Lecture 16

CS 6380 Artificial Intelligence

Solving CSPs

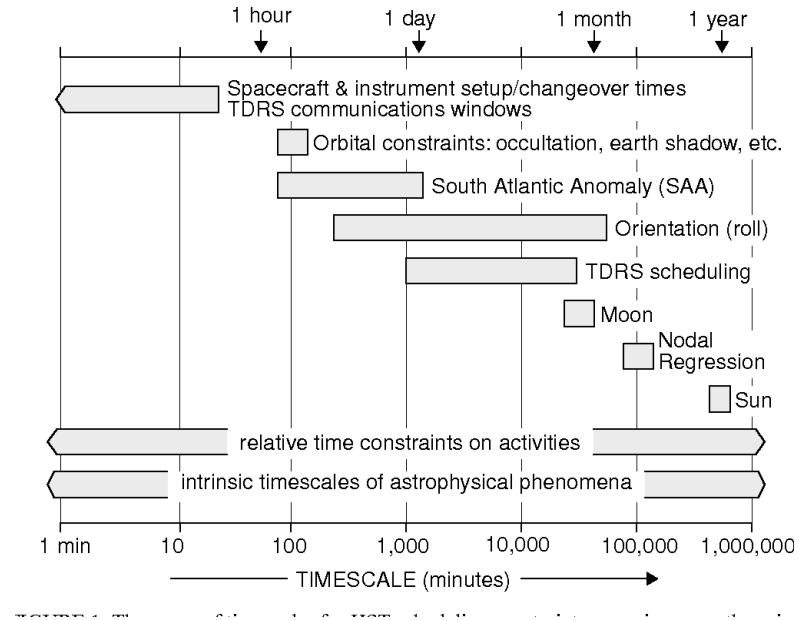
Department of Computer Science and Engineering

Indian Institute of Technology Madras (IITM), India

Material based on lecture slides for CSE 571: Artificial Intelligence by Siddharth Srivastava, ASU.

Recap

https://www.esa.int/Science_Exploration/Space_Science/Hubble_overview

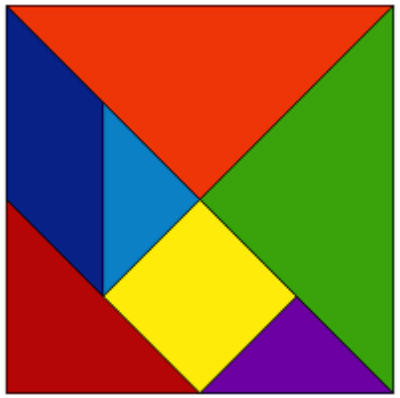


Johnston, M., & Miller, G. (1994). S PIKE : Intelligent Scheduling of Hubble Space Telescope Observations.

https://images.app.goo.gl/gESxMCeqNh8XtNPFA

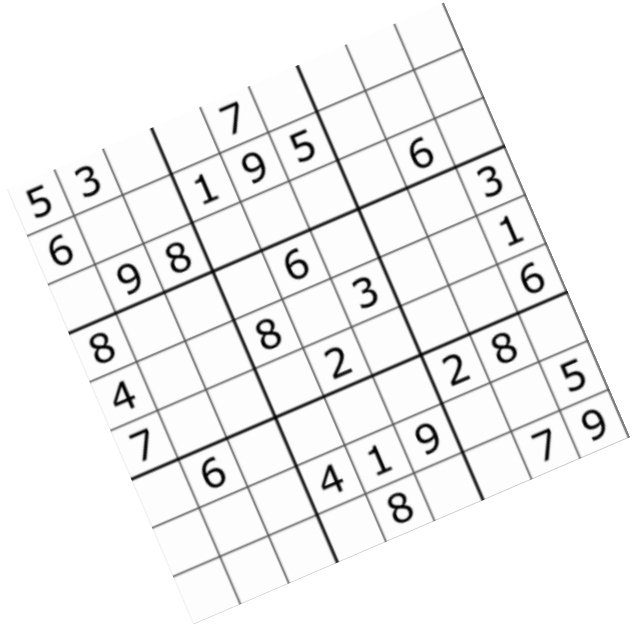
AI task:

input: puzzle; output: solution

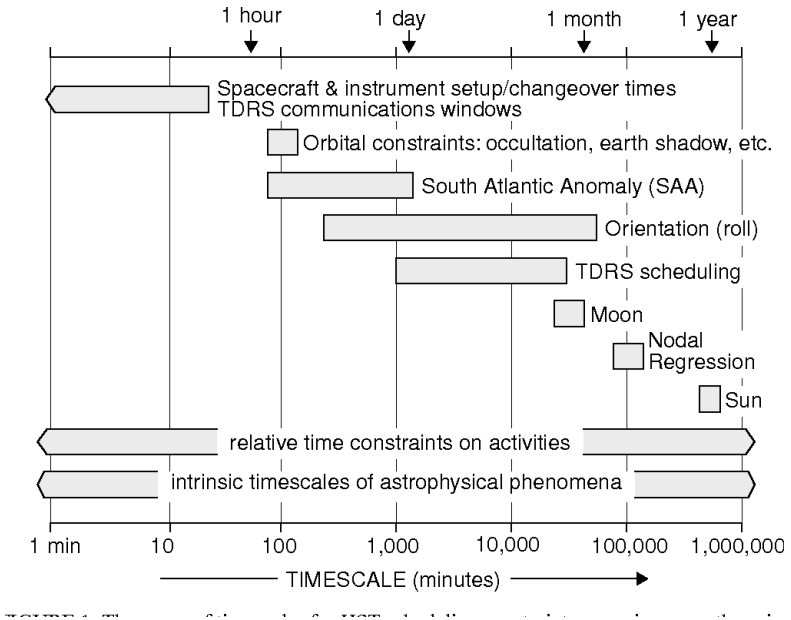


freepatternsarea.com

Which driver picks up which passenger

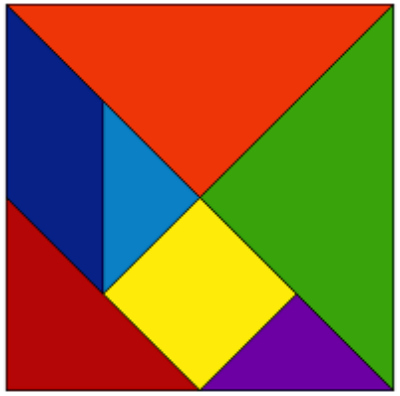


https://www.esa.int/Science_Exploration/Space_Science/Hubble_overview

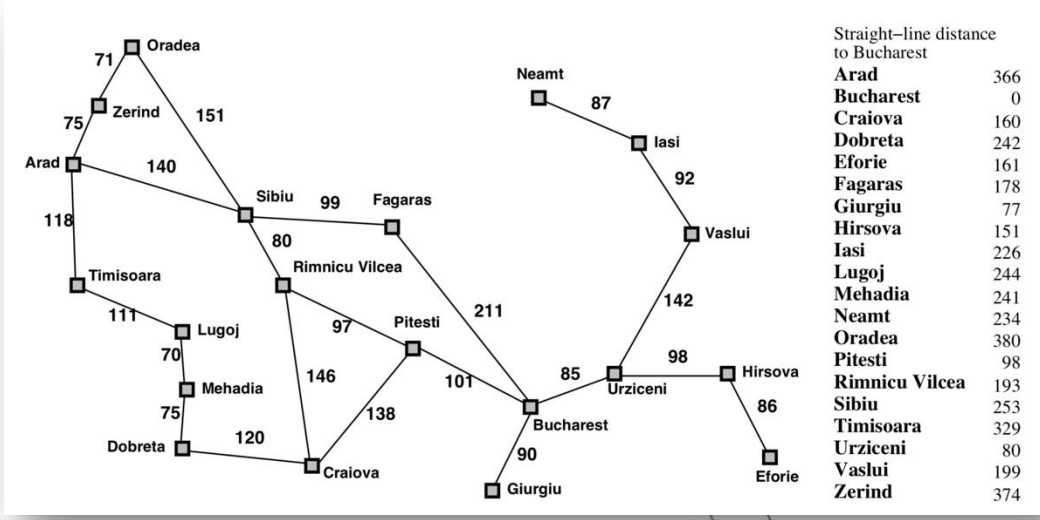


Johnston, M., & Miller, G. (1994). S PIKE : Intelligent Scheduling of Hubble Space Telescope Observations.

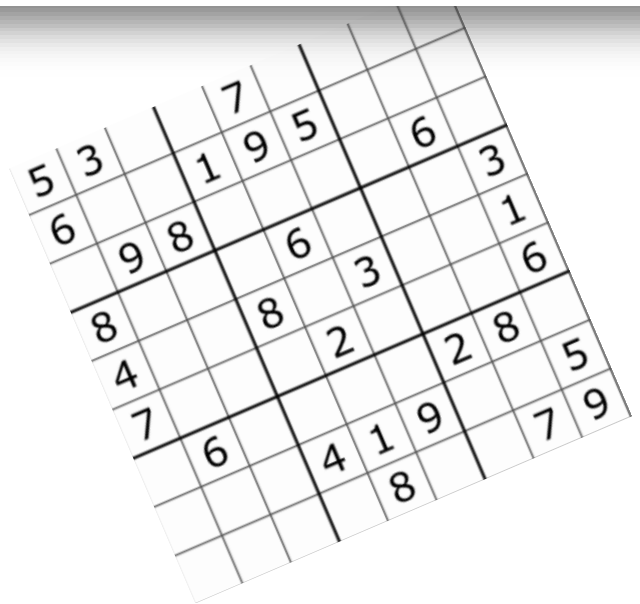
https://images.app.goo.gl/gESxMCeqNh8XtNPFA



freepatternsarea.com



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



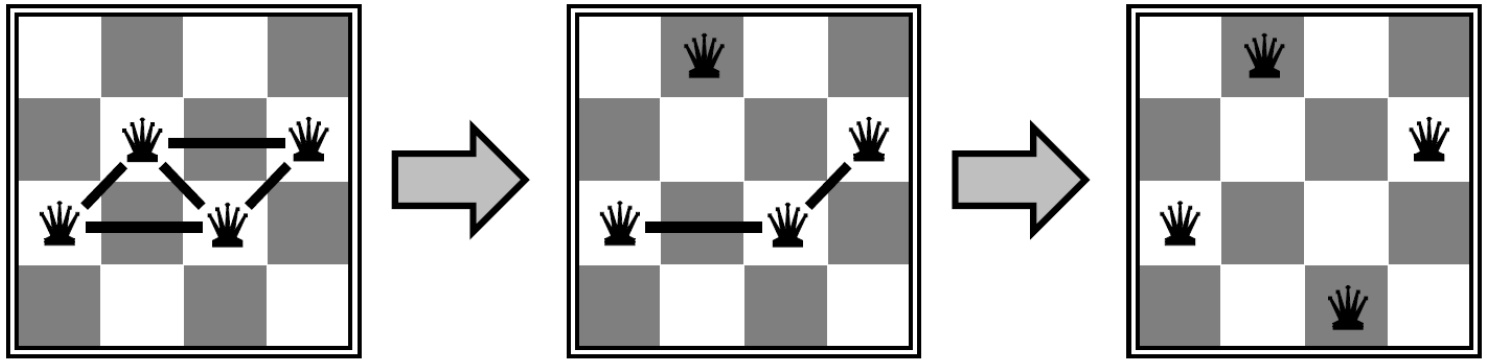
Unlike route planning, *describing a complete goal state* is the problem!

