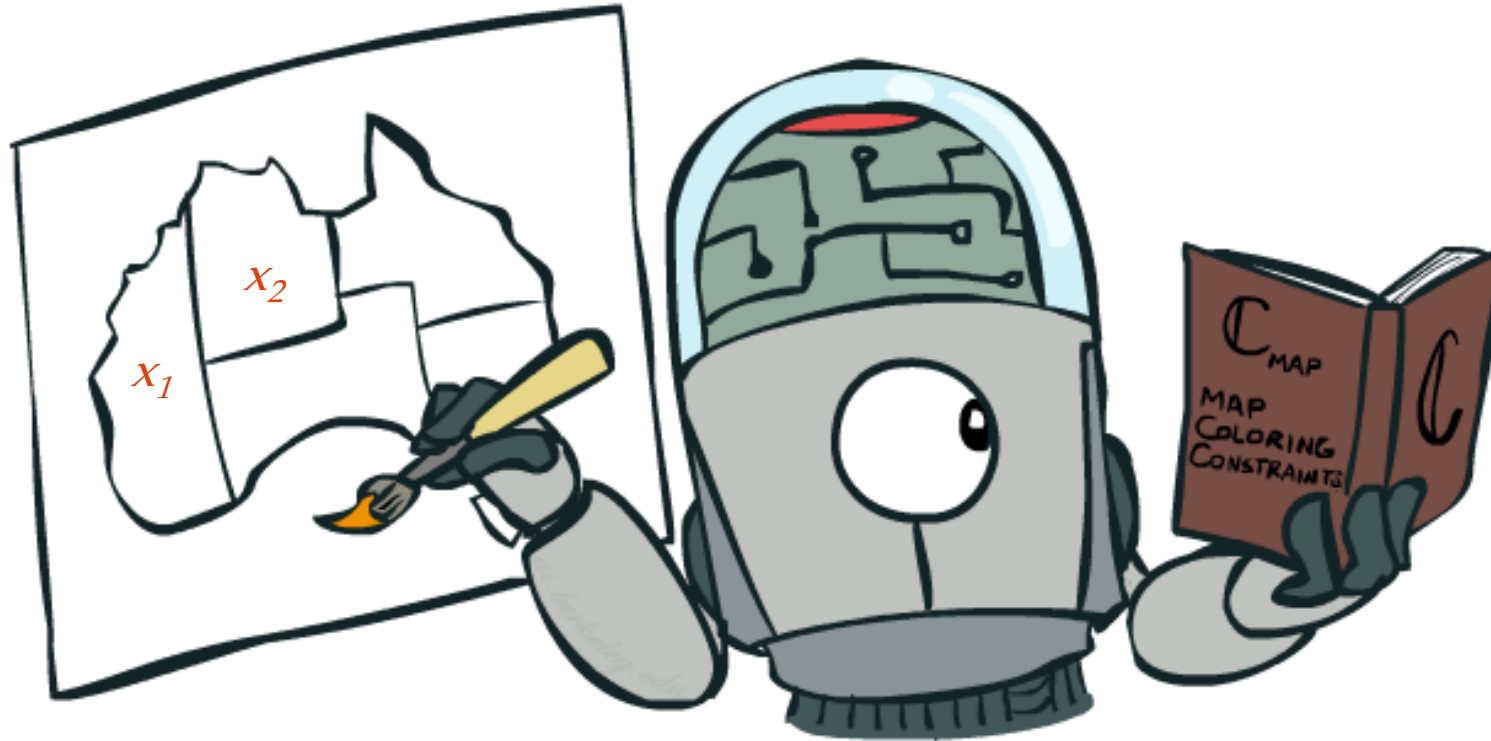
Constraint Satisfaction Problems

N variables
domain D
constraints

x_2

x_1

states
partial assignment

goal test
complete; satisfies constraints

*successor function*
assign an unassigned variable

What is Search For?

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

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

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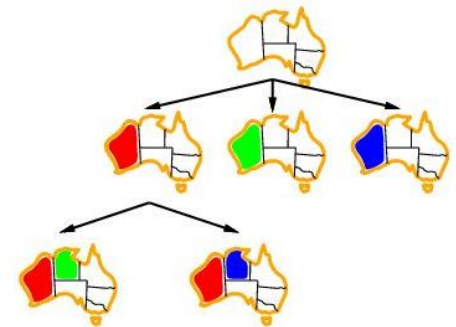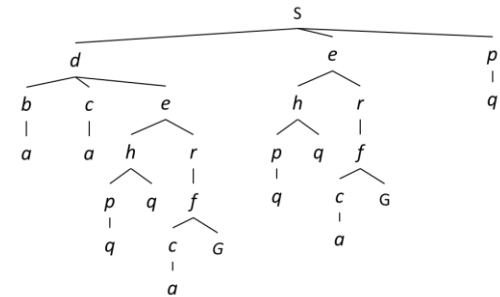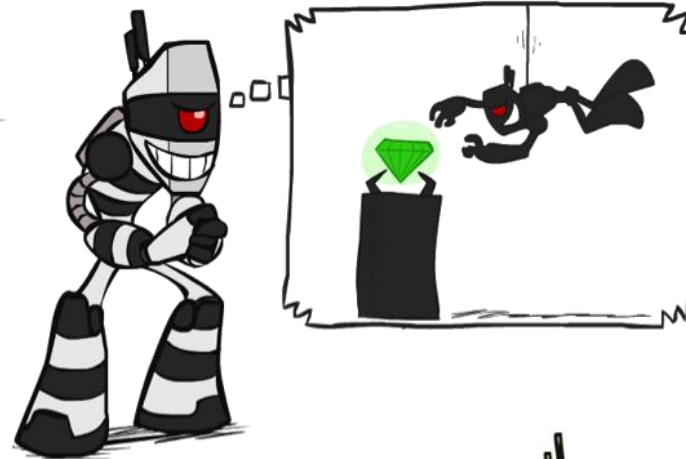
Constraint Satisfaction Problems

