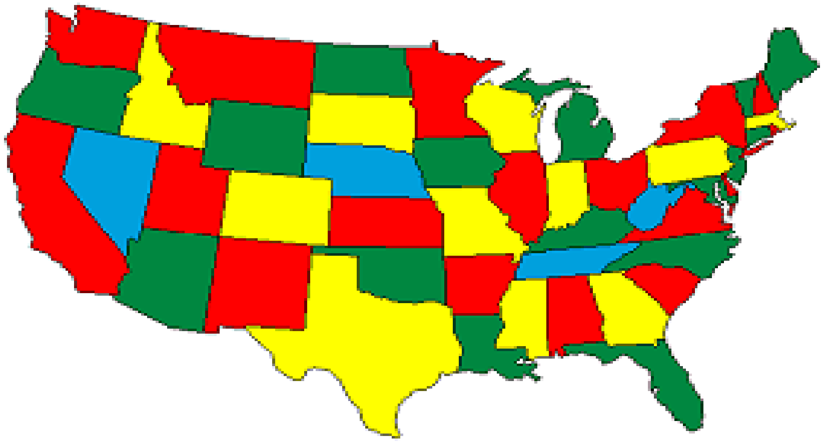
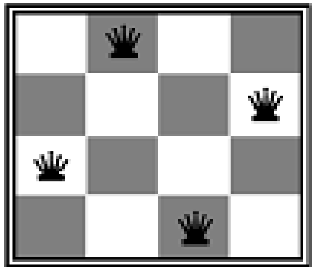
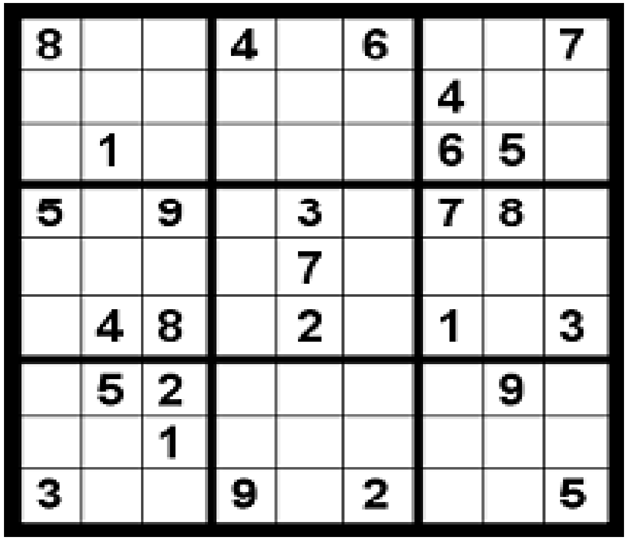
Constraint Satisfaction Problems

Chapter 5







Constraint Satisfaction Problem(CSP)

CSP:

- Set of variables $\{X1, X2, \ldots, Xn\}$ that each variable Xi has a value from domain *Di*
  - Usually *Di* is discrete and finite
- Set of constraints $\{C1, C2, \ldots, Cp\}$ that specify allowable combinations of values for subsets of variables (goal test)
- Solution(goal): Assign a value to every variable such that all constraints are satisfied

Example: map-coloring

- Variables:

  WA, NT, Q, NSW, V, SA, T

- Domain: D1={***red***, ***green***, ***blue***}
- Constraint: adjacent regions must have different colors

  $WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q$,

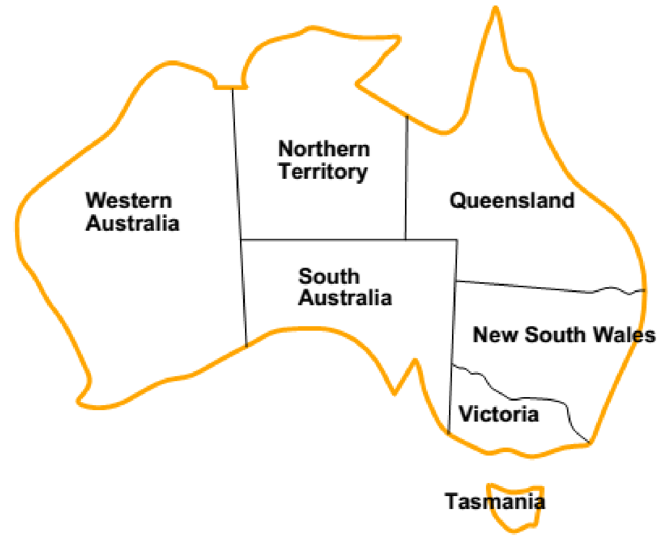
  $SA \neq Q$, $SA \neq NSW$, $SA \neq V$, $Q \neq NSW$, $NSW \neq V$

- Solutions are complete and consistent assignments

  e.g., WA = *red*, NT = *green*, Q = *red*,

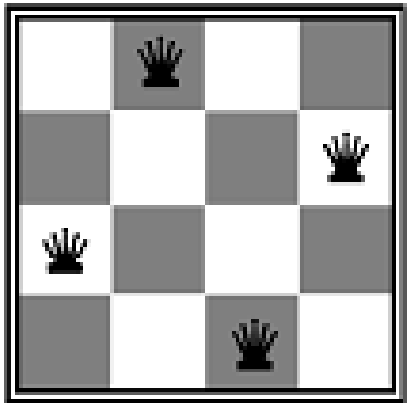
  NSW = *green*, V = *red*, SA = *blue*,