Sudoku

- Problem: fill in the blank squares such that
  - the cells in each row and column have distinct digits $\{1, 2, \ldots, 9\}$
- Given
  - A variable s_i for each open square
- Compute a digit assignment σ such that
  - All assignments in each row are different
  - All assignments in each column are different
  - All assignments in each region are different

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

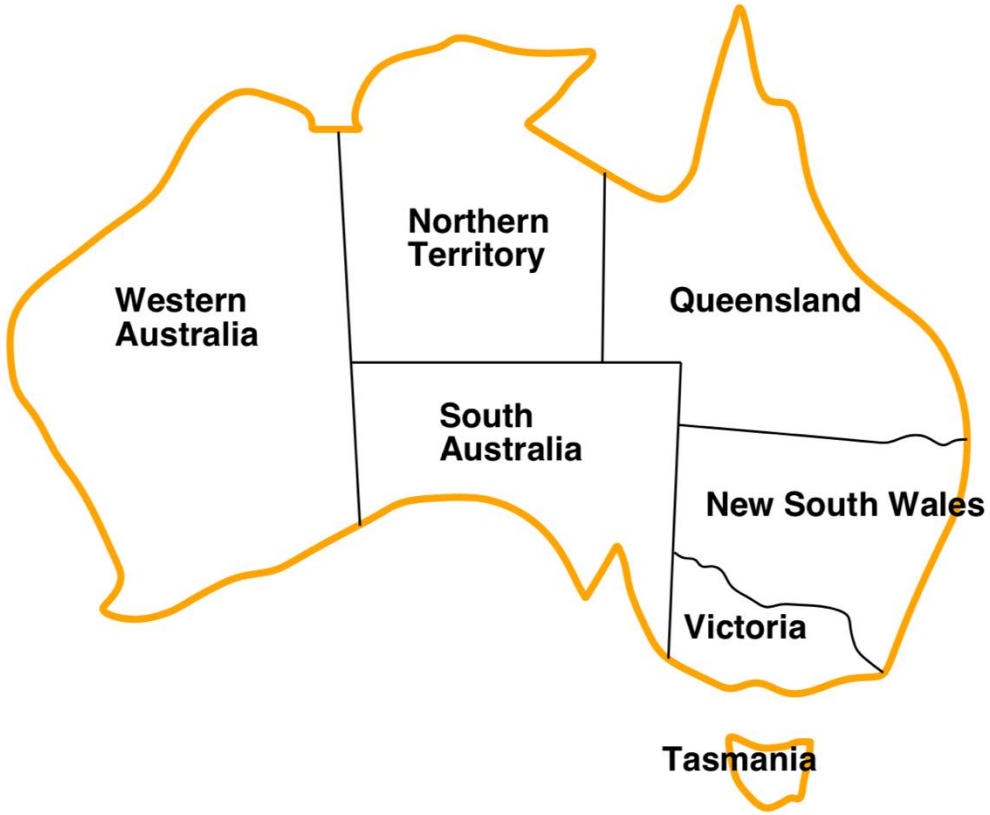
N-Queens



Map Coloring

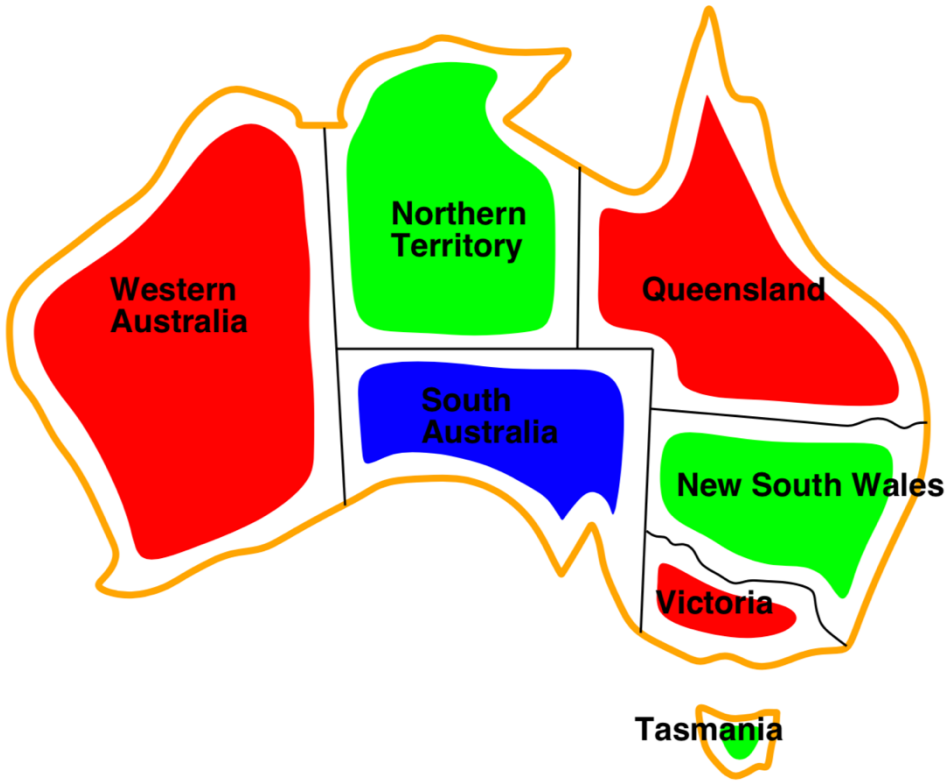
- Problem: color this map such that adjacent regions have different colors

- Colors: red, green, blue



Map Coloring

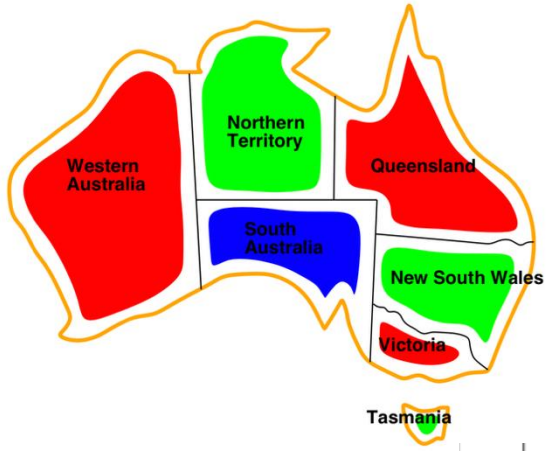
- Problem: color this map such that
  - adjacent regions have different colors
- Colors: red, green, blue
- Can formulate as:
  - Regions WA, NT, Q, NSW, SA, T
  - Possible colors: $red, green, blue$
- Compute:
  - A color mapping $\sigma: Region \rightarrow Color$ such that
  - For all adjacent r_1, r_2, $\sigma(r_1) \neq \sigma(r_2)$



Can such problems be solved automatically?

What's Common?

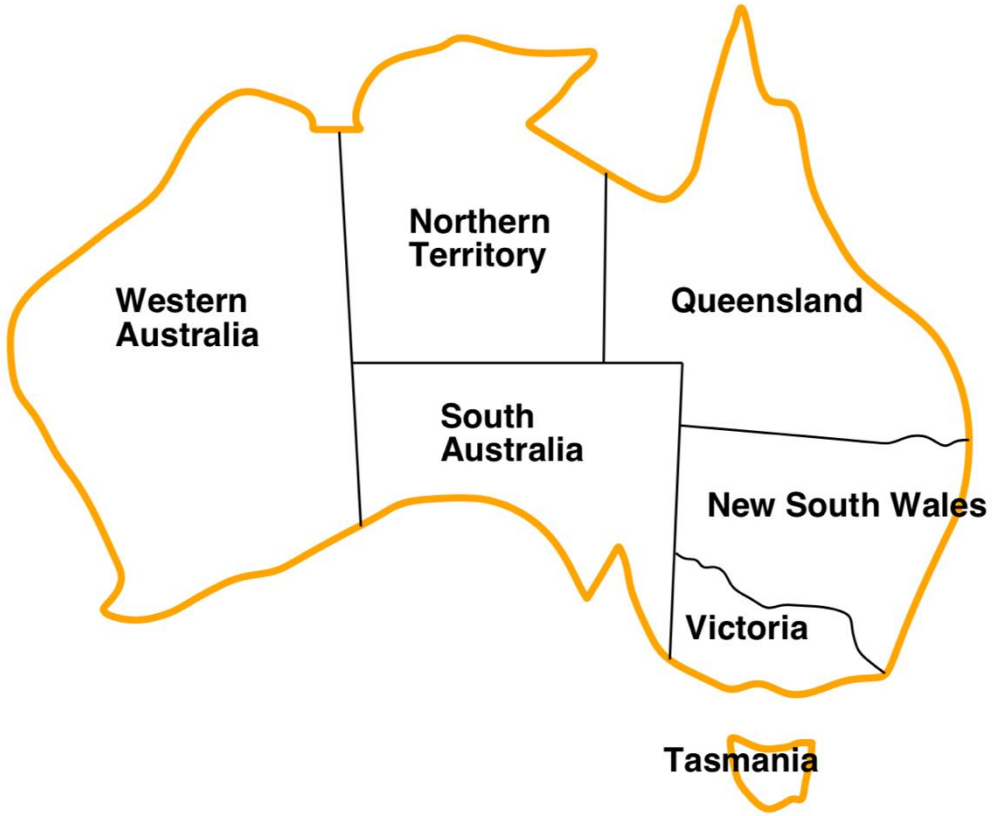
- Problem states are defined using
  - variables X_i with values from domains D_i
- Goal test is a set of constraints on variable values
- Constraint satisfaction problem:
  - Find an assignment to variable satisfying the constraints
- This captures a range of applications, including
  - Task allocation to heterogeneous robots,
  - Taxi services
  - Hubble telescope scheduling, Time tables…
- It seems to be a special case of a search problem!

