Outline

- Best-first search
  - Greedy best-first search
  - A* search
  - Heuristics
- Local search algorithms
  - Hill-climbing search
  - Beam search
  - Simulated annealing search
  - Genetic algorithms
- Constraint Satisfaction Problems

Constraint Satisfaction Problems

Special Type of search problem:

- 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

- Examples:
  - Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

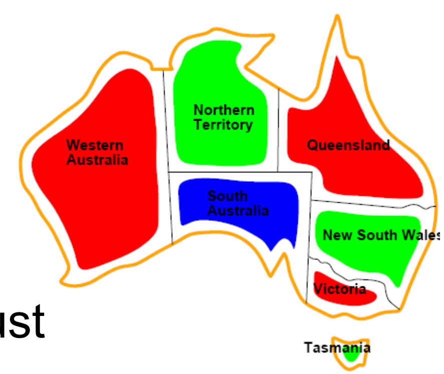
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

  - cryptarithmetic puzzle

```
   SEND
+  MORE
--------
  MONEY
```

  - n-queens
  - Graph/Map-Coloring

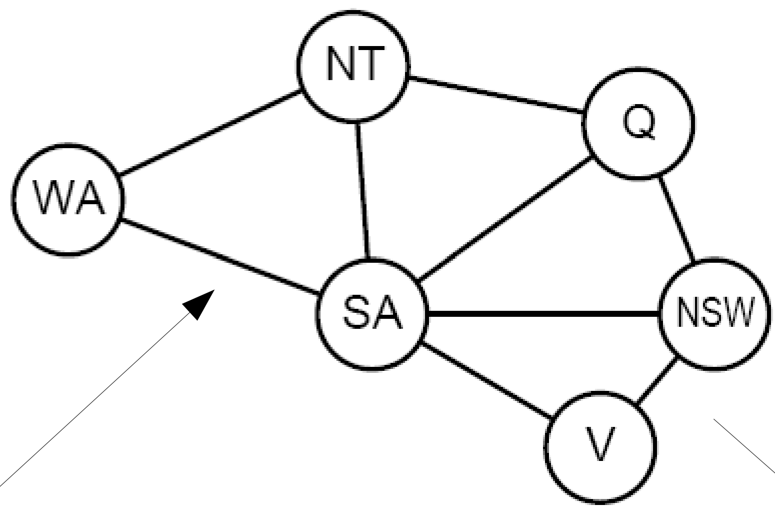
only n colors, neighbors must be different



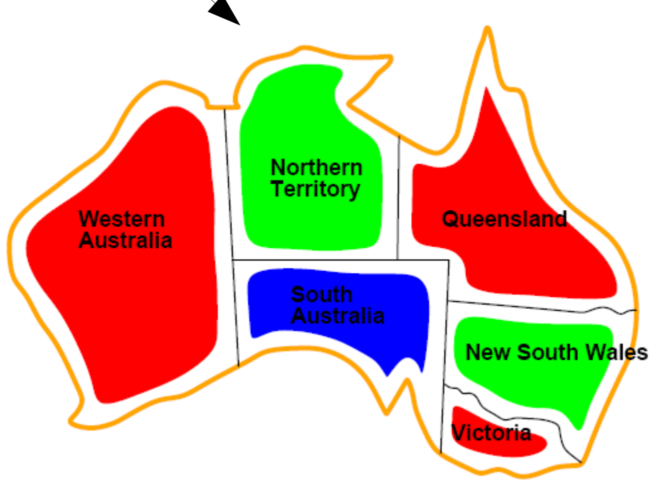
- Real-world:
  - assignment problems
  - timetables
    - classes, lecturers rooms, studies
  - ...