  T = *green*



solution

Example: 8-Queens Problem

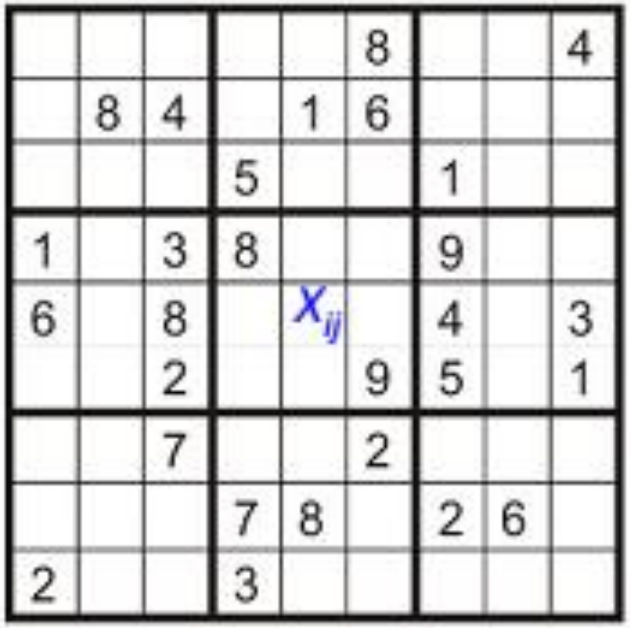


- 8 variables X_i, i = 1 to 8
- Domain for each variable {1,2,…,8}
- Constraints are of the forms:
  - $X_i = k \rightarrow X_j \neq k$ for all $j = 1$ to 8, $j \neq i$
  - $X_i = k_i, X_j = k_j \rightarrow |i-j| \neq |k_i - k_j|$ for all $j = 1$ to 8, $j \neq i$

Example: Sudoku

- Variables: *Xij*
- Domains: {1, 2, …, 9}
- Constraints:

  Alldiff (X in the same unit)

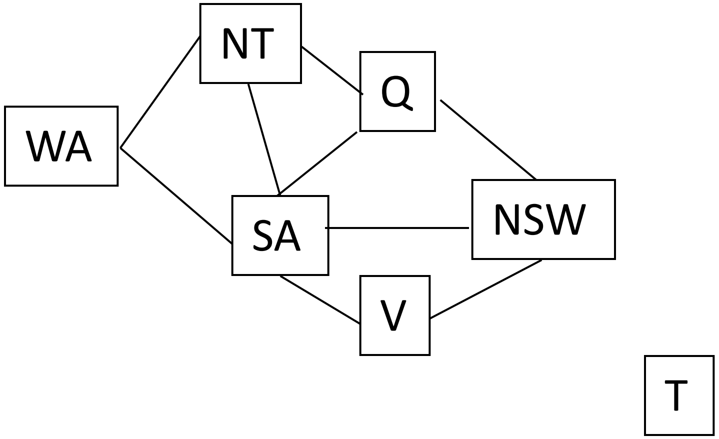


Varieties of constraints

- **Unary** constraints involve a single variable.
  - E.g., $SA \neq green$
- **Binary** constraints involve pairs of variables.
  - E.g., $SA \neq WA$
- **Higher-order** constraint involve 3 or more variables.
- **preferences** (soft constraints), e.g., red is better than green

Constraint Graph

- **Constraint graph:**
  - Nodes are variables
  - Arcs show constraints
    (Two variables are adjacent or neighbors if they are connected by an edge or an arc)



$WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q$, $SA \neq Q$, $SA \neq NSW$, $SA \neq V$, $Q \neq NSW$, $NSW \neq V$

Real-word CSPs

- Assignment problems

  –e.g., who teaches what class

- Timetable problems

  e.g., which class is offered when and where?

- Transportation scheduling

- More examples of CSPs: http://www.csplib.org/

Standard search formulation (incremental)

- **states:**
  - Values assigned so far
- **Initial state:**
  - The empty assignment { }
- **Successor function:**
  - Choose any unassigned variable and assign to it a value that does not violate any constraints.
- **Goal test:**
  - The current assignment is complete and satisfies all constraints

Standard search formulation (incremental)

- What is the **depth of any solution**(assuming ***n*** variables)?

  n–This is the good news.

- Given that there are ***m*** possible values for any variable, how many paths are there in the search tree?
- ***n!·m^n*** –This is the bad news
- How can we **reduce** the branching factor?

Backtracking search

- In CSP's, variable assignments are **commutative:**

  i.e., [*WA* = *red* then *NT* = *green*] is the same as [*NT* = *green* then *WA* = *red*]

