Constraint Satisfaction Problems

Presented by Yasin Ceran

October 8, 2024

1 Constraint Satisfaction Problems

2 Backtracking Search for CSPs

3 Constraint Propagation

4 Problem Structure

Constraint satisfaction problems (CSPs)

Standard search problem:
state is a "black box"—any old data structure
that supports goal test, eval, successor

CSP:
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

Simple example of a **formal representation language**

Allows useful **general-purpose** algorithms with more power
than standard search algorithms

Posing a Constraint Satisfaction Problem-I

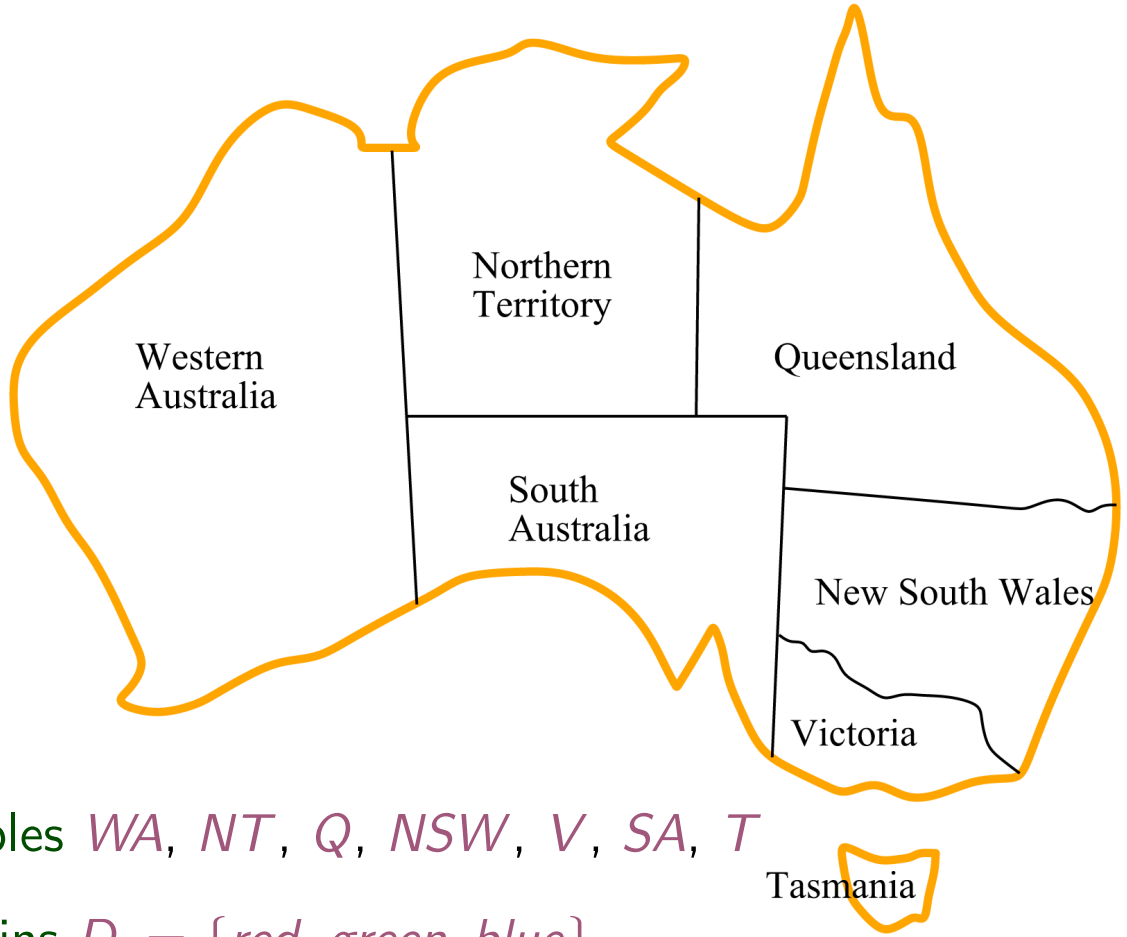
A CSP is characterized by

- A set of variables $X_1, X_2, \ldots, X_n$.
  - A discrete variable is one whose domain is finite or countably infinite.
  - A binary variable is a discrete variable with two values in its domain.
  - A Boolean variable, which is a variable with domain $\{false,\ true\}$
  - A variable whose domain is the real numbers or a range of the real numbers is a continuous variable.
- Each variable X_i has an associated domain D_i which specifies the set of possible values the variable can take.
  (We assume domains are finite.)
- Each domain D_i consists of allowable values $\{1, \ldots, v_k\}$ for variable X_i
- C is a set of constraints that specify allowable combinations of values.