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

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

o Allows useful general-purpose algorithms with more power than standard search algorithms
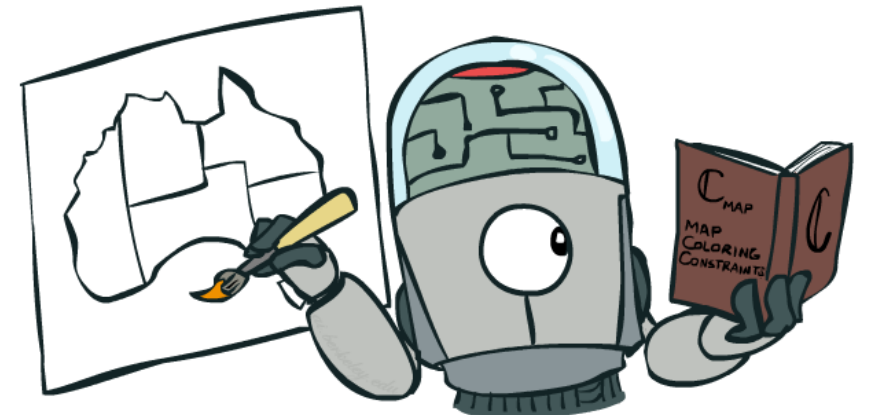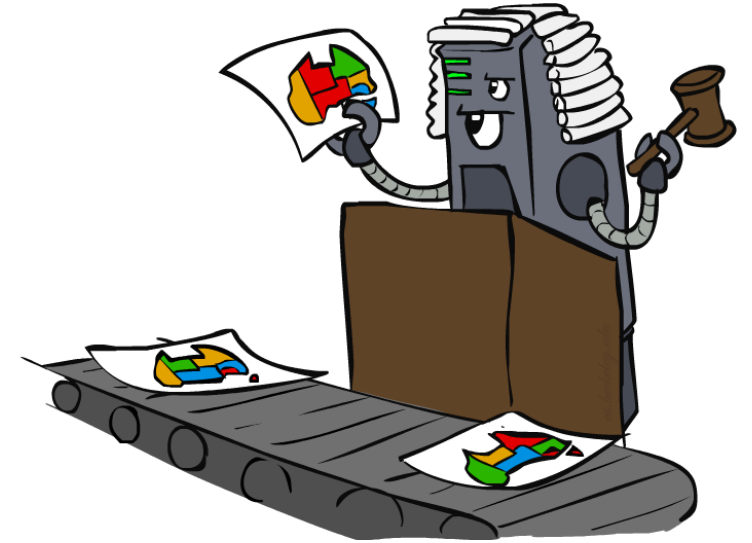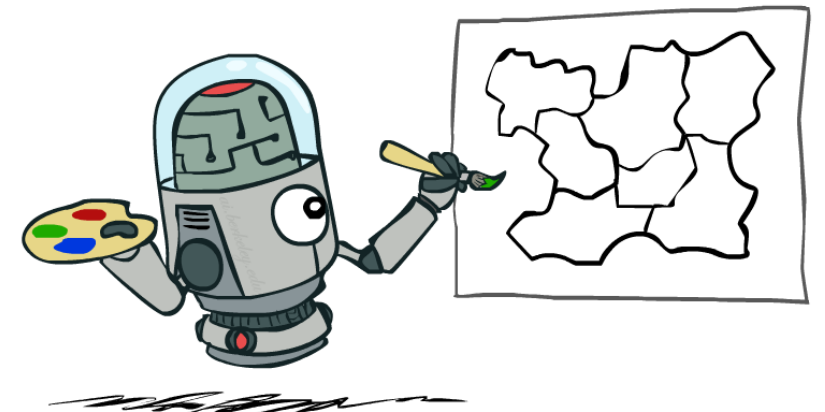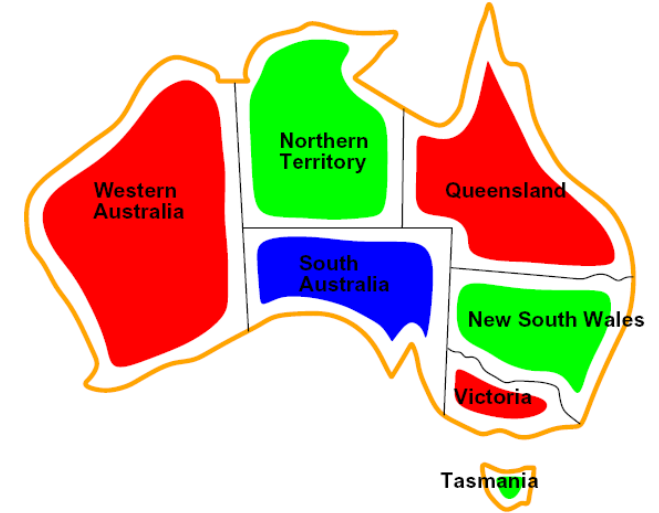
CSP Examples
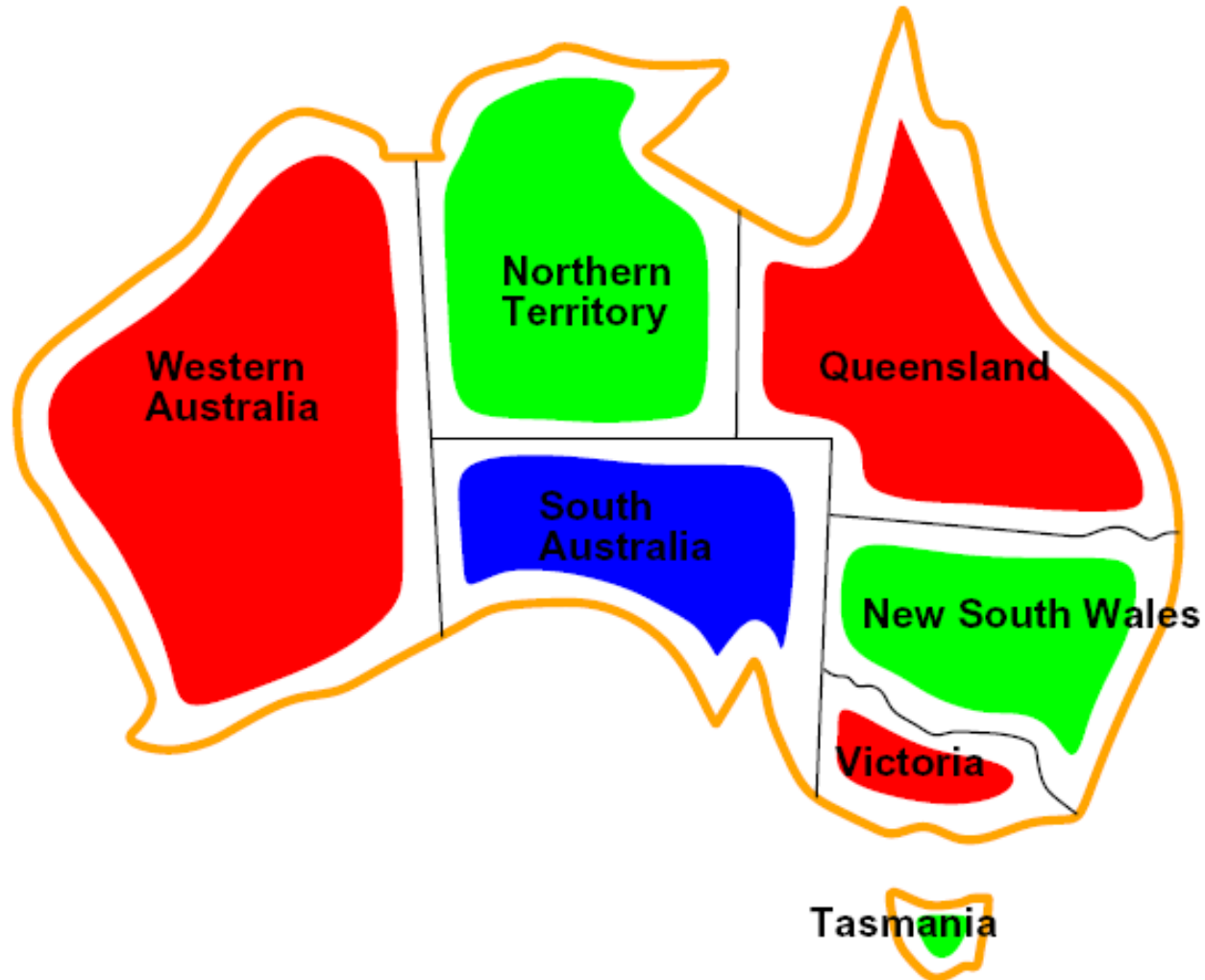
Example: Map Coloring