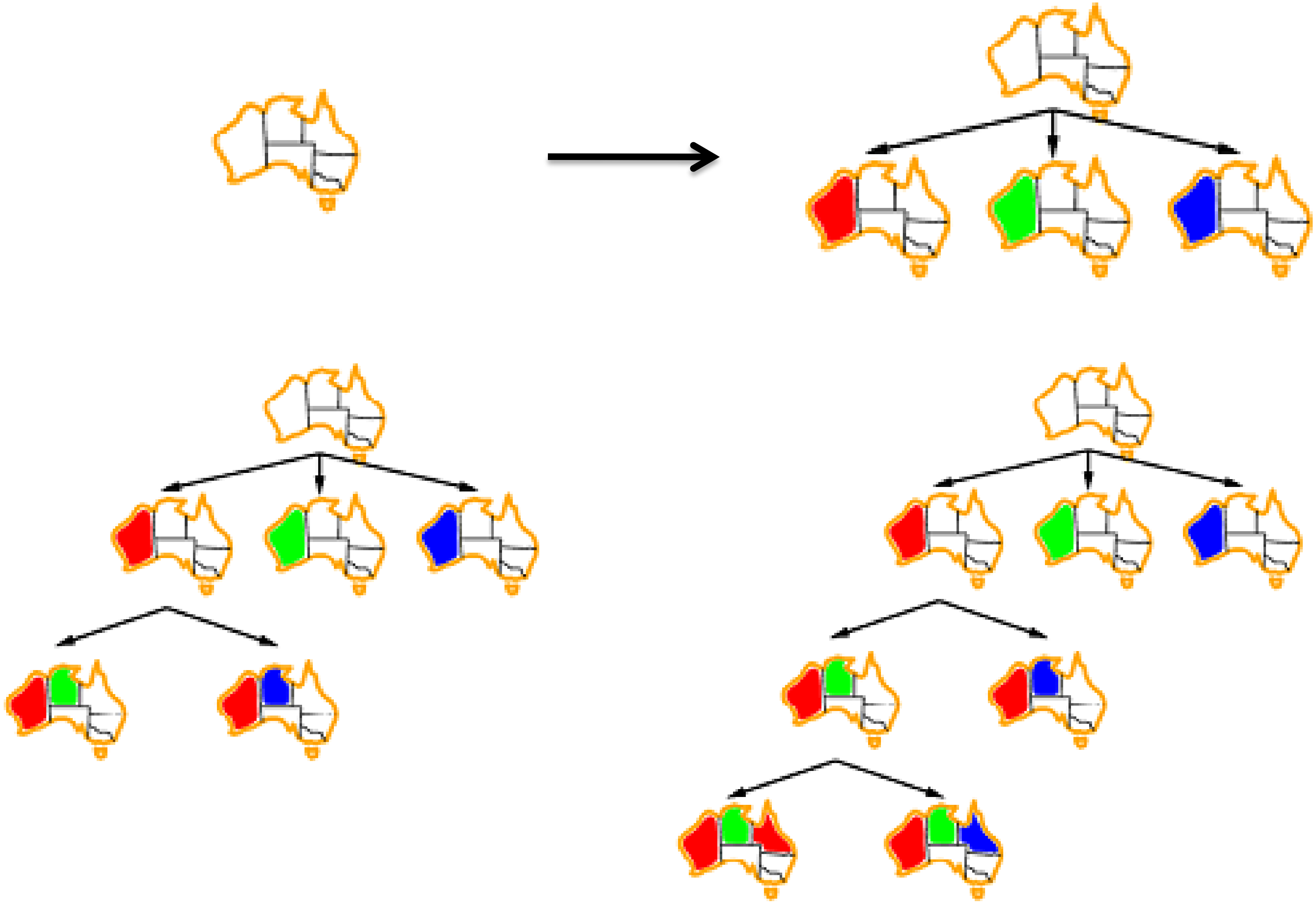
- only need to consider assignments to a single variable at each level ⟶ $b=d$ and there are d^n leaves.
- Depth-first search for CSPs with single-variable assignments is called **backtracking search.**
- Backtracking search is the basic uniformed algorithm for CSPs

