# Artificial Intelligence

CSE 440/EEE 333/ETE333

Chapter 6
Fall 2017

**Mirza Mohammad Lutfe Elahi**
Department of Electrical and Computer Engineering
North South University

# 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

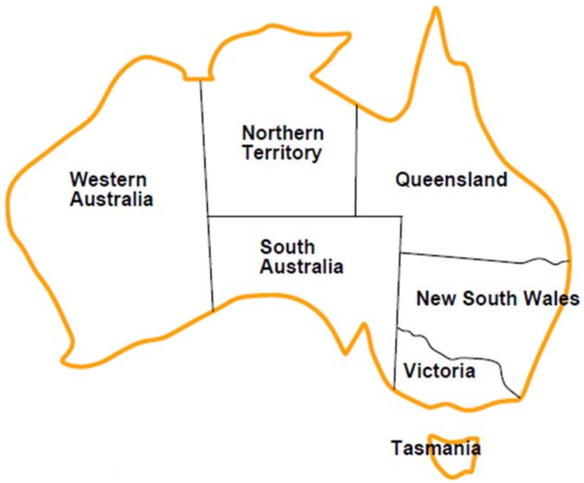
- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by $X$, a set of variables, $\{X_1, X_2, \ldots, X_n\}$
  - $D$, a set of domains for each $X$, $\{D_1, D_2, \ldots, D_n\}$
    - $D_i = \{v_1, v_2, \ldots, v_n\}$
  - $C$, a set of constraints.
    - $C_i = \langle scope, rel \rangle$

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

# CSP Example – Map Coloring


- Color each region either red, green or blue
- No adjacent region can have the same color

# CSP Example – Map Coloring



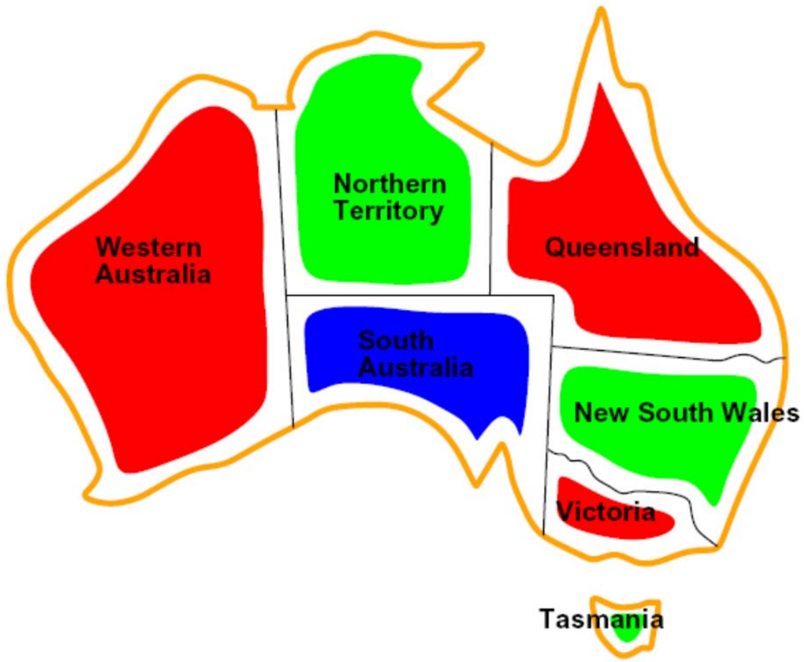

- Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$
- Domains: $D = \{red, green, blue\}$ for each $X_i \in X$
- Constraints: adjacent regions must have different colors