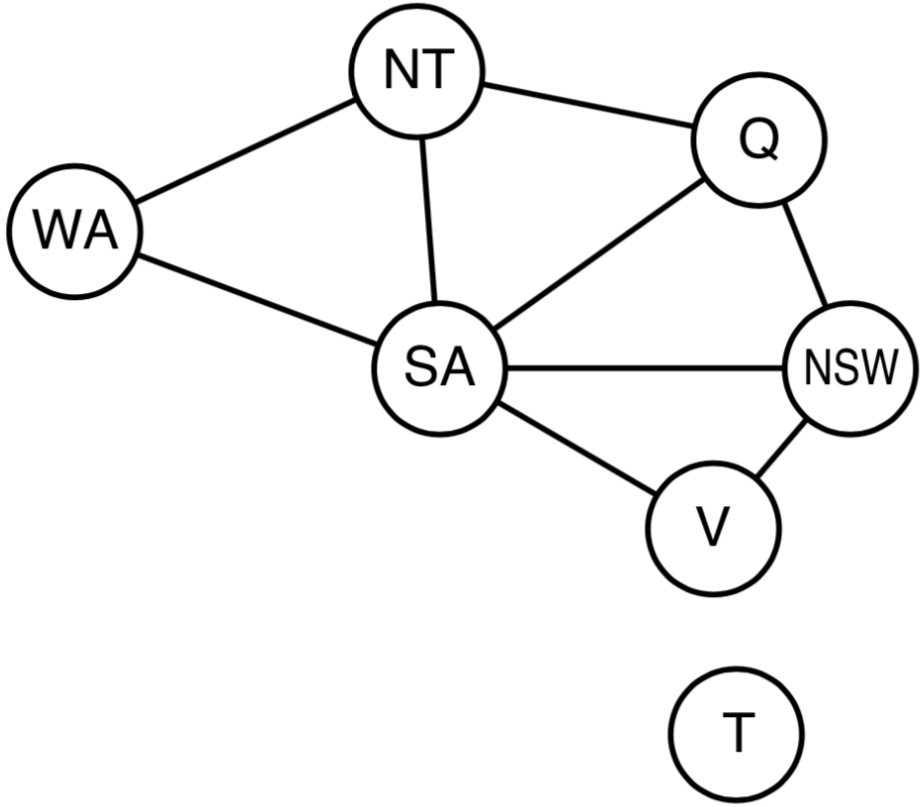


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Constraint Graph

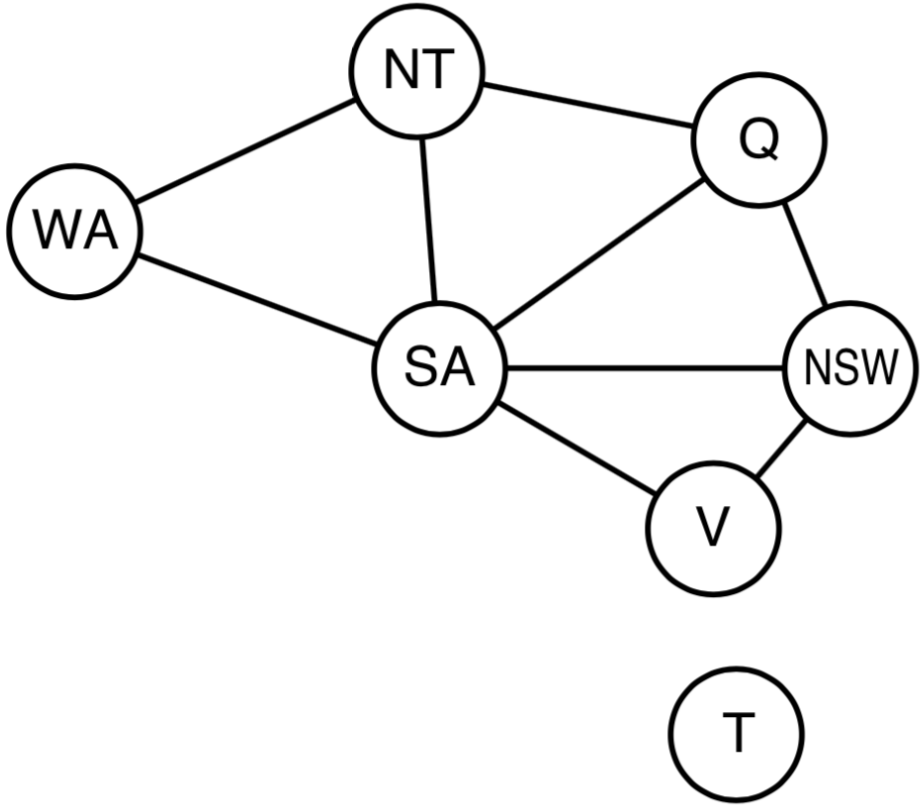
- Vertices: variables
- Edges between any pair of variables that are involved in a constraint





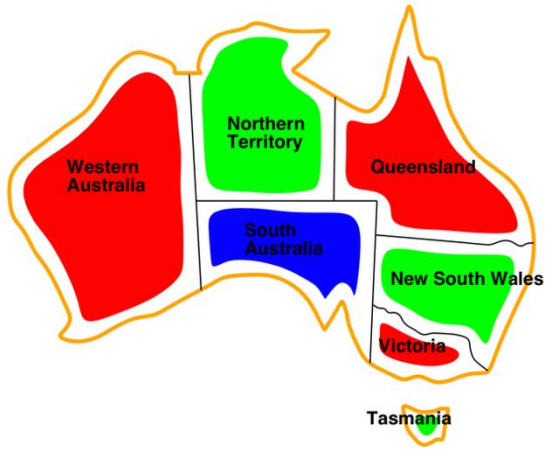
Varieties of CSPs

- Variables can be
  - Discrete
    - Finite domains (Boolean CSP: NP-complete)
      - d^n complete assignments
    - Infinite domains (integers, strings)
      - Project/Mission scheduling (start/end days)
  - Continuous
    - Start/end times for Hubble telescope
- Constraints
  - Linear, discrete variables: solvable
  - Nonlinear, discrete, infinite domains: undecidable



Types of Constraints

- Binary (recall)
- Unary constraints: involve a single variable
  - $SA \neq green$
- Higher-order constraints:
  - Involve 3 or more variables
  - E.g., Sudoku
- Preferences: soft constraints
  - E.g., Avoid making SA green
  - Can be associated with costs per variable assignment
- CSPs that include soft constraints are called constraint optimization problems



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

A More Complex Example: Cryptarithmetic

Assign unique values to letters to make this true:

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

A More Complex Example: Cryptarithmetic

Assign unique values to letters to make this true:

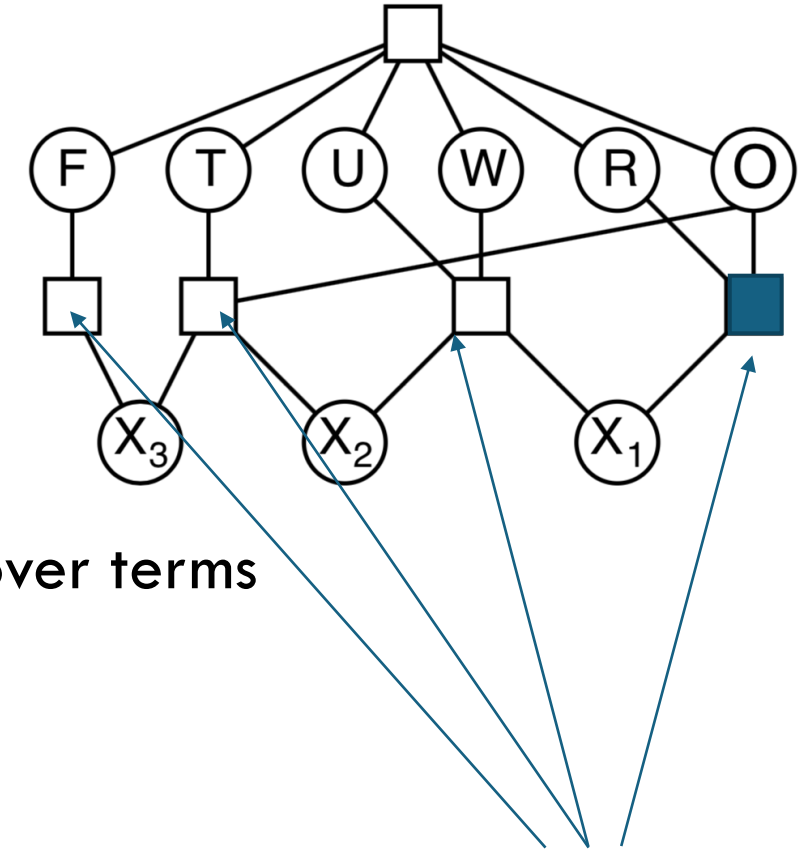
$$\begin{array}{r} \text{T W O} \\ +\ \text{T W O} \\ \hline \text{F O U R} \end{array}$$

- Need to add variables denoting carry-over terms
- Variables: $F, T, O, W, U, R, X_1, X_2, X_3$
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - alldiff(F, T, U, W, R, O)
  - $O + O = R + 10 \cdot X_1$
  - …

A More Complex Example: Cryptarithmetic

Assign unique values to letters to make this true:

$$\begin{array}{r} \text{T W O} \\ +\ \text{T W O} \\ \hline \text{F O U R} \end{array}$$



Higher order constraints

- Need to add variables denoting carry-over terms
- Variables: $F, T, O, W, U, R, X_1, X_2, X_3$
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - alldiff(F, T, U, W, R, O)
  - $O + O = R + 10 \cdot X_1$
  - …

