Artificial Intelligence

Constraint Satisfaction Problems (CSPs)
(Part 2)

CS 444 – Spring 2019
Dr. Kevin Molloy
Department of Computer Science
James Madison University

JAMES MADISON UNIVERSITY.

Constraint Satisfaction Problems (CSPs)

Standard Search Problem:

State is a "black box" – any old data structure that supports a goal test, eval, successors, etc.

CSPs:

State is defined by variables X_i with values from domain D_i.
Goal test is a set of constraints specifying allowable combinations of values for subsets of variables.

Varieties of Constraints

Unary constraints involve a single variable. e.g., SA ≠ green

Binary constraints involve pairs of variable. e.g., SA ≠ WA

Higher-order constraints involve 3 or more variables.

e.g. cryptarithmetic column constraints

Strong vs **soft** constraints

Preference (soft constraints)

- e.g., **red** is better than **green**
- Often representable by a cost for each variable assignment

⇒constrained optimization problems

Pruning the search space

Number of possible color assignments? $O(d^n)$ $O(3^6)= 729$

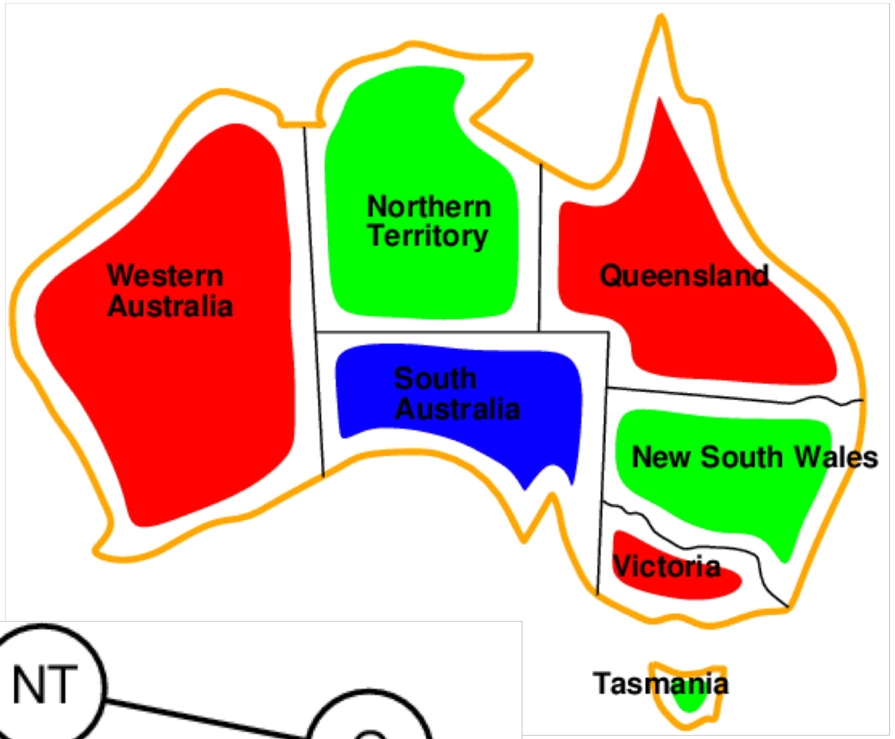
If South Australia is assigned blue? $O(3^5)= 243$

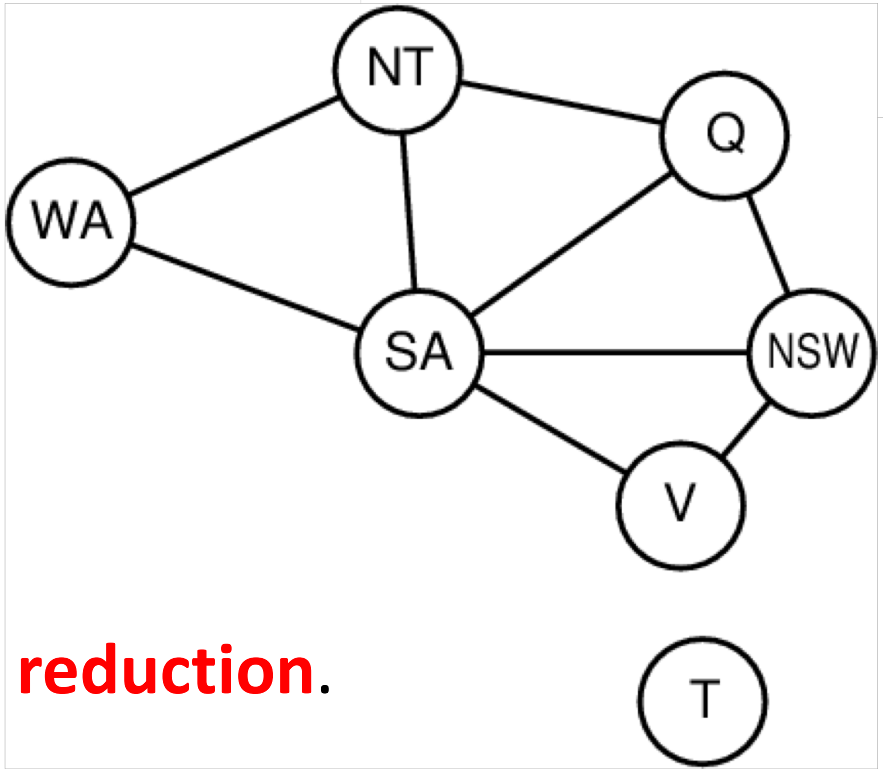
Can we do better?

Since South Australia is a neighbor to all other territories, we can eliminate blue from their domains.

$O(2^5)= 32$

This is an **87% reduction**.





Real-world CSPs

- **Assignment** problems: e.g., who teaches what class
- **Timetabling** problems: e.g., which class is offered when and where
- **Transportation schedules**
- **Factory scheduling**
- **Floor planning**

Real-world problems almost always involve real-valued variables

Standard Search Formulation (Incremental)

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

States are defined by the values assigned so far:

- **Initial state**: the empty assignment, 0
- **Successor function**: assign a value to an unassigned variable that does not conflict with current assignments.
- **Fails** if no legal assignments (a dead end, not fixable!)
- **Goal test**: the current is complete
    1. This is the same for all CSPs !
    2. Every solution appears at depth *n* with *n* variables (we can use DFS)
    3. Path is irrelevant, so can also use the complete-state formulation
    4. $b = (n\text{-}L)d$ at depth L, hence $n!d^n$ leaves (bad news)

Backtracking Search

Variable assignments are **commutative**, 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 node

$\Longrightarrow$ b = d (branching factor = depth) and there are d^n leaves

Depth-first search for CSPs with single-variable assignments is called backtracking search