C = {<($\forall X_i, X_j$ such that $X_i$ touches $X_j$), ($Color(X_i) \neq Color(X_j)$)>}

or

($WA$, $NT$) ∈ {(red, green), (red, blue), (green, red), (green, blue), ….}

# CSP Example – Map Coloring

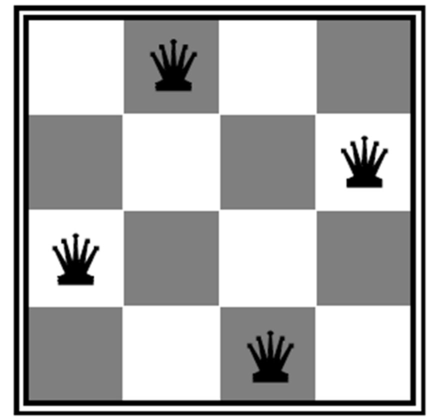



- Solutions are assignments satisfying all constraints, e.g.:

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

# CSP Example – N-Queens

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

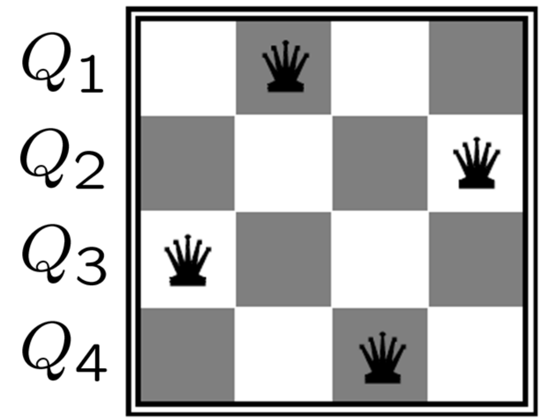


$$\forall i, j, k \;\; (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

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

# CSP Example – N-Queens

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

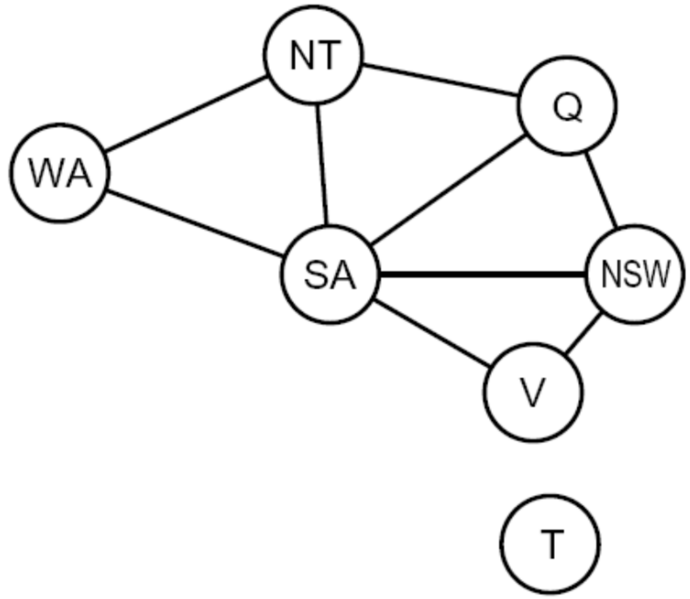



Implicit: $\forall i, j \;\; \text{non-threatening}(Q_i, Q_j)$

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

$\cdots$

# Constraint Graph




- 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!

# 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)

# Example: Cryptarithmetic

- Variables:

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

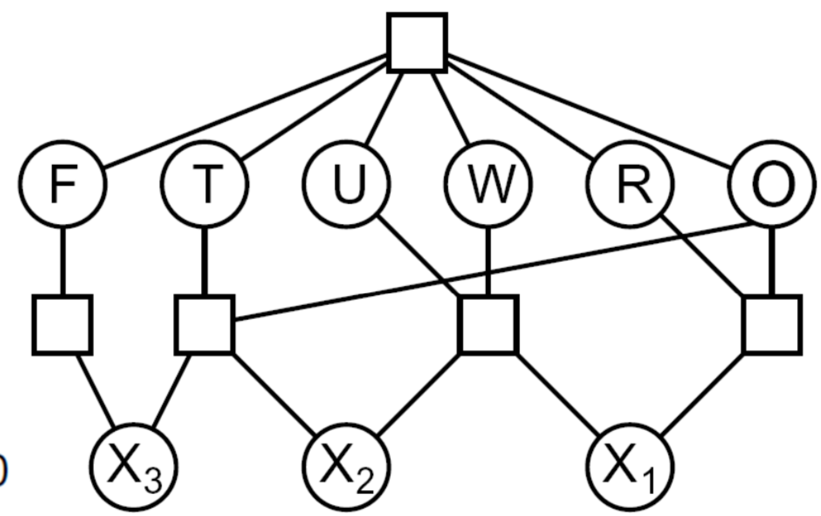
- Domains:

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

- Constraints:

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

$$AllDiff(F, T, U, W, R, O)$$
$$O + O = R + 10 \times C_{10}$$
$$C_{10} + W + W = U + 10 \times C_{100}$$
$$C_{100} + T + T = O + 10 \times C_{1000}$$
$$C_{1000} = F$$



# Real-World CPS

- Assignment problems

  e.g., who teaches what class
- Timetabling problems

  e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floor planning

- Notice that many real-world problems involve real-valued variables

# Solving CPS

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

- Initial state: the empty assignment, { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment.
  - fail if no legal assignments (not fixable!)
- Goal test: the current assignment is complete

1) This is the same for all CSPs!

2) Every solution appears at depth $n$ with $n$ variables
   - use depth-first search

3) Path is irrelevant, so can also use complete-state formulation

4) Branching factor $b = (n-l)d$ at depth $l$, hence $n!d^n$ leaves!!!!

# Solving CPS – 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
    $b = d$ and there are $d^n$ leaves
- 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
- Depth-first search for CSPs with single-variable assignments is called backtracking search

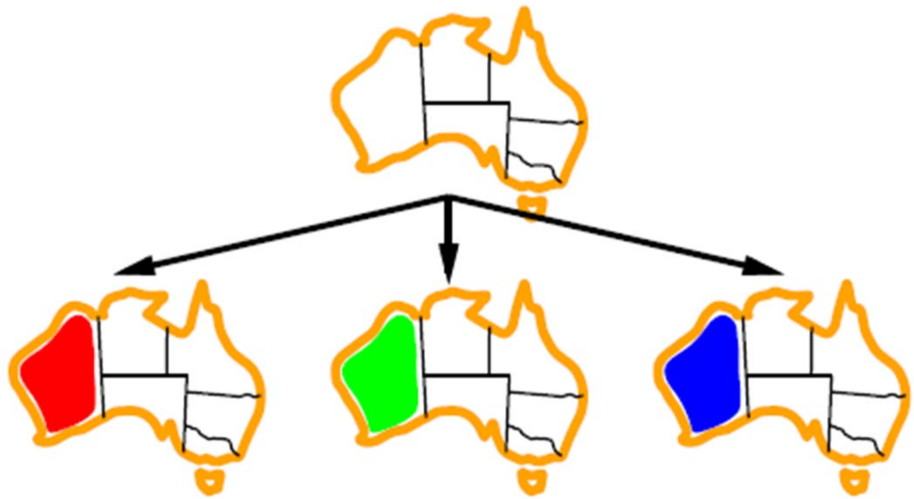
- Can solve *n*-queens for $n \approx 25$

