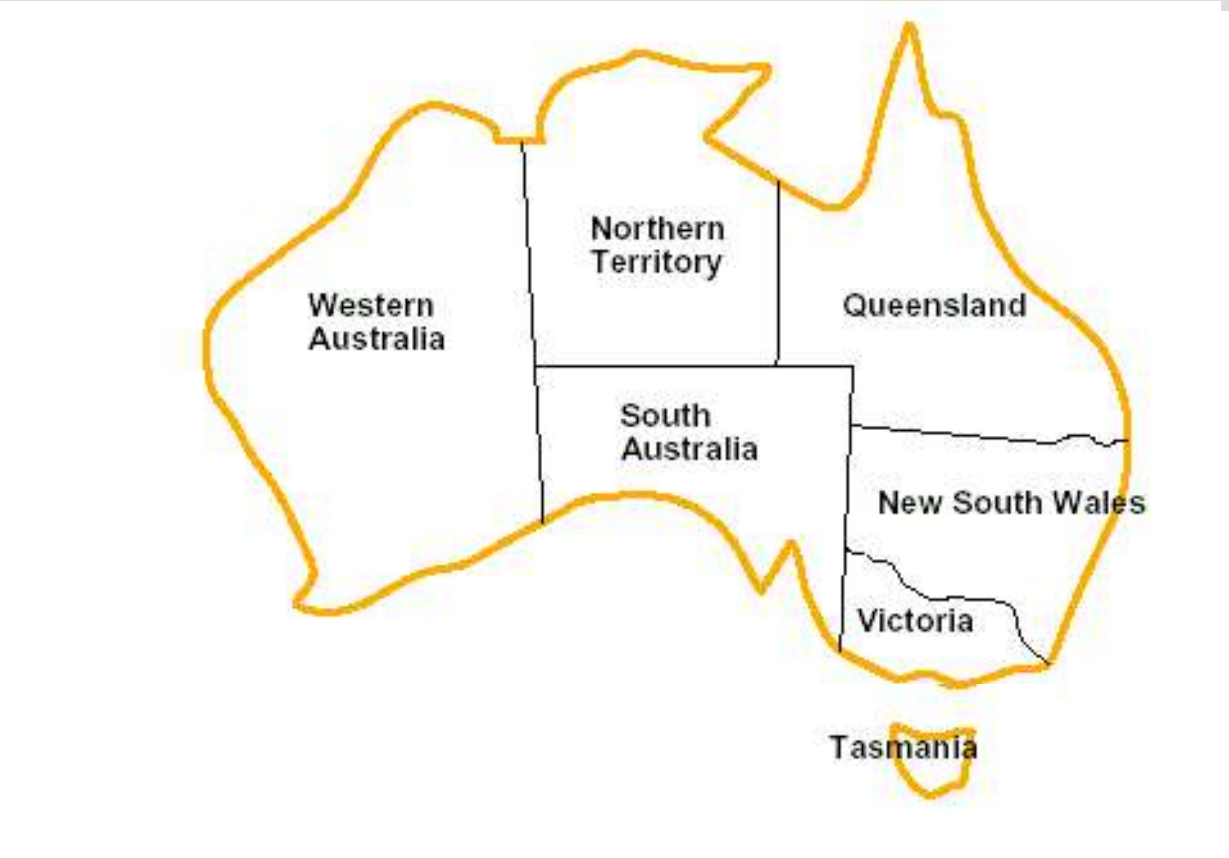
CSC421 Intro to Artificial Intelligence

UNIT 06:
Constraint Satisfaction Problems II

Example: Map coloring



Constraints:
adjacent regions must have different colors

WA ≠ NT (if the language allows it)
or
(WA, NT) ∈ {(red, green), (red, blue), (blue, green), (blue, red), (green, red), (green, blue)}

VARIABLES: WA, NT, Q, NSW, V, SA, T
Domains : {red, green, blue}

More map coloring

How would you formulate as a "standard" search problem ?

Other examples of CSPs ?