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

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

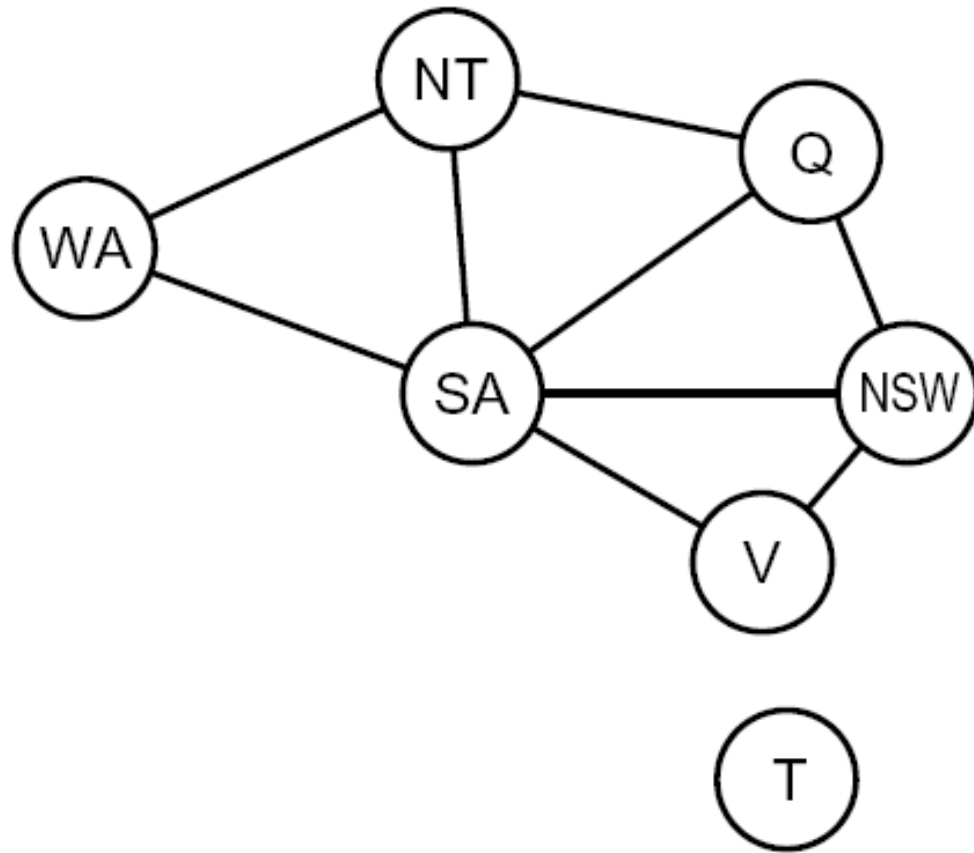o Constraints: adjacent regions must have different colors

   Implicit: $WA \neq NT$

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

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

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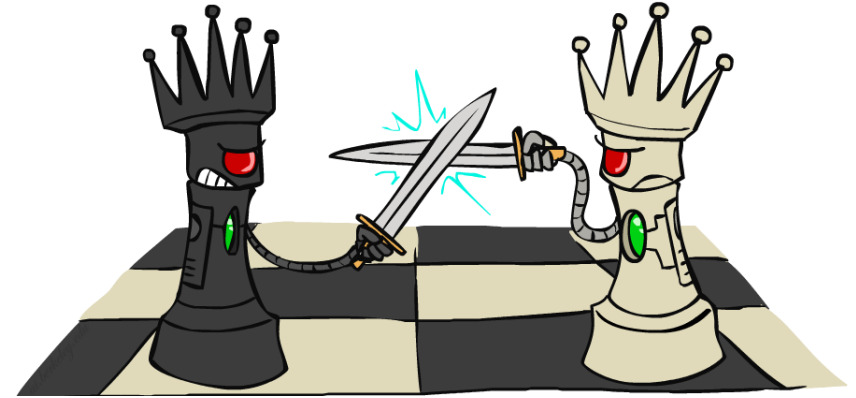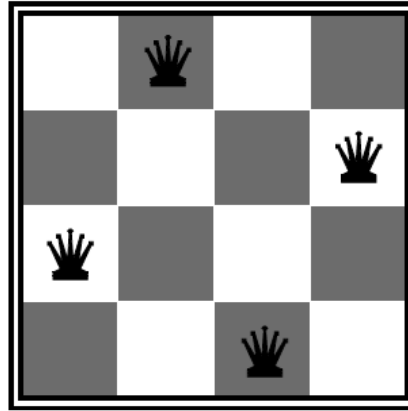
Constraint Graphs

Example: N-Queens

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

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

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

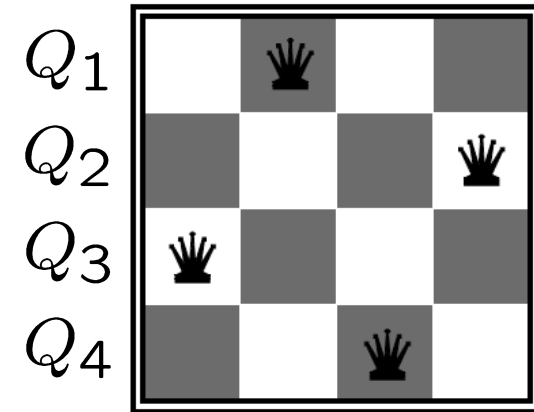# Example: N-Queens

o Formulation 2:

    o Variables: Q_k

    o Domains: $\{1, 2, 3, \ldots N\}$

    o Constraints:

Implicit: $\forall i, j$ 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:
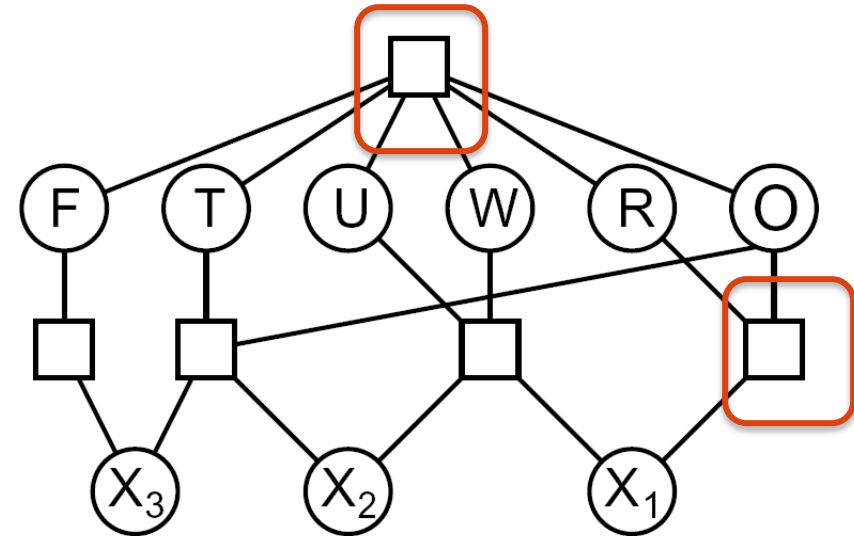
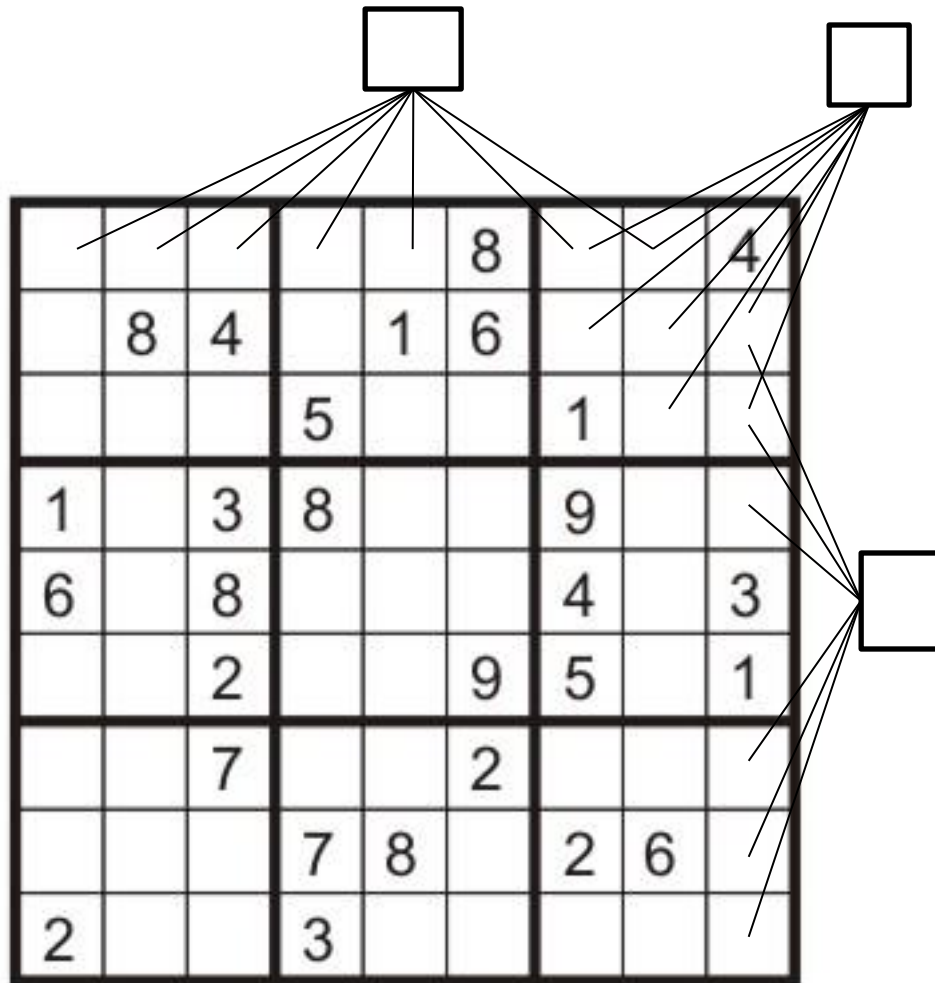 $\text{alldiff}(F, T, U, W, R, O)$

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

  $\cdots$

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

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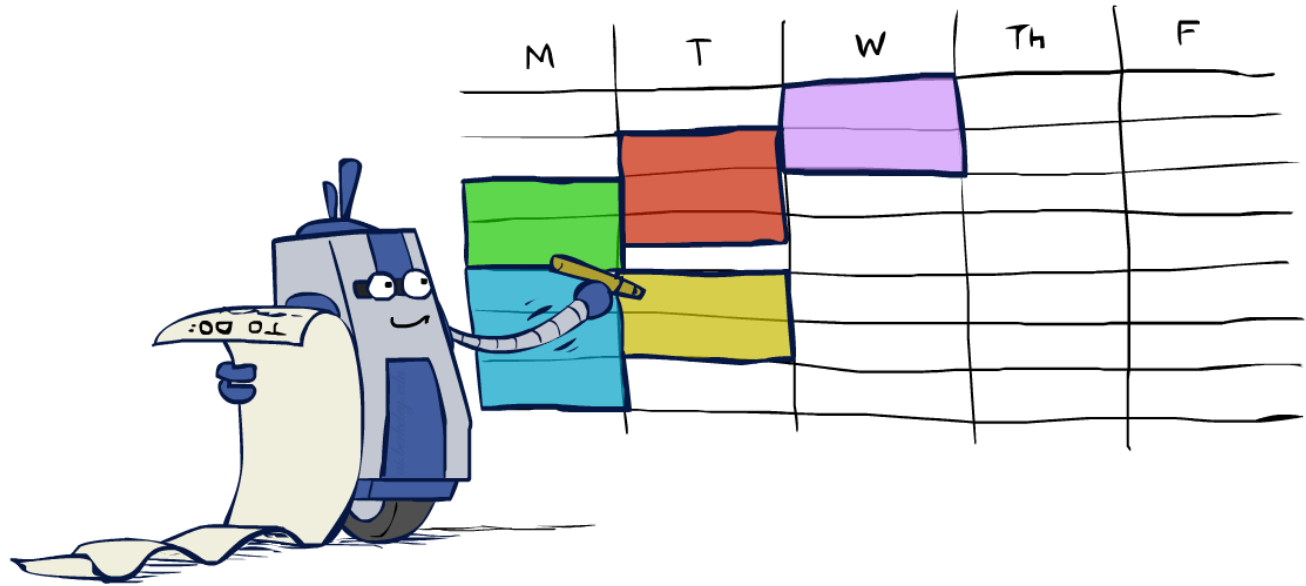
o Many real-world problems involve real-valued variables…

Solving CSPs

Standard Search Formulation

o Standard search formulation of CSPs

o States defined by the values assigned so far (partial assignments)
  - o Initial state: the empty assignment, {}
  - o Successor function: assign a value to an unassigned variable
  - o Goal test: the current assignment is complete and satisfies all constraints