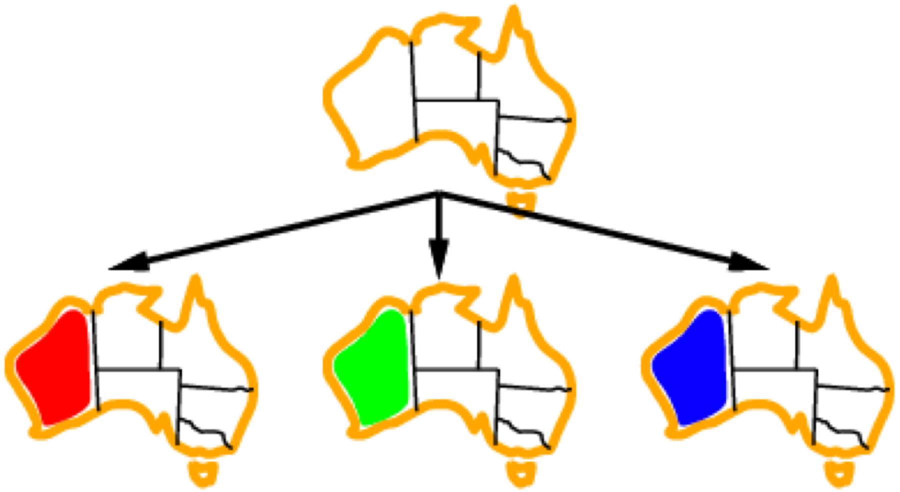
Can solve n-queens for n $\approx$ 25 in a reasonable amount of time.