Solutions are assignments that satisfy all the constraints
{WA = red, NT = green, SA = blue, Q = red, NS = green,
V = red, T = green}

Standard Search Formulation

Standard Search Formulation

- Straightforward, dumb approach
  - States := the values assigned so far
  - Initial state: the empty assignment, {}
  - Succesor function: assign a value to an unassigned variable that doesn't conflict with current assignment
  - Goal test: the current assignment is complete
- Every solution appears at depth n with n variables (DFS) - path irrelevant
- But
  - b = (n-l)* d at depth l, so n! d^N leaves

Backtracking search

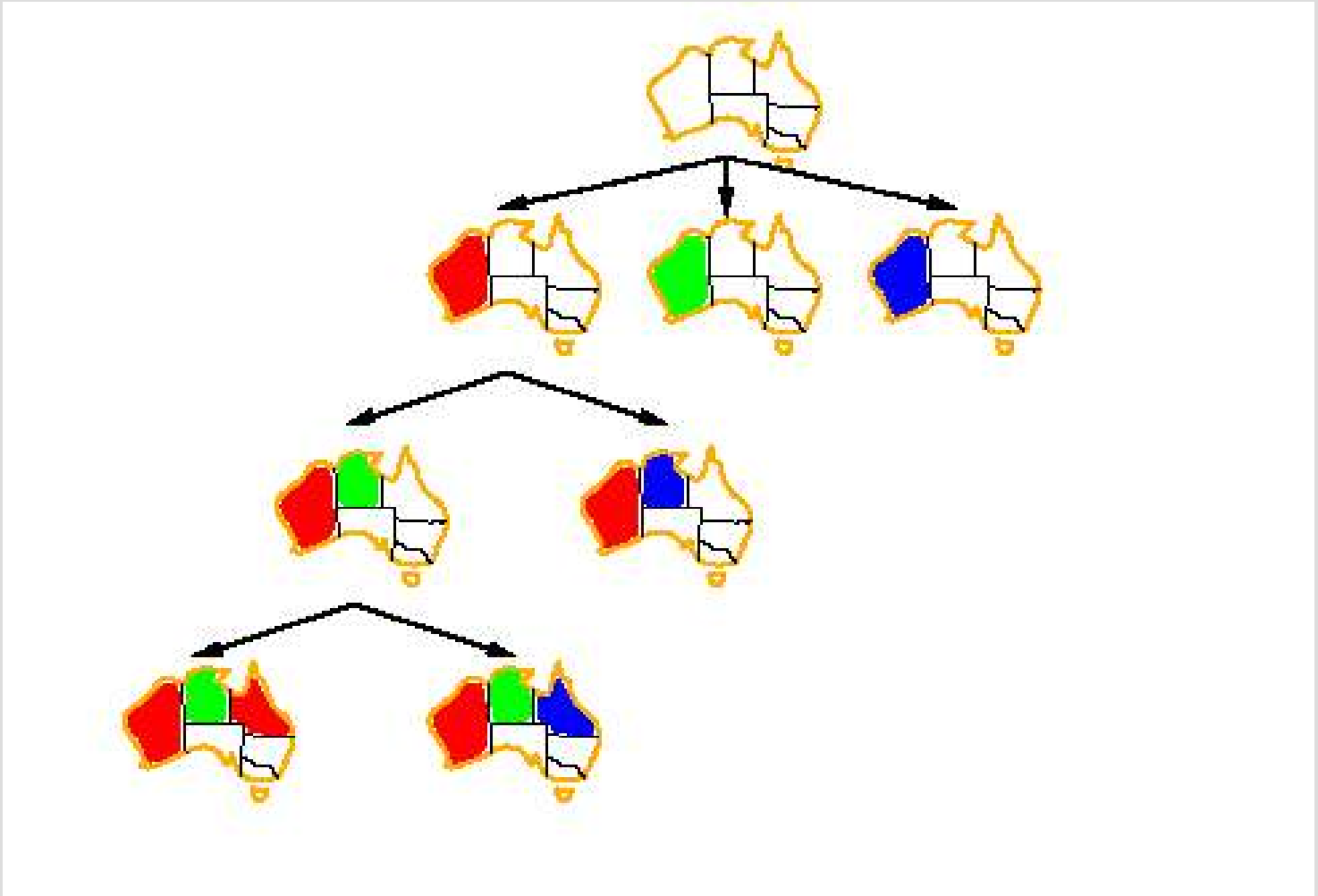
- Variable assignments are commutative, i.e.,
  - [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each node b = d and there are d^N
- DFS for CSP with single variable assignment is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSP

Backtracking

```
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
```

Example



Improving backtracking efficiency

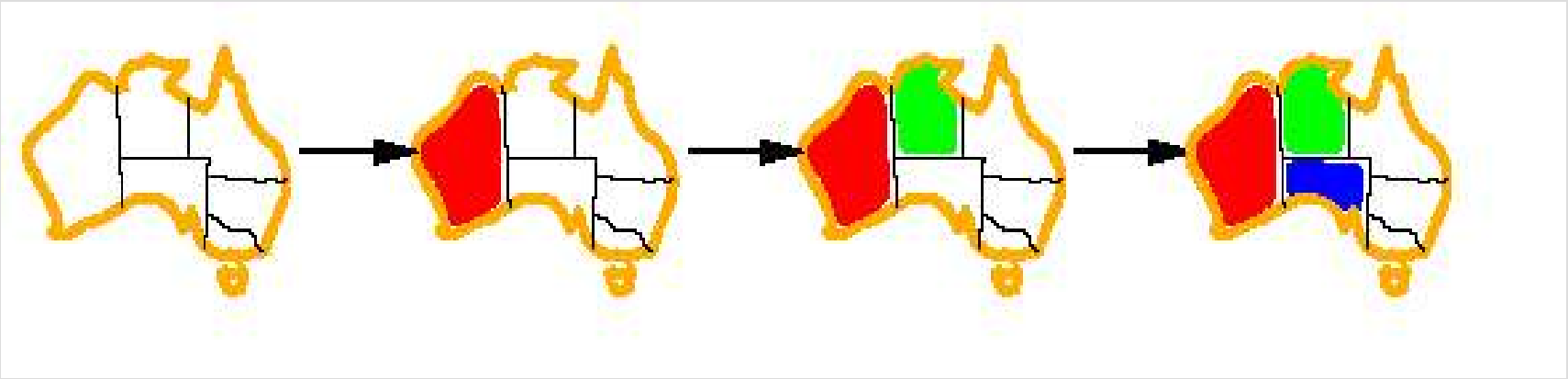
- Any ideas ?

Improving Backtracking Efficiency

- General-purpose methods can give huge gains in speed:
  - Which variables should be assigned next ?
  - In what order should its values be tried ?
  - Can we detect inevitable failure early ?
  - Can we take advantage of problem structure ?