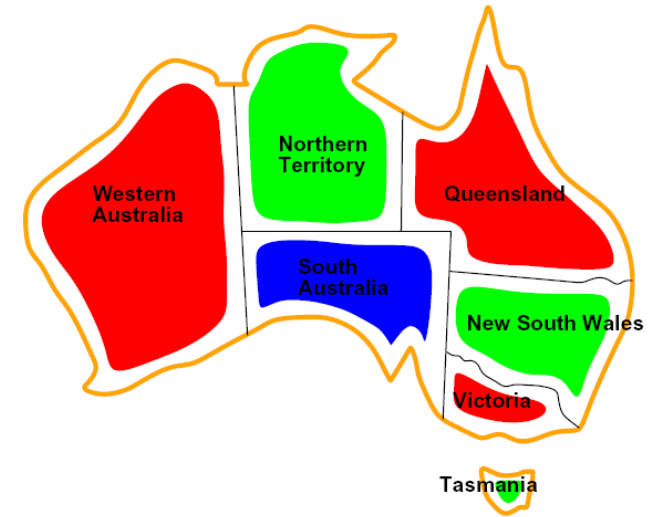o We'll start with the straightforward, naïve approach, then improve it
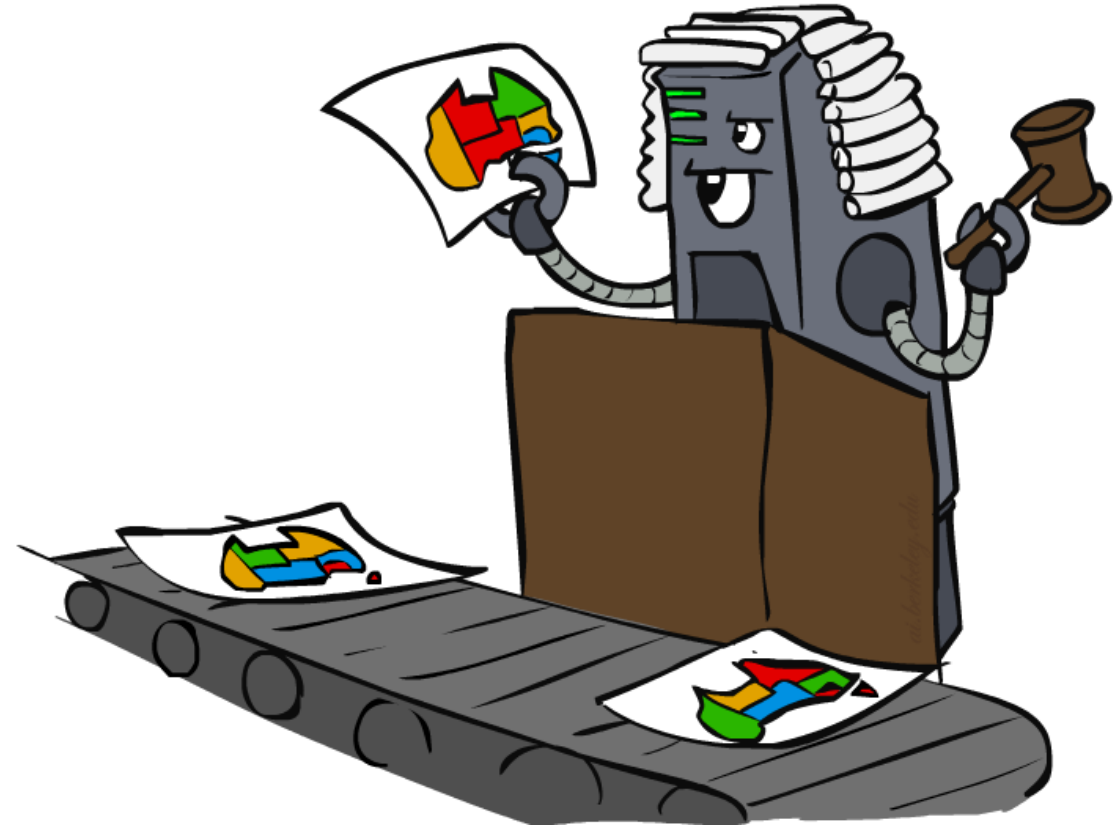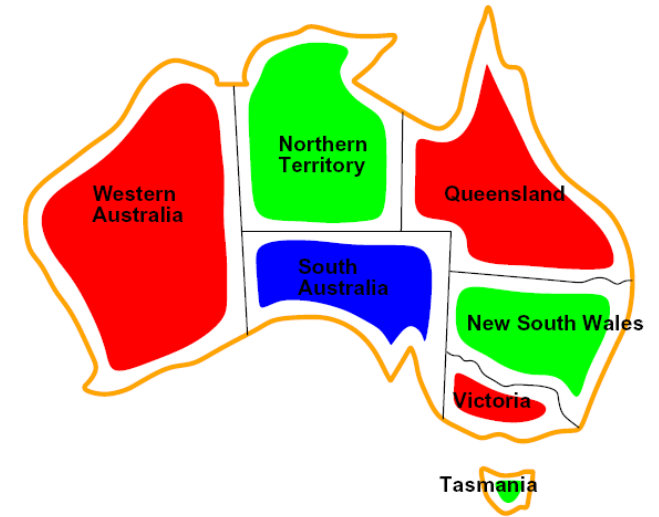
Search Methods

o What would BFS do?

{}

{WA=g} {WA=r} … {NT=g} …
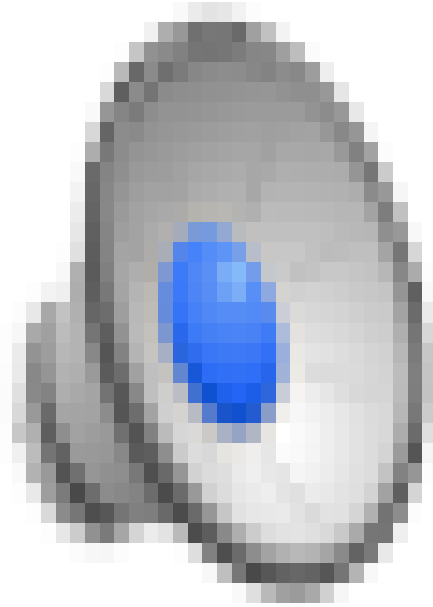
Search Methods

o What would BFS do?

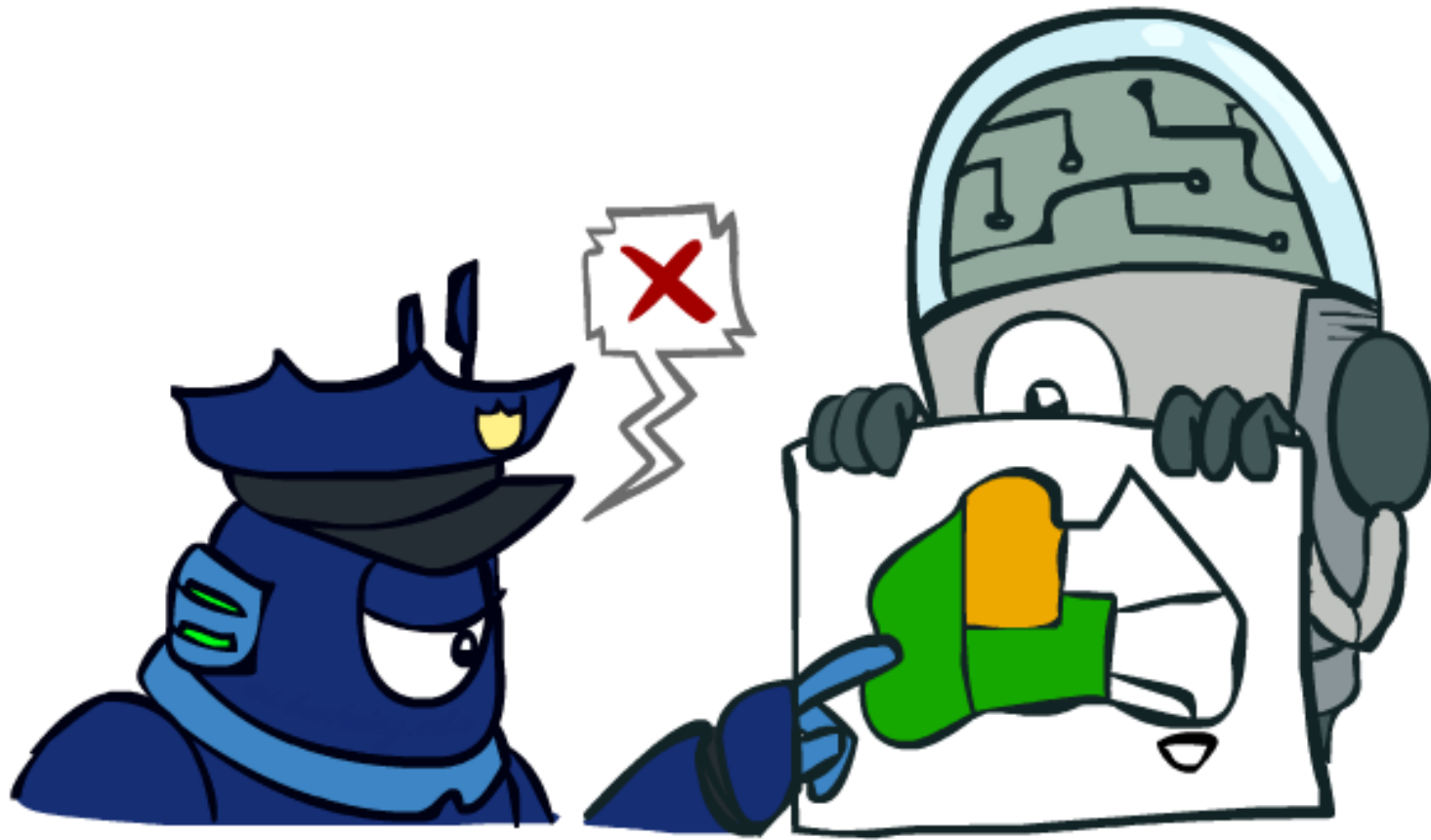o What would DFS do?
  o let's see!