Posing a Constraint Satisfaction Problem-II

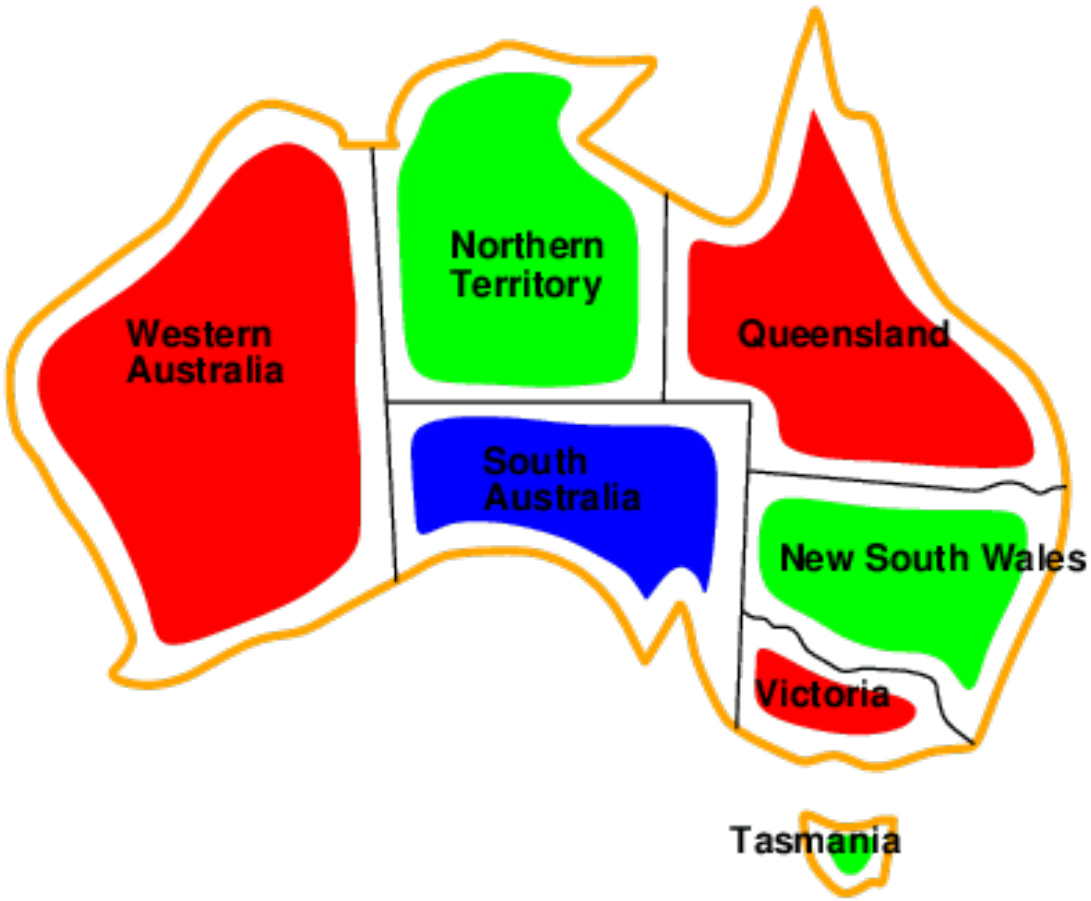
- Each constraint C_i consists of a pair $< scope, rel >$, where *scope* is a tuple of variables that participate in the constraint and *rel* is a relation that defines the values that those variables can take on.
- For example: If X_1 and X_2 both have the domain $\{A, B\}$, then the constraint saying the two variables must have different values can be written as $< (X_1, X_2), X_1 \neq X_2 >$
- Given a set of variables, an assignment on the set of variables is a function from the variables into the domains of the variables.
- We write an assignment on $\{X_1, X_2, \ldots, X_n\}$ as $\{X_1 = v_1, X_2 = v_2, \ldots, X_k = v_k\}$.
- An assignment that does not violate any constraints is called a consistent or legal assignment.
- A complete assignment is one in which every variable is assigned
- A solution to a CSP is a consistent, complete assignment.
- A partial assignment is one that assigns values to only some of the variables.
- A solution to CSP is total assignment that satisfies all the constraints.

Example: Map-Coloring



- Variables WA, NT, Q, NSW, V, SA, T
- Domains $D_i = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors, e.g., $WA \neq NT$ (if the language allows this), or
  $(WA, NT) \in$
  $\{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.

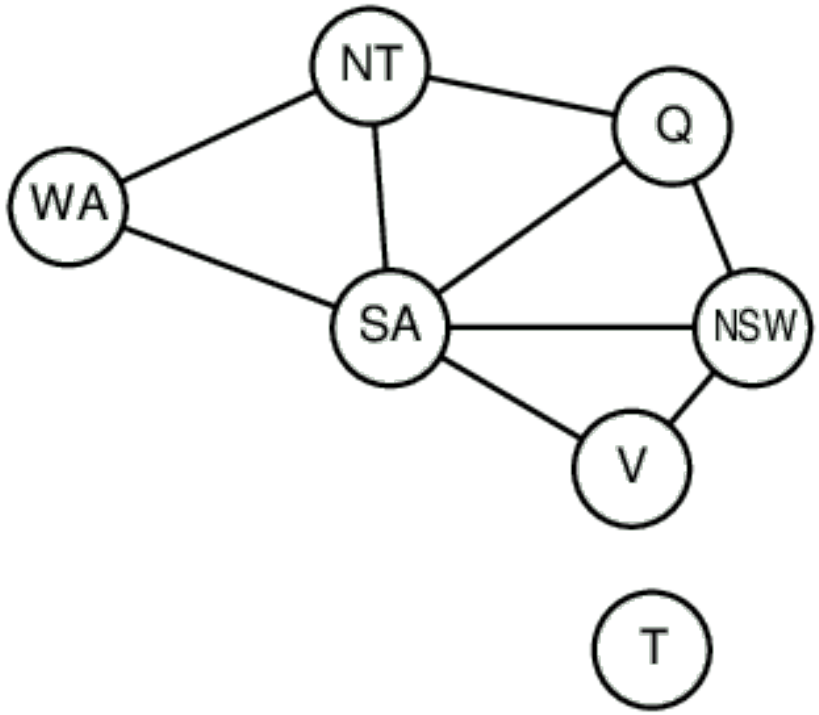


Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Varieties of CSPs

- The simplest kind of CSP involves variables that have discrete, finite domains.
- size d with n variables $\Longrightarrow$ $O(d^n)$ complete assignments
- Example: The 8-queens problem with the variables $Q_, \ldots, Q_8$ are the positions of each queen in columns $1, \ldots, 8$ and each variable has the domain $D_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$.
- Example: job scheduling, variables are start/end days for each job
- need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- linear constraints solvable, nonlinear undecidable
- Constraint satisfaction problems with continuous domains are common in the real world and are widely studied in the field of operations research.
- e.g., start/end times for Hubble Telescope observations
- The best-known category of continuous-domain CSPs is that of linear programming problems, where constraints must be linear equalities or inequalities.

Varieties of constraints

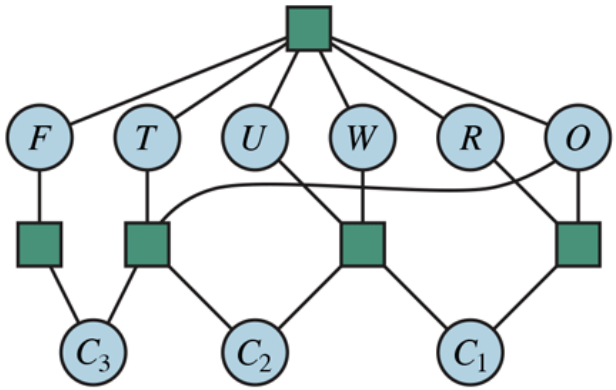
- Unary constraints involve a single variable, e.g., $SA \neq green$
- Binary constraints involve pairs of variables, e.g., $SA \neq WA$
- Higher-order constraints involve 3 or more variables,e.g., Y is between X and Z
- Preferences (soft constraints), e.g., *red* is better than *green*; often representable by a cost for each variable assignment.
  $\rightarrow$ constrained optimization problems

Example: Cryptarithmetic

A constraint involving an arbitrary number of variables is called a global constraint. One of the most common global constraints is Alldiff, which says that all of the variables involved in the constraint must have different values.

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

(a)



(b)

- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints:
  - mat*alldiff* (F, T, U, W, R, O)
  - $O + O = R + 10 \cdot C_1$, etc.

Real-world CSPs

- Assignment problems, e.g., who teaches what class
- Timetabling problems, e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floorplanning

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

CSP-Depth-Limited Search

- Consider a standard depth-limited search for the Australia map problem
- A State would be a partial assignment and an action would be extending the assignment
- For a CSP with n variables of domain size d
  - the branching factor at the top level is nd because any of d values can be assigned to any of n variables
  - At the next level, the branching factor is $(n-1)d$, and so on for n levels
  - We generate a tree with $n! \cdot d^n$ leaves, even though there are only d^n possible complete assignments!

Backtracking search

A problem is commutative if the order of application of any given set of actions has no effect on the outcome.

Variable assignments are commutative, i.e.,
[$WA = red$ then $NT = green$] same as [$NT = green$ then $WA = red$]

Only need to consider assignments to a single variable at each node
For a CSP with n variables of domain size d,there are d^n leaves

Depth-first search for CSPs with single-variable assignments
is called backtracking search

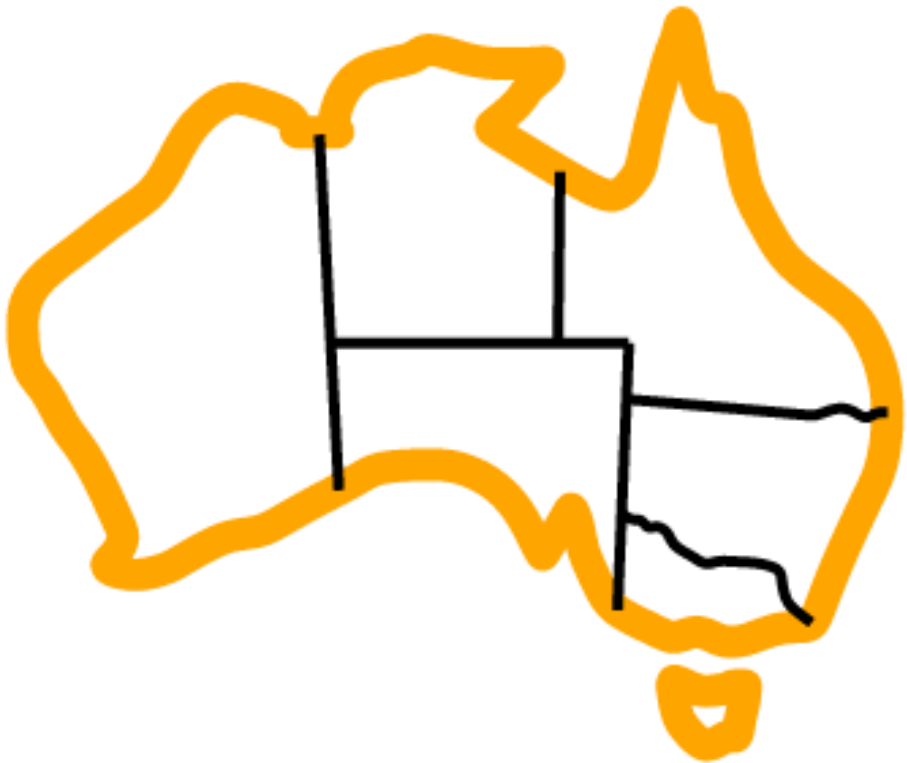
Backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

Backtracking search

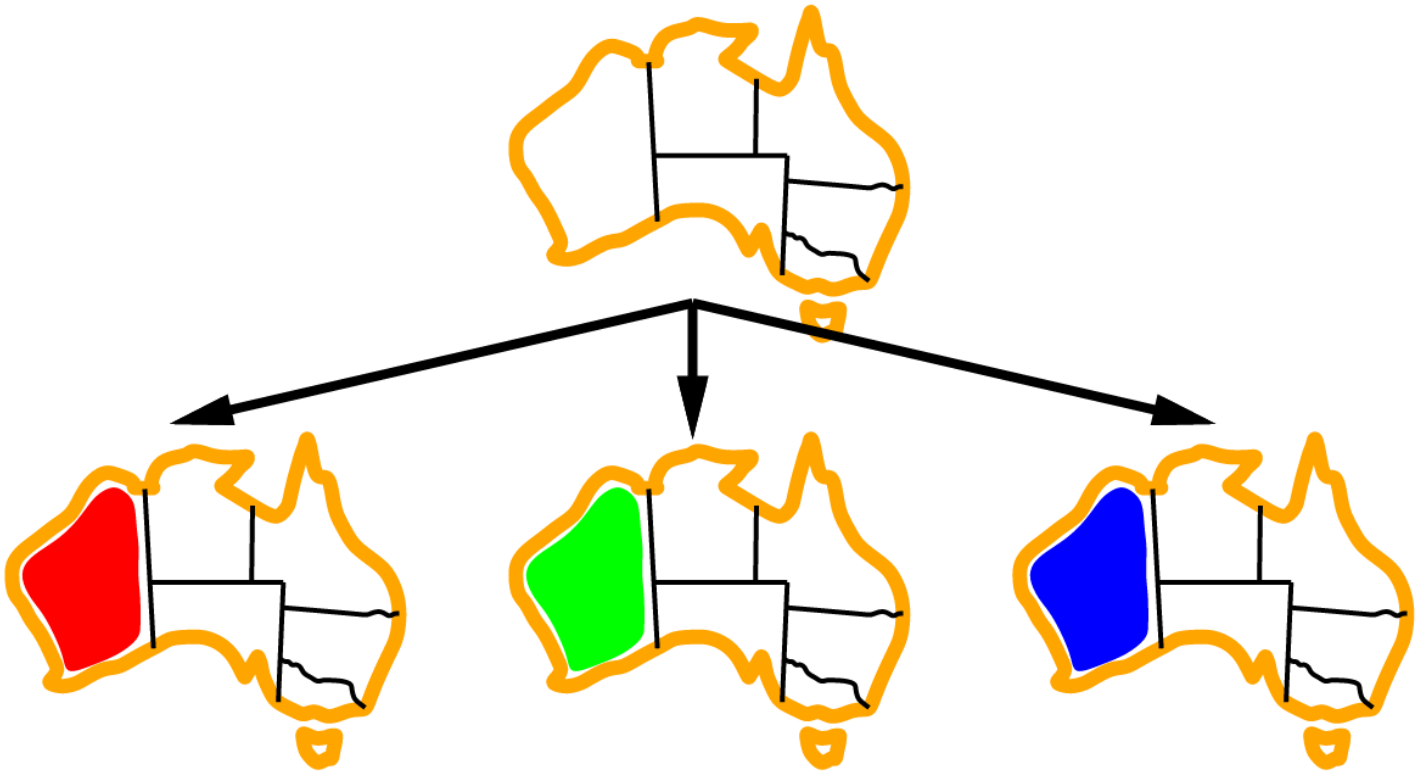
```
function Backtracking-Search(csp) returns solution/failure
    return Recursive-Backtracking({ }, csp)

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
```

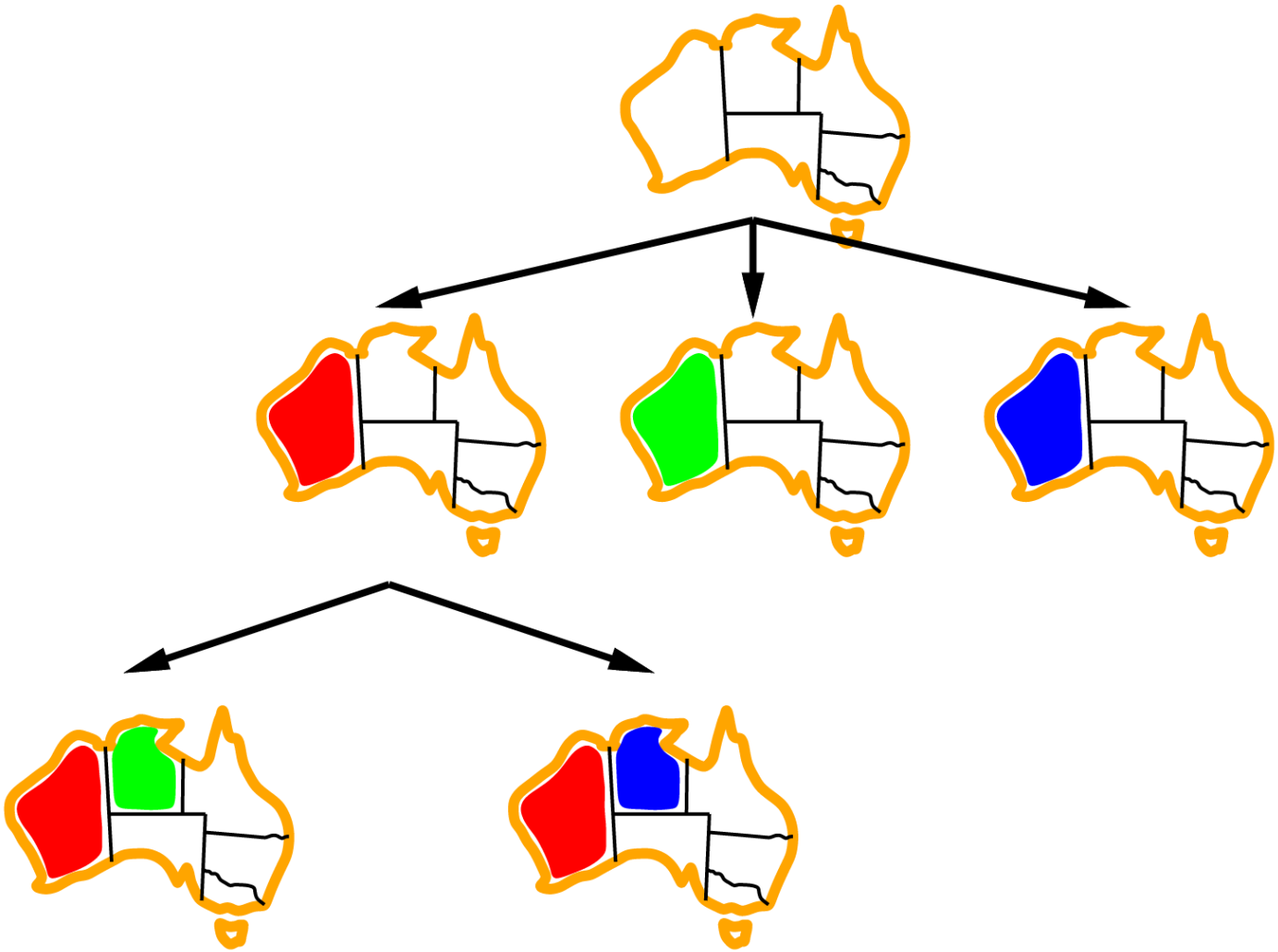
Backtracking example



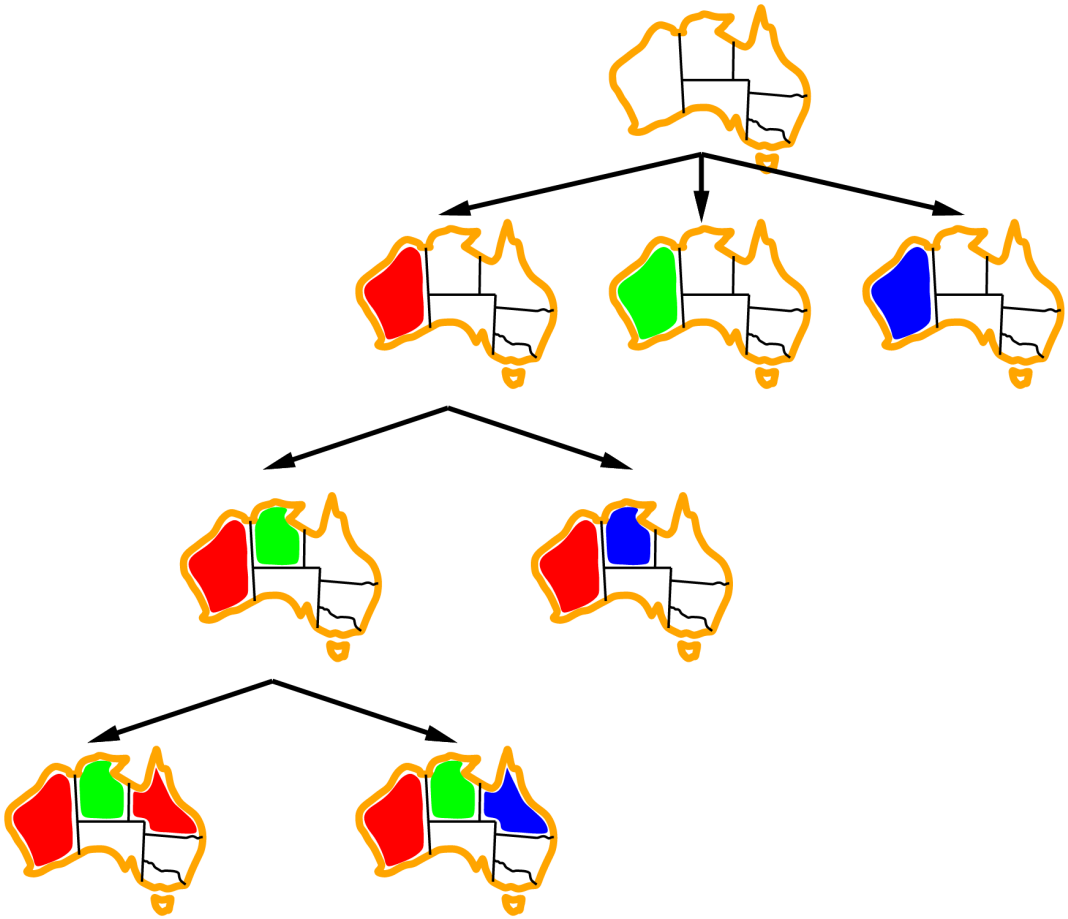
Backtracking example



Backtracking example



Backtracking example



Improving backtracking efficiency

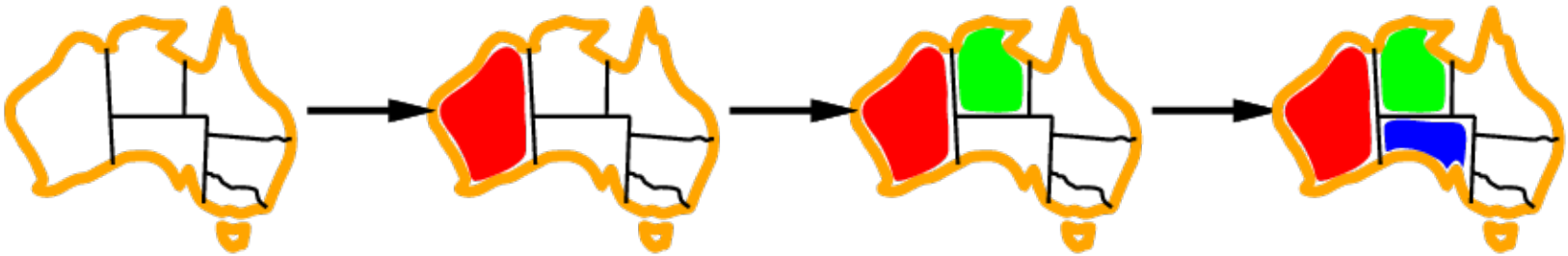
General-purpose methods can give huge gains in speed:

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 save and reuse partial results from the search ?

Variable and Value Ordering

- The backtracking algorithm contains the line $var \leftarrow SELECT\_UNASSIGNED\_VARIABLE(csp)$.
- The simplest strategy for $SELECT\_UNASSIGNED\_VARIABLE$ is to choose the next unassigned variable in order, $X_1, X_2, \ldots$

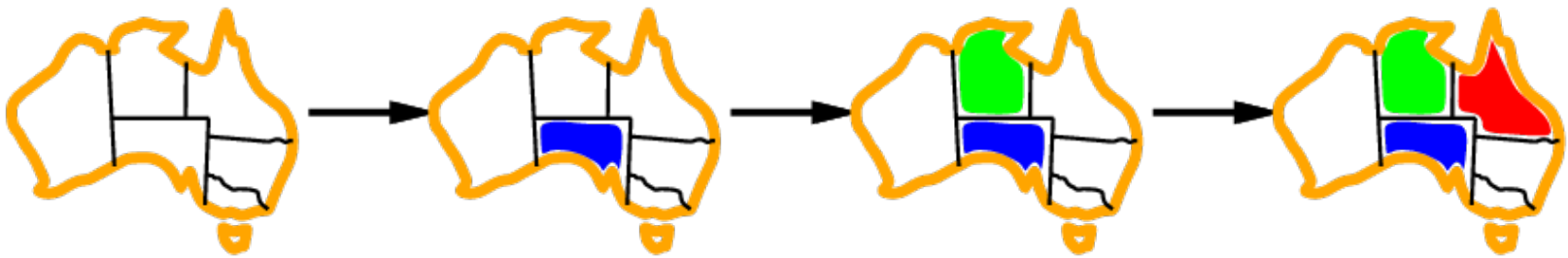
Minimum remaining values (MRV):
choose the variable with the fewest legal values