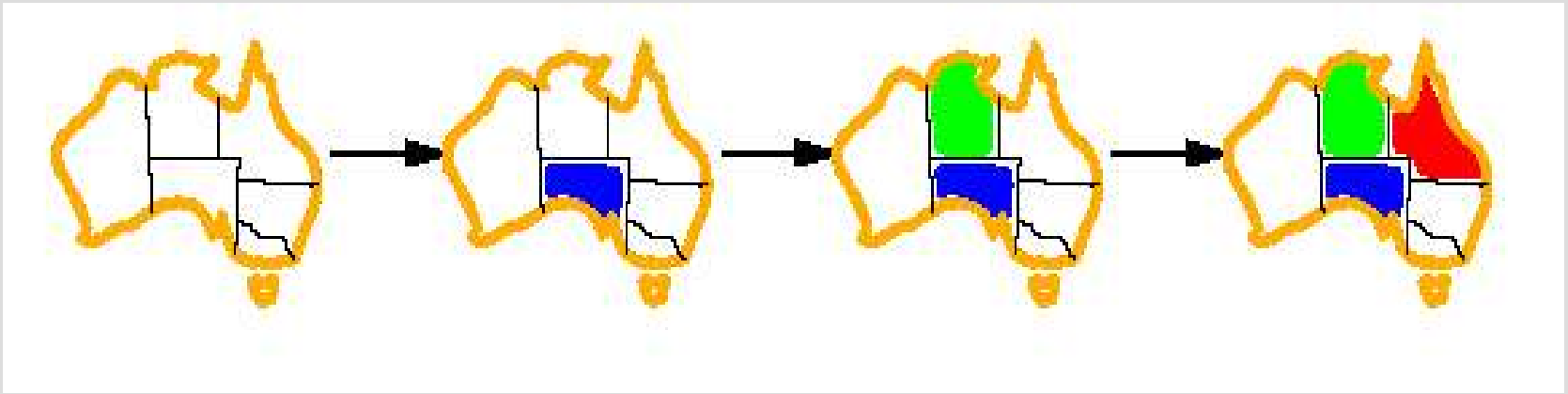
Minimum Remaining Values

- Choose the variable with the fewest legal values



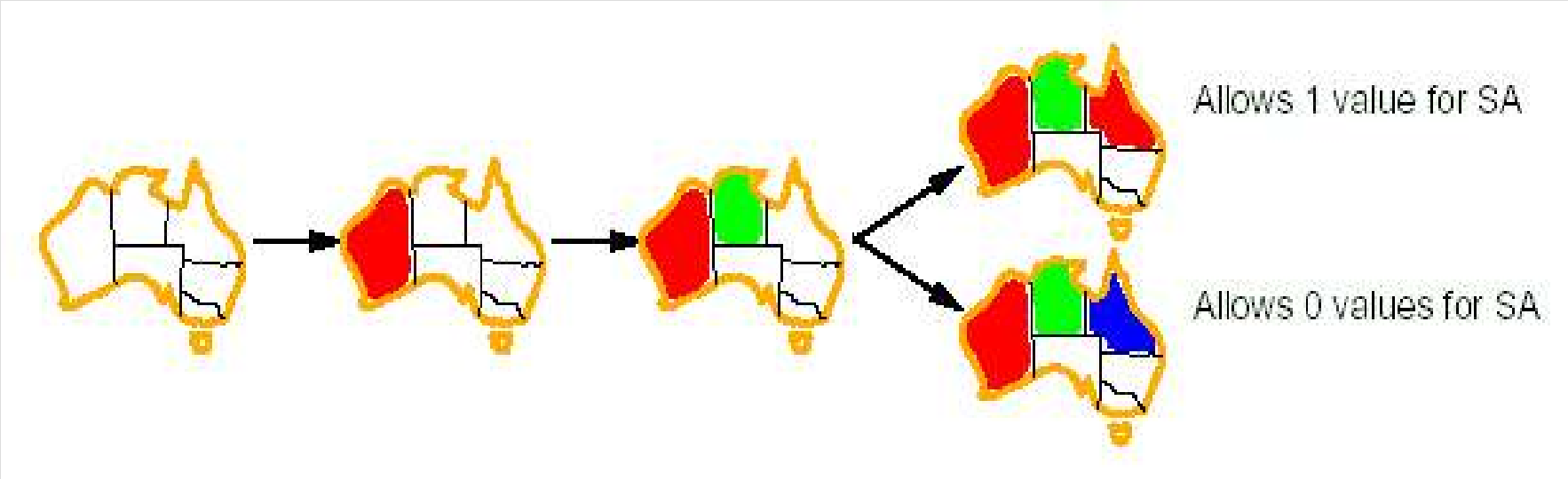
Degree Heuristics

- Tie-breaker among MRV variables
- Degree heuristic:
  - Choose the variable with the most constraints on remaining variables
  - For example SA has the highest degree