o What problems does naïve search have?
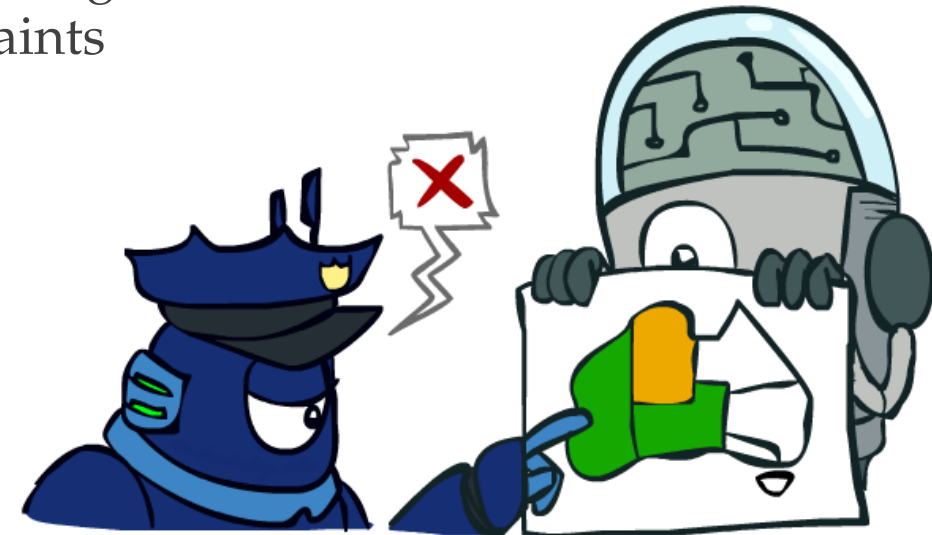
Video of Demo Coloring -- DFS
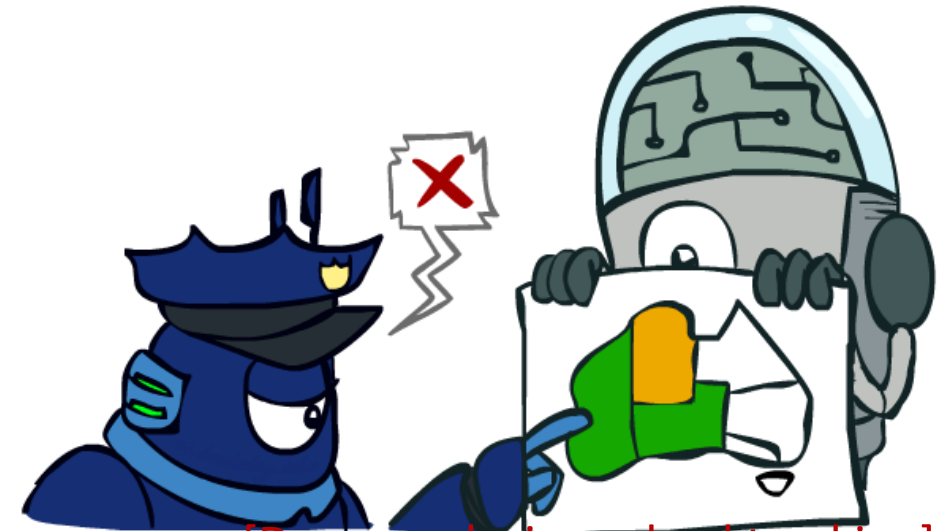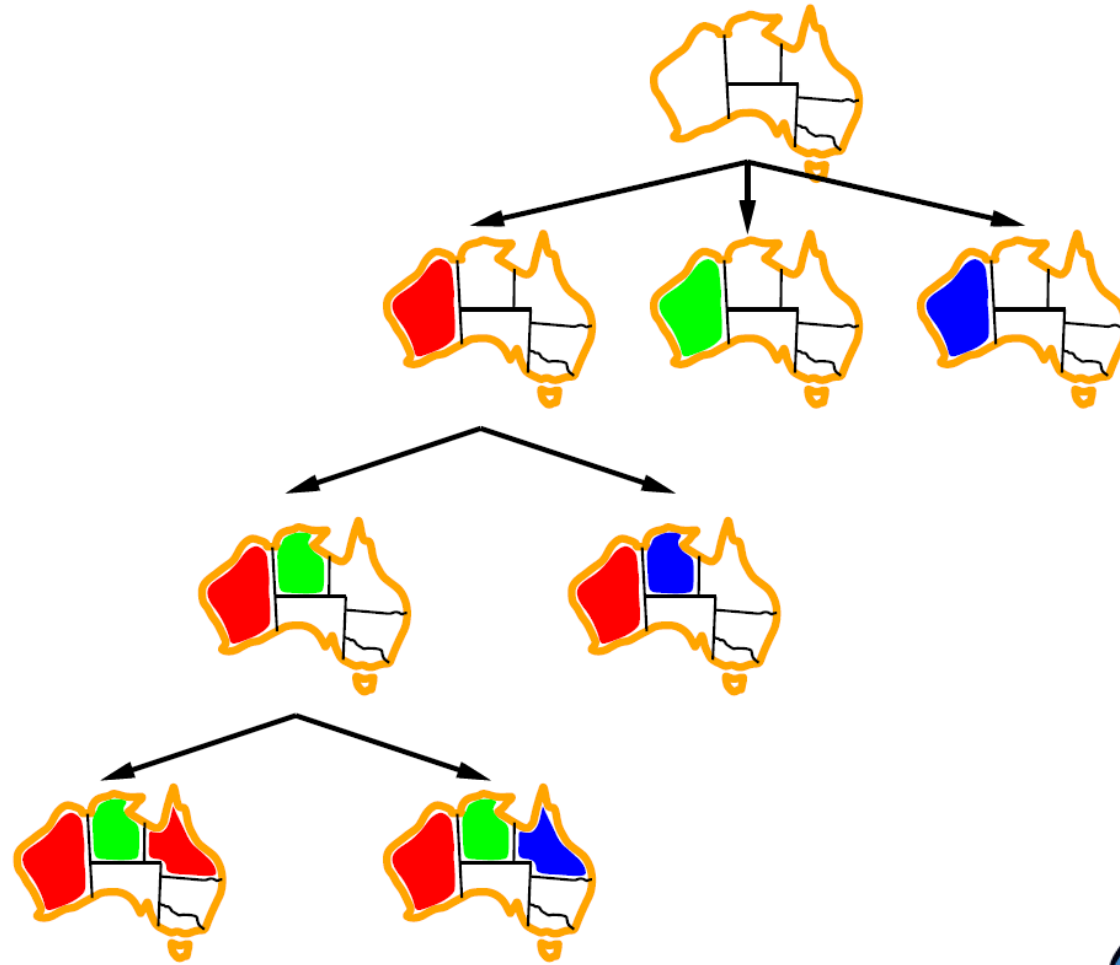
Backtracking Search