Backtracking Search

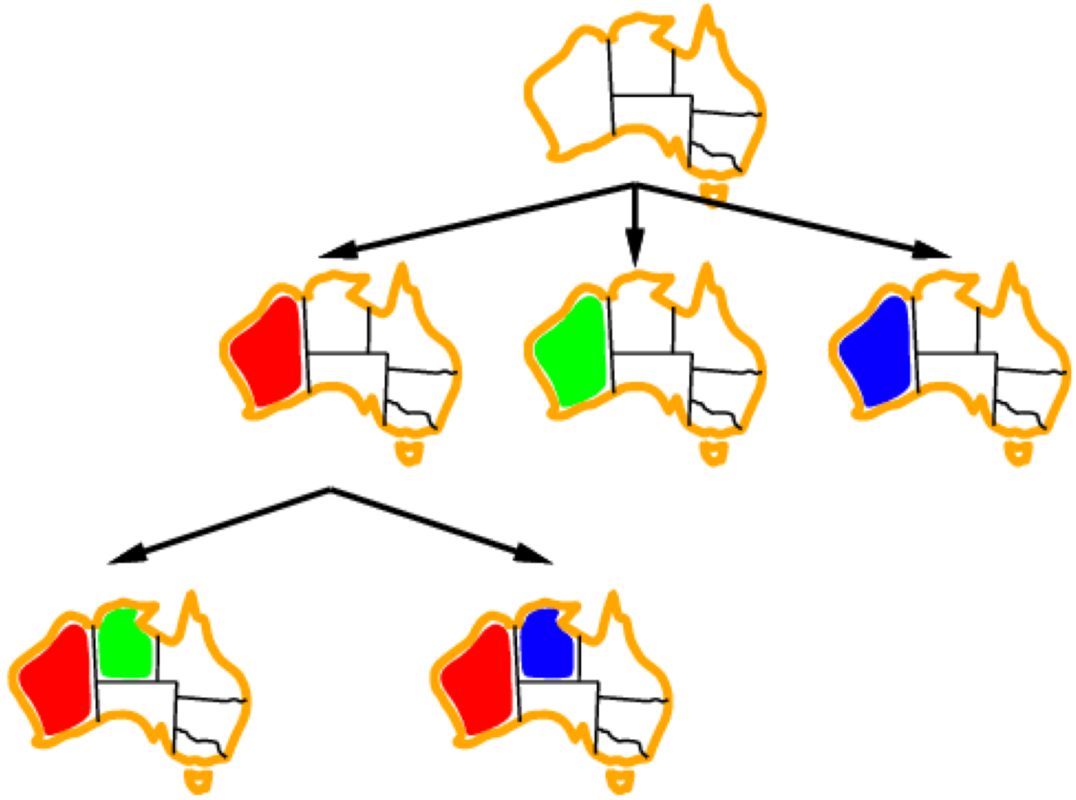
```
function Backtracking-Search(csp) returns solution/failure
    return Backtrack({}, csp)

function Backrack(assignment, csp) returns solution/failure
    If assignment is complete then return assignment
    Var ← Select-Unassigned-Variable(csp, assignment)
    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(var, assignment, csp)
            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

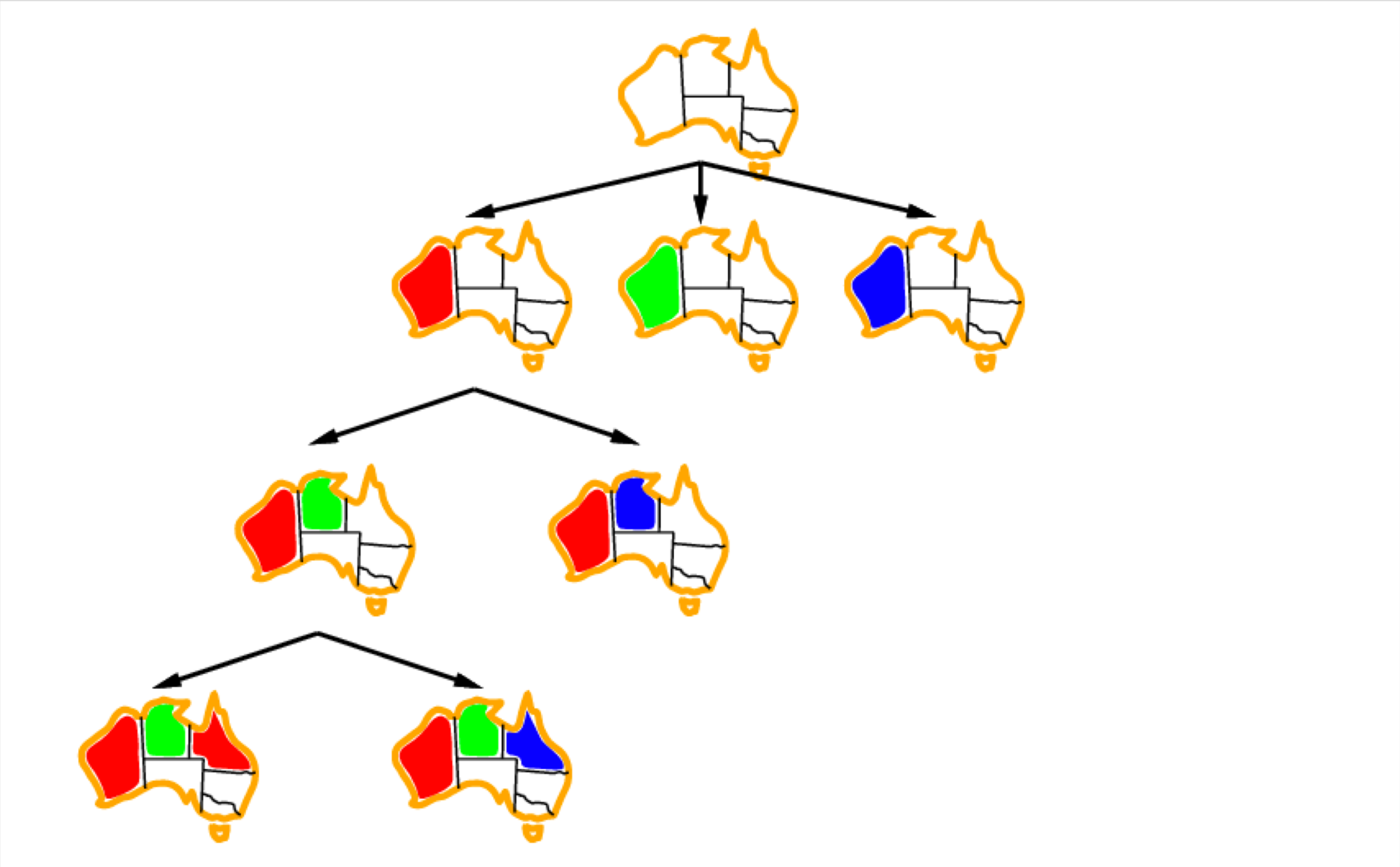
Backtracking Example



Backtracking Example



Backtracking Example



Improving Backtracking Efficiency

General purpose methods can give huge gains in speed:

1. Which variable should be assigned next? [Select-Unassigned-Variable]
2. In what order should its values be tried? [Order-Domain-Values]
3. Can we detect inevitable failure early? [Inference]
4. Can we take advantage of problem structure?

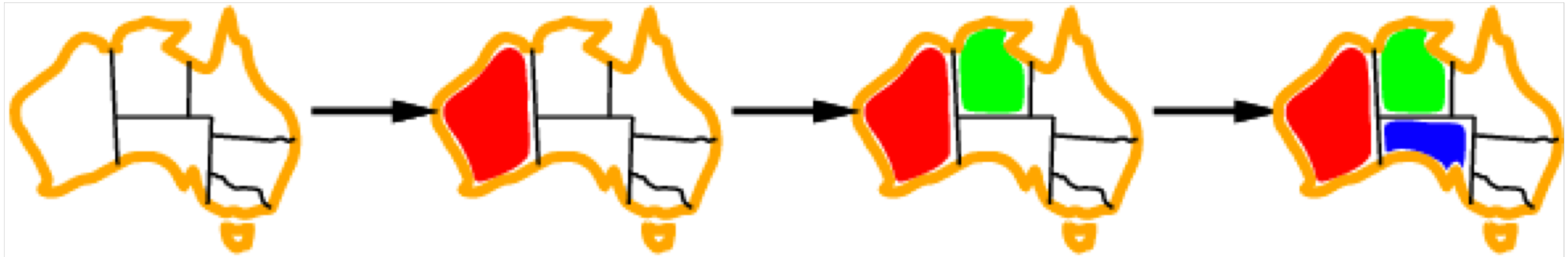
Minimum Remaining Values

Minimum remaining values (MRV) for:

var ← SELECT-UNASSIGNED-VAR(csp, assignment)

Choose the variable with the fewest legal values to prune the search tree.