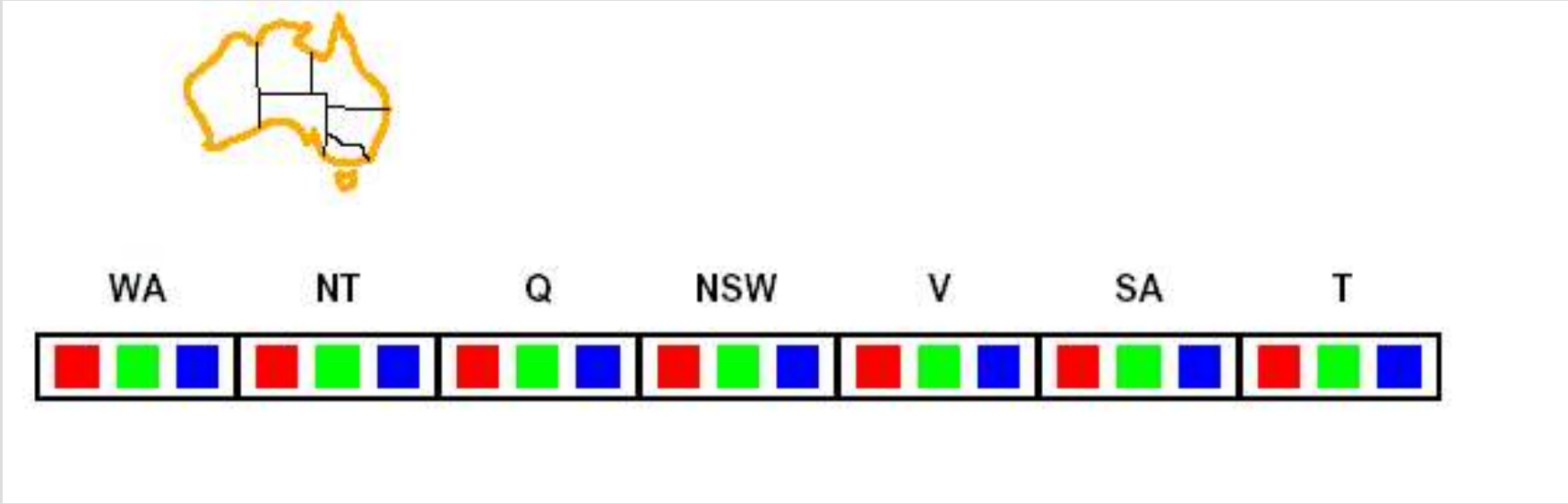
Least constraining value

- Prefers value that rules out the fewest choices for the neighboring variables in the constraint graph