Backtracking Search

o Backtracking search is the basic uninformed algorithm for solving CSPs

o Idea 1: One variable at a time
  o Variable assignments are commutative, so fix ordering -> better branching factor!
  o I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  o Only need to consider assignments to a single variable at each step

o Idea 2: Check constraints as you go
  o I.e. consider only values which do not conflict previous assignments
  o Might have to do some computation to check the constraints
  o "Incremental goal test"

o Depth-first search with these two improvements
  is called *backtracking search* (not the best name)
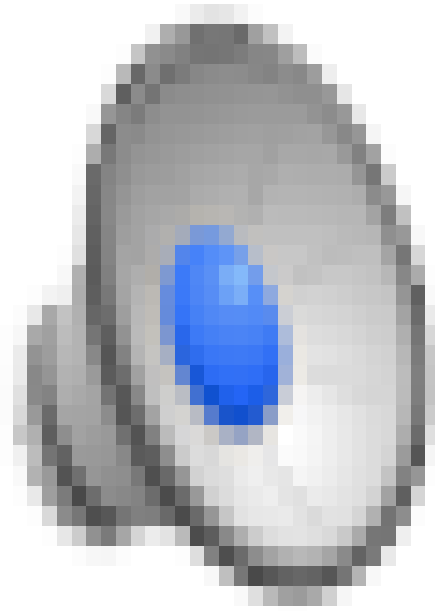
o Can solve n-queens for n ≈ 25

Backtracking Example

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

o General-purpose ideas give huge gains in speed

o Ordering:
  o Which variable should be assigned next?
  o In what order should its values be tried?

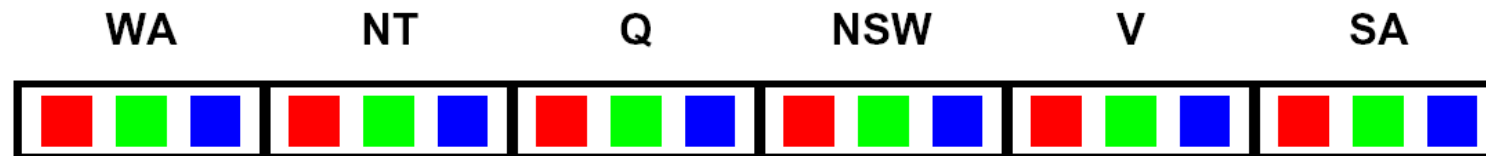o Filtering: Can we detect inevitable failure early?
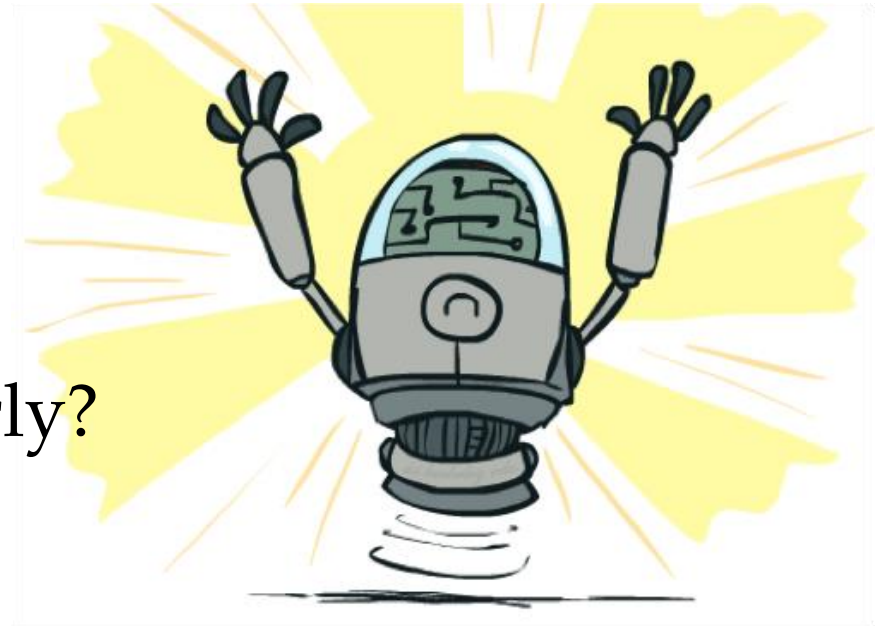
Filtering

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
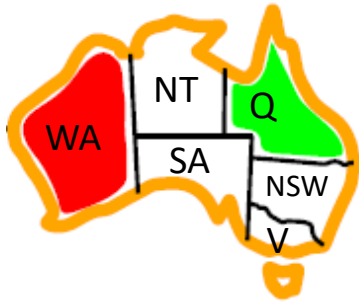
Video of Demo Coloring – Backtracking with Forward Checking
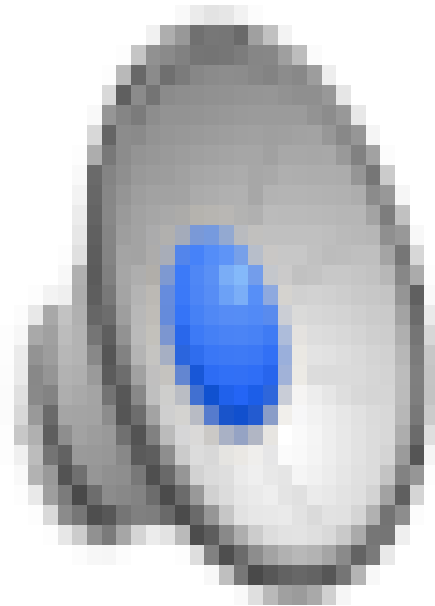
Filtering: Constraint Propagation