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
```

Example



Improving backtracking efficiency

1. which variable should be assighned next?
2. In what order should its values be tried?
3. can we detect invitable failure early?
4. can we take advantage of problem structure?

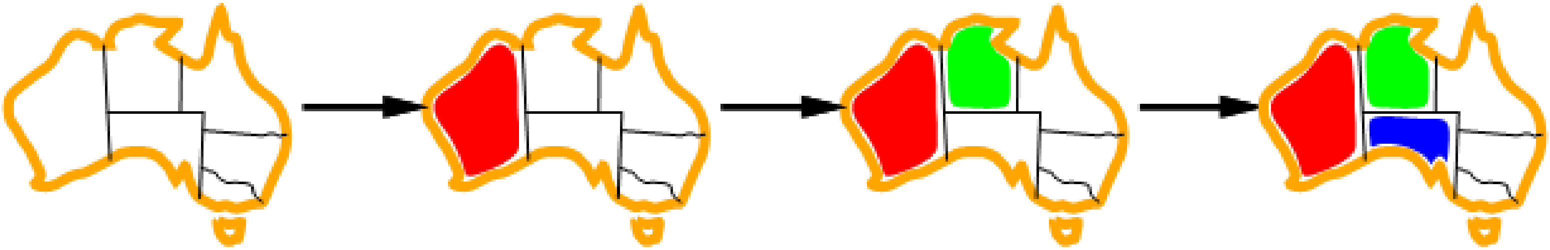
Which variable should be assigned next?

- **Minimum remaining values** (MRV)
  - Choose the variable with **the fewest** legal values

- **Tie-breaker among among most constrained variables (degree heuristic)**
