Constraint Satisfaction Problems II

PROF LIM KWAN HUI

50.021 Artificial Intelligence

The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.

Recap: Constraint Satisfaction Problems Formulation

- Finite set of ***variables X*** $= \{X_1, X_2, \ldots, X_n\}$
- Non-empty ***domain D*** of k possible values for each variable D_i where $D_i = \{v_1, \ldots, v_k\}$
- Finite set of ***constraints C*** $= \{C_1, C_2, \ldots, C_m\}$
  - Each constraint C_i limits the values that variables can take, e.g., $V_1 \neq V_2$
- In relation to a search problem, we have:
  - **State** is defined by *variables* X_i that take on values from *domain* D_i
  - **Goal Test** is a set of *constraints* C_i specifying *allowable combinations of values* for subsets of variables

Recap: CSPs as Standard Search

- There are potentially $n!d^n$ leaves in the search tree
  - *How did we get this?*

- For a CSP with n variables with d domains, we have:
  - Depth 0: branching factor of nd
  - Depth 1: branching factor of $(n-1)d$
  - Depth 2: branching factor of $(n-2)d$
  - …
  - Depth n: branching factor of d

Recap: CSPs as Standard Search

- There are potentially $n!d^n$ leaves in the search tree
  - *How did we get this?*

- $n!d^n$ leaves is too many to search through, can we reduce it?
  - Now: Backtracking search with various enhancements

Commutativity

- In CSPs, variable assignments are *commutative*
  - Does not matter which order you assign the variables
  - E.g., [WA=red then NT=green] same as [NT=green then WA=red]

- Only need to consider assignments to a single variable at each level/depth
  - Branching factor is the domain size d
  - d^n leaves now, instead of $n!d^n$ leaves

Backtracking Search

- Backtracking Search is essentially like Depth-First Search for CSPs with single-variable assignments
  - The backtracking occurs when there are no legal values for a variable
  - Uses *commutativity* to reduces search from $n!d^n$ leaves to d^n leaves

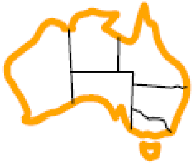
- Backtracking search is the basic uninformed algorithm for CSPs

Backtracking Search

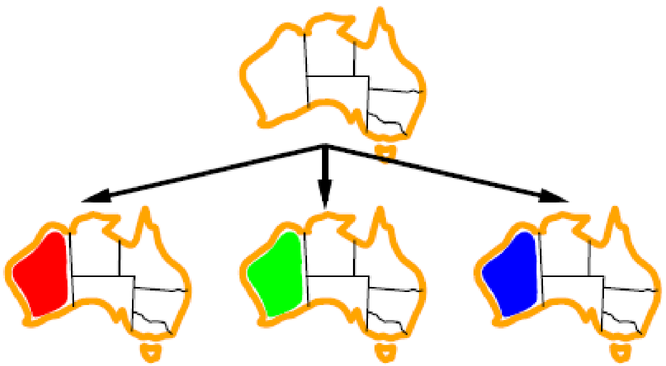
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