Constraint Graph

- nodes are variables
- edges indicate constraints between them

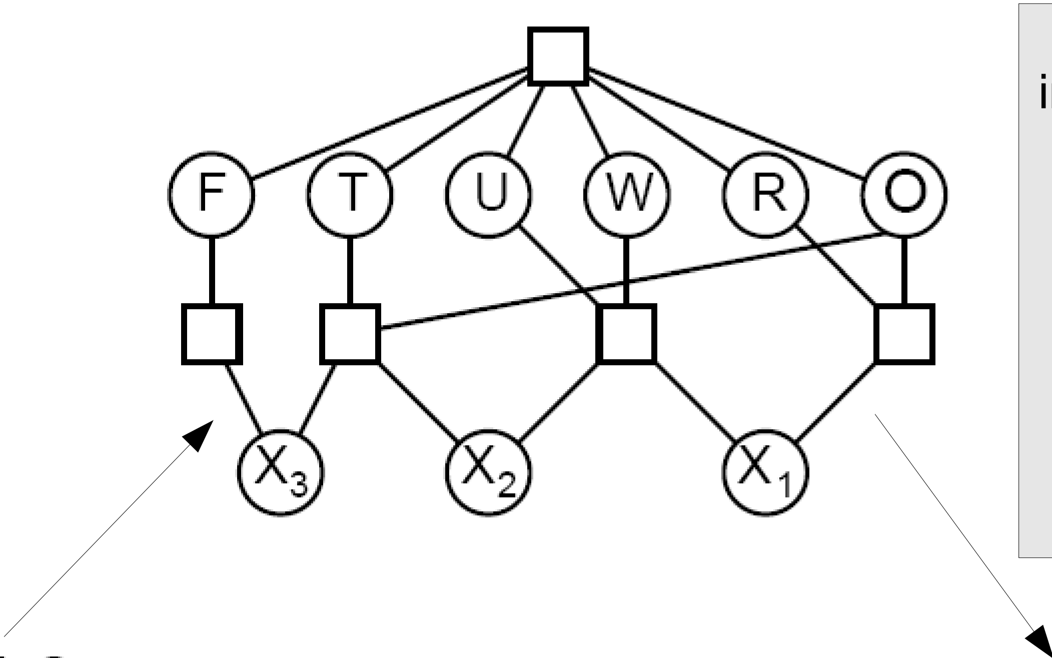


Two neighboring nodes must not have the same color



Constraint Graph

- nodes are variables
- edges indicate constraints between them



Connected nodes are involved in (in-)equations:

$$2 \cdot O = 10 \cdot X_1 + R$$
$$2 \cdot W + X_1 = 10 \cdot X_2 + U$$
$$2 \cdot T + X_2 = 10 \cdot X_3 + O$$
$$F = X_3$$
$$F \neq T \neq U \neq W \neq R \neq O$$

```
  T W O
+ T W O
-------
F O U R
```

```
  7 3 4
+ 7 3 4
-------
1 4 6 8
```

Types of Constraints

- Unary constraints involve a single variable,
  - e.g., $South\ Australia \neq green$
- Binary constraints involve pairs of variables,
  - e.g., $South\ Australia \neq Western\ Australia$
- Higher-order constraints involve 3 or more variables
  - e.g., $2 \cdot W + X_1 = 10 \cdot X_2 + U$
- Preferences (soft constraints)
  - e.g., *red is better than green*
  - are not binding, but task is to respect as many as possible

→ constrained optimization problems

Backtracking Search

- CSP are typically solved with backtracking
  - add one constraint at a time without conflict
  - succeed if a legal assignment is found

```
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
```

Worst-Case Complexity of Backtracking Search

- Assumptions
  - we have n variables
    → all solutions are a depth n in the search tree
  - all variables have v possible values
- Then
  - at level 1 we have $n \cdot v$ possible assignments
    (we can choose one of n variables and one of v values for it)
  - at level 2, we have $(n-1) \cdot v$ possible assignments for each previously assigned variable
    (we can choose one of the remaining $n-1$ variables and one of the v values for it)
  - In general: branching factor at depth l: $(n-l+1) \cdot v$
- Hence
  - The search tree has $n!v^n$ leaves

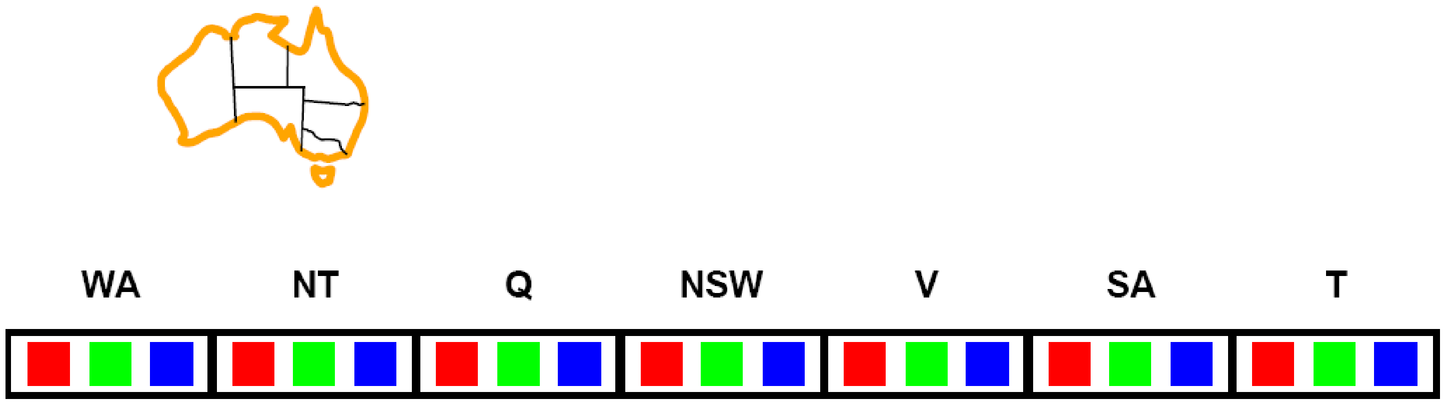
→ heuristics are needed in SELECT-UNASSIGNED-VARIABLE

General Heuristics for CSP

- Domain-Specific Heuristics
  - Depend on the particular characteristics of the problem
  - Obviously, a heuristic for the 8-puzzle can not be used for the 8-queens problem
- General-purpose heuristics
  - For CSP, good general-purpuse heuristics are known:
  - Mininum Remaining Value Heuristic
    - choose the variable with the fewest consistent values
  - Degree Heuristic
    - choose the variable that imposes the most constraints on the remaining values
  - Least Constraining Value Heuristic
    - Given a variable, choose the value that rules out the fewest values in the remaining variables
  - used in this order, these three can greatly speed up search
    - e.g., n-queens from 25 queens to 1000 queens

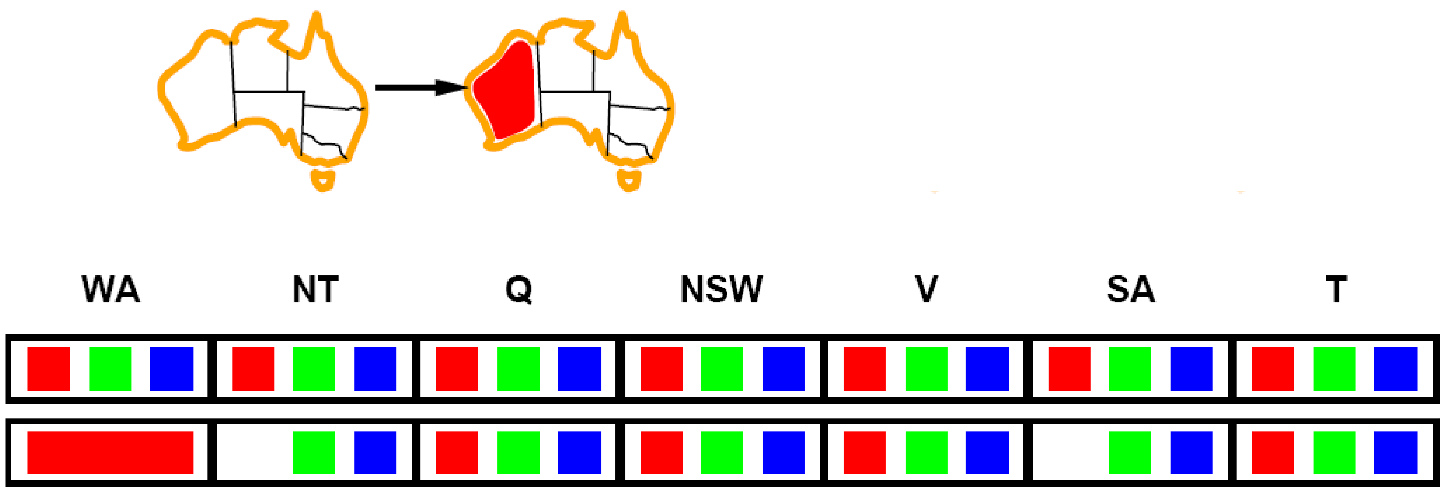
Forward Checking

- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable has no more legal values



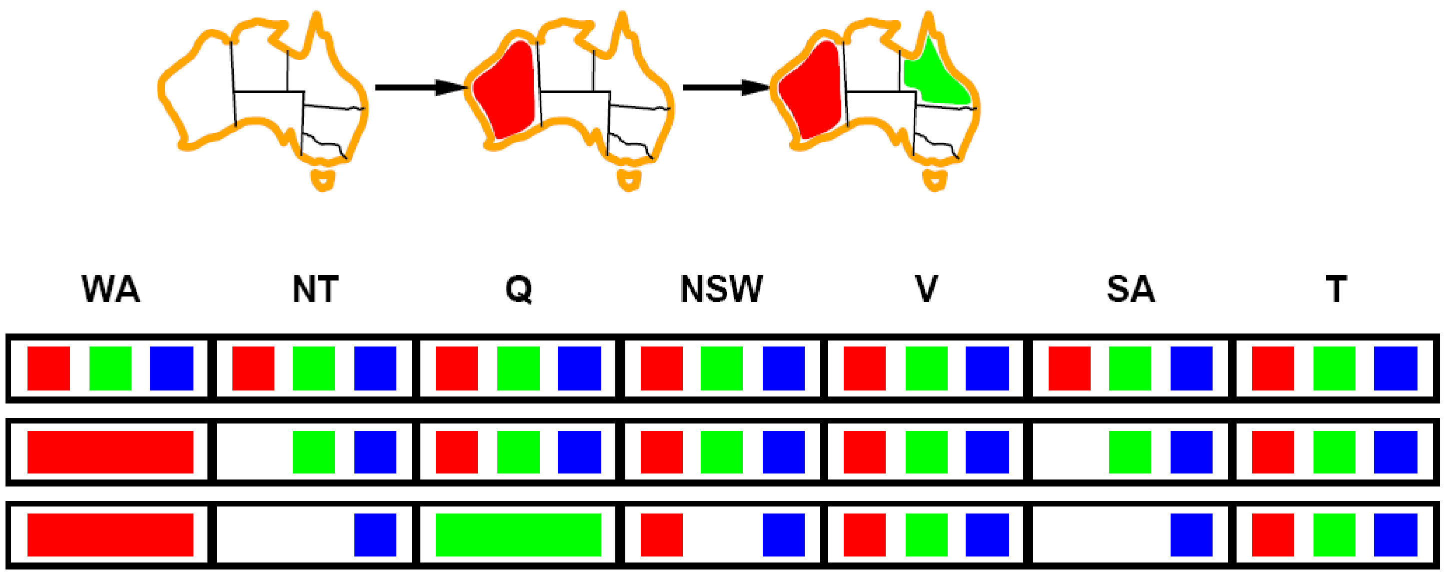
Forward Checking

- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable no legal values



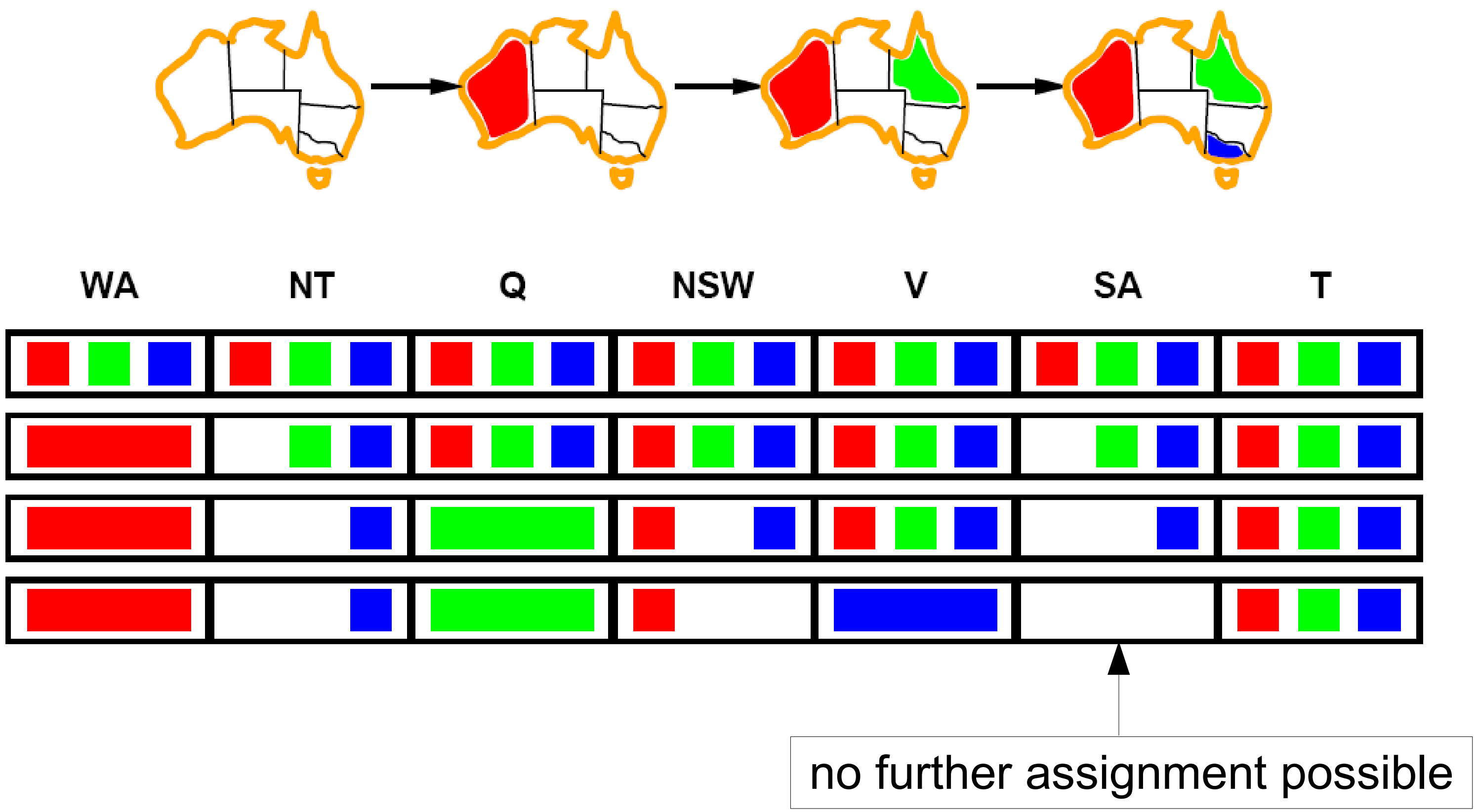
Forward Checking

- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable no legal values



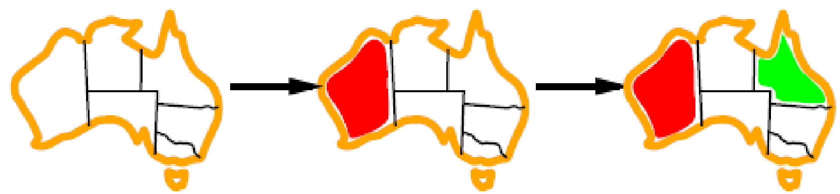
Forward Checking

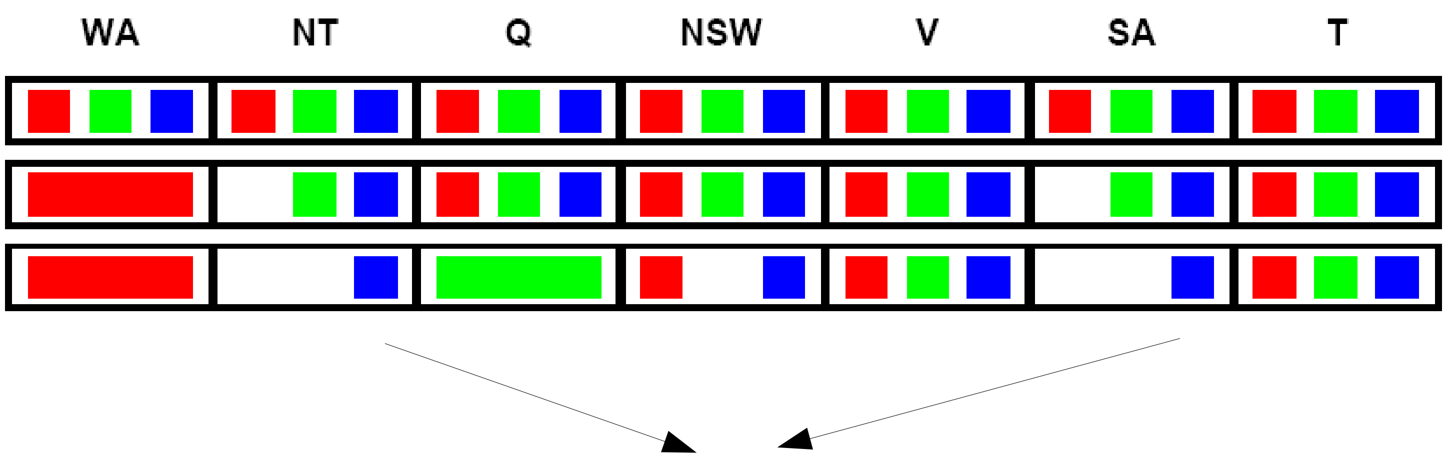
- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable no legal values



Constraint Propagation

- Problem:
  - forward checking propagates information from assigned to unassigned variables
  - but doesn't provide early detection for all failures

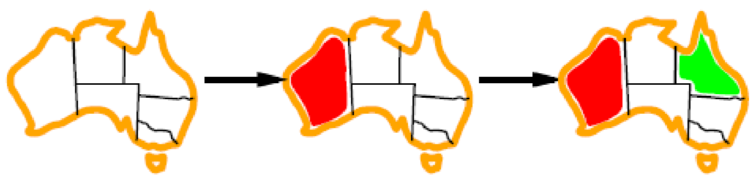


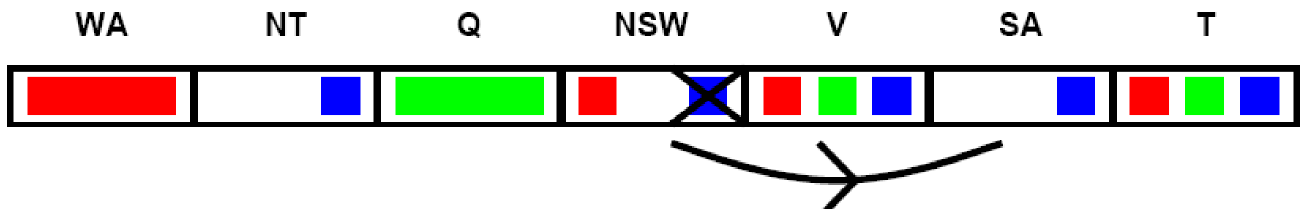


only one of them can be blue!

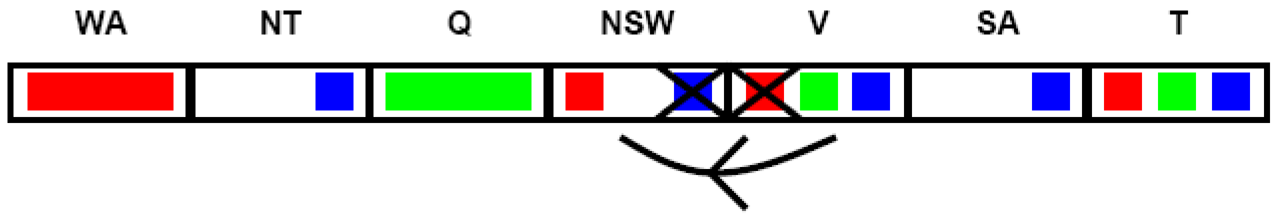
Arc Consistency

A binary constraint between variables X and Y is consistent iff for every value of X, there is some legal value for Y





- If one variable (NSW) looses a value (blue), we need to recheck its neighbors as well:



Arc Consistency Algorithm

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*

> If X loses a value, neigbors of X need to be rechecked.

---

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*

- Run-time: $O(n^2d^3)$ (can be reduced to $O(n^2d^2)$)
  more efficient than forward checking

Local Search for CSP

- Modifications for CSPs:
  - work with complete states
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Min-conflicts Heuristic:
  - randomly select a conflicted variable
  - choose the value that violates the fewest constraints
  - hill-climbing with $h(n) = \#$ of violated constraints
- Performance:
  - can solve randomly generated CSPs with a high probability
  - except in a narrow range of

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