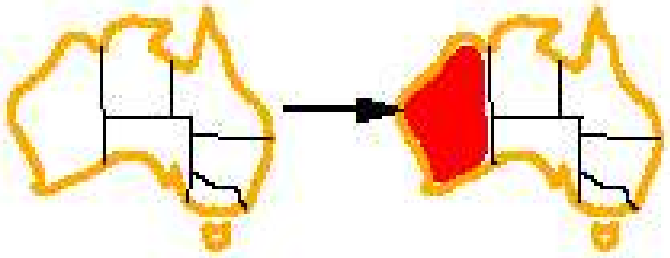


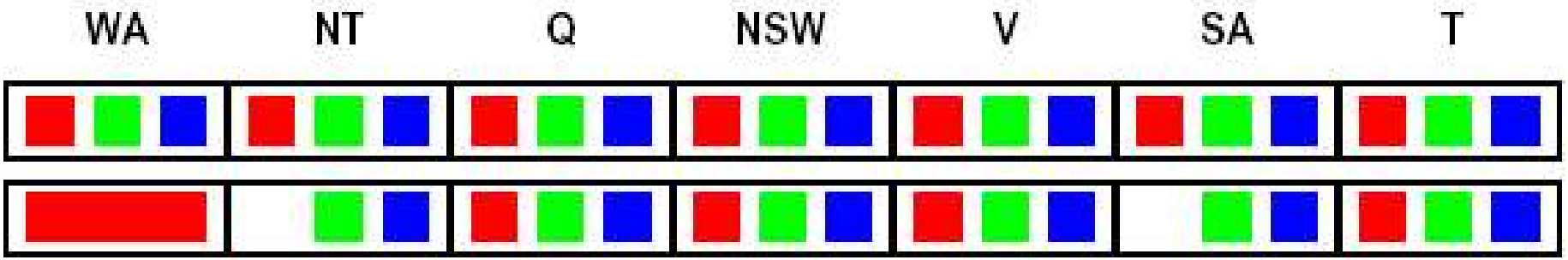
Forward Checking

- Idea: keep track of remaining legal values for unassigned variables

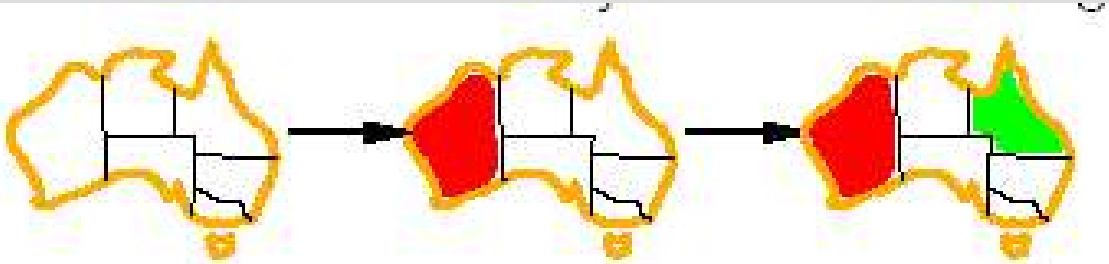


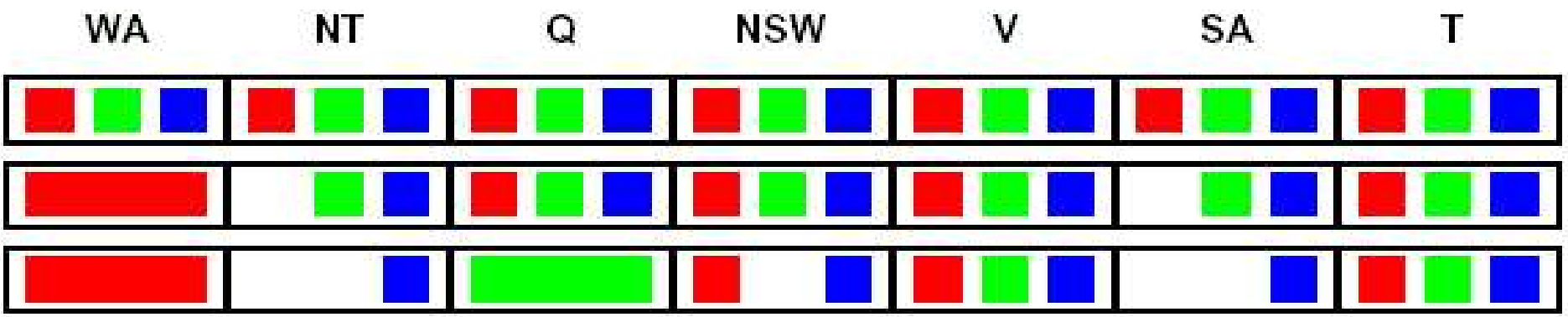
Forward Checking





Forward Checking

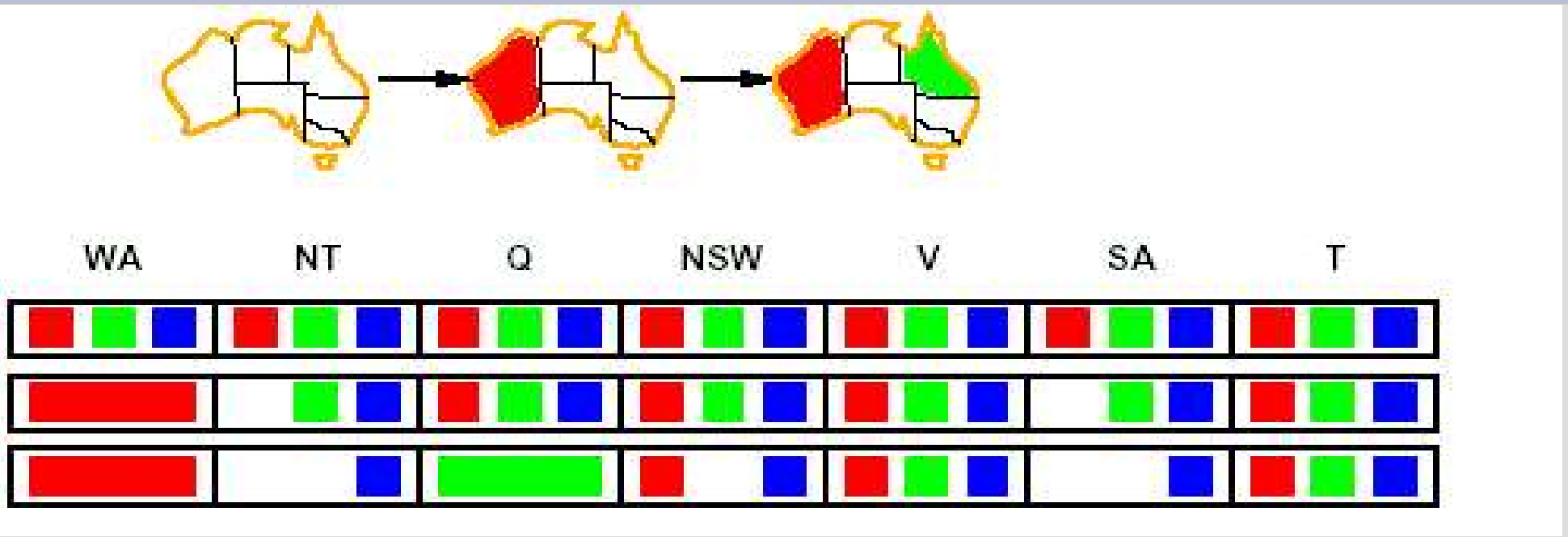