  - Choose the variable with **the most** constraints on remaining variable

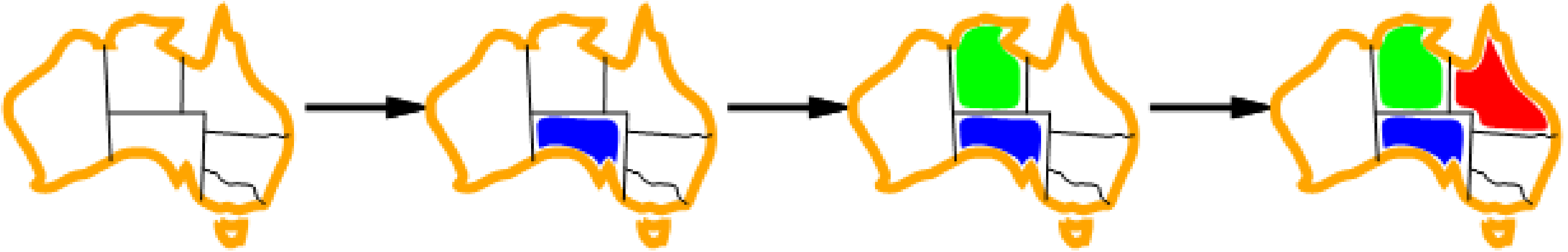
Minimum remaining values

– Choose the variable with **the fewest** legal values



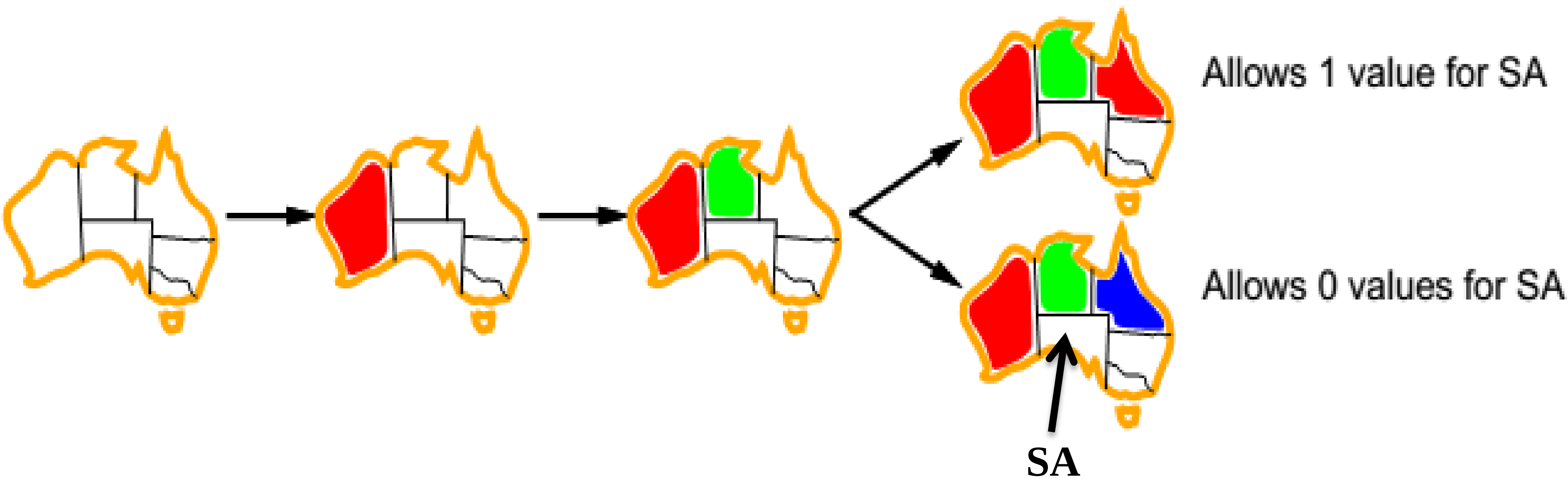
degree heuristic

- Choose the variable with **the most** constraints on remaining variable



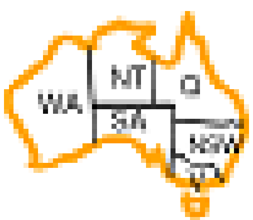
Given a variable, in which order should its values be tried?

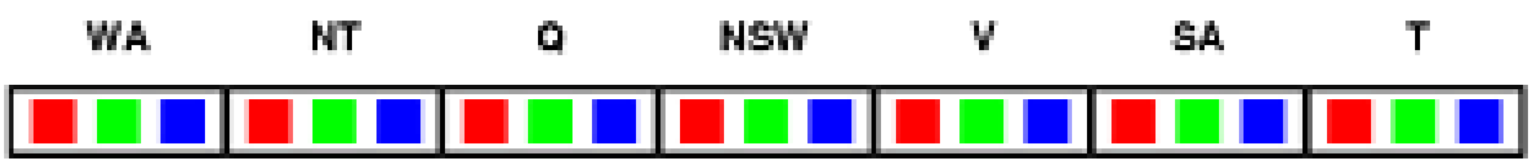
- Choose the **least constraining value**:
  - The value that rules out **the fewest** values in the remaining variables.



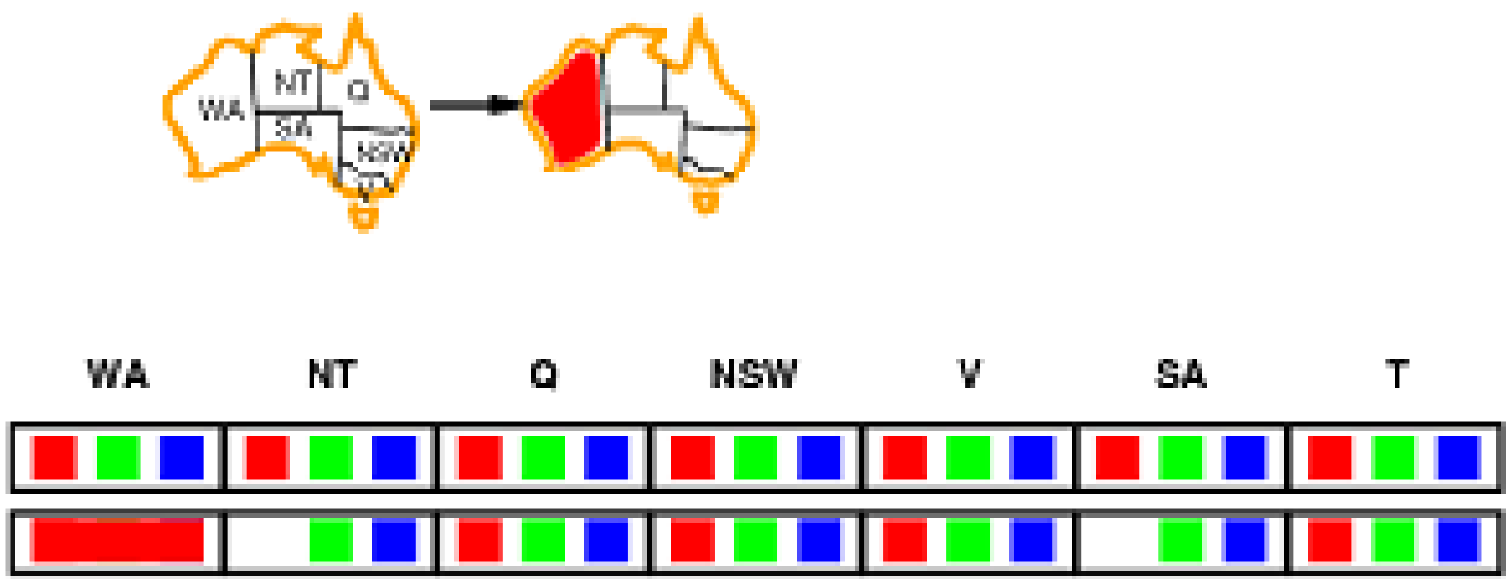
Early detection of failure: Forward checking

- **Idea:** keep track of remaining legal values for unassigned variables
  - Terminate search when any variables has no legal values

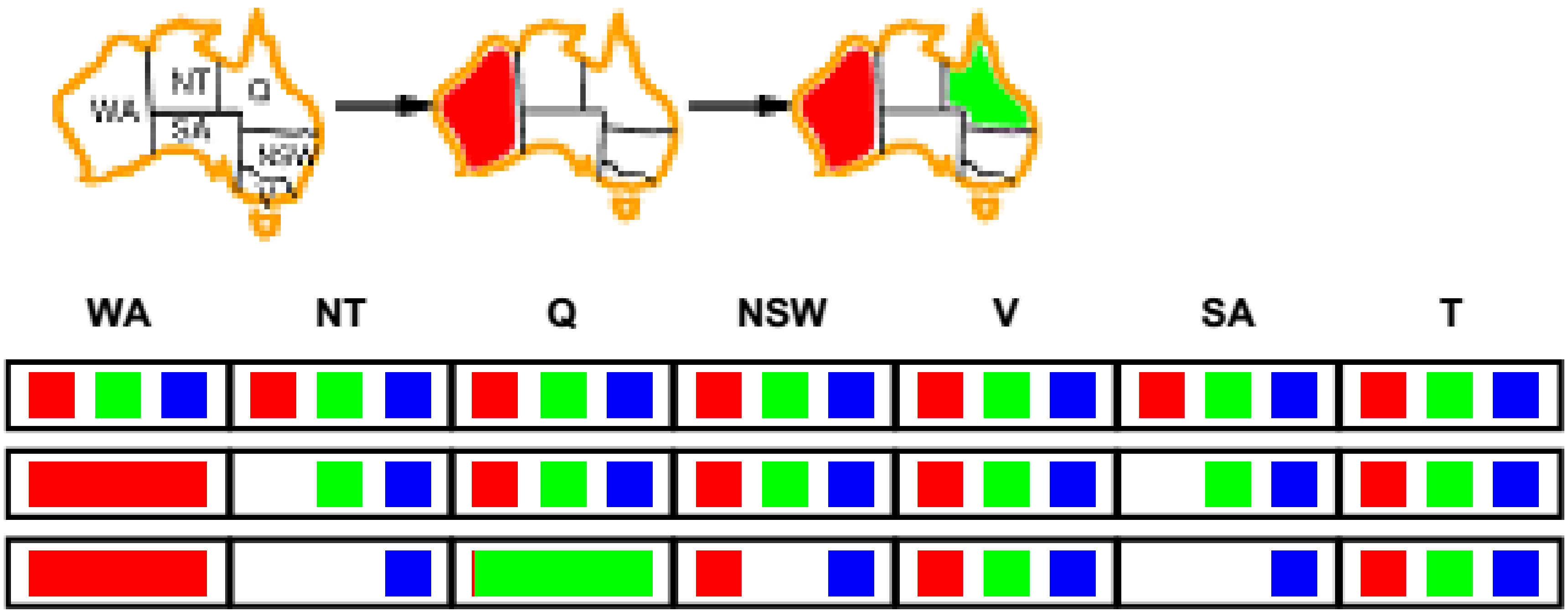




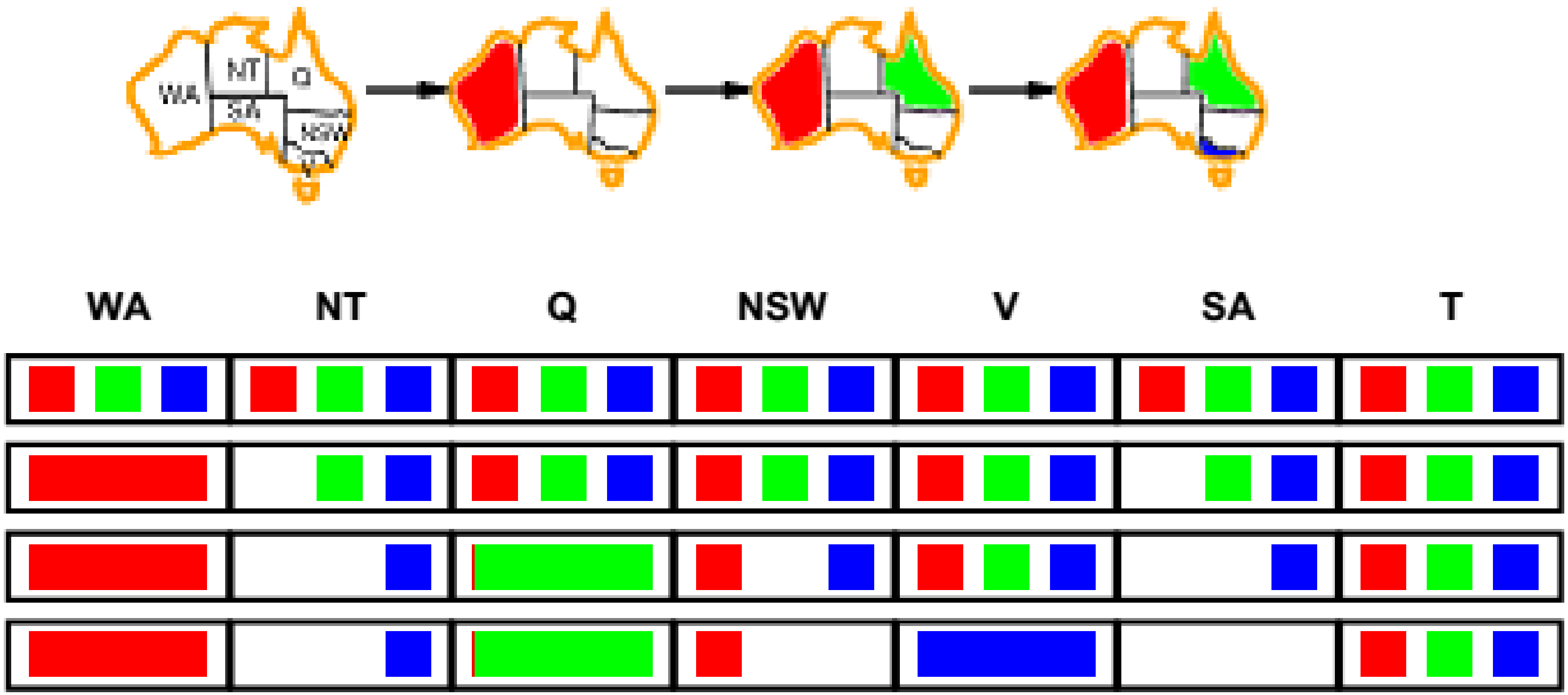
Forward checking



Forward checking

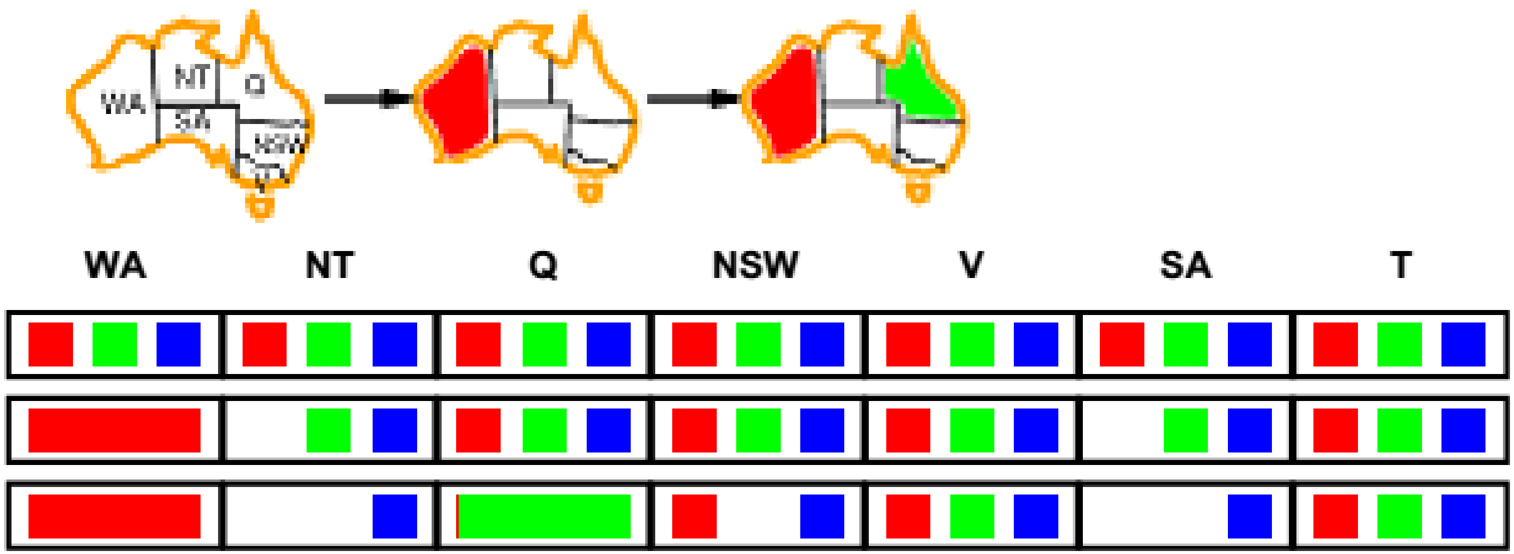


Forward checking



Constraint propagation