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
```

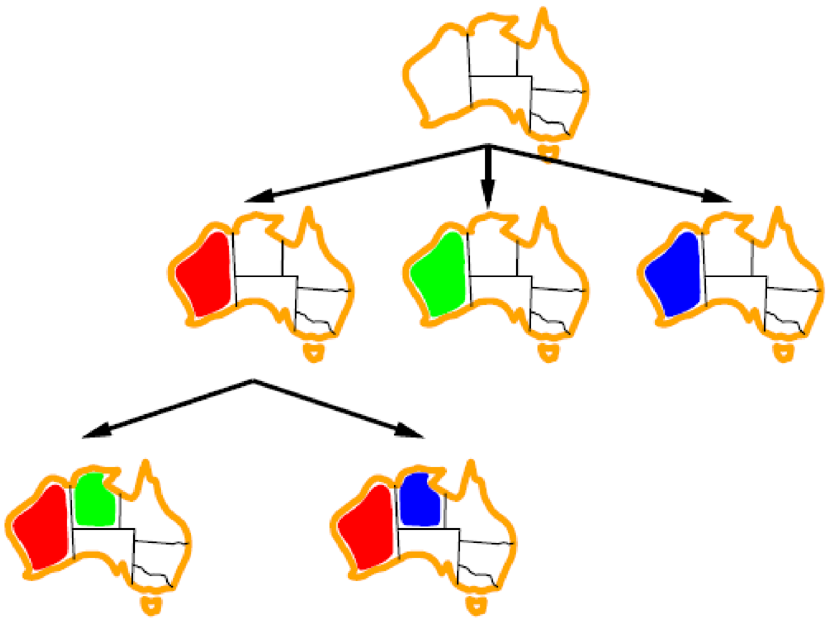
Backtracking Example



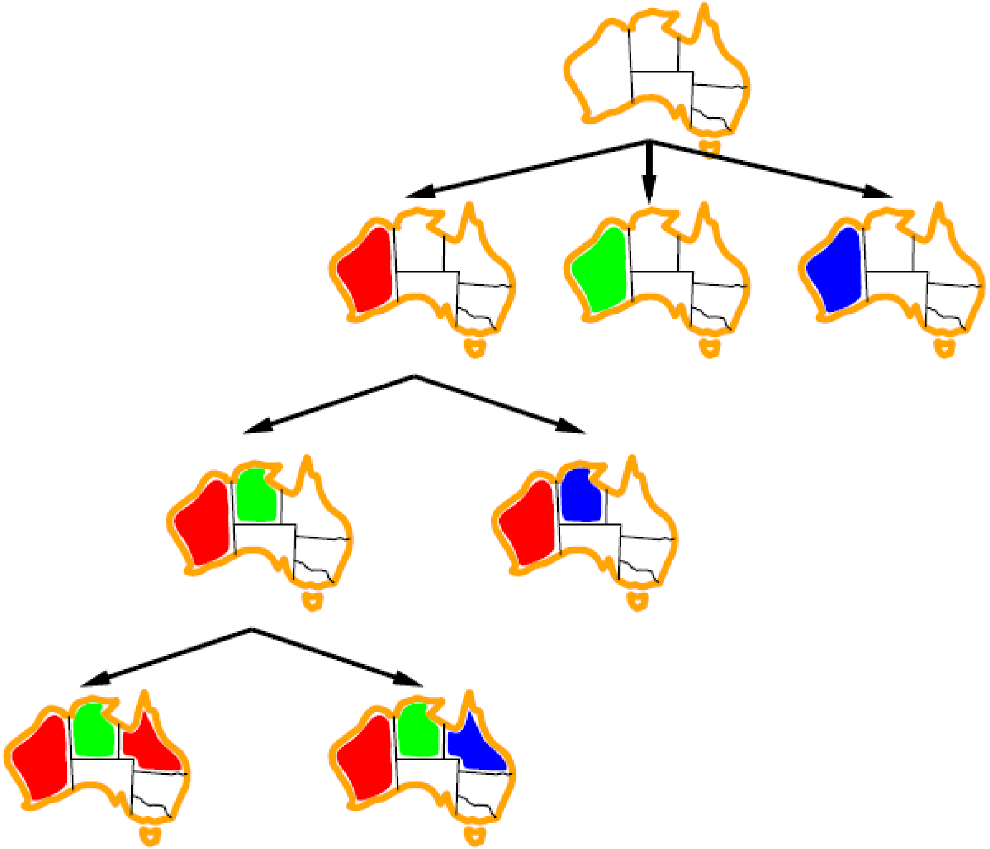
Backtracking Example



Backtracking Example



Backtracking Example

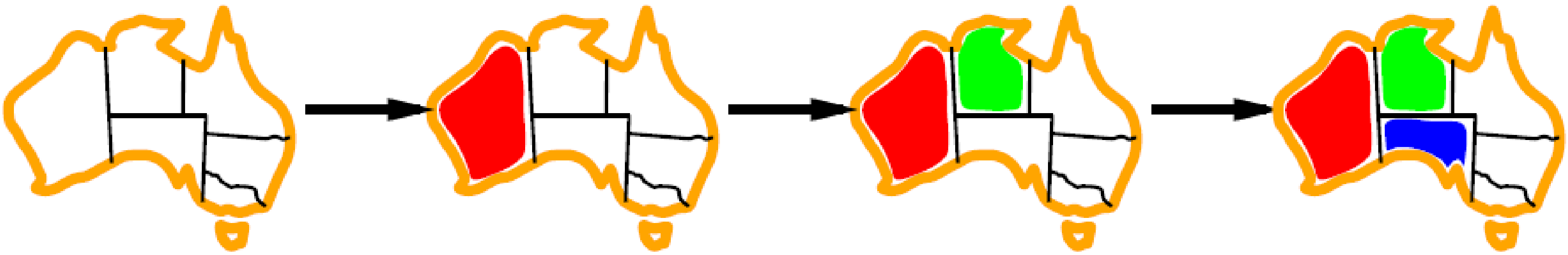


Improving Backtracking Search

- *General-purpose* methods can give *huge gains in speed*:
  - Which variable should be assigned next?
    - Idea 1: Minimum remaining values
    - Idea 2: Degree heuristic
  - In what order should its values be tried?
    - Idea 3: Least constraining value
  - Can we detect inevitable failure early?
    - Idea 4: Forward checking / AC-3 Algorithm