Also called "most constrained variable" or "fail-first heuristic"



… but MRV heuristic does not help in selecting the first variable.

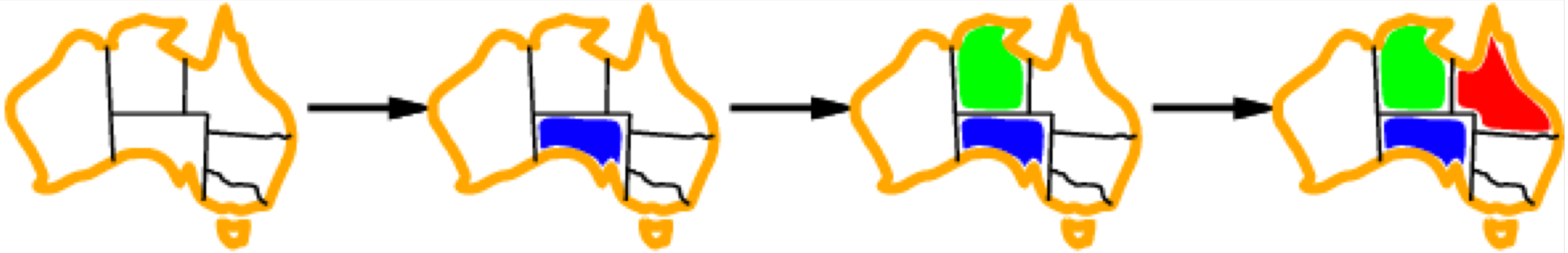
Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:

Choose the variable with the most constraints on remaining variables

var ← SELECT-UNASSIGNED-VAR(csp, assignment)



Called **degree heuristic** because information is available in constraint graph

Attempts to reduce branching factor on future choices

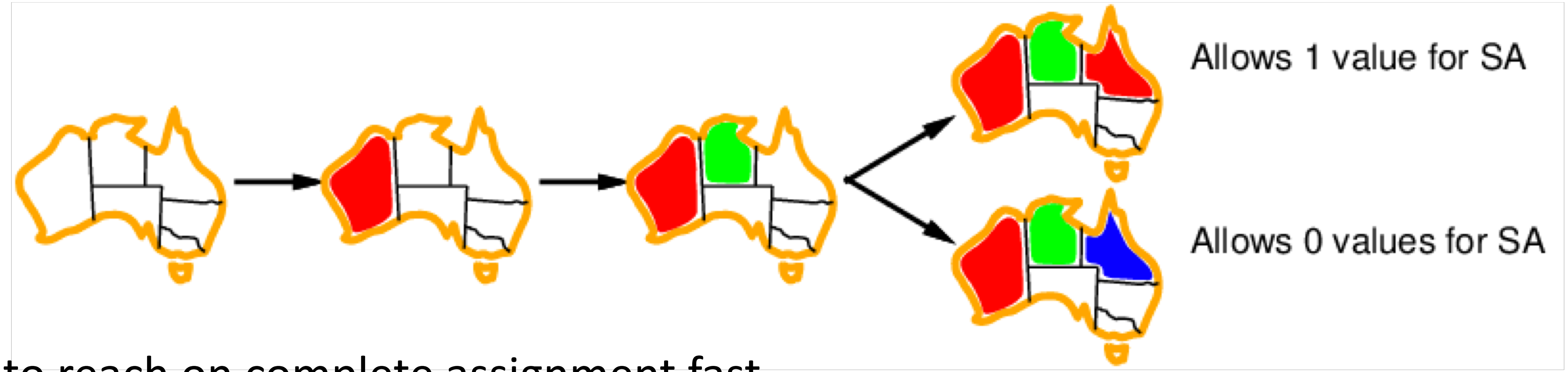
Least Constraining Value Heuristic

Least constraining value heuristic for:

Var ← Order-Domain-Values(var, assignment, csp)

Given a variable, choose the least constraining value:

Selects value that rules out the fewest values in the remaining variables:



Goal is to reach on complete assignment fast.

Combining above heuristics make **1000 queens feasible**

When all solutions/complete assignments needed, LCV is irrelevant

Inference

Idea: Infer reductions in the domain of variables

When: Before and/or during the backtracking search itself

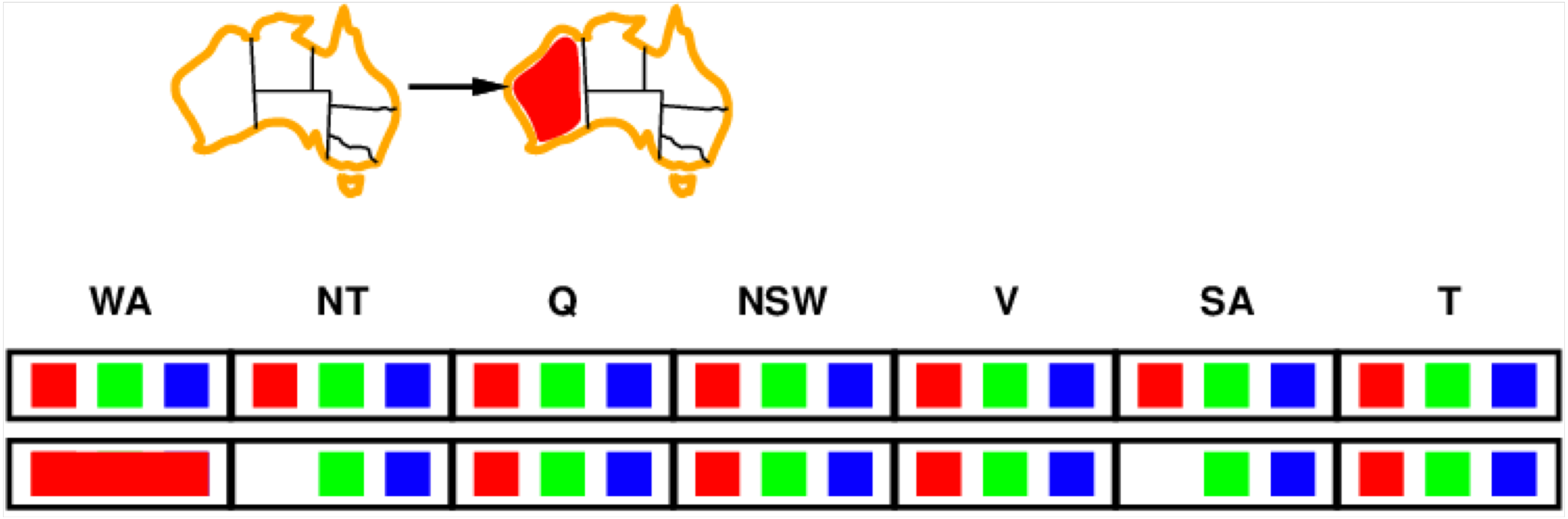
How: Constraint propagation

Algorithms: forward checking, AC-3