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

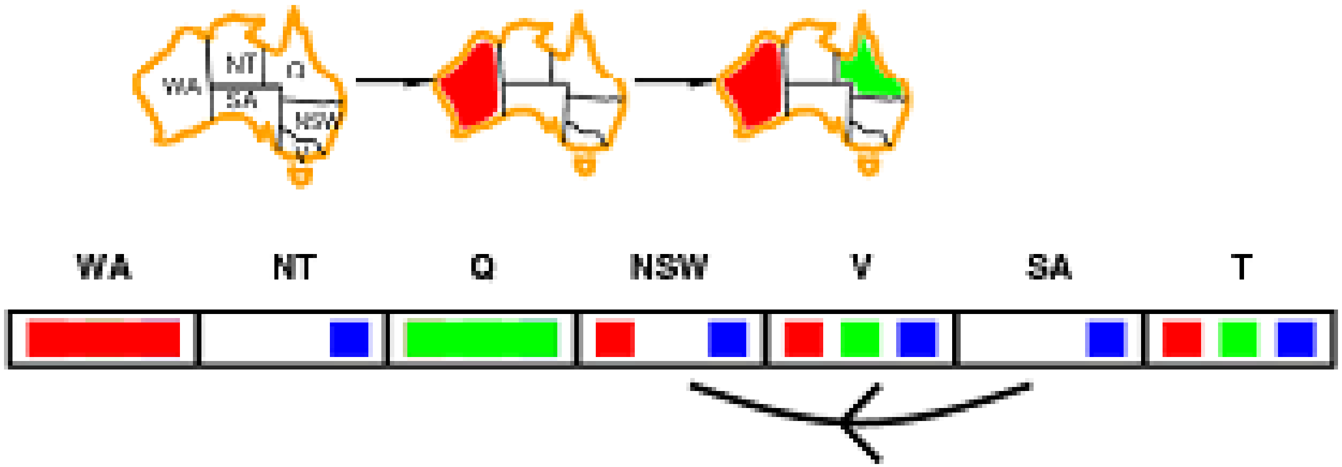


- *NT* and *SA* can not both be blue!!!

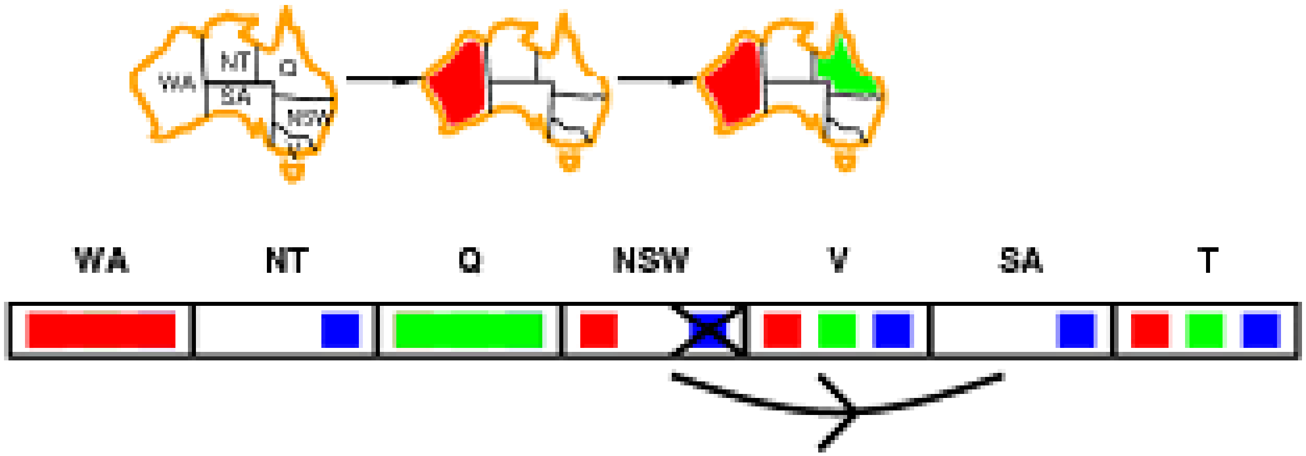
Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is **consistent** iff

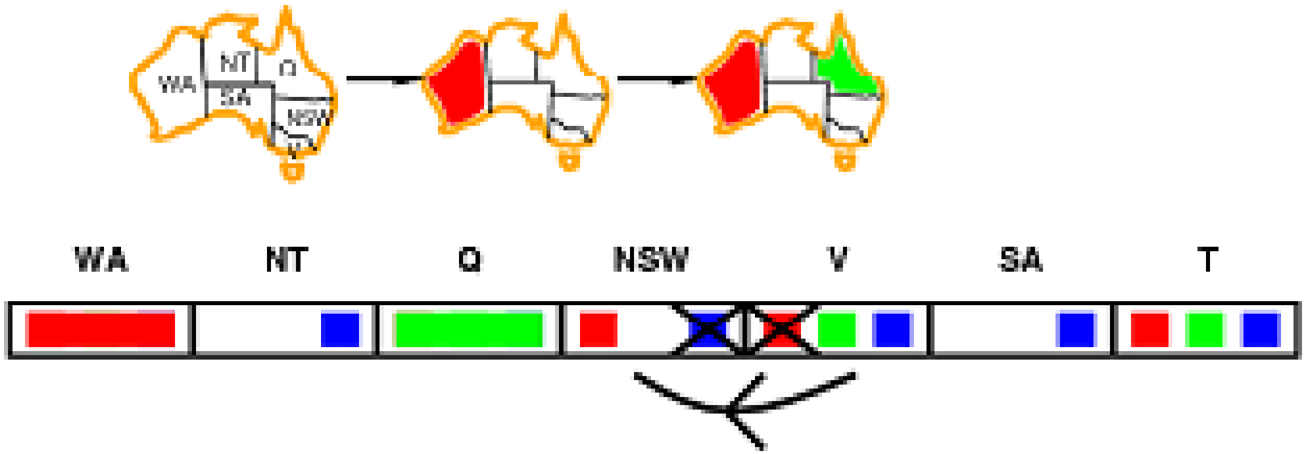
  for **every** value x of X there is **some** allowed value of Y
- When checking $X \rightarrow Y$ throw out any values of X for which there isn't an allowed value of Y



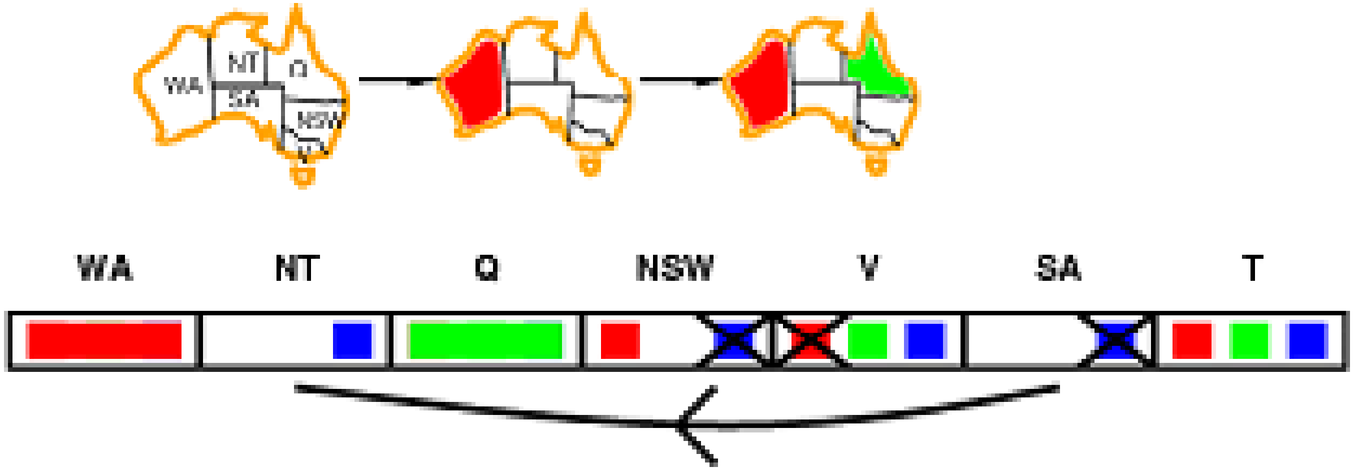
Arc consistency