Constraint Propagation

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

Example



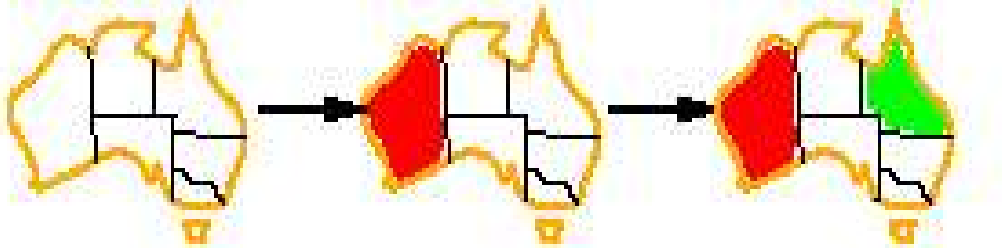
NT and SA can not
both be blue

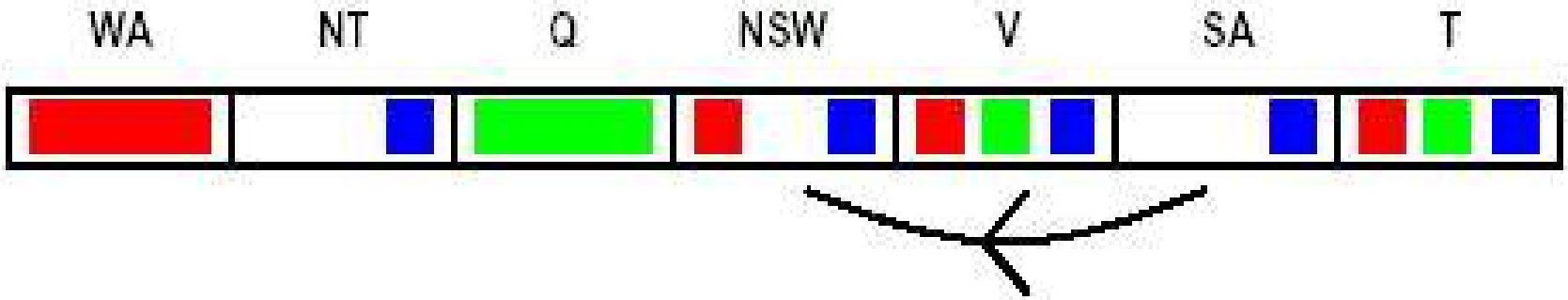
Constraint propagation repeatedly enforces constraints locally – has to be done efficiently otherwise simple search might be faster