Simplest Form of Inference: Forward Checking

Idea: Keep track of remaining legal values for unassigned variables

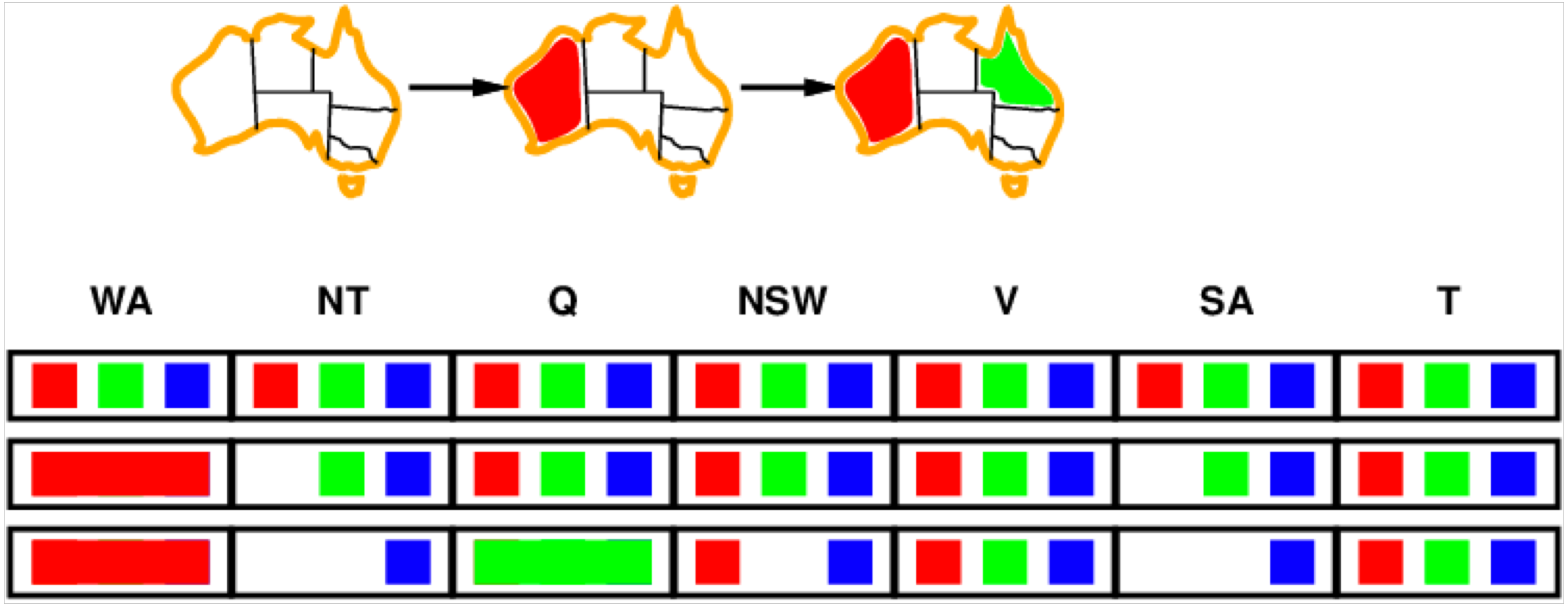
Terminate search when any variable has no legal values



Simplest Form of Inference: Forward Checking

Idea: Keep track of remaining legal values for unassigned variables

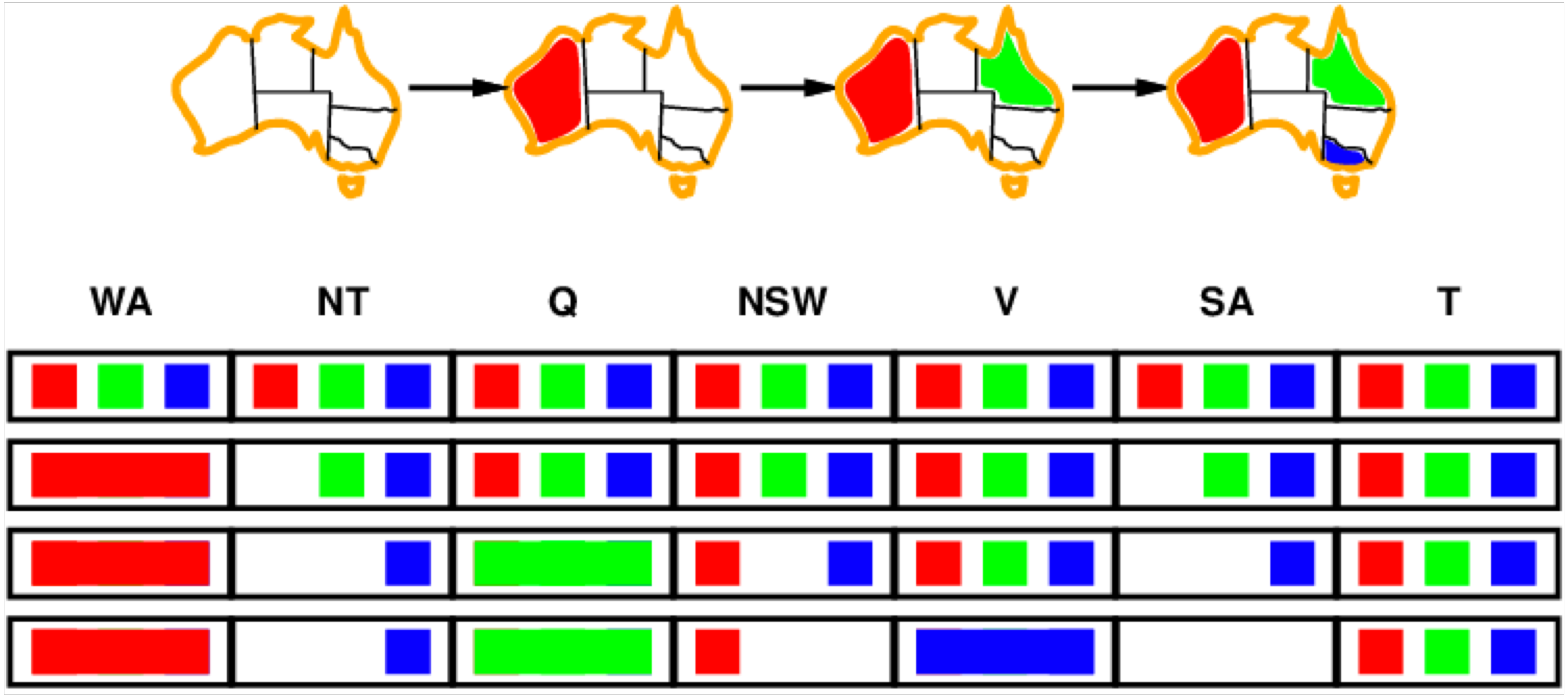
Terminate search when any variable has no legal values



Simplest Form of Inference: Forward Checking

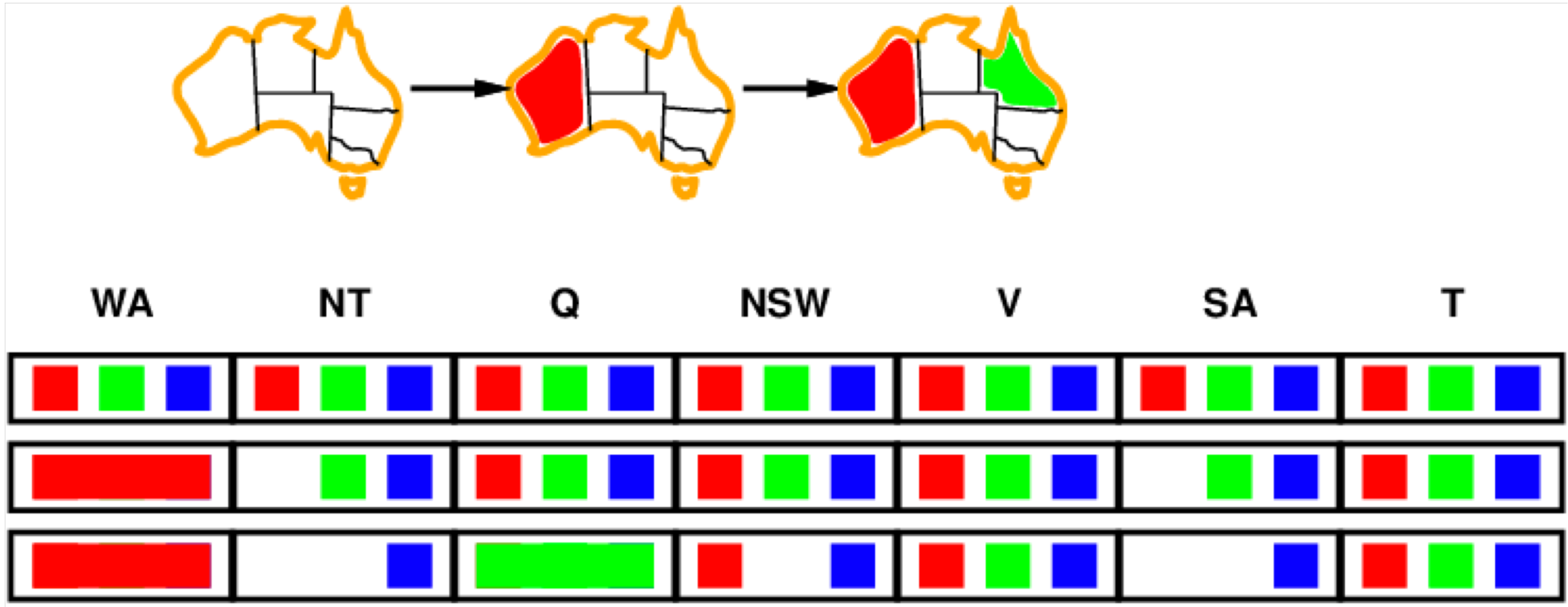
Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



Constraint Propagation

Forward checking propagates information from assigned to unassigned variables:



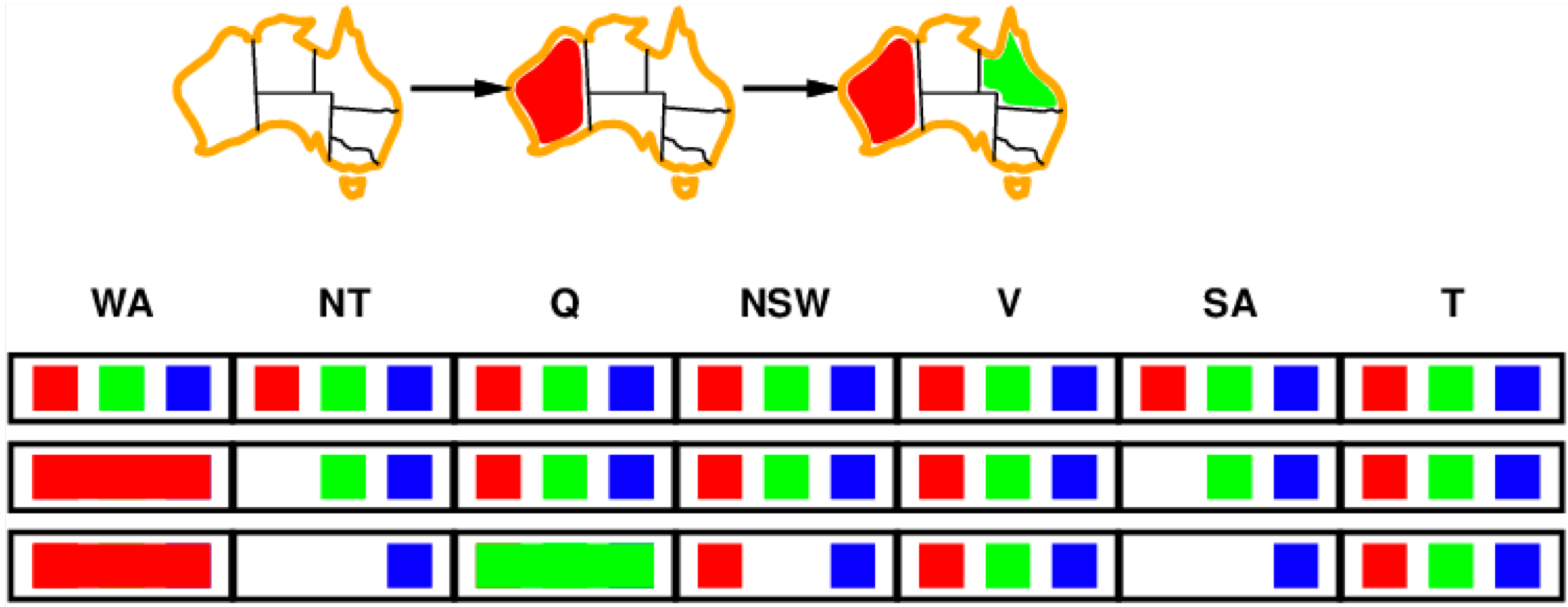
Forward checking establishes arc consistency

Whenever a var X is assigned, domains of neighbors Y of X in constraint graph are reduced

For each unassigned var Y that is connected to X by a constraint, delete from Y's domain any value that is inconsistent with the value choosen for X

Constraint Propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detectio



BUT: NT and SA cannot both be blue!

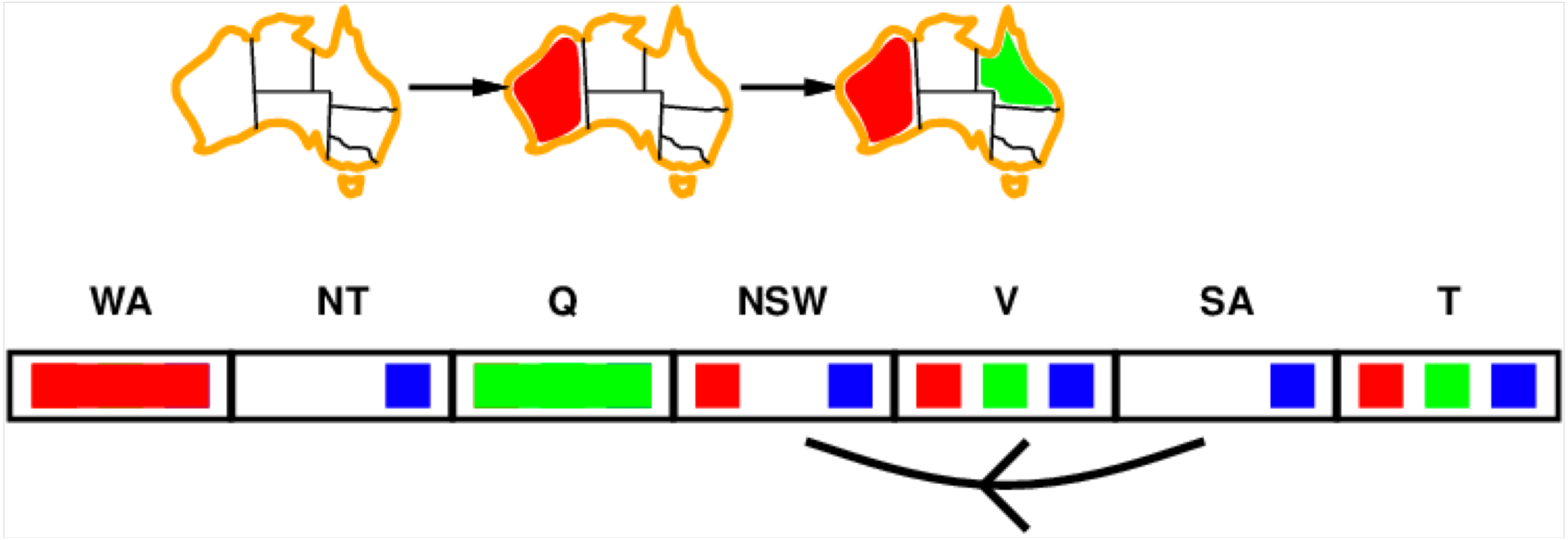
Constraint propagation enforces constraints locally at each step (over and over), and does not "chase" arc consistency

When the domain of a neighbor Y of X is reduced, domains of neighbors of Y may also become inconsistent (e.g., NT and SA).

Back to Arc Consistency

Simplest form of constraint propagation makes each arc consistent

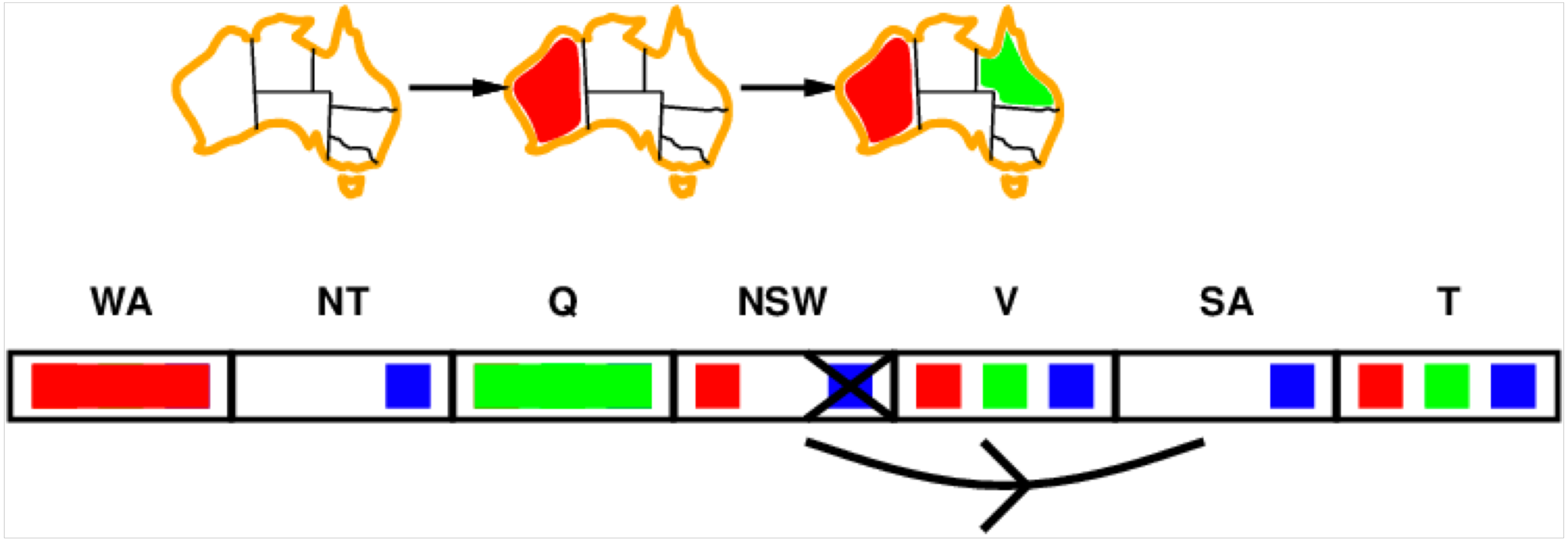
X → Y is consistent iff for every value of x of X there is some allowed value y of Y



Back to Arc Consistency

Simplest form of constraint propagation makes each arc consistent

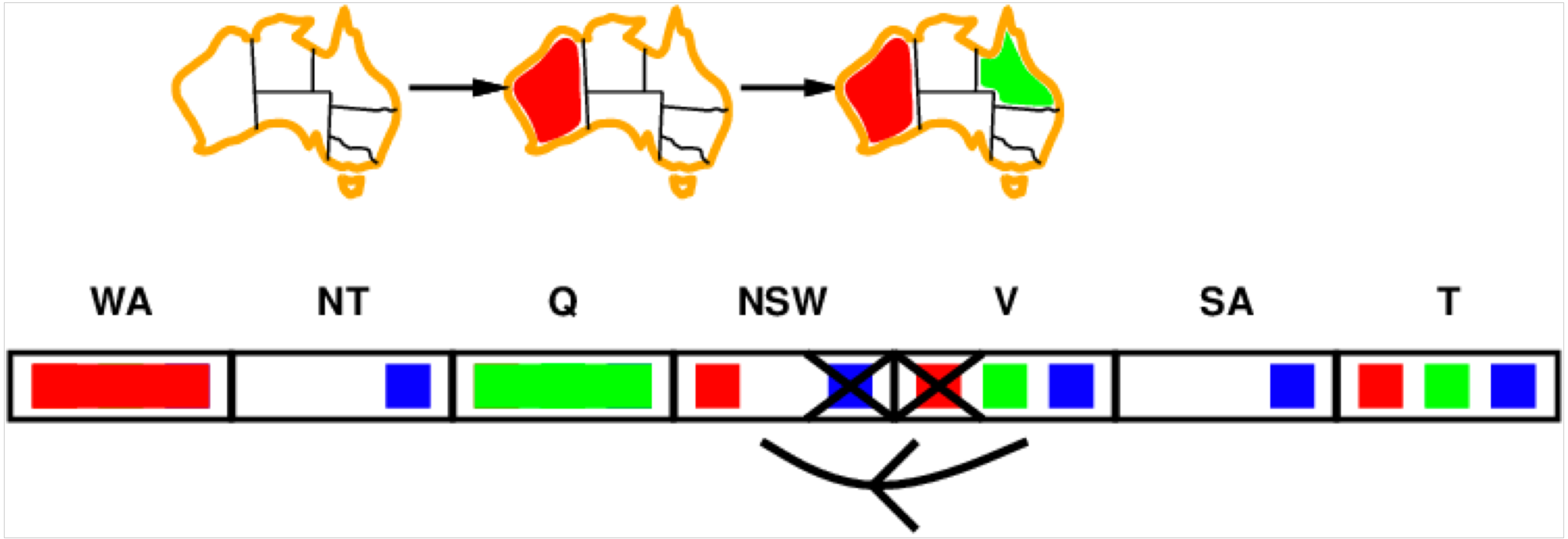
X → Y is consistent iff for every value of x of X there is some allowed value y of Y



Back to Arc Consistency

Simplest form of constraint propagation makes each arc consistent

X → Y is consistent iff for every value of x of X there is some allowed value y of Y



If a variable loses a value, its neighbors in the constraint graph need to be rechecked

Iterative Algorithms for CSPs

Hill-climbing, simulated annealing typically work with "complete" states (all variables assigned)

To apply to CSPs:

- Allow states with unsatisfied constraints