  - Can we take advantage of problem structure?
    - Idea 5: Tree-structured CSPs

Minimum remaining values

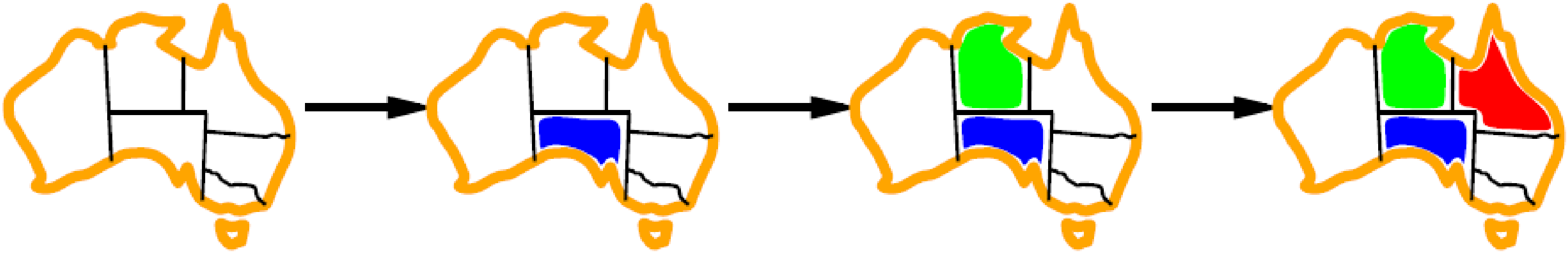
- Minimum remaining values (MRV)
  - Choose the variable with the *fewest legal values*, i.e., the most constrained variable



Degree heuristic

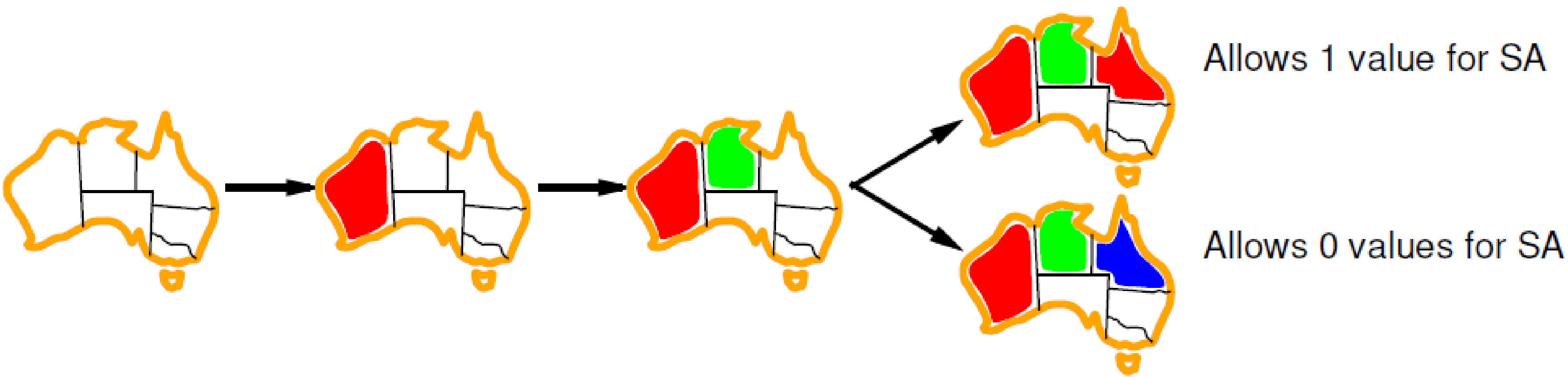
- What happens when multiple variables have the same MRV?

- Degree heuristic
  - Choose the variable with the *most constraints on remaining variables*



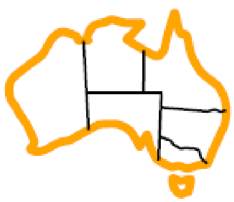
Least constraining value

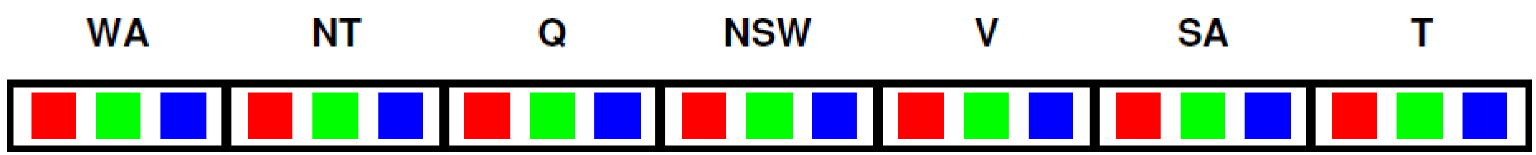
- Given a variable, choose the least constraining value
  - The one that *rules out the fewest values* in the remaining variables



Forward checking

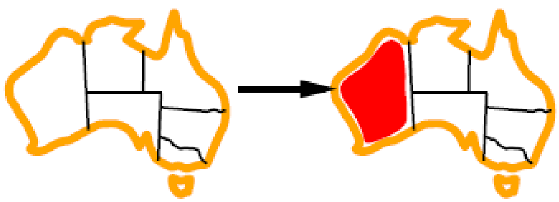
- Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

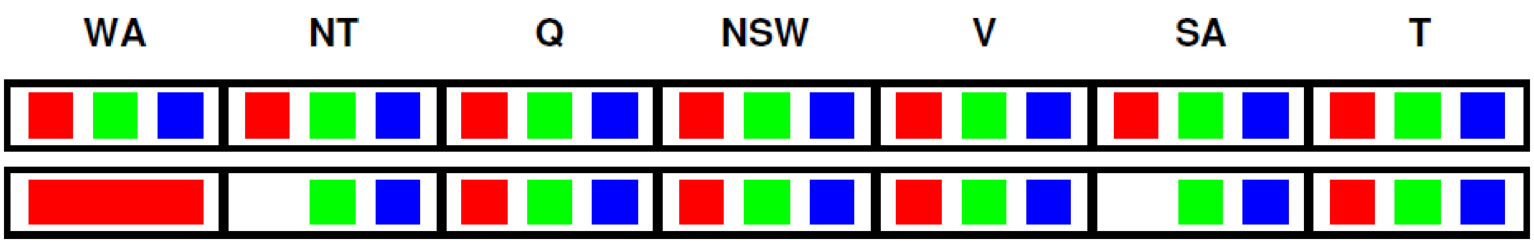




Forward checking

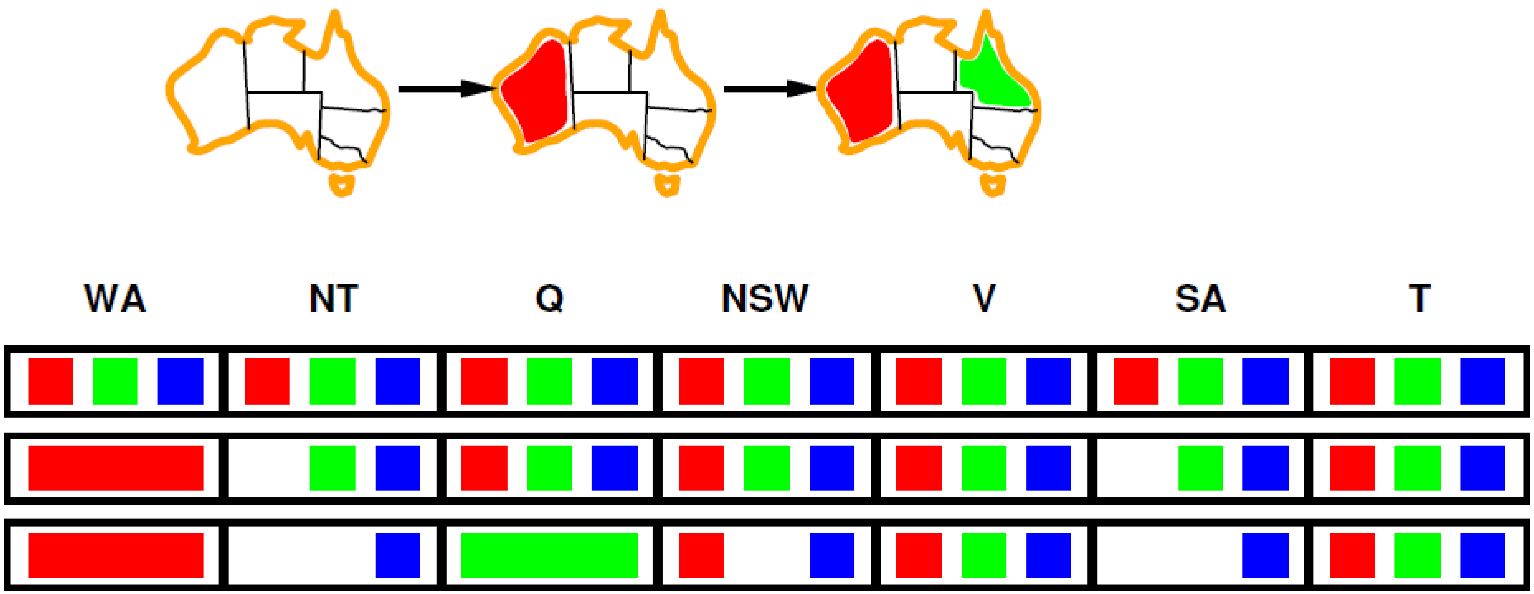
- Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values





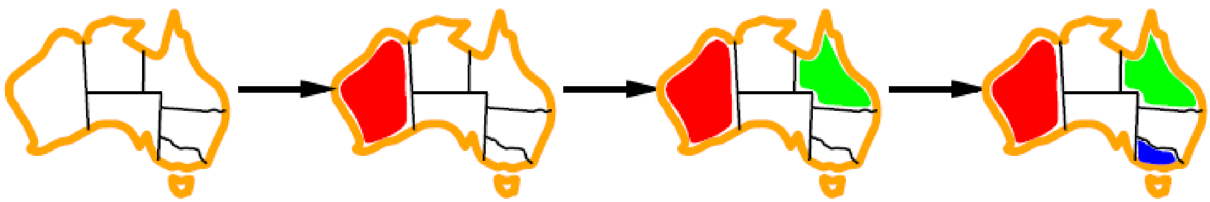
Forward checking

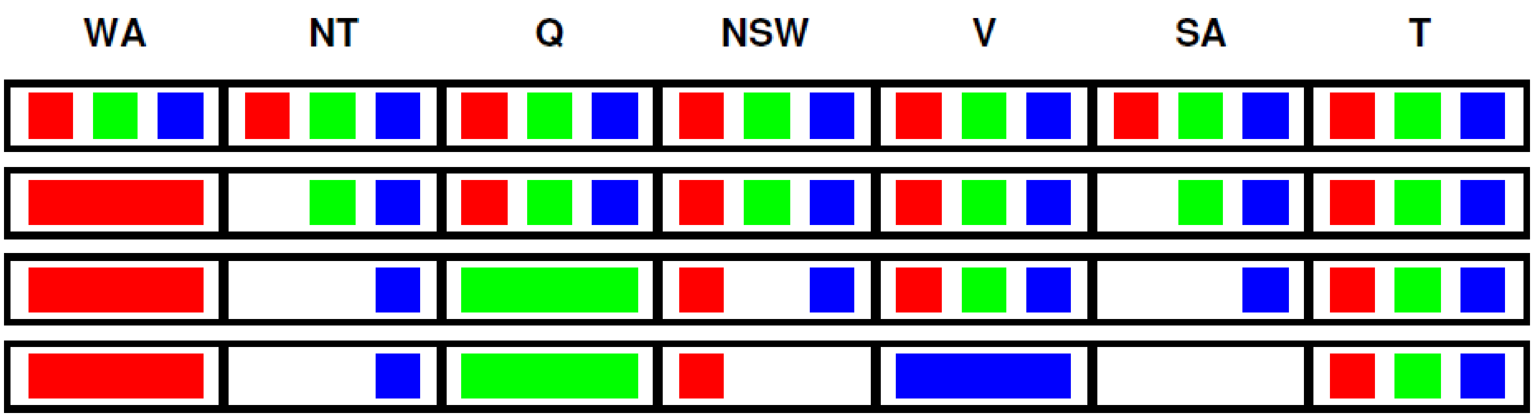
- Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



Forward checking

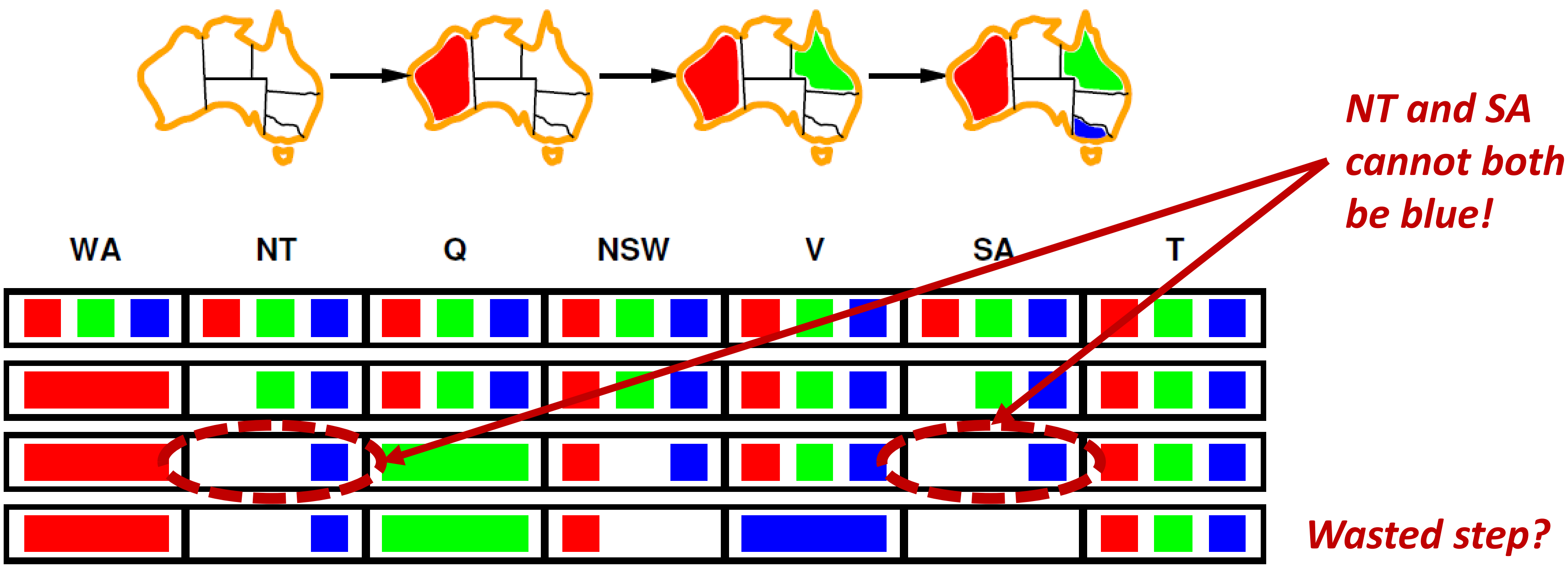
- Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values





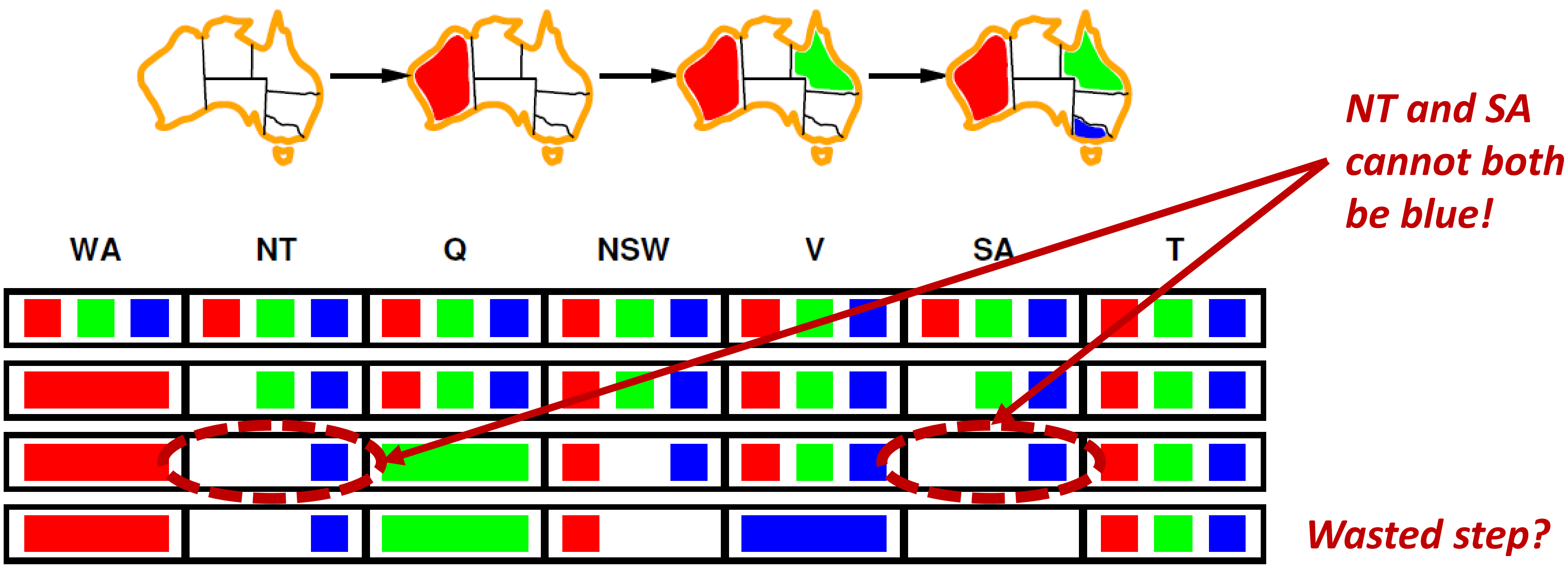
Limitation of forward checking