# Backtracking Algorithm

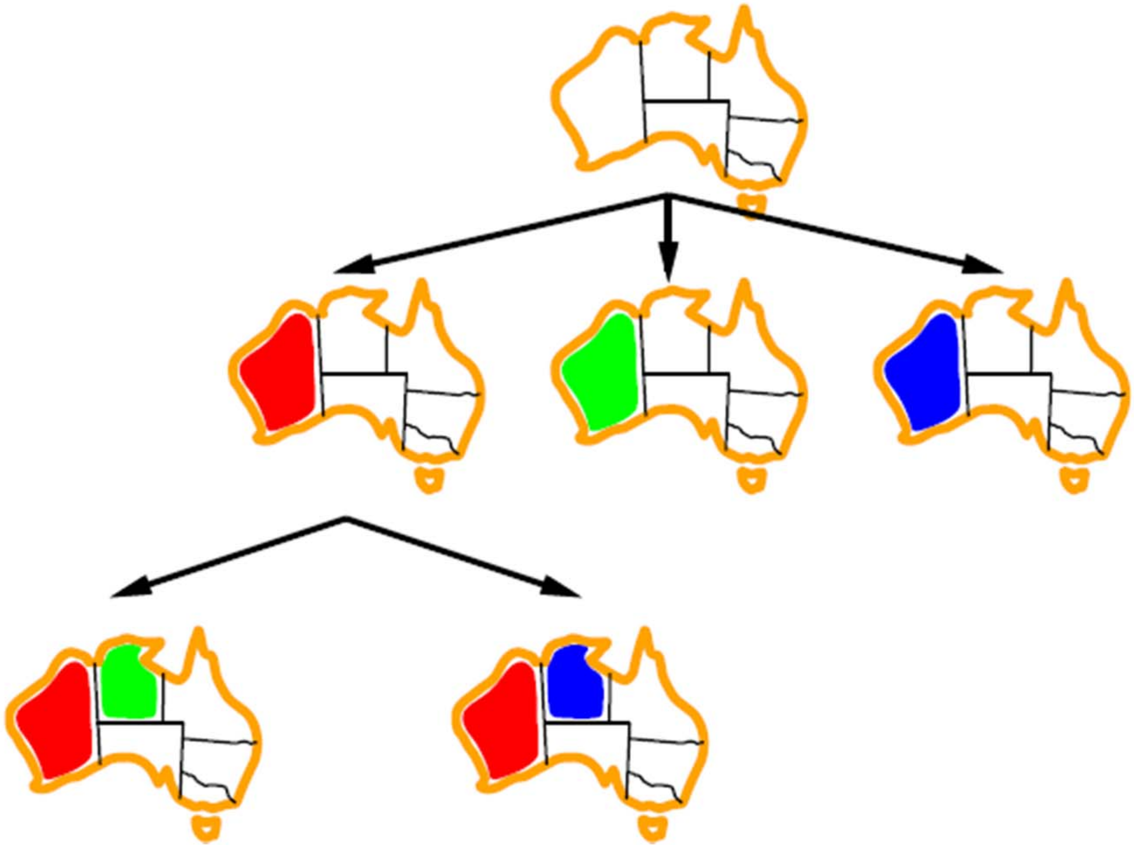
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
   return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
         add {var = value} to assignment
         inferences ← INFERENCE(csp, var, value)
         if inferences ≠ failure then
            add inferences to assignment
            result ← BACKTRACK(assignment, csp)
            if result ≠ failure then
               return result
      remove {var = value} and inferences from assignment
   return failure
```

# Backtracking Example

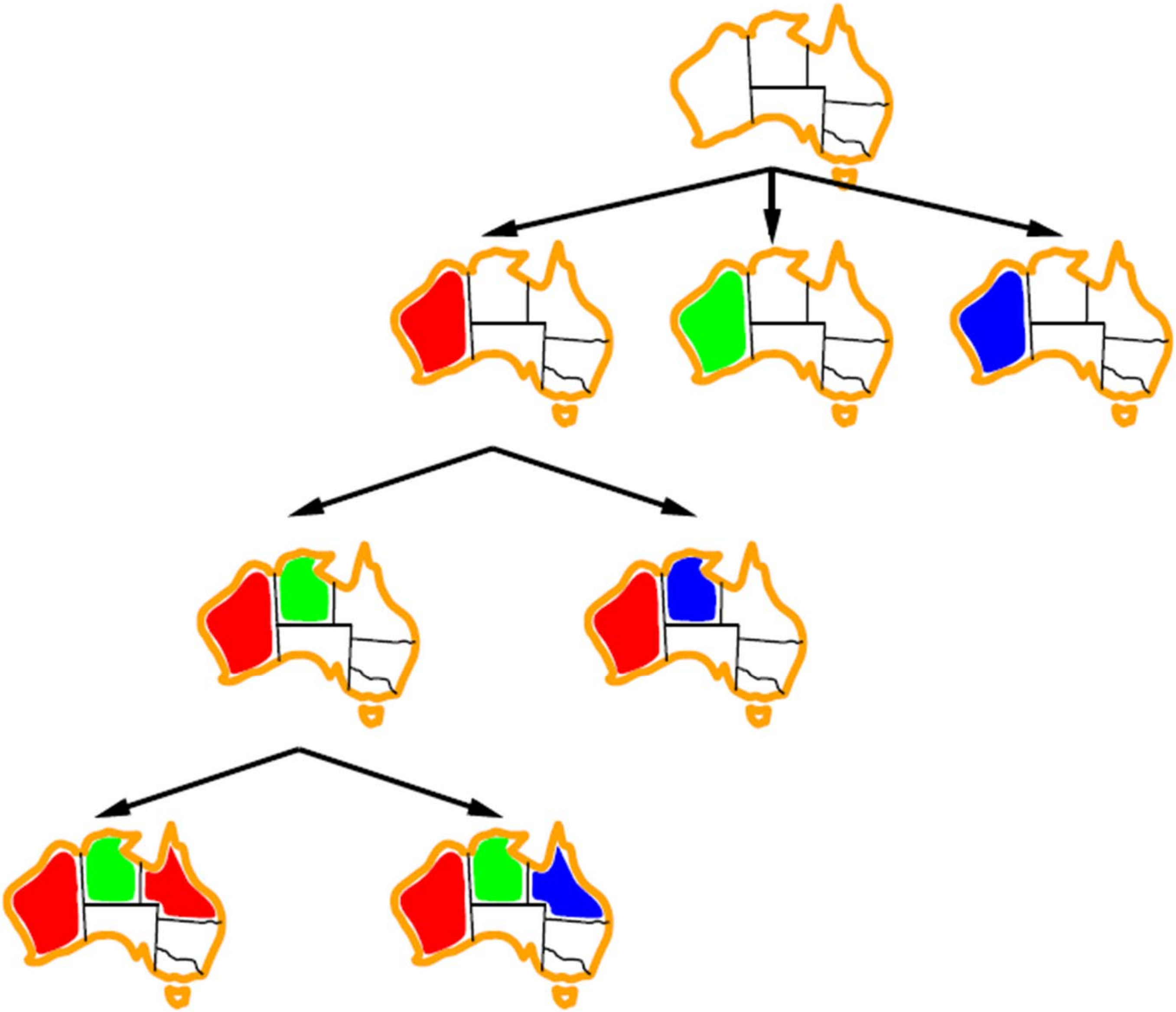
# Backtracking Example



# Backtracking Example



# Backtracking Example



# Improve Backtracking Efficiency

- General-purpose methods can give huge gains in speed:

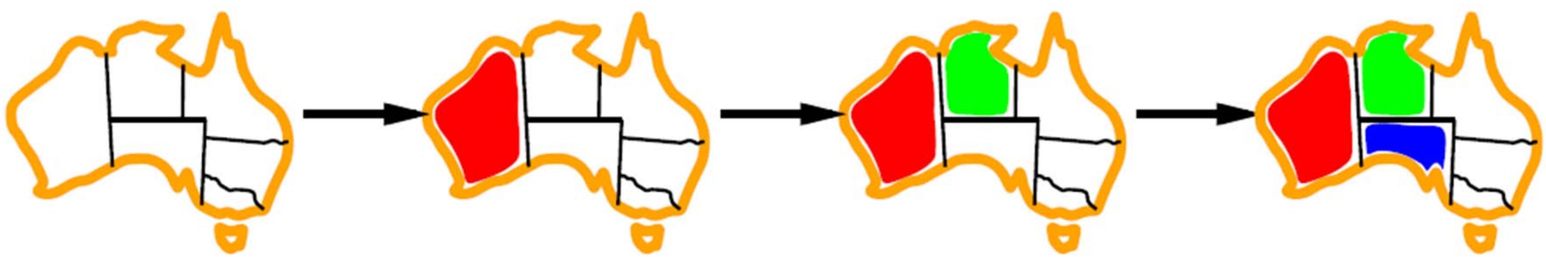