- Operators reassign variable values

Variable selection: randomly select any conflicted variable

Value selection by min-conflicts heuristic:

- Choose value that violates the fewest constraints
- i.e., hill-climber with h(n) = total number of violated constraints

Probabilistic Search and Energy Guidance for Biased Decoy Sampling in Ab-initio Protein Structure Prediction. Molloy et al. IEEE Trans in Computational Biology and Bioinformatics, 2013.

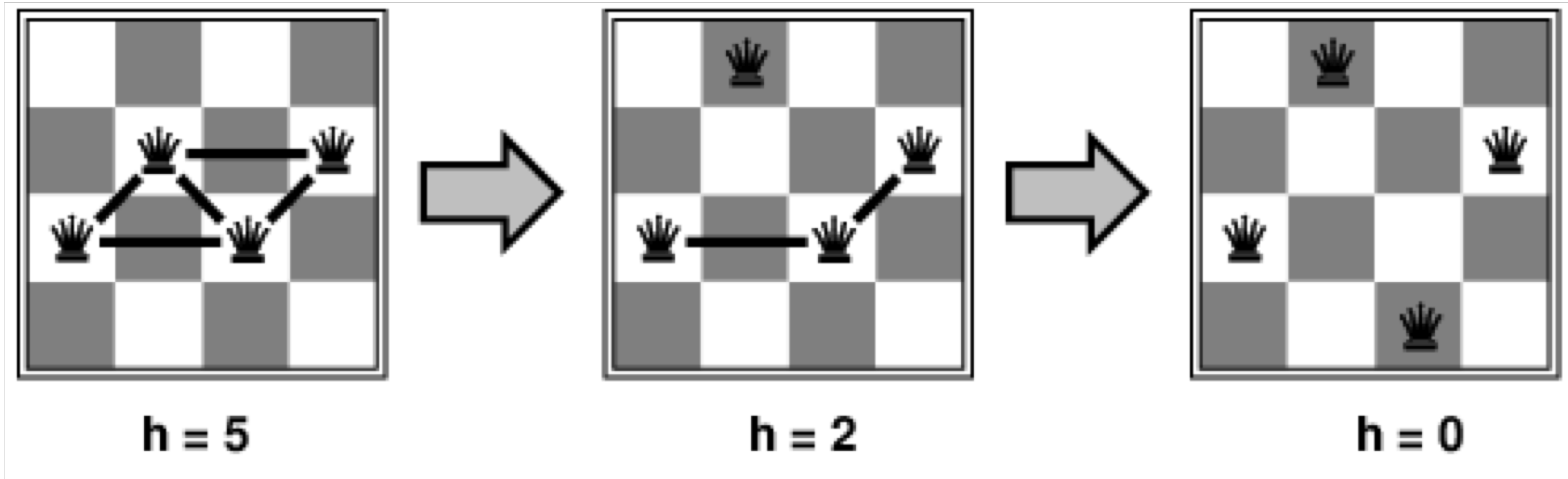
Example: 4-Queens as CSP

States: 4 queens in 4 columns (4^4 = 256 state)

Operators: move queen in column

Goal test: no attacks

Evaluation: h(n) = number of attacks



4 Queens as a CSP

Work through the 4-queens as CSP in greater detail

Assume one queen in each column. Which row does each one go in?

Variables Q_1, Q_2, Q_3, Q_4 Domains D_i = {1, 2, 3, 4}

Constraints:

$Q_i \neq Q_j$ (cannot be in the same row)

$| Q_i - Q_j | \neq |i - j|$ (or same diagonal)

Translate each constraint into set of allowable values for its variables

E.g., values for (Q1, Q2) are {(1,3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2) }

Min-conflict

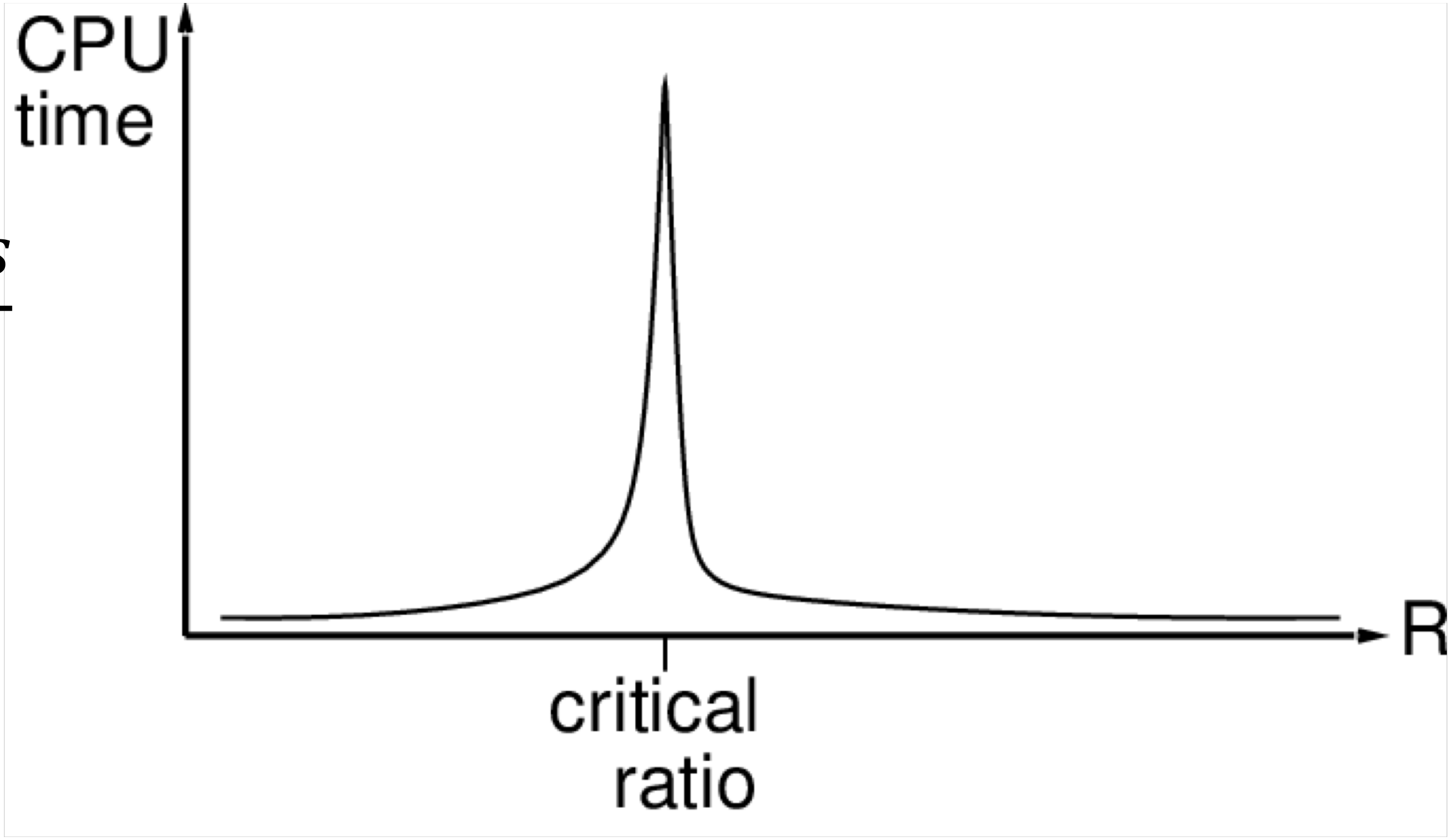
```
function Min-Conflict(csp, max-steps) returns solution/failure
    current ← an initial complete assignment for csp
    for i = 1 to max_steps do
        if current is a solution for csp then return current
        var ←random selected conflict variable
        value ← the value v for var that minimizes Conflicts(var, v, current, csp)
        set var = value in current
    Return failure
```

Performance of Min-conflicts

Given random initial state can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

The same appears to be true for any randomly-generated CSP except in a narrow range of the ratio.

$$R = \frac{number\ of\ constraints}{number\ of\ variables}$$



CSP Summary

CSPs are a special kind of search problem:

States defined by values of a fixed set of variables

Goal test defined by constraints on variable values

- Backtracking = DFS with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- The CSP representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice

Define

Take a minute a write down a definition for the following:

- Backtracking search
- Min-conflicts
- Cutset cycle