- Forward checking propagates information from assigned to unassigned variables
  - However, it does not provide early detection for all failures



Limitation of forward checking

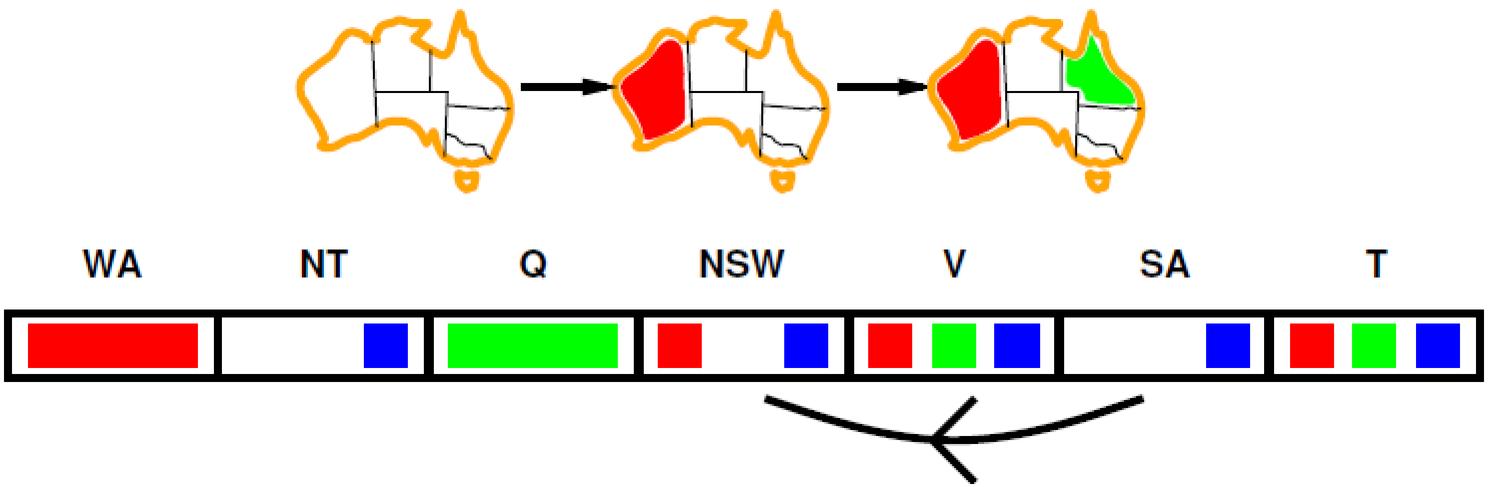
- Forward checking does not provide early detection for all failures
  - We need to repeatedly enforce constraints locally, i.e., constraint propagation



Arc consistency

- Simplest form of propagation make each arc *consistent*
- *X → Y* is consistent iff

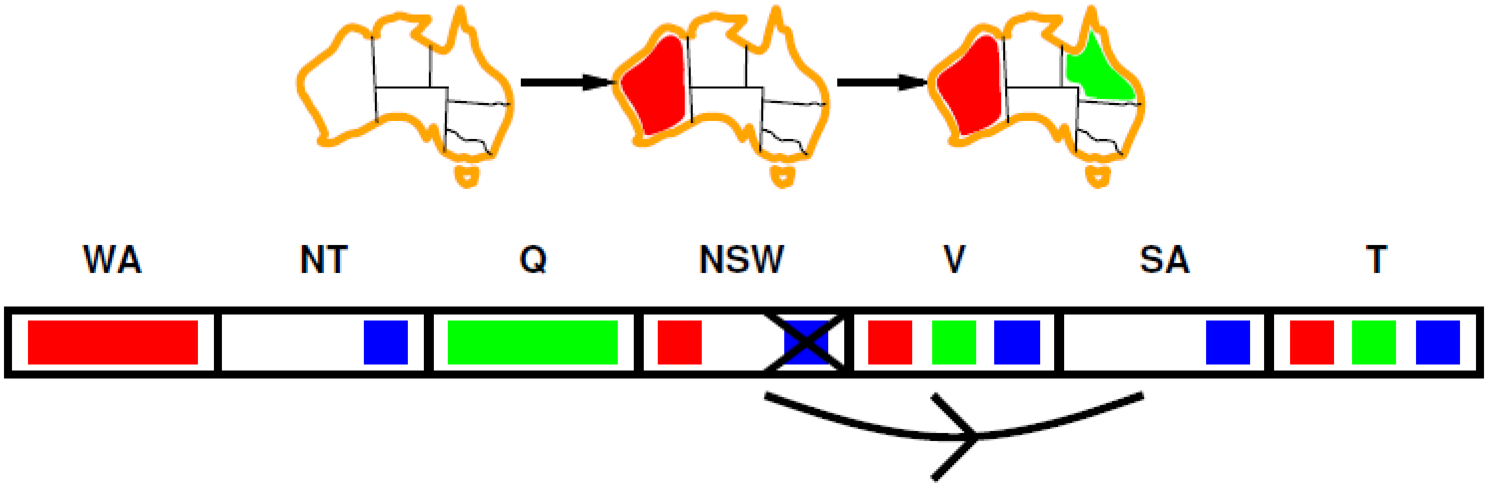
  for *every value x* of X there is *some allowed y*



Arc consistency

- Simplest form of propagation make each arc *consistent*
- *X → Y* is consistent iff