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?

- Filtering: Can we detect inevitable failure early?
- Structure: Can we take advantage of problem structure?

# Minimum Remaining Values

Minimum remaining values (MRV):

choose the variable with the fewest legal values

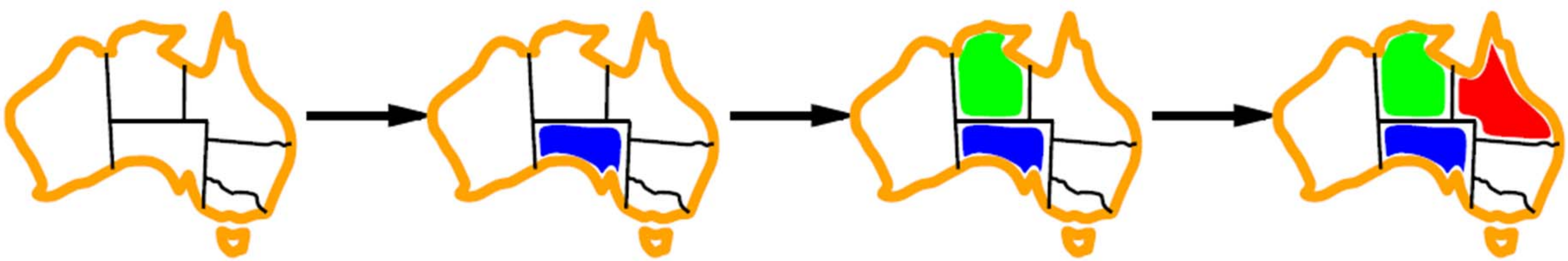


# Degree Heuristic

Tie-breaker among MRV variables

Degree heuristic:

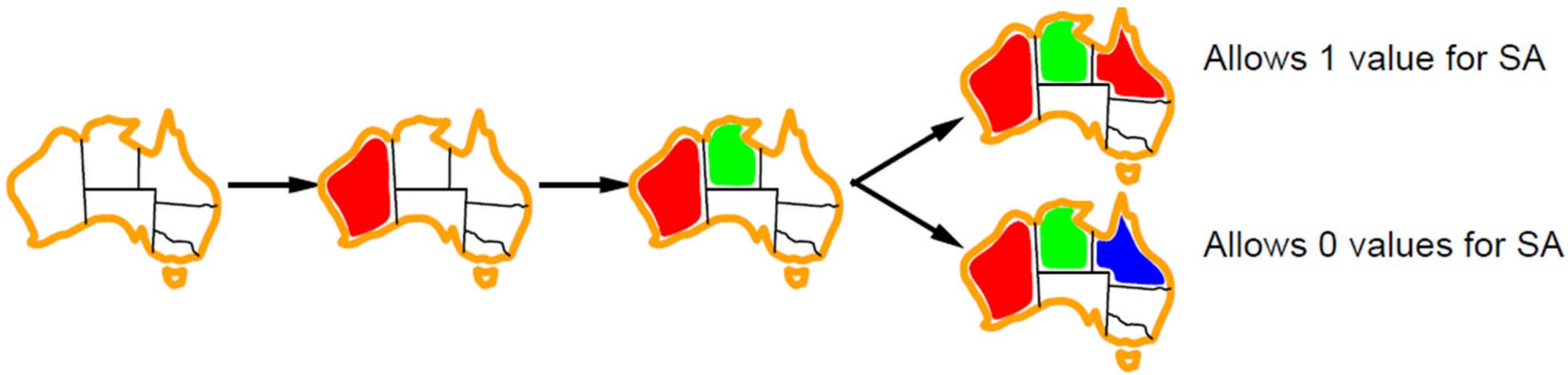
choose the variable with the most constraints on remaining variables



# Least Constraining Value

Given a variable, choose the least constraining value:

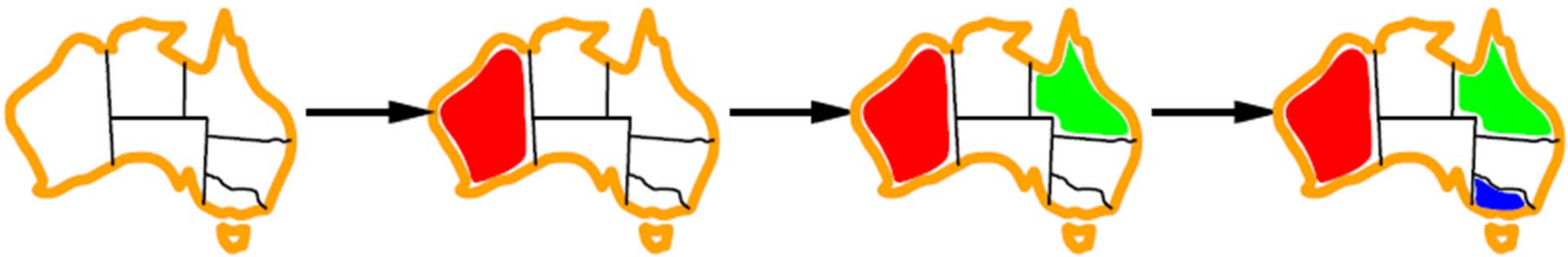
the one that rules out the fewest values in the remaining variables




Combining these heuristics makes 1000 queens feasible

# Forward Checking

Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values



```
function ForwardChecking(csp) //returns a new domain for
     each var
    for each variable X in csp do
        for each unassigned variable Y connected to X do
            for each value d in Domain(Y)
                if d is inconsistent with Value(X)
                    Domain(Y)={Domain(Y)-d}
    return csp //whith modified domains
```

# Forward Checking

| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| Domains | RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| After WA | R | GB | RGB | RGB | RGB | GB | RGB |
| After Q | R | B | G | RB | RGB | B | RGB |
| After V | R | B | G | R | B | | RGB |

# Forward Checking

| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| Domains | RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| After WA | R | GB | RGB | RGB | RGB | GB | RGB |
| After Q | R | B | G | RB | RGB | B | RGB |
| After V | R | B | G | R | B | | RGB |

# Forward Checking

| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| Domains | RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| After WA | R | GB | RGB | RGB | RGB | GB | RGB |
| After Q | R | B | G | RB | RGB | B | RGB |
| After V | R | B | G | R | B | | RGB |

# Forward Checking

| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| Domains | RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| After WA | R | GB | RGB | RGB | RGB | GB | RGB |
| After Q | R | B | G | RB | RGB | B | RGB |
| After V | R | B | G | R | B | | RGB |

# Constraint Propagation

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

| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| Domains | RGB | RGB | RGB | RGB | RGB | RGB | RGB |
| After WA | R | GB | RGB | RGB | RGB | GB | RGB |
| After Q | R | B | G | RB | RGB | B | RGB |
| After V | R | B | G | R | B | | RGB |

*NT* and *SA* cannot both be blue!

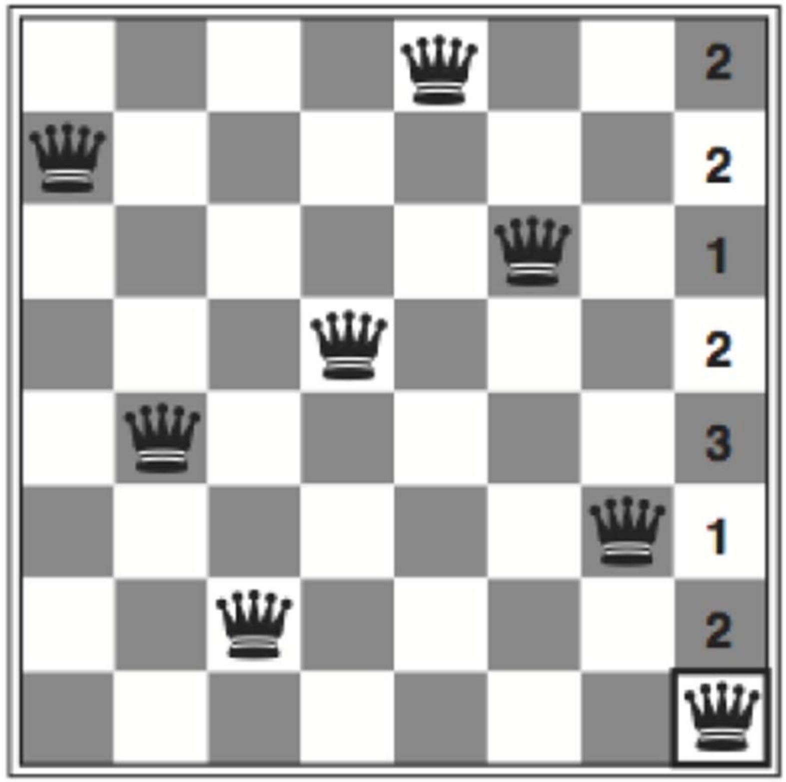
Constraint propagation repeatedly enforces constraints locally

# Minimum Conflicts

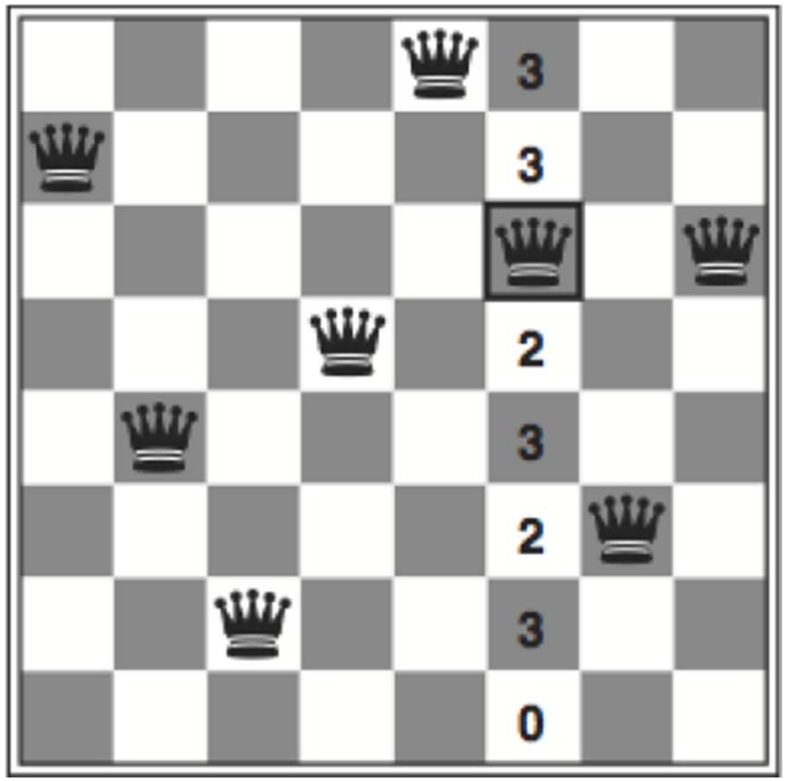
**function** MIN-CONFLICTS($csp$, $max\_steps$) **returns** a solution or failure
  **inputs**: $csp$, a constraint satisfaction problem
    $max\_steps$, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for $csp$
  **for** $i$ = 1 to $max\_steps$ **do**
    **if** $current$ is a solution for $csp$ **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from $csp$.VARIABLES
    $value \leftarrow$ the value $v$ for $var$ that minimizes CONFLICTS($var$, $v$, $current$, $csp$)
    set $var = value$ in $current$
  **return** $failure$

# Minimum Conflicts



# Minimum Conflicts



# Minimum Conflicts