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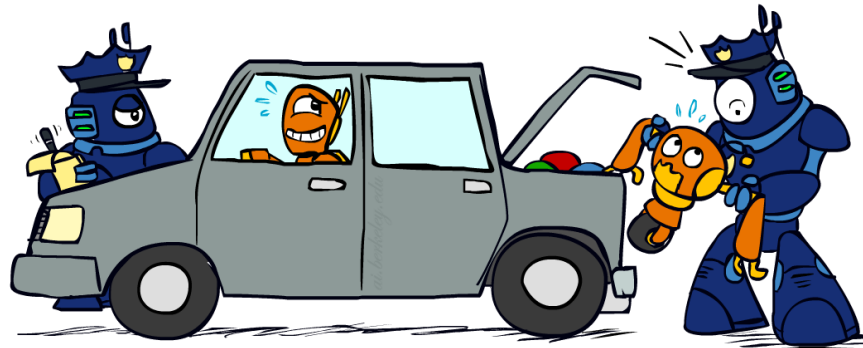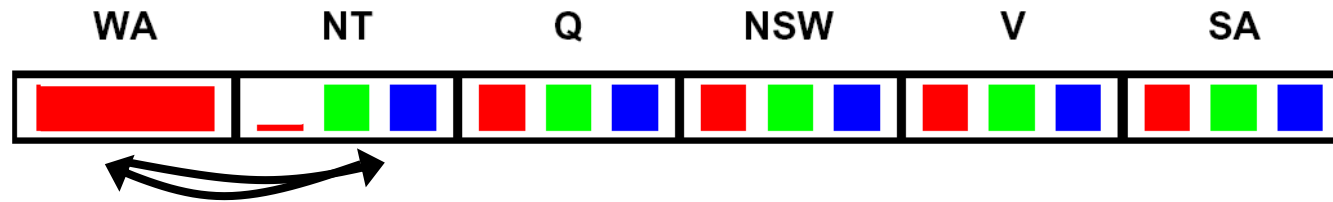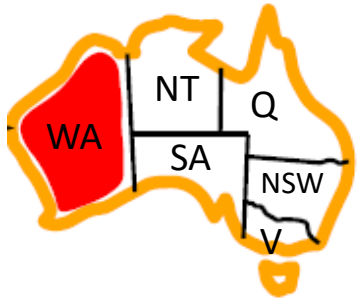

o NT and SA cannot both be blue!
o Why didn't we detect this yet?
o *Constraint propagation:* reason from constraint to constraint

Consistency of A Single Arc

o An arc X → 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
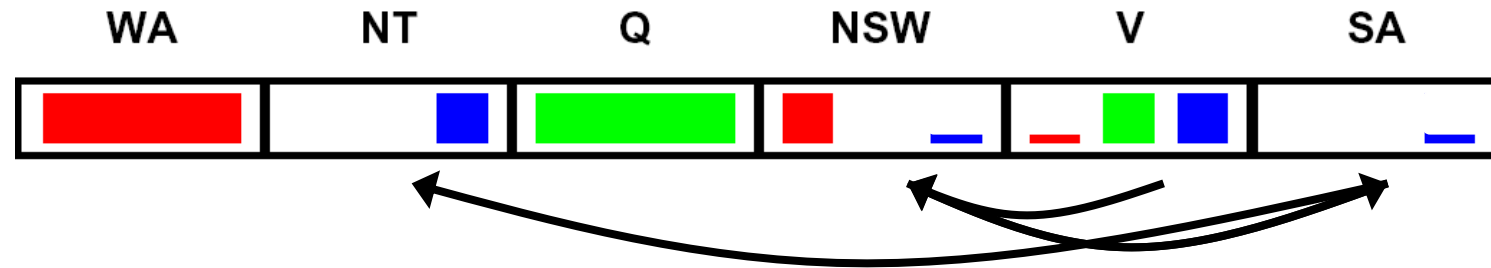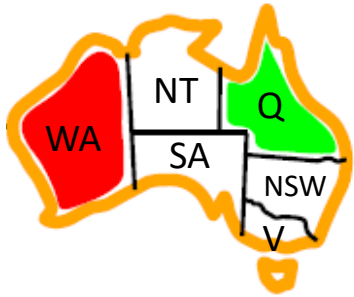

Delete from the tail!

Forward checking?
Enforcing consistency of arcs pointing to each new assignment

Arc Consistency of an Entire CSP

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

o Important: If X loses a value, neighbors of X need to be rechecked!
o Arc consistency detects failure earlier than forward checking
o Can be run as a preprocessor or after each assignment
o 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: *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 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 $X_i \leftrightarrow X_j$

    then delete x from DOMAIN[X_i]; *removed* ← *true*

  return *removed*

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

Limitations of Arc Consistency

o After enforcing arc consistency:

  o Can have one solution left

  o Can have multiple solutions left

  o Can have no solutions left (and not know it)

o Arc consistency still runs inside a backtracking search!
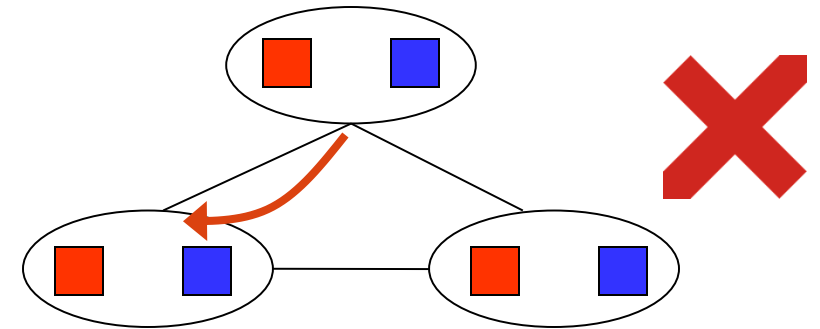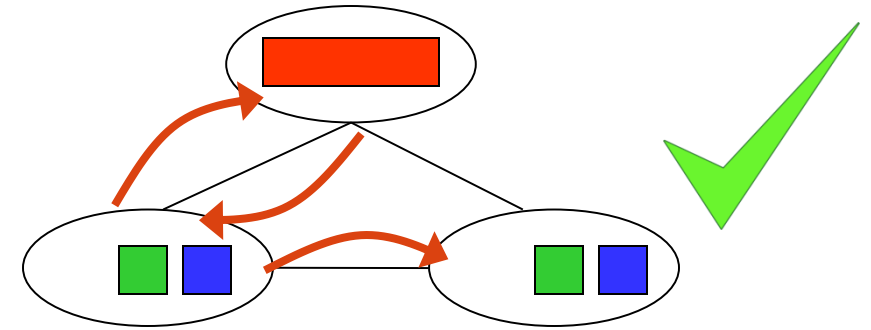
Video of Demo Coloring – Backtracking with Forward Checking – Complex Graph
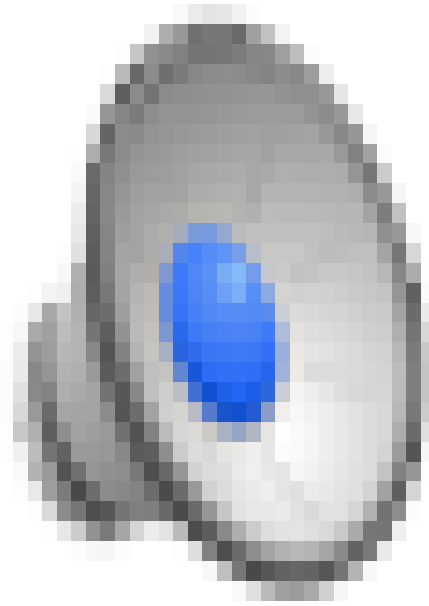
Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph