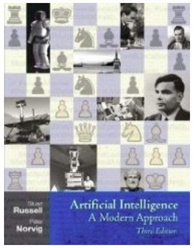
Outline

- 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
  - Backtracking Search
  - Forward Checking
  - Constraint Propagation
  - Local Search
  - Tree-Structured CSPs



Many slides based on Russell & Norvig's slides *Artificial Intelligence: A Modern Approach*

Constraint Satisfaction Problems

Special Type of search problem:

- state is defined by variables X_i with d values from domain D_i
- goal test is a set of constraints specifying allowable combinations of values for subsets of variables

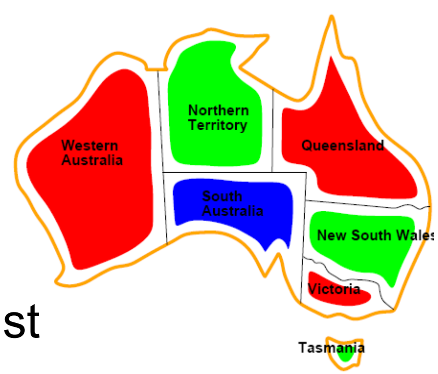
- Examples:
  - Sudoku

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

  - Graph/Map-Coloring

only n colors, neighbors must be different



- cryptarithmetic puzzle

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

- n-queens

Real-World CSPs

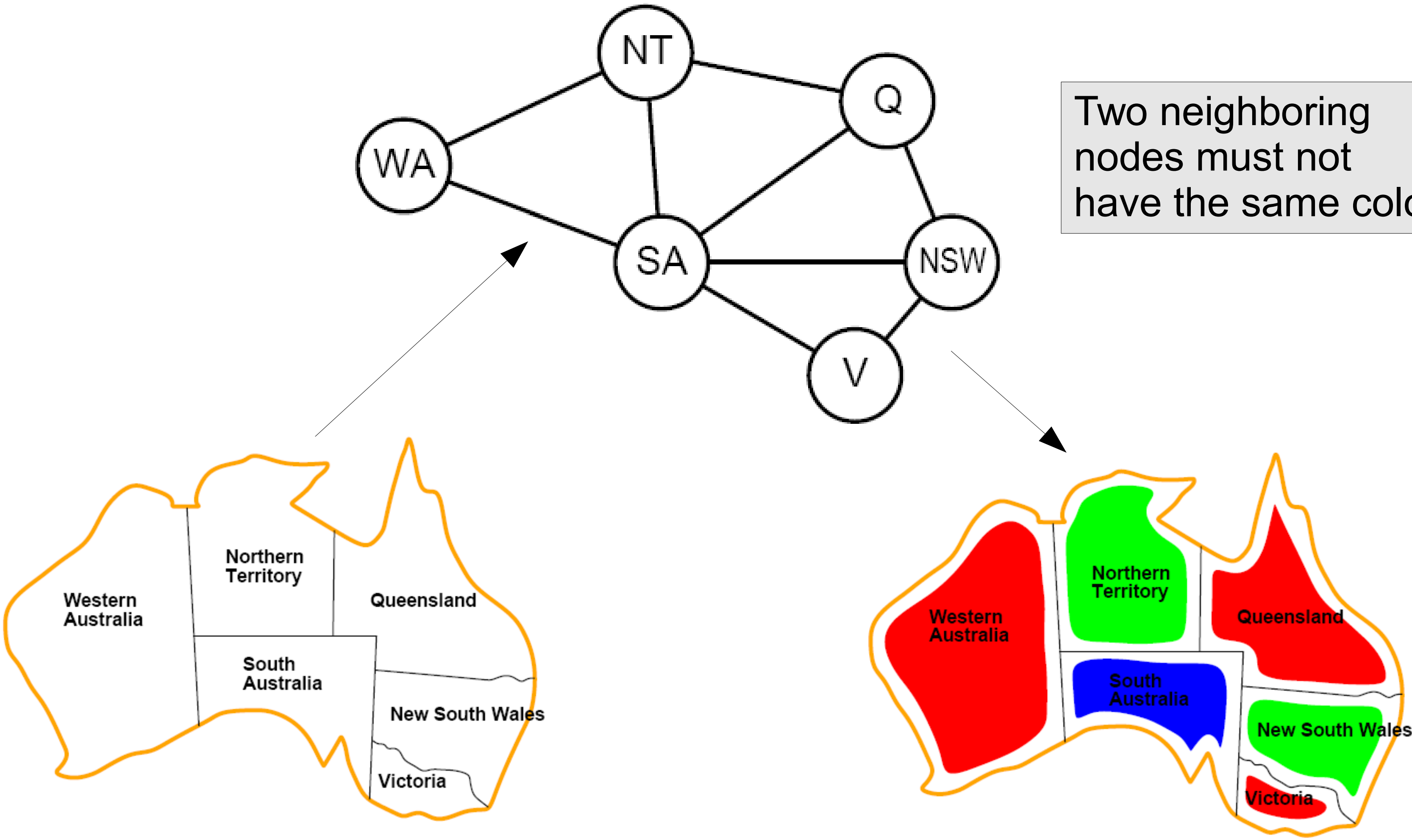
- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Scheduling
  - Job scheduling
    - Constraints are, e.g., start and end times for each job
  - Transportation scheduling
  - Factory scheduling
- Floorplanning

Notice that many real-world problems involve real-valued variables

- Linear constraints solvable in polynomial time using linear programming
- Problems with nonlinear constraints undecidable

Constraint Graph

- nodes are variables
- edges indicate constraints between them

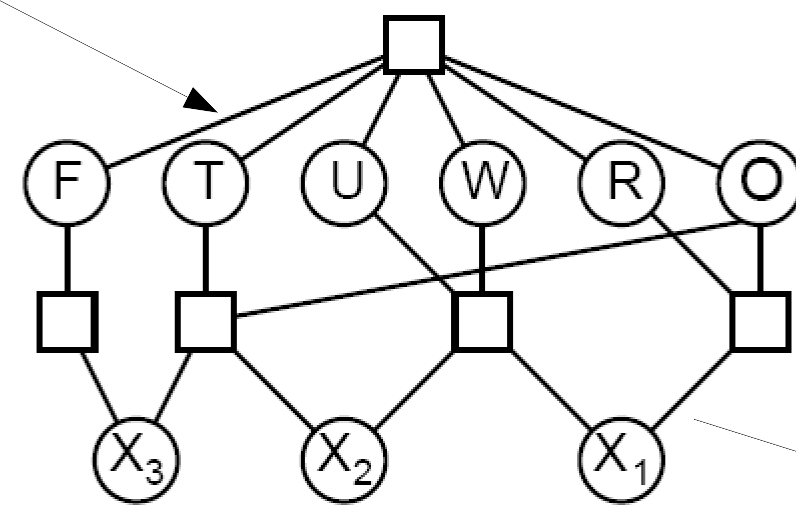


Two neighboring nodes must not have the same color

Constraint Graph

- nodes are variables
- edges indicate constraints between them

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



Connected nodes are involved in (in-)equations:

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

```
   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

Solving CSP Problems

Two principal approaches:

- Search:
  - successively assign values to variable
  - check all constraints
  - if a constraint is violated → backtrack
  - until all variables have assigned values
- Constraint Propagation:
  - maintain a set of possible values D_i for each variable X_i
  - try to reduce the size of D_i by identifying values that violate some constraints

Solving Constraint Problems with Search

- Constraint problems define a simple search space:
  - The start node is an empty assignment of values to variables
  - Its successors are all possible ways of assigning one value to a variable (depth 1)
  - Their successors are those with 2 variables assigned (depth 2)
  - ….
  - Until at the end all variables have been assigned a value (depth n)
- Goal test:
  - Does a node at depth n satisfy all constraints?
- Observation:
  - All solution nodes will appear at depth $n \rightarrow$ depth-first search is feasible without losing completeness

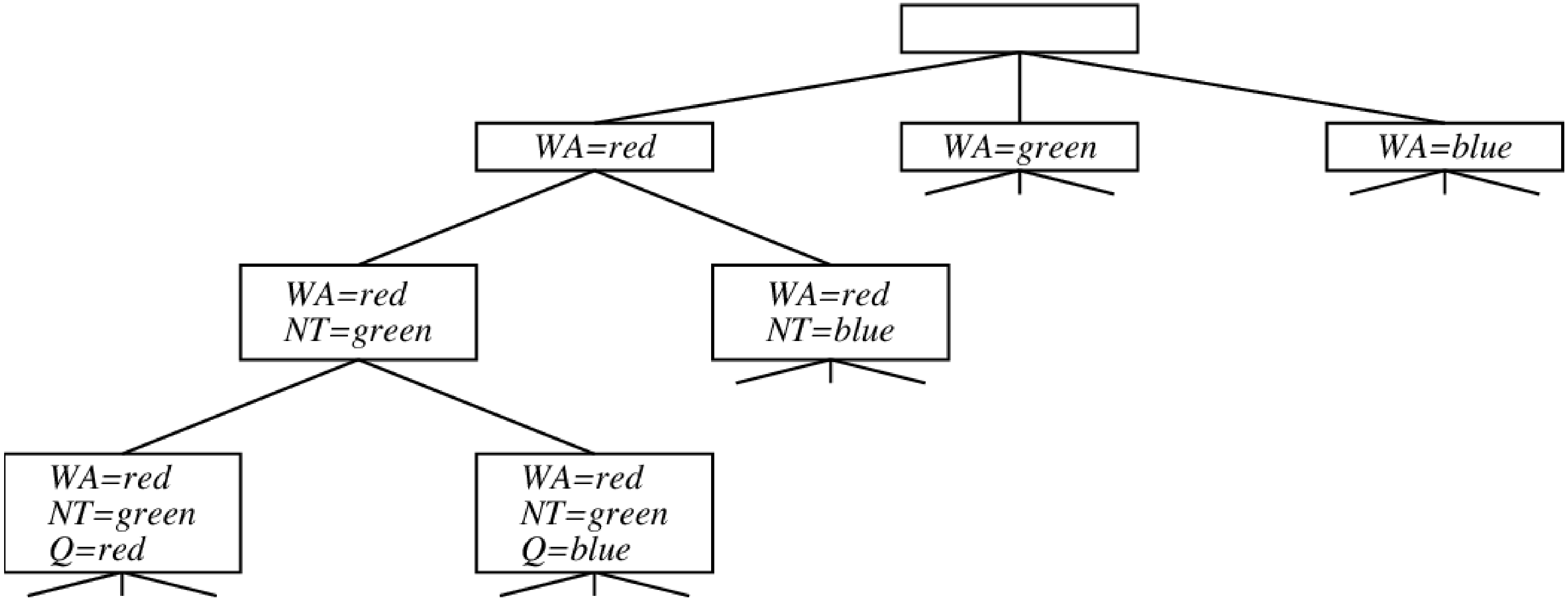
Complexity of Naive Search

- Assumptions
  - we have n variables
    $\rightarrow$ 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

Commutative Variable Assignments

- Variable assignments are commutative
  - [$WA = red$ then $NT = green$] is the same as [$NT = green$ then $WA = red$]
- Thus, at each node, we only need to make assignments for one of the variables

  → Total complexity reduces to v^n



Backtracking Search

- Depth-first search with single variable assignments per level is also called backtracking search

- Backtracking is the basic uninformed search algorithm for CSPs
  - add one constraint at a time without conflict
  - succeed if a legal assignment is found
  - Can solve n-queens problems for up to $n \simeq 25$

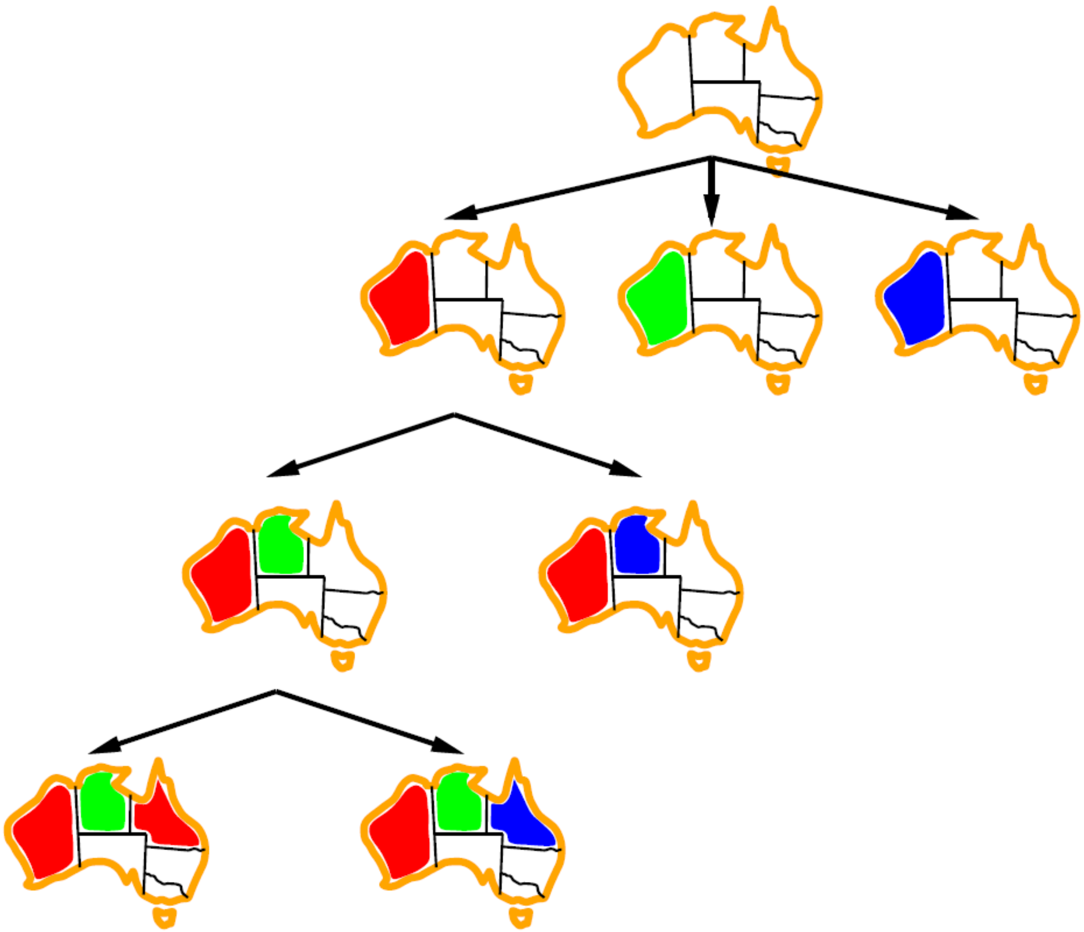
- Complexity:
  - Worst case is still exponentional
  - heuristics for selecting variables (SELECTUNASSIGNEDVARIABLE) and for ordering values (ORDERDOMAINVALUES) can improve practical performance

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 Search

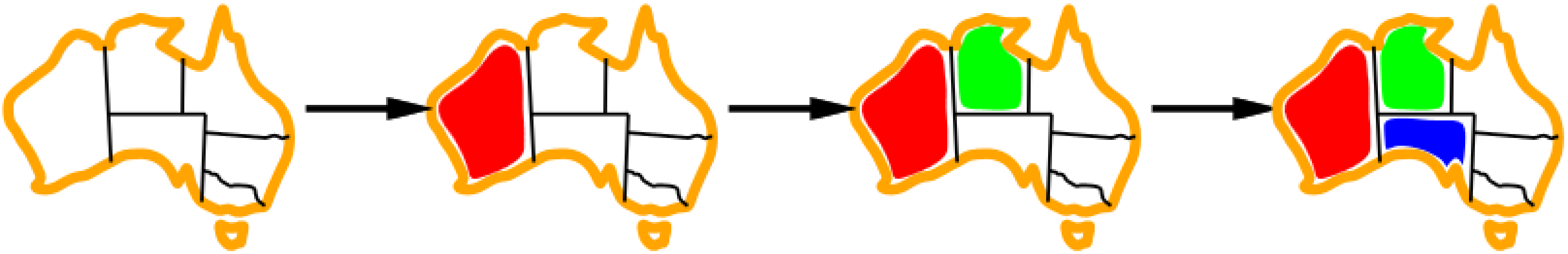


General-purpose methods can give huge gains in speed:

1) Which variable should be assigned next?
2) In what order should its values be tried?
3) Can we detect inevitable failure early?
4) Can we take advantage of problem structure?

General Heuristics for CSP

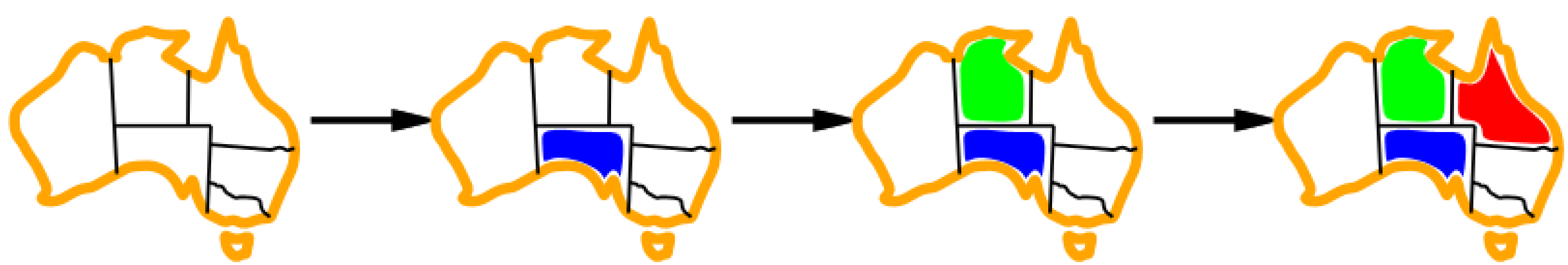
- Domain-Specific Heuristics
  - Depend on the particular characteristics of the problem
  - Obviously, a heuristic for the 8-puzzle can not be used for the 8-queens problem
- General-purpose heuristics
  - For CSP, good general-purpuse heuristics are known:
  - Mininum Remaining Values Heuristic
    - choose the variable with the fewest consistent values



General Heuristics for CSP

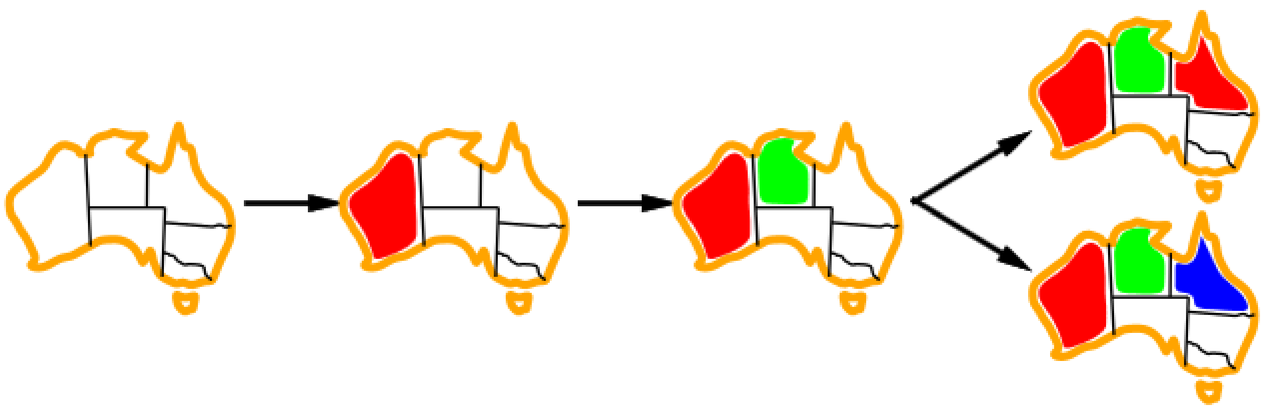
- Domain-Specific Heuristics
  - Depend on the particular characteristics of the problem
  - Obviously, a heuristic for the 8-puzzle can not be used for the 8-queens problem
- General-purpose heuristics
  - For CSP, good general-purpuse heuristics are known:
  - Mininum Remaining Values Heuristic
    - choose the variable with the fewest consistent values
  - Degree Heuristic
    - choose the variable with the most constraints on remaining variables

SelectUnassignedVariable



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



OrderDomainValues

- Least Constraining Value Heuristic
  - Given a variable, choose the value that rules out the fewest values in the remaining variables

General Heuristics for CSP

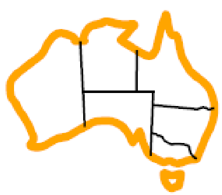
- Domain-Specific Heuristics
  - Depend on the particular characteristics of the problem
  - Obviously, a heuristic for the 8-puzzle can not be used for the 8-queens problem
- General-purpose heuristics
  - For CSP, good general-purpuse heuristics are known:
  - Mininum Remaining Values Heuristic
    - 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

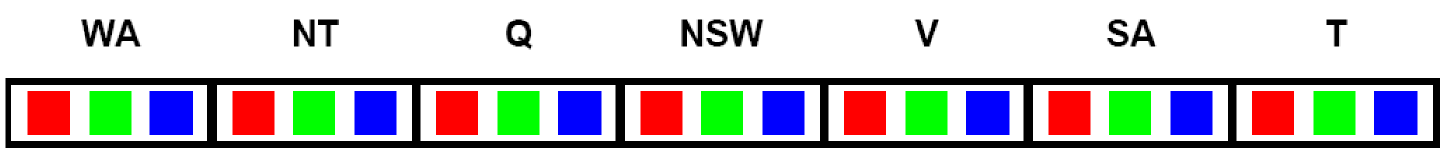
SelectUnassignedVariable

OrderDomainValues

Forward Checking

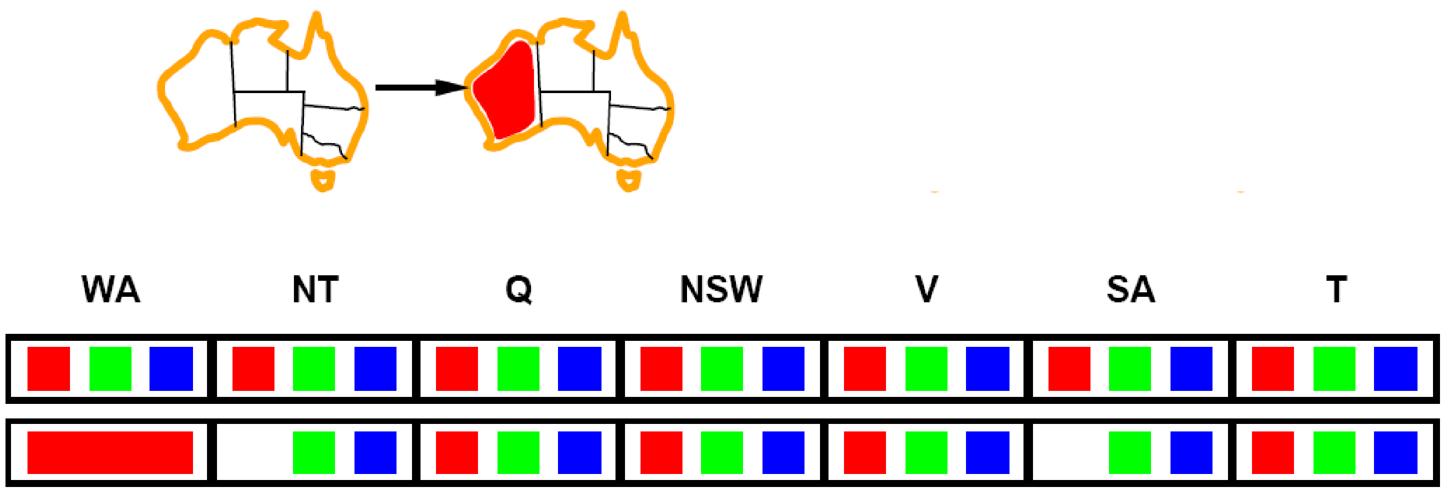
- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable has no more legal values





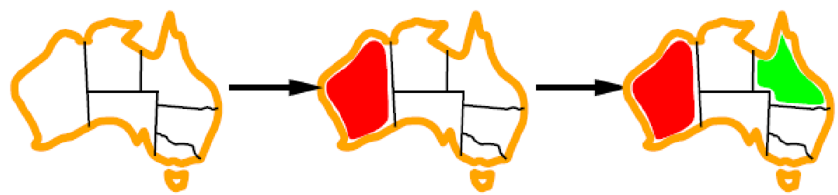
Forward Checking

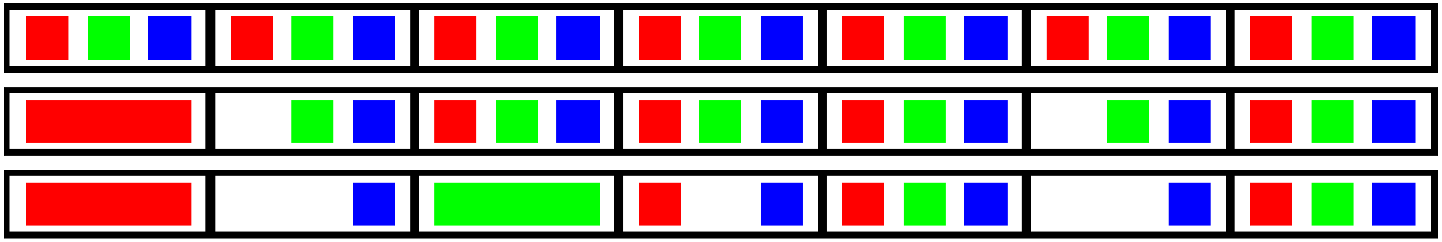
- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable has no more legal values



Forward Checking

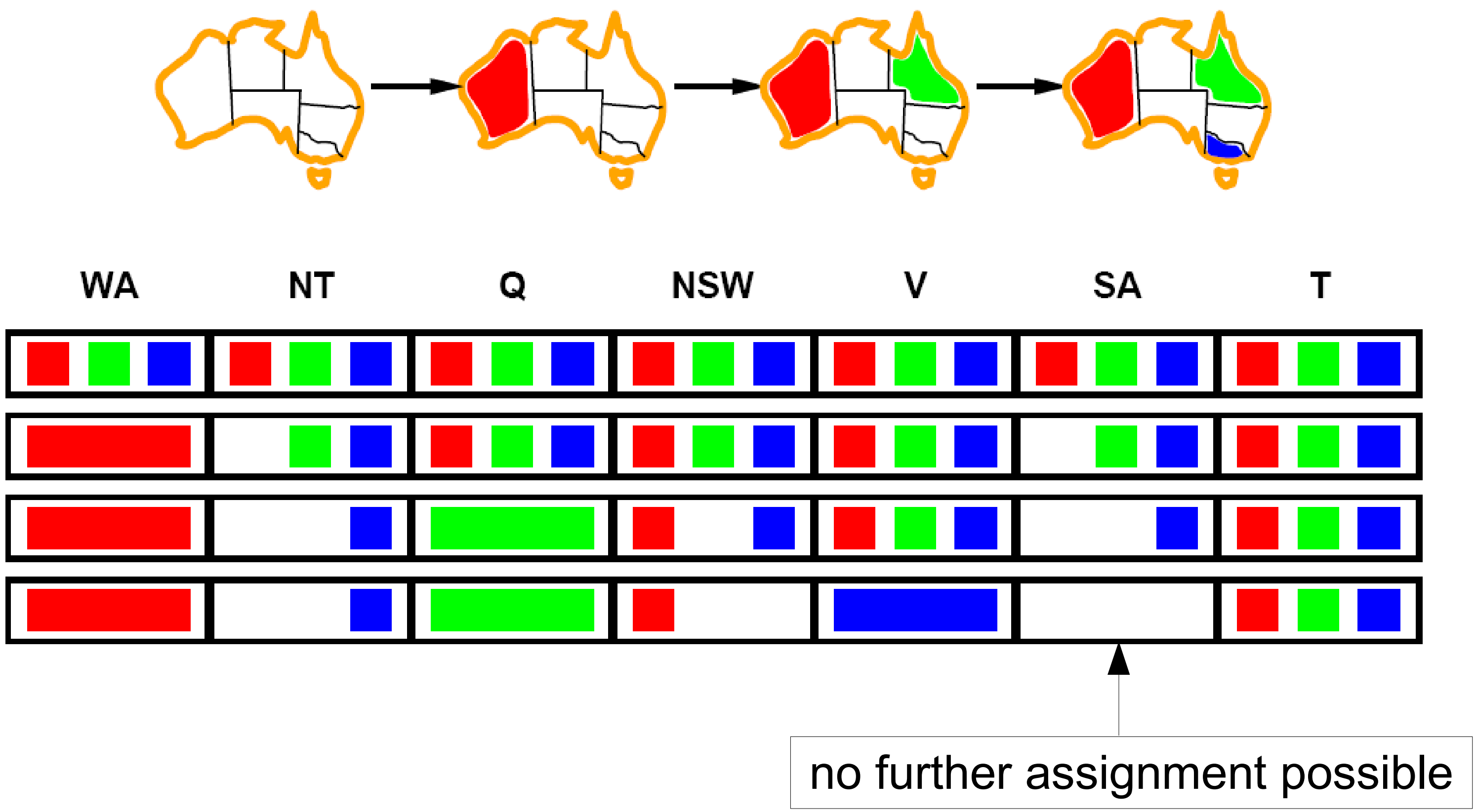
- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable has no more legal values





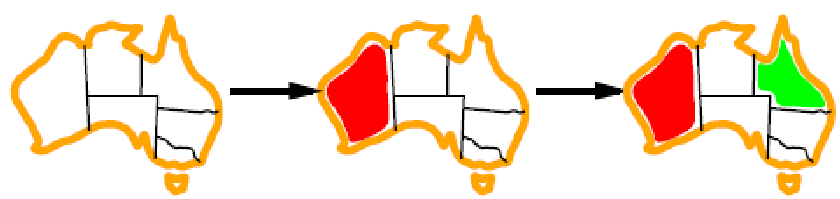
Forward Checking

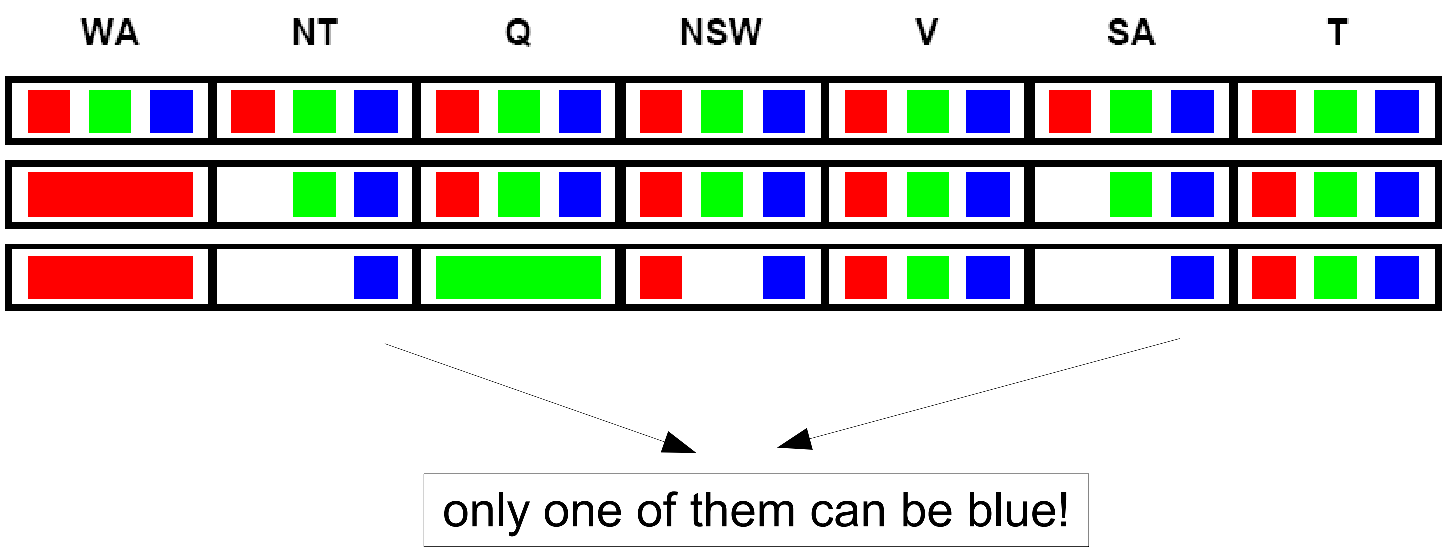
- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable has no more legal values



Constraint Propagation

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





Constraint Propagation - Sudoku

- Problem
  - CSP with 81 variables
- Constraints
  - some values are assigned in the start (unary constraints)
  - 27 constraints on 9 values that must all be different

    (9 rows, 9 columns, 9 squares)
- Constraint Propagation
  - People often write a list of possible values into empty fields
  - try to successively eliminate values
- Status
  - Automated constraint solvers can solve the hardest puzzles in no time

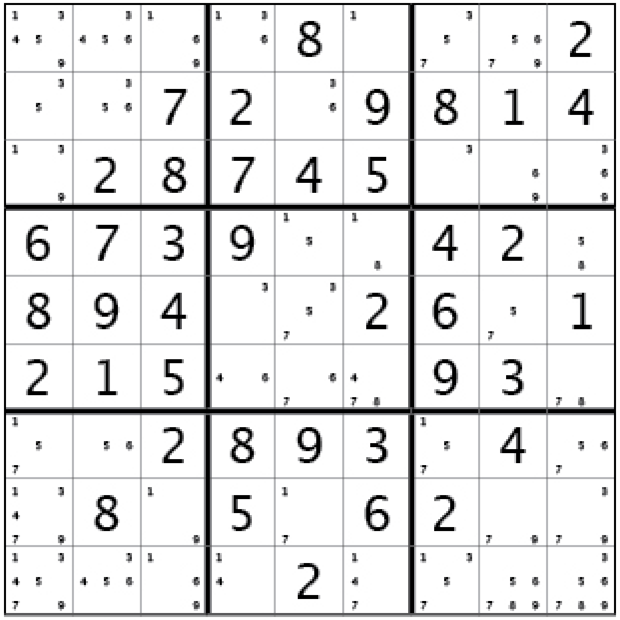


Figure 6

Node Consistency

Node Consistency

- the possible values of a variable must conform to all unary constraints
- can be trivially enforced
- Example:
  - Sudoku: Some nodes are already filled out, i.e., constrained to a single value

More General Idea: Local Consistency

- make each node in the graph consistent with its neighbors
- by (iteratively) enforcing the constraints corresponding to the edges

Arc Consistency

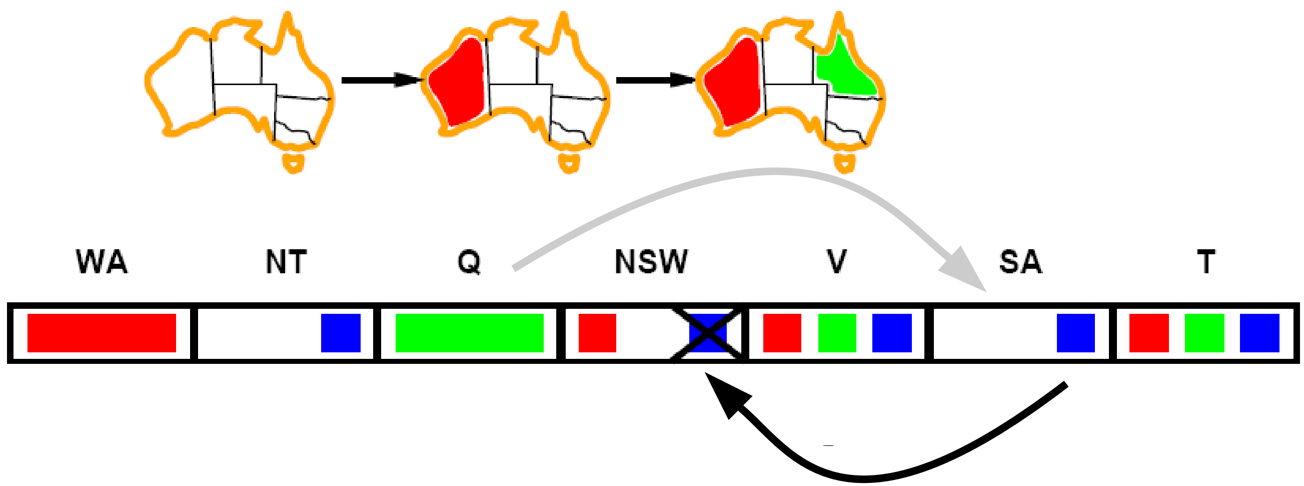
- every domain must be consistent with the neighbors:

> A variable X_i is arc-consistent with a variable X_j if
> - for every value in its domain D_i
> - there is some value in D_j
> - that satisfies the constraint on the arc (X_i, X_j)

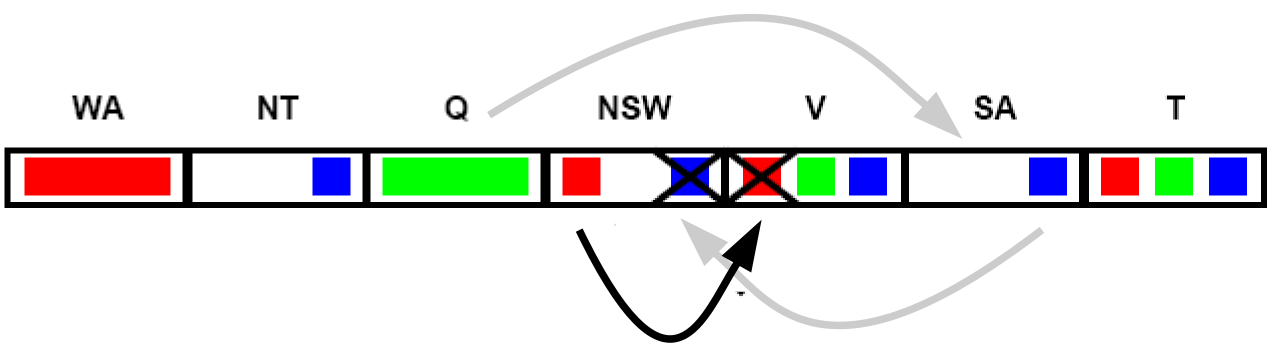
- can be generalized to n-ary constraints
  - each tuple involving the variable X_i has to be consistent

Maintaining Arc Consistency (MAC)

- After each new assignment of a value to a variable, possible values of the neighbors have to be updated:



- If one variable (NSW) looses a value (blue), we need to recheck its neighbors as well because they might have lost a possible value



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
```

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

```
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
```

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

Path Consistency

- Arc Consistency is often sufficient to
  - solve the problem (all domains have size 1)
  - show that the problem cannot be solved (some domains empty)
- but may not be enough
  - there is always a consistent value in the neighboring region

→ Path consistency
  - tightens the binary constraints by considering triples of values

A pair of variables (X_i , X_j) is path-consistent with X_m if
- for every assignment that satisfies the constraint on the arc (X_i,X_j)
- there is an assignment that satisfies the constraints on the arcs (X_i,X_m) and (X_j,X_m)

  - Algorithm AC-3 can be adapted to this case (known as PC-2)

k-Consistency

- The concept can be generalized so that a set of k values need to be consistent
  - 1-consistency = node consistency
  - 2-consistency = arc consistency
  - 3-consistency = path consistency
  - ....
- May lead to faster solution ($O(n^2d)$)
  - but checking for k-Consistency is exponentional in k in the worst case
- therefore arc consistency is most frequently used in practice

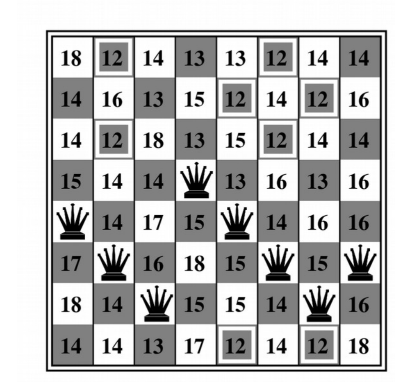
Sudoku

- simple puzzles can be solved with AC-3
  - the puzzle has 9 constraints on the rows, 9 on the columns and 9 on the square (27 in total)
    - each such constraint requires that 9 values are all different
  - the 9-valued AllDiff constraints can be converted into pairwise binary constraints
    - 9x8/2 = 36 pairwise constraints
  - therefore 27x36 = 972 arc constraints
- somewhat more with PC-2
  - there are 255,960 path constraints
- however, not all problems can be solved with constraint progapagation alone
  - to solve all puzzles we need a bit of search

Integrating Constraint Propagation and Backtracking Search

- Performance of Backtracking can be further sped up by integrating constraint propagation into the search
- Key idea:
  - each time a variable is assigned, a constraint propagation algorithm is run in order to reduce the number of choice points in the search
- Possible algorithms
  - Forward Checking
  - AC-3, but initial queue of constraints only contains constraints with the variable that has been changed

Local Search for CSP



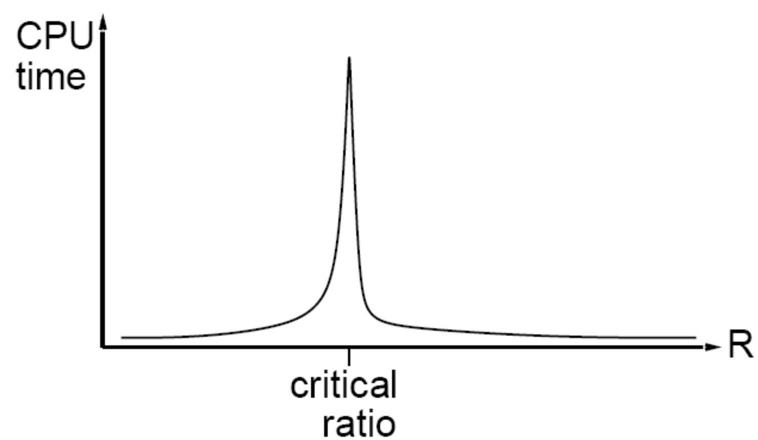
- Modifications for CSPs:
  - work with complete states
  - allow states with unsatisfied constraints
  - operators reassign variable values

- Min-conflicts Heuristic:
  - randomly select a conflicted variable
  - choose the value that violates the fewest constraints
  - hill-climbing with $h(n) =$ # of violated constraints

Min-conflicts is the heuristic that we studied for the 8-queens problems.

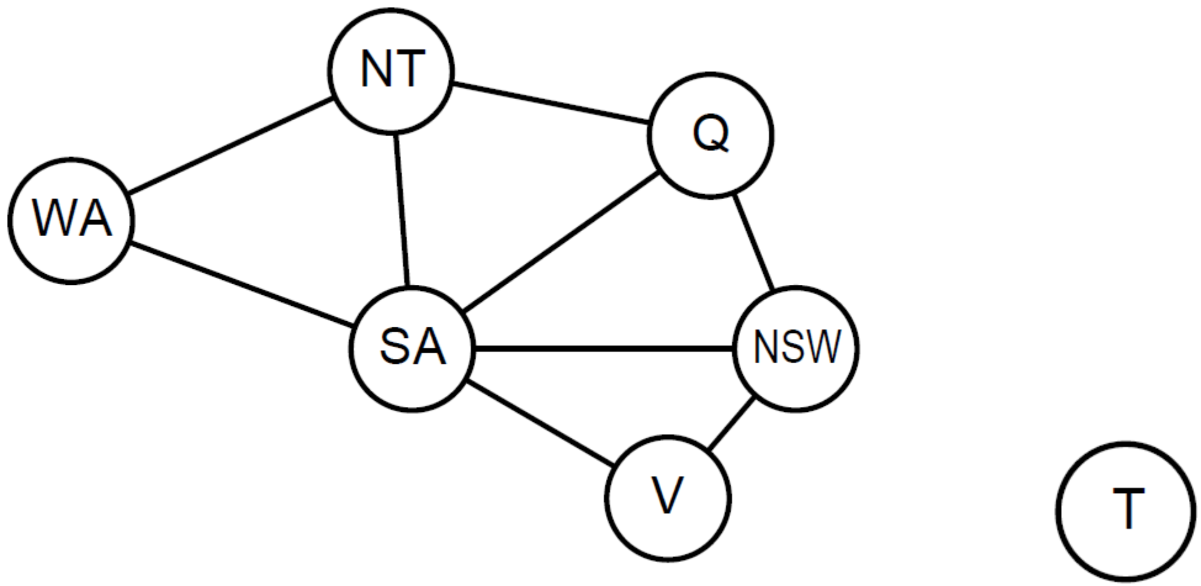
- Performance:
  - can solve randomly generated CSPs with a high probability
  - except in a narrow range of

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



Problem Structure

- Decomposing the problem into independent subproblems



  - The problem of coloring Tasmania is independent of the problem of coloring the mainland of Australia

The Power of Problem Decomposition

- Search space for a constraint satisfaction with n variables, each of which can have d values $= O(d^n)$
- Decomposing the problem into subproblems with c variables each:
  - Each problem has complexity $= O(d^c)$
  - There are n/c such problems

  $\rightarrow$ Total complexity $= O(n/c \cdot d^c)$
- Thus the total complexity can be reduced from exponential in n to linear in n!
- Example:

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

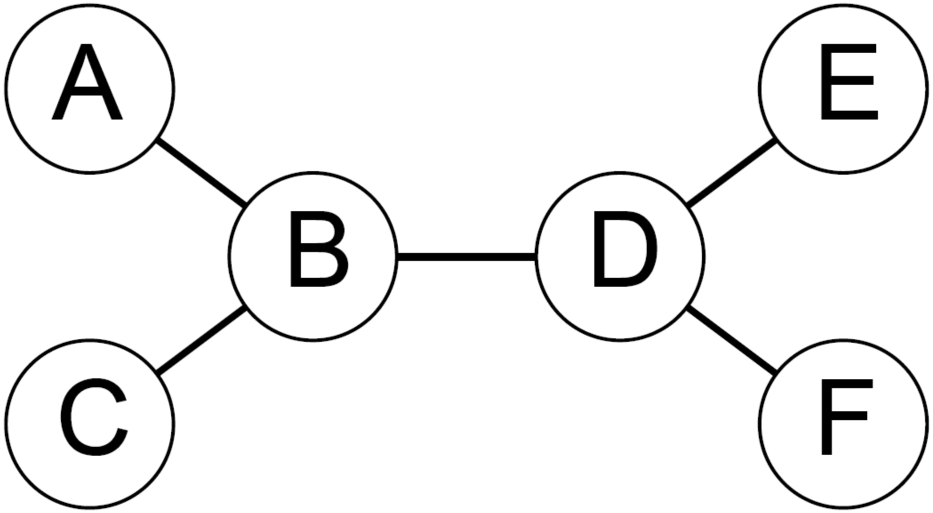
  2^{80} = 4 billion years at 10 million nodes/sec

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

Unconditional Independence is powerful but rare!

Tree-Structured CSP

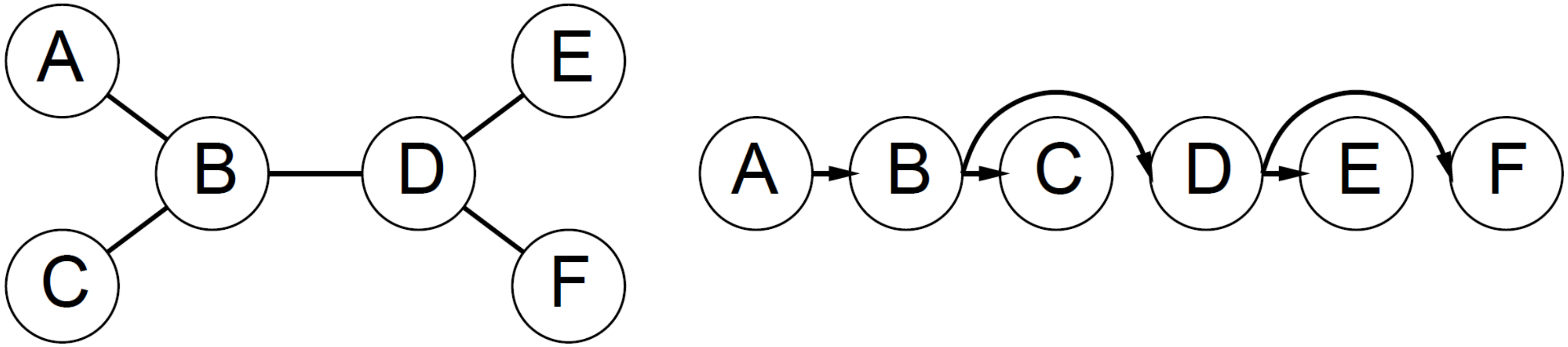
- A CSP is tree-structured if in the constraint graph any two variables are connected by a single path



Theorem: Any tree-structured CSP can be solved in linear time in the number of variables (more precisely: $O(n \cdot d^2)$)

Linear Algorithm for Tree-Structured CSPs

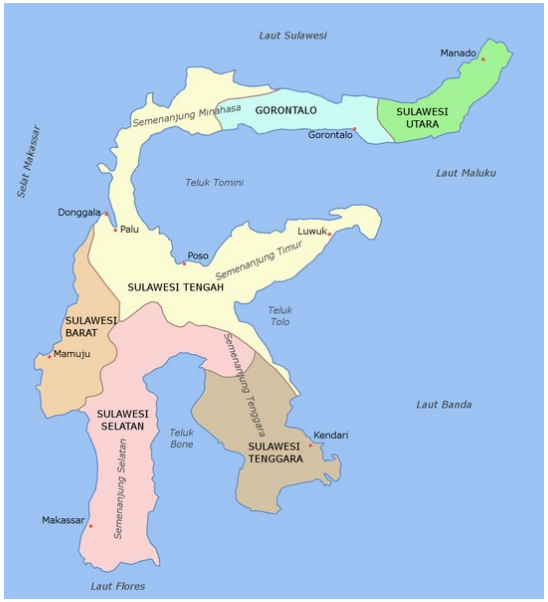
1) Choose a variable as a root, order nodes so that a parent always comes before its children (each child can have only one parent)



2) For $j = n$ downto 2
   - Make the arc (X_i, X_j) arc-consistent, calling REMOVE-INCONSISTENT-VALUE(X_i, X_j)
3) For $i = 1$ to n
   - Assign to X_i any value that is consistent with its parent.

Nearly Tree-structured Problems

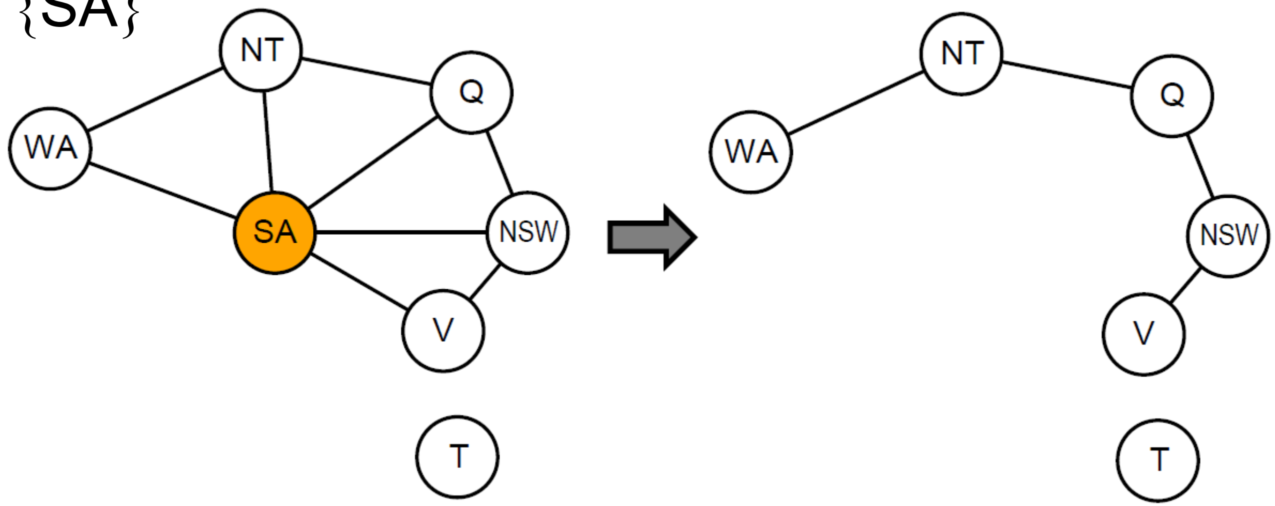
- Tree-structured problems are also rare.
- Most maps are clearly not tree-structured…
  - Exception: Sulawesi



- Two approaches for making problems tree-structured:
  - Removing nodes so that the remaining nodes form a tree (cutset conditioning)
  - Collapsing nodes together (decompose the graph into a set of independent tree-shaped subproblems)

Cutset Conditioning

1) Choose a subset S of the variables such that the constraint graph becomes a tree after removal of S (= the cycle cutset)
   - Example: $S = \{\text{SA}\}$



2) Choose a (consistent) assignment of variables for S
3) Remove from the remaining variables all values that are inconsistent with the variable of S
4) Solve the CSP problem for the remaining variables
5) If no solution → choose a different assignment for variables in 2)

Summary

- 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
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work
  - to constrain values and detect inconsistencies
- The CSP representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time