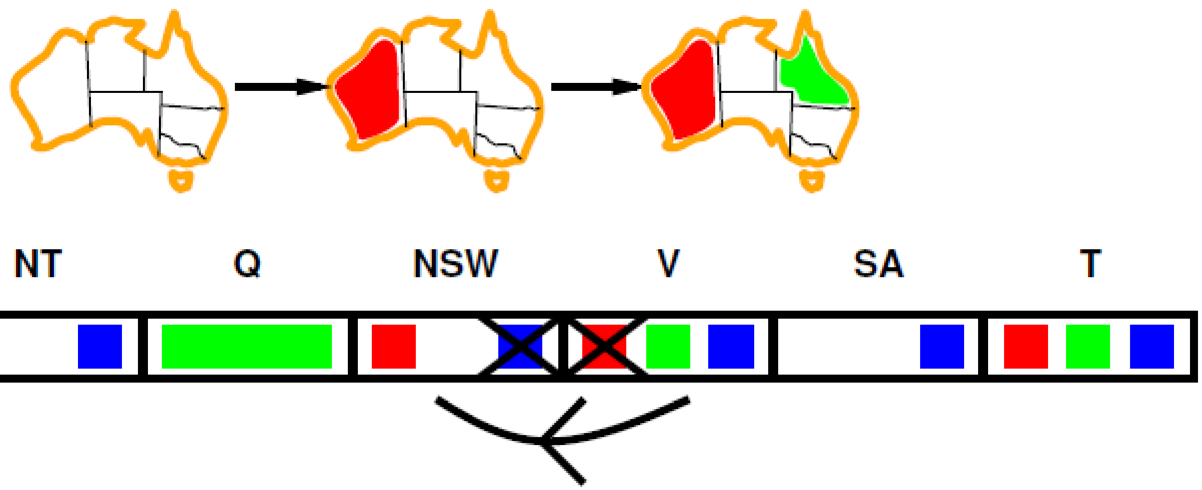
  for *every value x* of X there is *some allowed y*



Arc consistency

- Simplest form of propagation make each arc *consistent*
- *X* → *Y* is consistent iff

  for *every value x* of X there is *some allowed y*

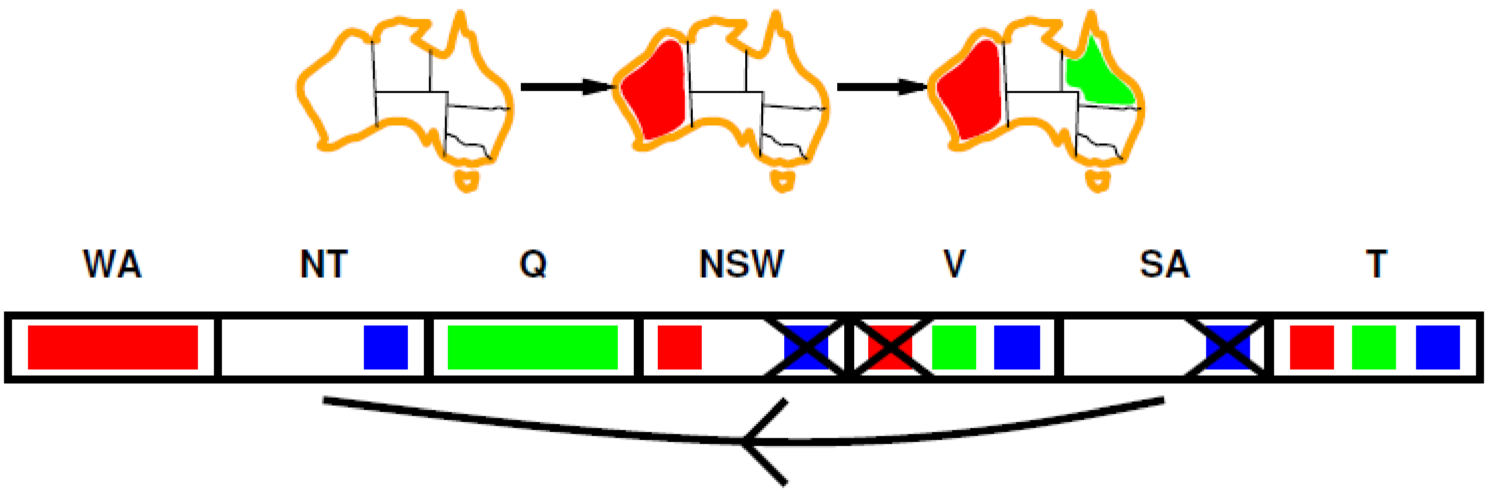


If *X* loses a value, neighbors of *X* need to be *rechecked*

Arc consistency

- Simplest form of propagation make each arc *consistent*
- *X → Y* is consistent iff

  for *every value x* of X there is *some allowed y*



Arc consistency detects failure earlier than forward checking

Can be run as a *preprocessor or after* each assignment

Arc consistency algorithm

```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

Exercise: Application of AC-3

- Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA=green, V=red} for the problem of colouring the map of Australia

Exercise: Application of AC-3

- Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA=green, V=red} for the problem of colouring the map of Australia

- Does the ordering of arcs matter? For one specific ordering of arcs:
  1. SA –> WA ; $D_{SA}=R\cancel{G}B$, $D_{WA}=G$ (delete G from SA)
  2. SA –> V ; $D_{SA}=\cancel{RG}B$, $D_{V}=R$, (delete R from SA, leaving only B)
  3. NT –> WA ; $D_{NT}=R\cancel{G}B$, $D_{WA}=G$, (delete G from NT)

Exercise: Application of AC-3

- Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA=green, V=red} for the problem of colouring the map of Australia

- Does the ordering of arcs matter? For one specific ordering of arcs:
  1. SA –> WA ; $D_{SA}=R\cancel{G}B$, $D_{WA}=G$ (delete G from SA)
  2. SA –> V ; $D_{SA}=\cancel{R}\cancel{G}B$, $D_{V}=R$, (delete R from SA, leaving only B)
  3. NT –> WA ; $D_{NT}=R\cancel{G}B$, $D_{WA}=G$, (delete G from NT)
  4. NT –> SA ; $D_{NT}=R\cancel{G}\cancel{B}$, $D_{SA}=\cancel{R}\cancel{G}B$ (delete B from NT, leaving only R)
  5. NSW –> SA ; $D_{NSW}=RG\cancel{B}$, $D_{SA}=\cancel{R}\cancel{G}B$ (delete B from NSW)
  6. NSW –> V ; $D_{NSW}=\cancel{R}G\cancel{B}$, $D_{V}=R$ (delete R from NSW, leaving only G)
  7. Q –> NT ; $D_{Q}=\cancel{R}GB$, $D_{NT}=R\cancel{G}\cancel{B}$ (delete R from Q)
  8. Q –> SA ; $D_{Q}=\cancel{R}G\cancel{B}$, $D_{SA}=\cancel{R}\cancel{G}B$ (delete B from Q)
  9. Q –> NSW ; $D_{Q}=\cancel{R}\cancel{G}\cancel{B}$, $D_{NSW}=\cancel{R}G\cancel{B}$ (delete G from Q, leaving no domain for Q)

Exercise: Arc consistency algorithm

- Task: AC-3 has a complexity of $O(n^2d^3)$, show how this is derived

Exercise: Arc consistency algorithm

- Task: AC-3 has a complexity of $O(n^2d^3)$, show how this is derived

$O(n^2)$: Need to check for all edges, potentially n^2 edges

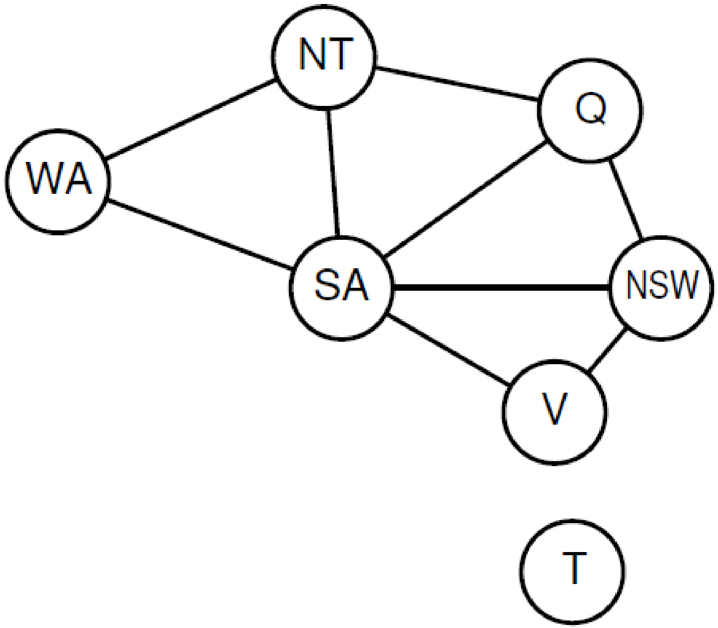
$O(d^2)$: For each edge, comparing their two domains

$O(d)$: Each variable change re-propagate to its neighbours, max d times

$O(n^2d^3)$: $O(n^2) * O(d^2) * O(d)$

Problem structure

- Tasmania and mainland are *independent subproblems*
- Identifiable as *connected components* of constraint graph

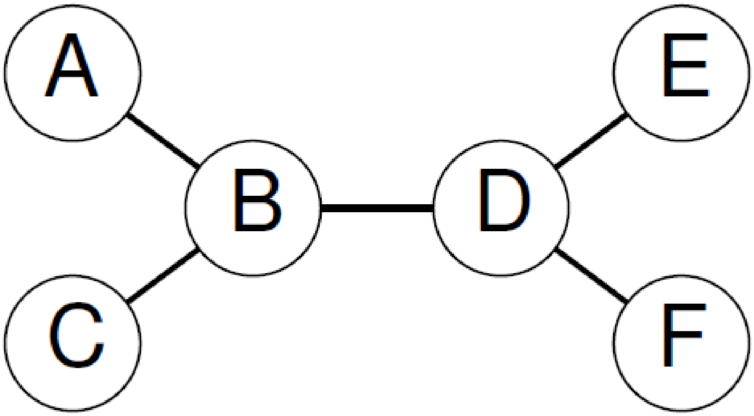


Problem structure

- Suppose each subproblem has *c* variables out of *n* total
- Worst-case solution cost is $(n/c) \cdot d^c$, linear in n
- E.g., *n=80, d=2, c=20*
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - 4×2^{20} = 0.4 seconds at 10 million nodes/sec

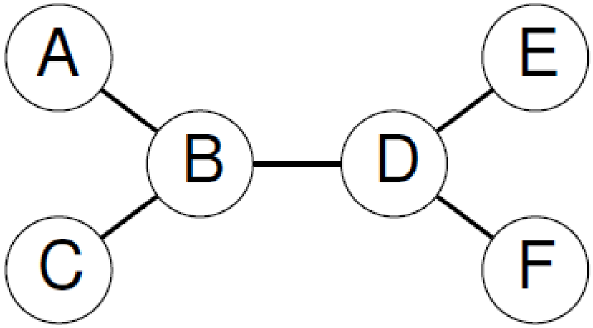
Tree-structured CSPs

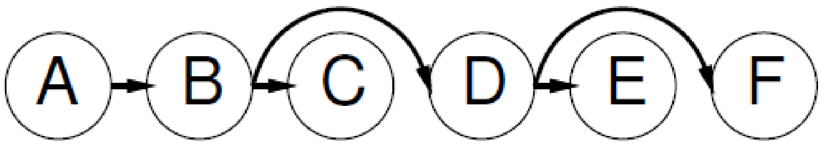
- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time
- Compare to general CSPs, where worst-case time is $O(d^n)$



Algorithm for tree-structured CSPs

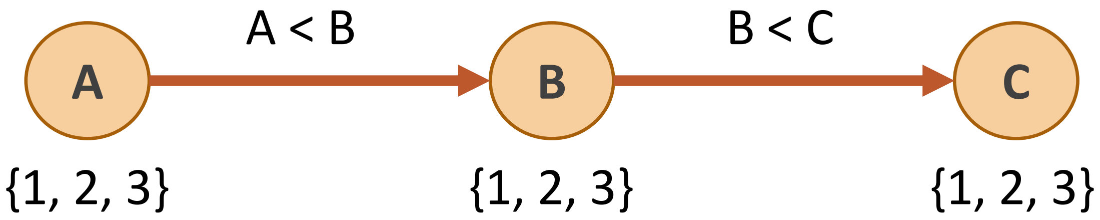
1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
2. For *j* from *n* down to *2*, apply *RemoveInconsistent(Parent(X_j),X_j)*
3. For *j* from *1* to *n*, assign X_j consistently with *Parent(X_j)*





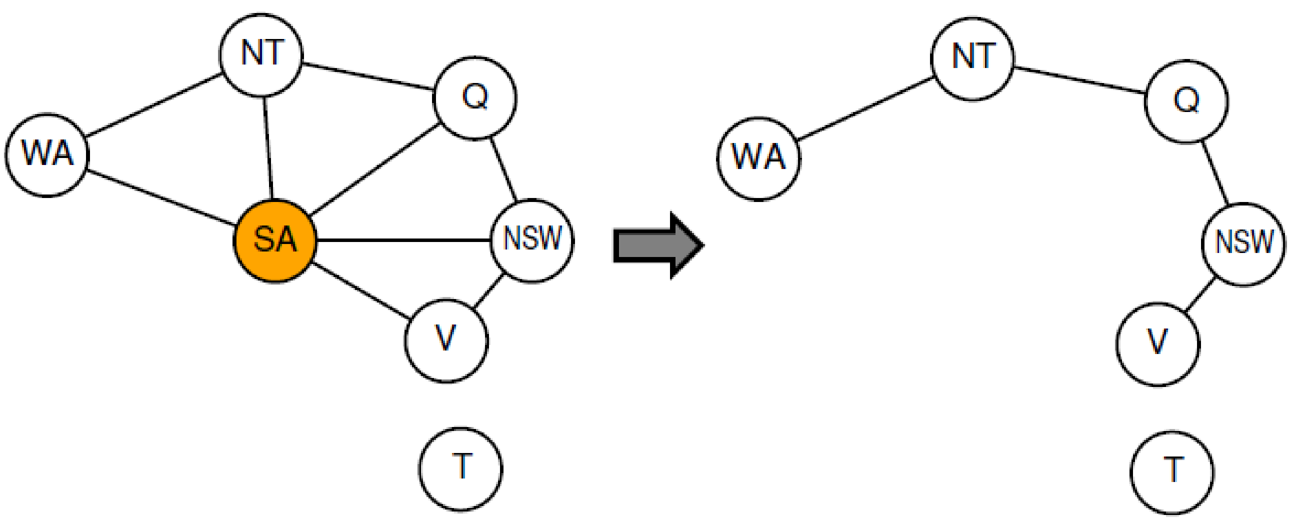
Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
2. For *j* from *n* down to *2*, apply *RemoveInconsistent(Parent(X_j),X_j)*
3. For *j* from *1* to *n*, assign X_j consistently with *Parent(X_j)*



Nearly tree-structured CSPs

- *Conditioning*: instantiate a variable, prune its neighbors' domains
- *Cutset conditioning*: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- *Cutset size c* ⇒ runtime $O(d^c \cdot (n - c)d^2)$, very fast for small c



Summary & Objectives

- CSPs are a special kind of problem:
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Speed-ups to backtracking via:
  - Variable ordering, value selection, early failure detection, problem structure
- Able to apply backtracking search and the various speed-ups/heuristics on CSP problems.