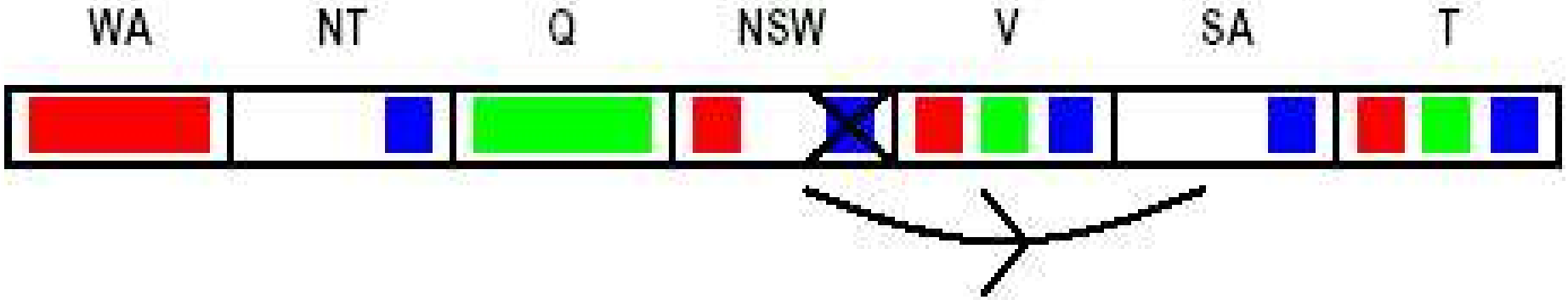
Arc consistency

- Stronger form of constraint propagation than forward checking
- Arc consistent
  - X -> Y is consistent iff for every value x of X there is some allowed value y of Y
- Does X->Y imply Y->X ?

Arc consistency







K-consistency

- Arc consistency either as pre-processor or after every assignment
- Stronger forms k-consistency
- Determining appropriate level of consistency checking is mostly an empirical science
- Special constraints -> special purpose constraint propagation
    - Alldiff
    - Resource constraint – bounds propagation

Problem structure

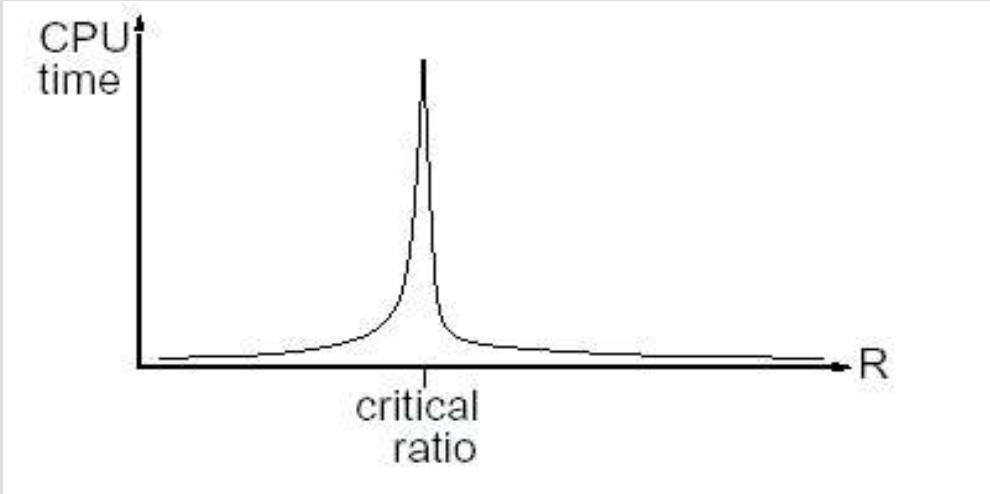
- Connected components
- Tree structured CSP
  - Relation of syntactic restrictions and complexity of reasoning
- Nearly tree-structured CSP
  - Cutstet conditioning

Iterative algorithms for CSP

- Hill-climbing, simulated annealing typically work with "complete" states, i.e. 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: Min-conflict heuristic
  - Choose value that violates the fewest constraints
  - Hill climbing with h(n) = # violated constraints

Performance of min-conflict

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability
- Appears to be true for any randomly generated CSP except narrow range of critical ratio R = # constraints / # of variables



Summary I

- CSPs are a special kind of problem
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints of variable values
- Back-tracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics can help significantly
- Forward checking prevents assignments that guarantee later failure

Summary II

- Constraint propagation such as arc consistency does additional work to constrain values and detect inconsistencies
- The CSP representation allows analysis of problem structure
- Tree-structured CSP can be solved in linear time
- Iterative min-conflicts is usually effective in practice