If *X* loses a value, neighbors of *X* need to rechecked

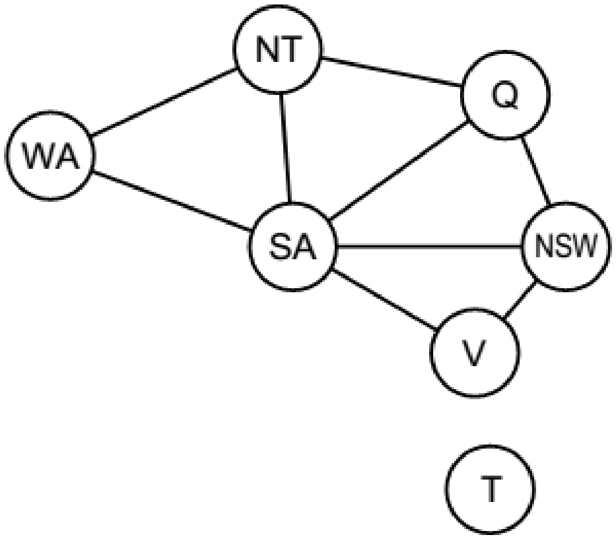


Arc consistency



Arc consistency detects failures earlier than forward checking

Can we take advantage of problem structure?



- *T* is **independent subproblems**
- Suppose each subproblem has ***C*** variables out of ***n*** total
  - Worst-case solution cost is ***n/c .d^c***

E.g., $n = 80$, $d = 2$, $c = 20$

2^{80} = 4 billion years at 10 million nodes/sec

$4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

End of chapter 5