Solving CSPs

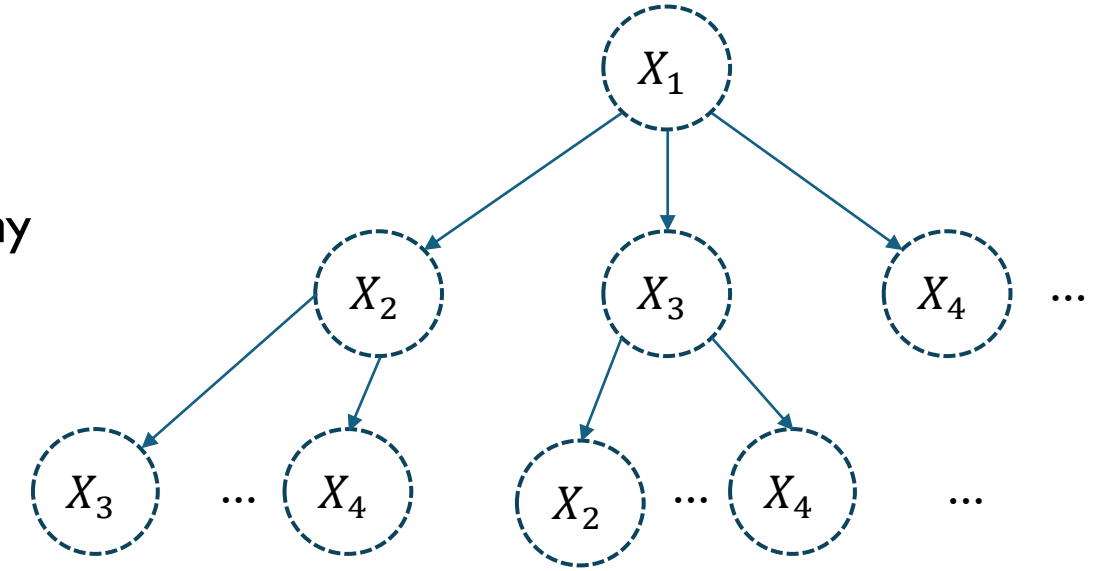
Standard Search Formulation

- Initial state: empty assignment $\sigma = \{\}$
- "Actions"/Successor function:
  - Select an unassigned variable
  - Assign a value from its domain that doesn't break any constraints
- Goal test: complete, constraint satisfying assignment

Standard Search Formulation

- Initial state: empty assignment $\sigma = \{\}$
- Actions/Successor function:
  - Select an unassigned variable
  - Assign a value from its domain that doesn't break any constraints
- Goal test: complete, constraint satisfying assignment
- Works for all CSPs!
  - All solutions appear at depth n for problems with n variables
  - Use DFS?
  - Number of leaves (nodes in the fringe):
    - At depth m, #branches per node $= b_m = (n - m)D$

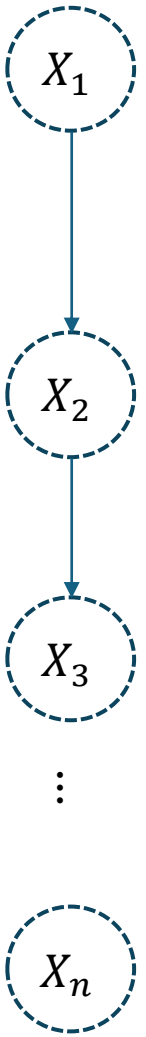
    $\rightarrow n!\, D^n$ leaves



Each dotted circle represents a set of choice
E.g., $X_1 = x_1, or\ X_1 = x_2, \ldots or\ X_1 = x_d$

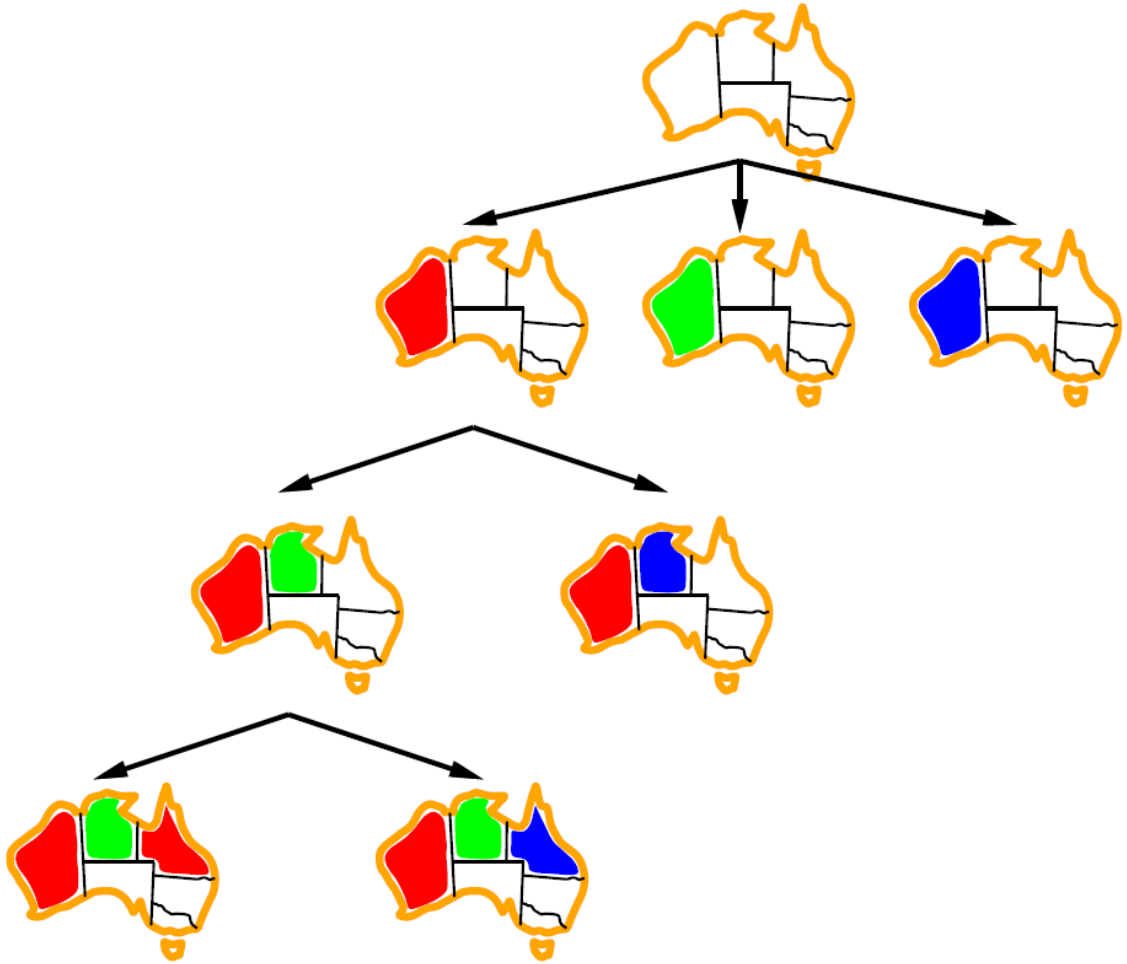
Backtracking Search

- Observation: variable assignments are commutative
  - Order of assignment doesn't matter
  - Can force each "level" to consider assignments to a single variable
  - Makes $b = D$; D^n leaves
- Backtracking search: DFS for CSPs, making a single variable assignment at each level
  - Baseline uninformed search algorithm for CSPs



Backtracking Search

- Observation: variable assignments are commutative
  - Order of assignment doesn't matter
  - Can force each "level" to consider assignments to a single variable
  - Makes $b = D$; D^n leaves
- Backtracking search: DFS for CSPs, making a single variable assignment at each level
  - Baseline uninformed search algorithm for CSPs



```
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
```

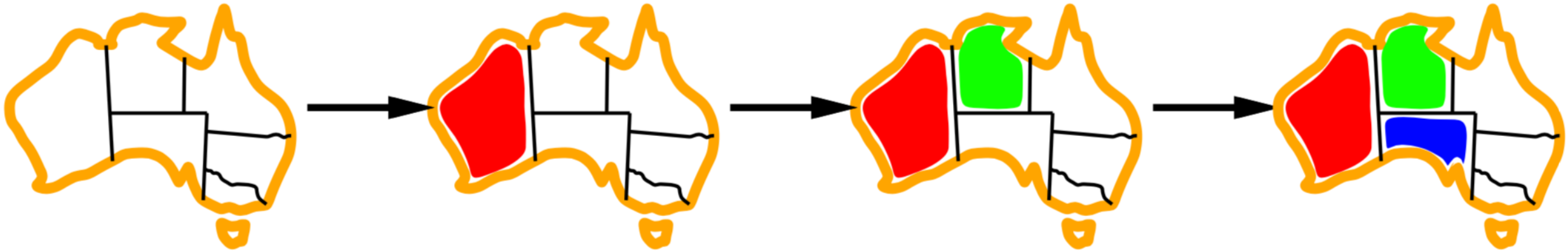
Optimizing Backtracking Search

- As we noted, there's a lot more "structure" here. Can we exploit it?

- Ideas:
  - Better selection of the next variable to consider
  - Better ordering of its values for assignment
  - It would be great if we could…
    - Detect failure early (before a complete assignment)
    - Exploit the structure of the constraint graph

Variable Selection: Minimum Remaining Values

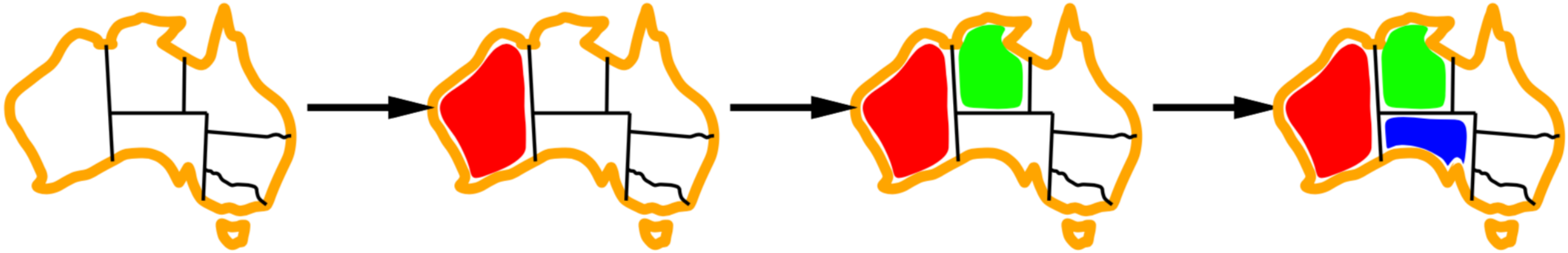
- MRV: choose the variable with the fewest legal values



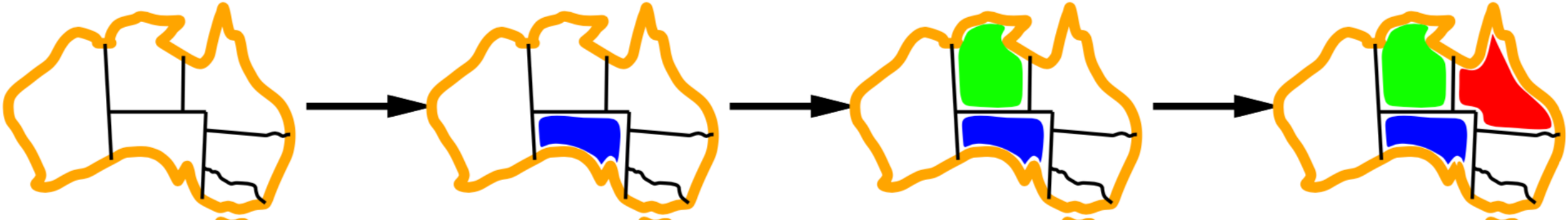
- Intuition: lower branching factor for those variables; can reveal infeasibility earlier

Variable Selection: Minimum Remaining Values

- MRV: choose the variable with the fewest legal values
- Tie-breaker: choose variable with the most constraints on the remaining variables (idea: settle the variable that affects the most variables)

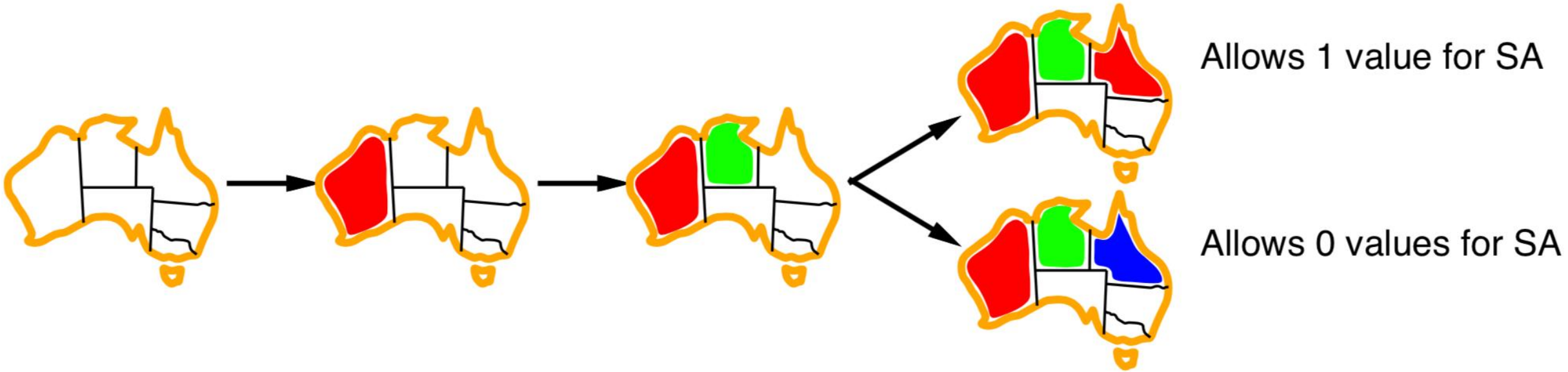


With tie-breaking:



Value Selection: Least Constraining Value

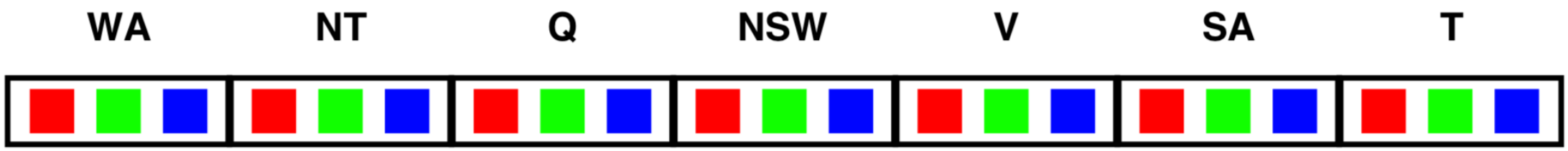
- After selecting a variable, choose the least constraining value
  - The one that leaves the most options available to other variables



- Combination of these heuristics: $40\times$ bump in size of solvable n-queens

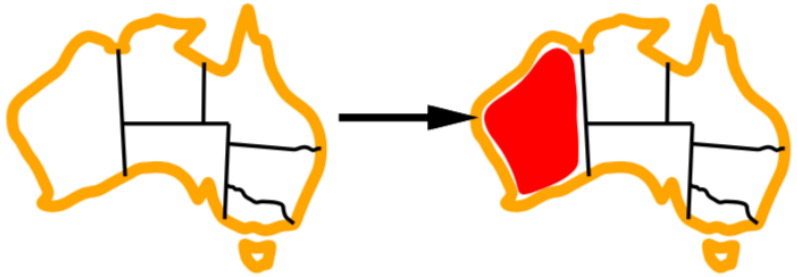
Early Failure Detection: Forward Checking

- Keep track of remaining legal values per unassigned variable
- Terminate search path when any variable has no legal values



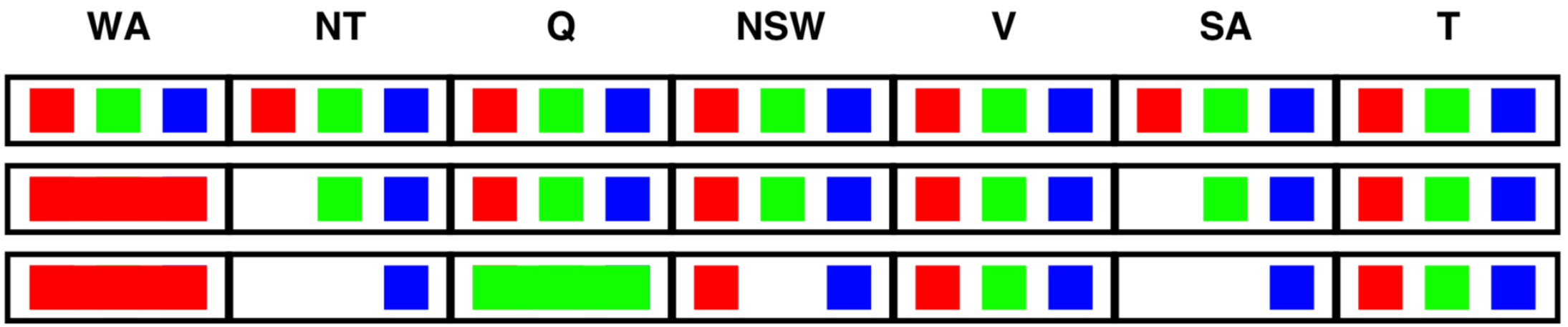
Early Failure Detection: Forward Checking

- Keep track of remaining legal values per unassigned variable
- Terminate search path when any variable has no legal values



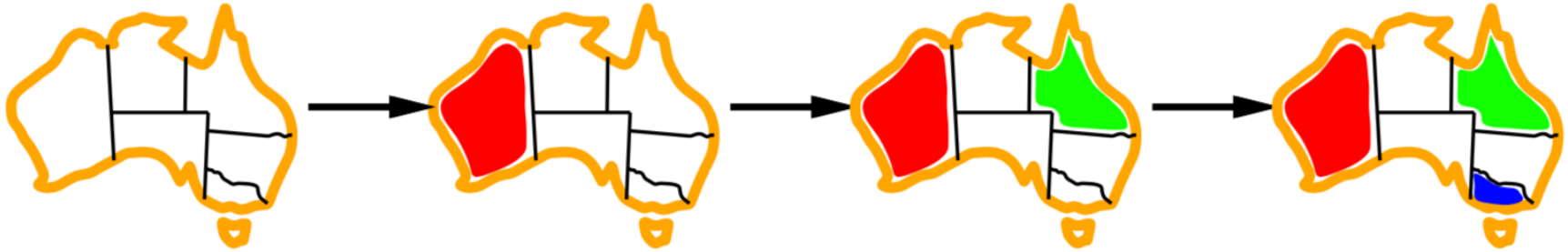
Early Failure Detection: Forward Checking

- Keep track of remaining legal values per unassigned variable
- Terminate search path when any variable has no legal values



Early Failure Detection: Forward Checking

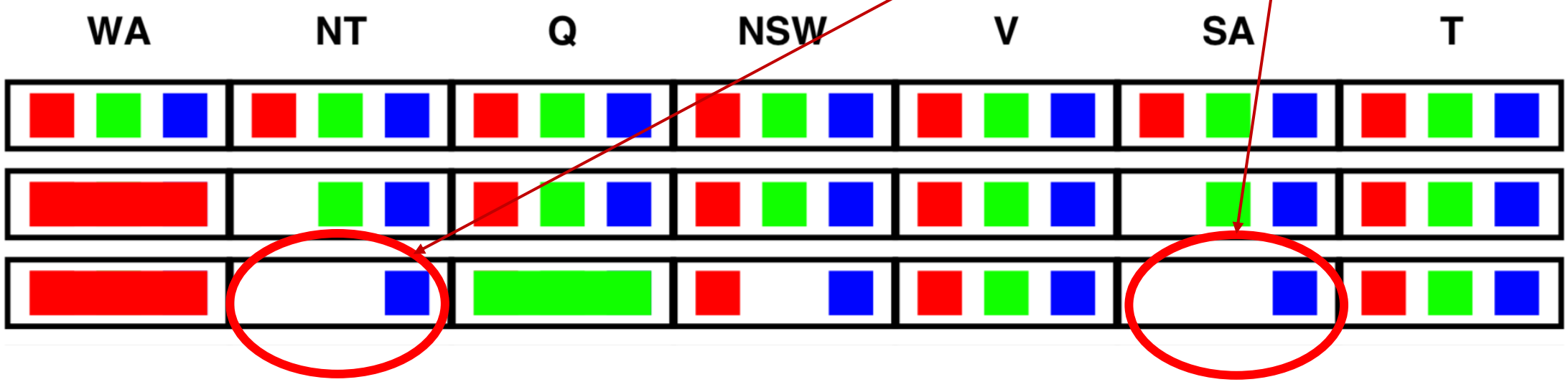
- Keep track of remaining legal values per unassigned variable
- Terminate search path when any variable has no legal values



Early Failure Detection: Constraint Propagation

- We could have detected failure even earlier:

What can we do to detect this early?



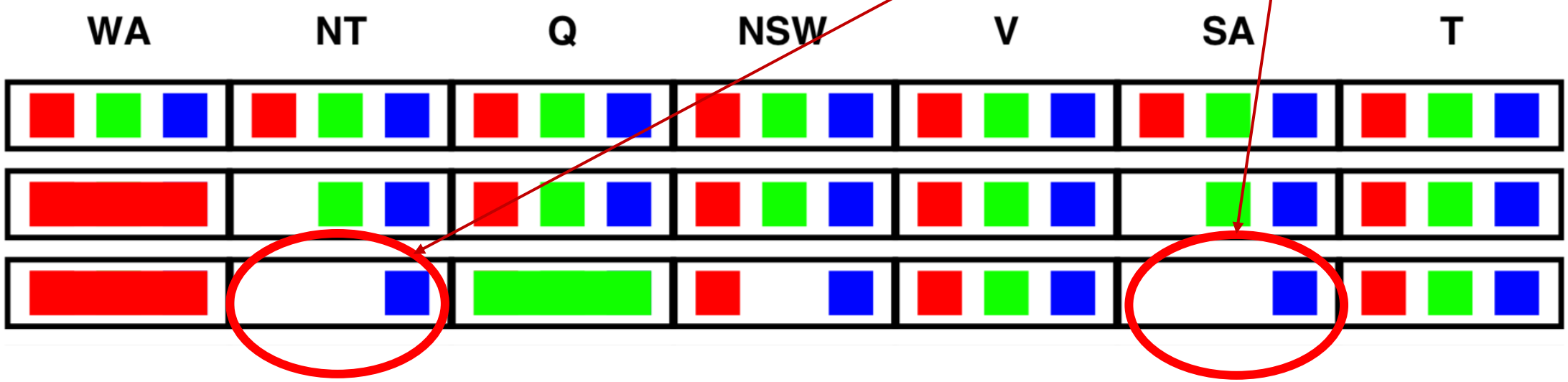
Node Consistency

- Enforcing local consistency in each part of the constraint graph eliminates inconsistent values throughout the graph
- The simplest form is node consistency, defined on a single variable
  - A variable is node consistent if all values in its domain satisfy its unary constraints
- Consider a variant of the Australia problem where South Australians dislike green
  - SA starts with domain {red, green, blue}
  - Eliminating green leaves SA with the reduced domain {red, blue}, making SA node consistent
- A network is node consistent if every variable in it is node consistent
- Running node consistency can always eliminate all unary constraints
  - All n-ary constraints can also be transformed into binary ones, so CSP solvers commonly work with only binary constraints

Early Failure Detection: Constraint Propagation

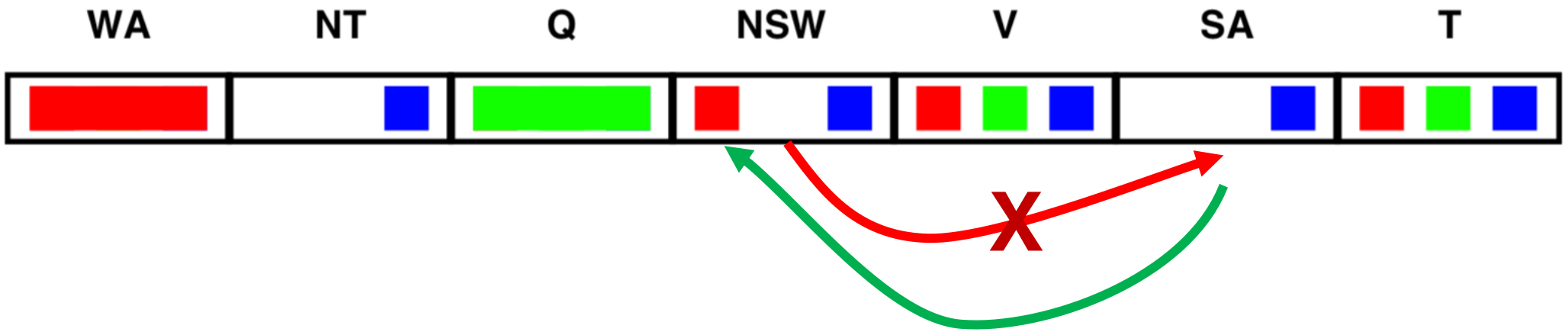
- We could have detected failure even earlier:

What can we do to detect this early?



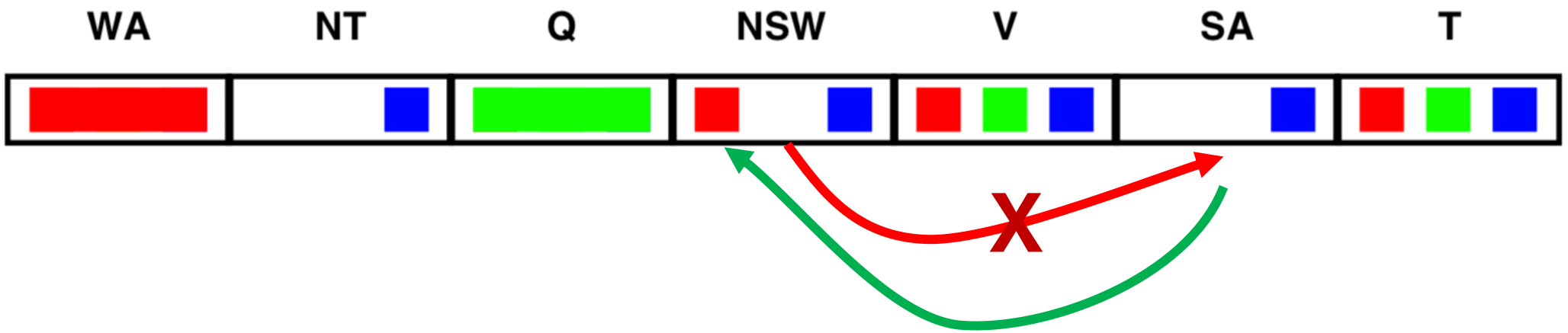
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
  - Note that this adds a direction to the dependency!
  - Two for each edge in the constraint graph



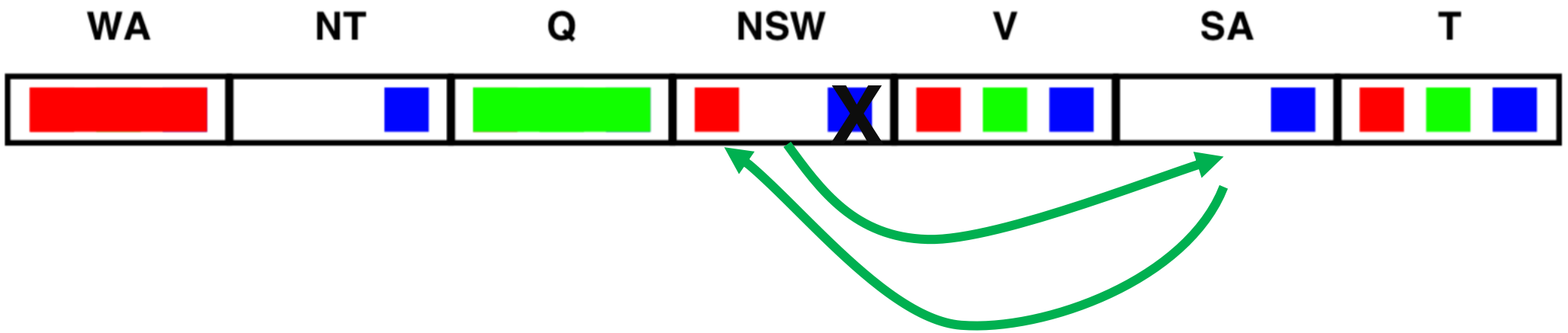
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X



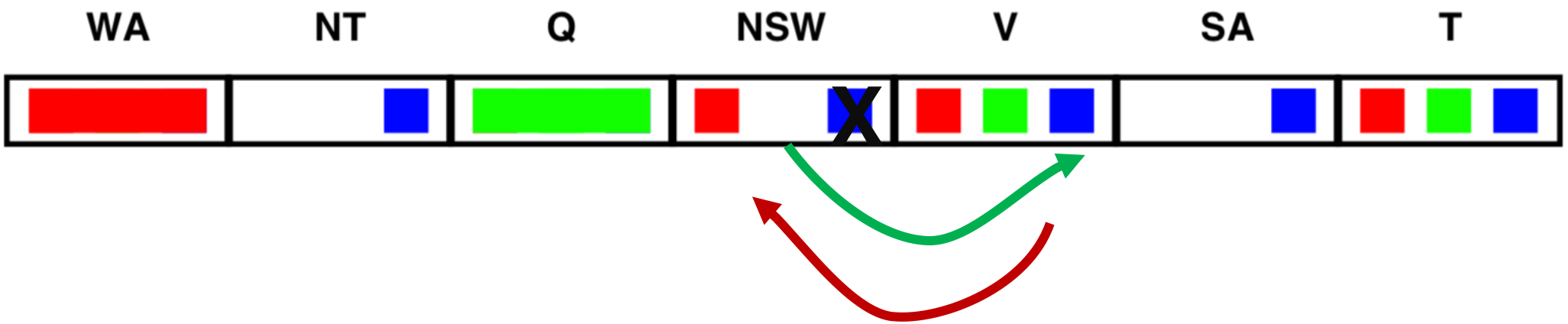
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X



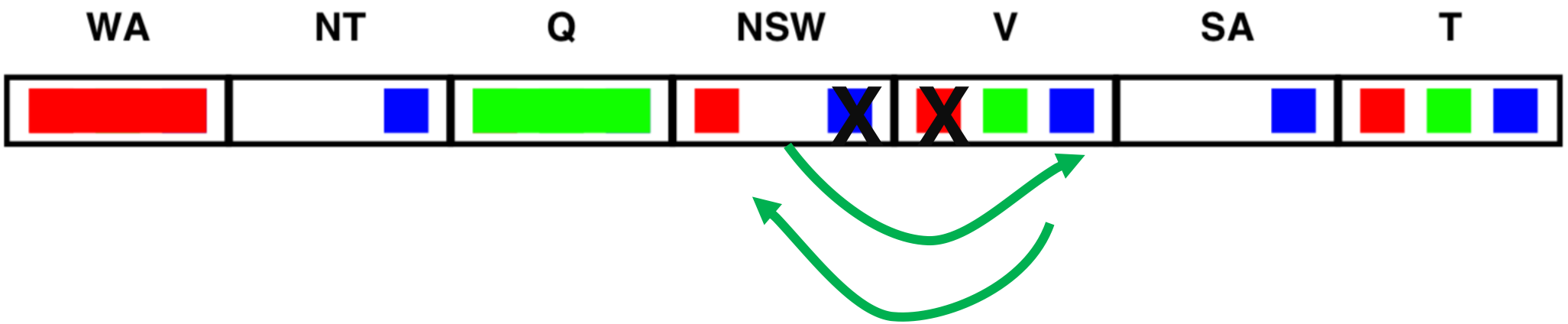
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X
- If X loses a value, its neighbors need to be rechecked



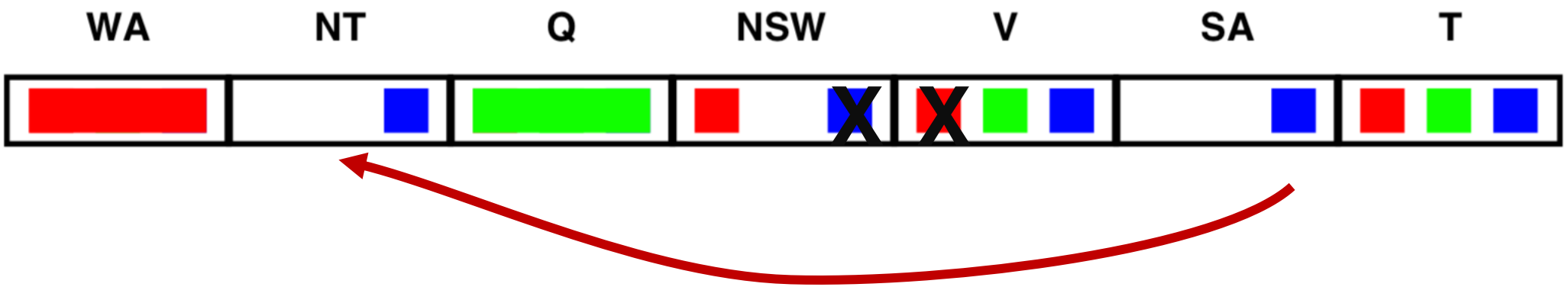
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X
- If X loses a value, its neighbors need to be rechecked



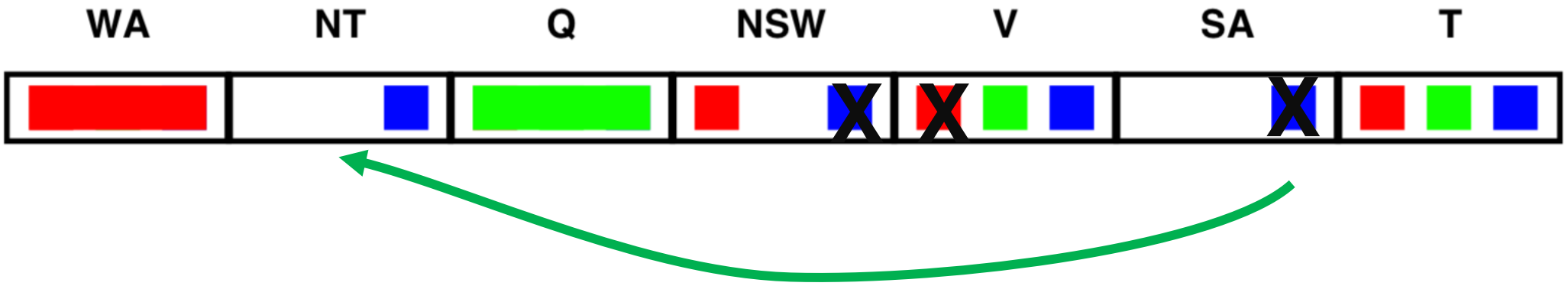
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X
- If X loses a value, its neighbors need to be rechecked



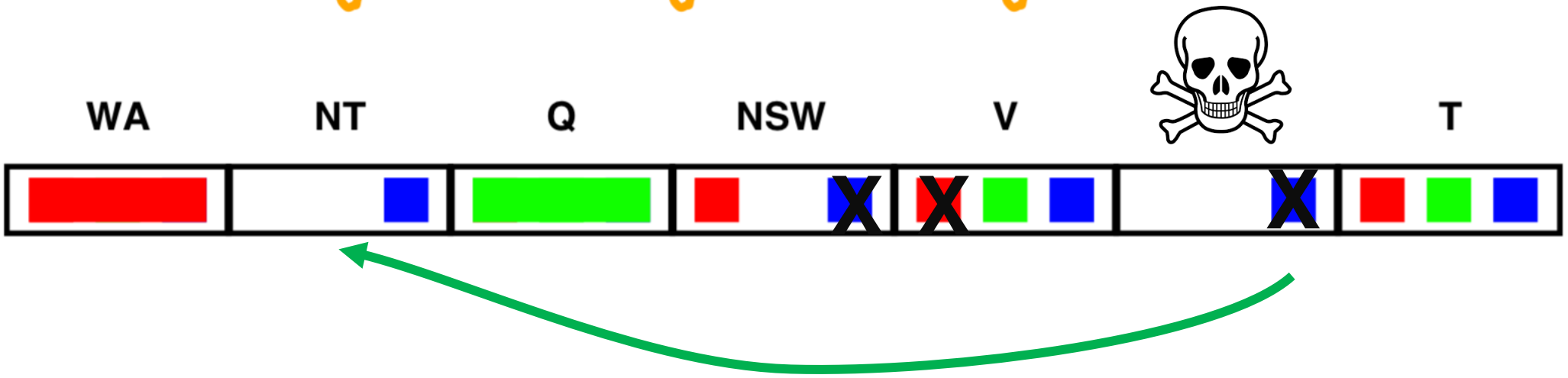
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X
- If X loses a value, its neighbors need to be rechecked



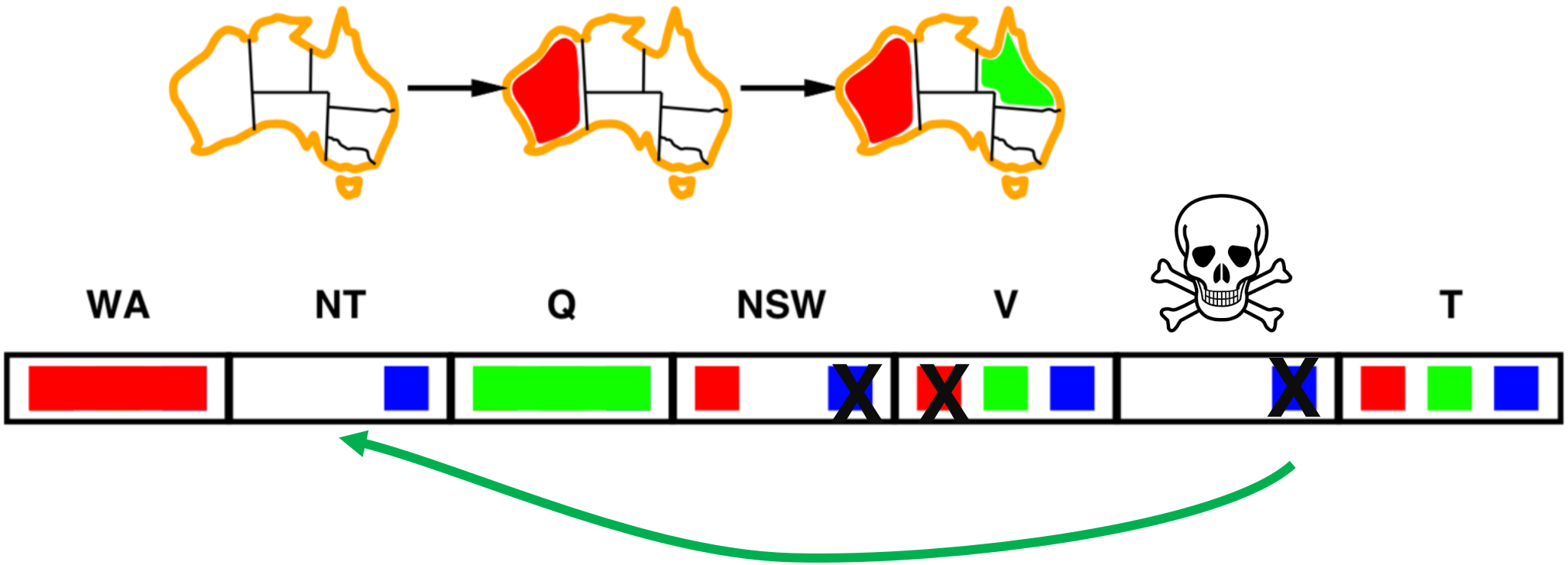
Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- To make $X \rightarrow Y$ consistent remove "bad" values from X
- If X loses a value, its neighbors need to be rechecked



Constraint Propagation: Arc Consistency

- Make each arc consistent
  - $X \rightarrow Y$ is consistent iff for all values of X, some value of Y is consistent
- Arc consistency reveals infeasibility earlier than forward checking (remaining values)
- Can be used as a postprocessor after each assignment + preprocessor before start
  - Carries out arc-consistency operations instead of making assignments, backtracking…



AC-3 Algorithm for Forcing Arc Consistency

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*

---

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds
  removed $\leftarrow$ *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* $\leftarrow$ *true*
  return *removed*

Path Consistency

- Arc consistency sometimes fails to make enough inferences
  - Try coloring the Australia map with only two colors, red and blue
  - Every variable is already arc consistent, since each can be red with blue at the other end of the arc, or vice versa
  - Yet there is no solution, because WA, NT, and SA all touch each other and need at least three colors between them
- Path consistency tightens the binary constraints using implicit constraints inferred by looking at triples of variables
  - A two variable set {Xi, Xj} is path consistent with respect to a third variable Xm if, for every assignment {Xi = a, Xj = b} consistent with the constraints on {Xi, Xj}, there is an assignment to Xm satisfying the constraints on {Xi, Xm} and {Xm, Xj}
  - The name comes from thinking of a path from Xi to Xj with Xm in the middle

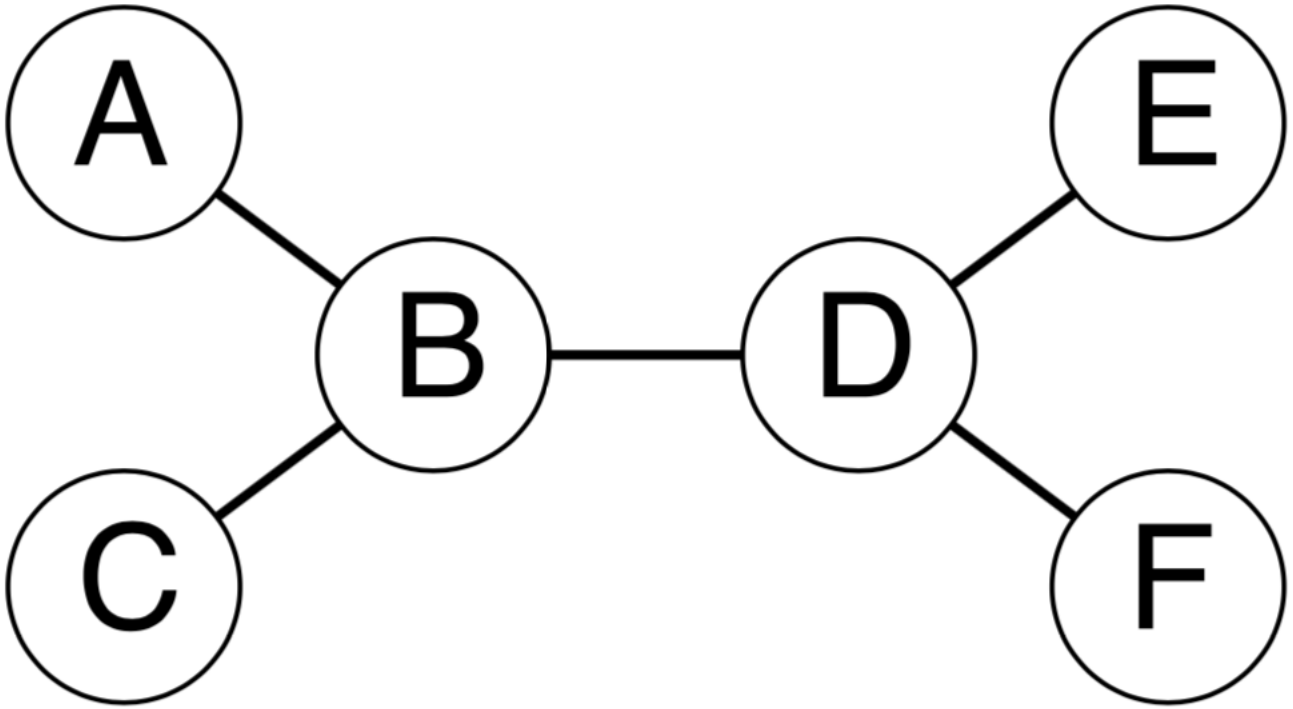
Path Consistency

- How does path consistency fare on the two color version of Australia?
- Make the set {WA, SA} path consistent with respect to NT
  - Enumerate the consistent assignments to the set, and there are only two, {WA = red, SA = blue} and {WA = blue, SA = red}
  - With either assignment NT can be neither red nor blue, since it would conflict with WA or with SA
  - There is no valid choice for NT, so both assignments are eliminated and no valid assignments remain for {WA, SA}
- Therefore the two color problem has no solution, detected without any search
- The PC-2 algorithm achieves path consistency in much the same way that AC-3 achieves arc consistency

Optimizing Backtracking Search

- Some considerations to improve speed:
  - Which variable should be assigned next? (MRV)
  - In what order should its values be tried? (LCV)
  - Can failure be detected early (before a complete assignment)? (FC, Arc Consistency)
  - Can we exploit the structure of the constraint graph?

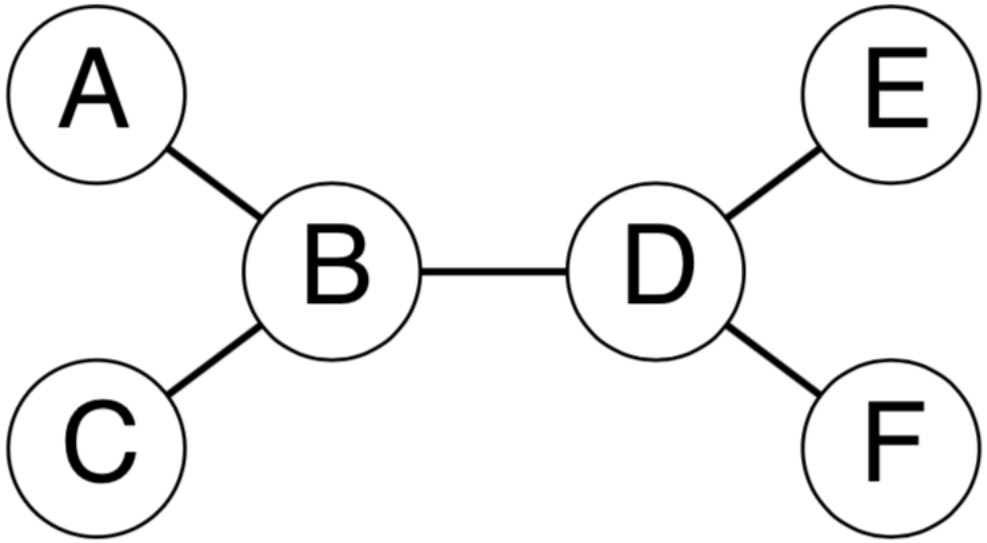
Tree-Structured CSPs

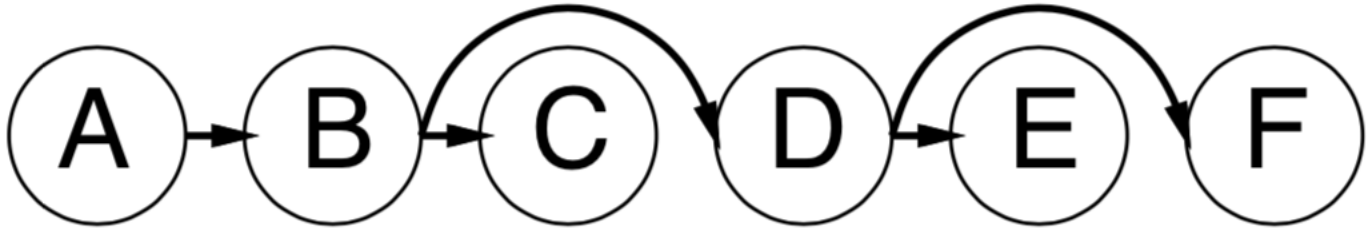


- If the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time
  - General CSPs have worst case time $O(d^n)$

Solving Tree-Structured CSPs

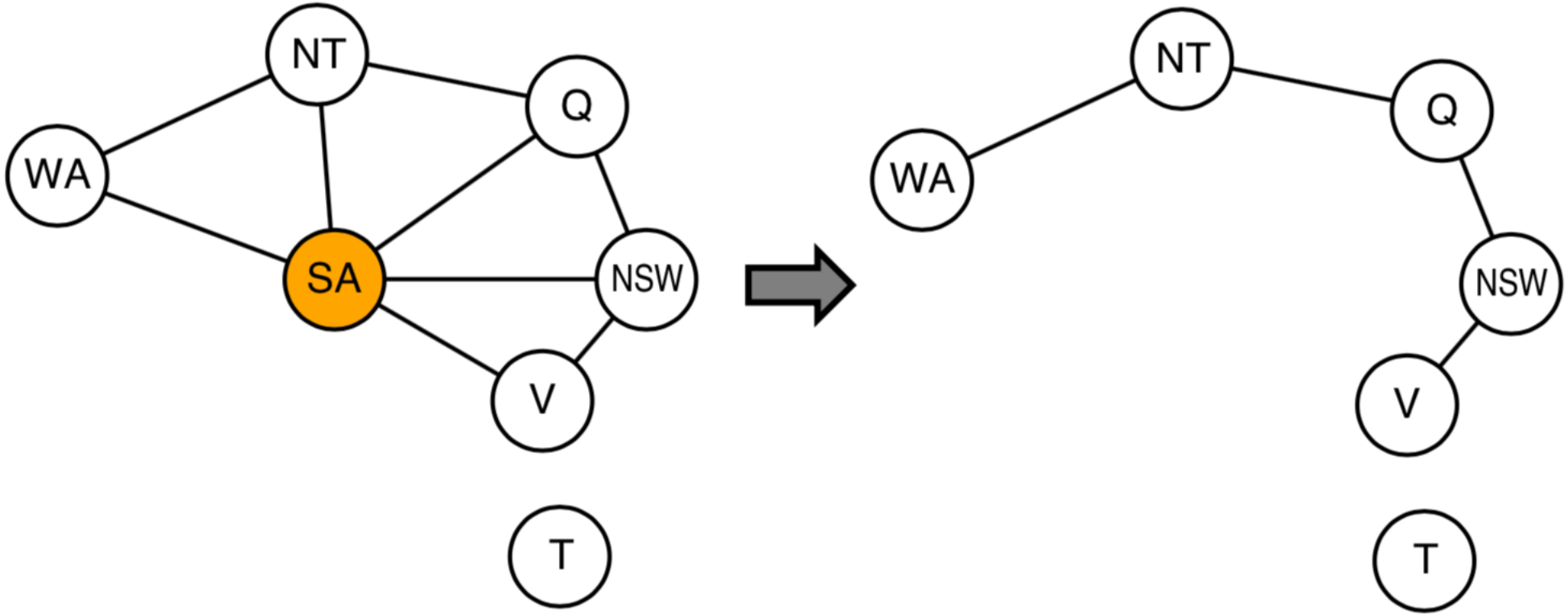
- Choose a root
- Order variables such that parent nodes precede children
- For j from n down to 2, apply RemoveInconsistent$(Parent(X_j), X_j)$
- For j from 1 to n assign X_j to make it consistent with $Parent(X_j)$





Almost Tree-Structured CSPs

- Conditioning: assign a variable, prune its neighbors' domains to generate a tree



- Cutset conditioning: select a set to remove. Condition on all possible assignments
  - Cutset size c $\rightarrow$ time complexity $O(d^c(n-c)d^2)$