Degree heuristic

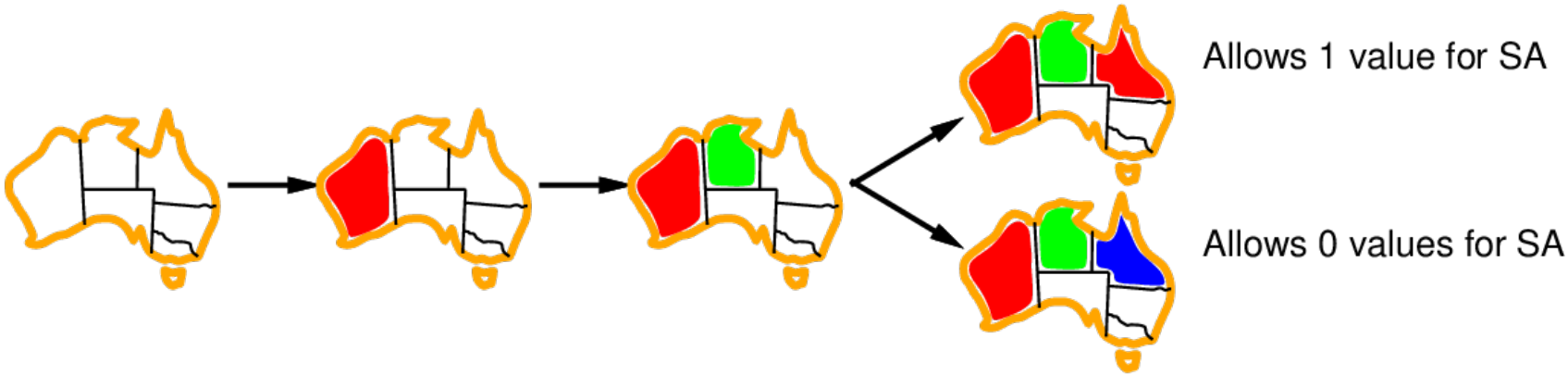
Tie-breaker among MRV variables

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

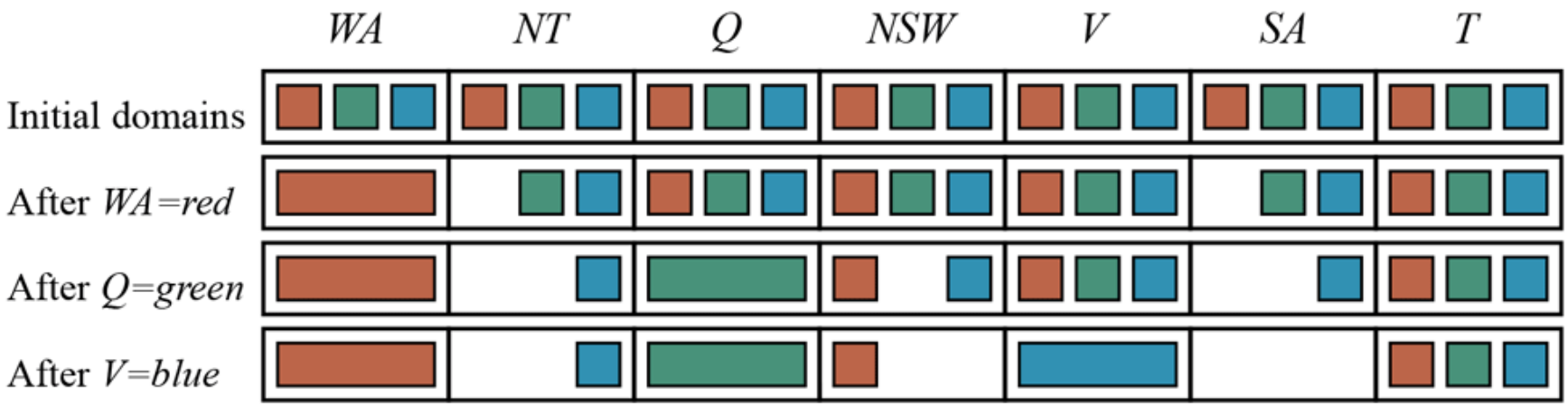


Combining these heuristics makes 1000 queens feasible

Forward checking

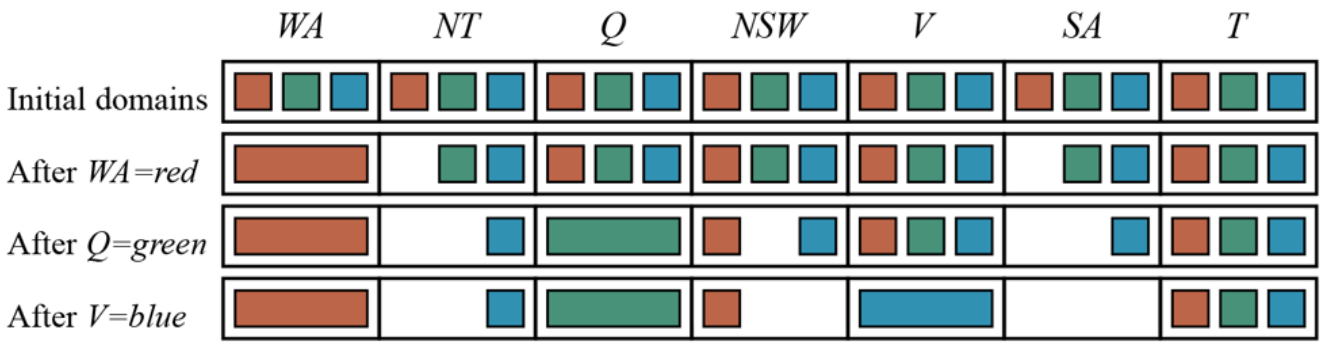
Idea: Keep track of remaining legal values for unassigned variables

Idea:Terminate search when any variable has no legal values



Constraint propagation

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



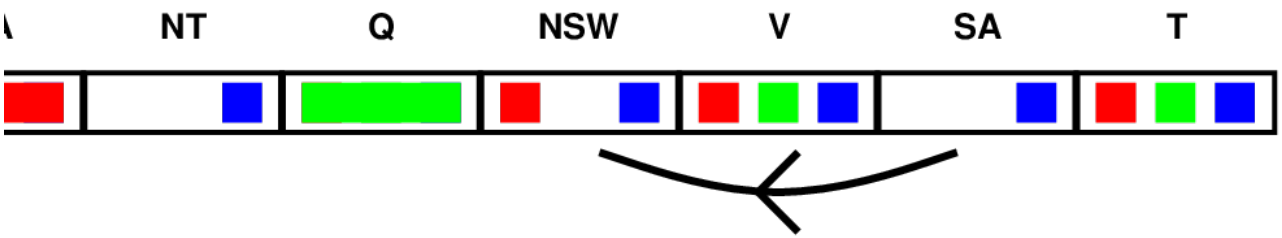
NT and *SA* cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc consistent

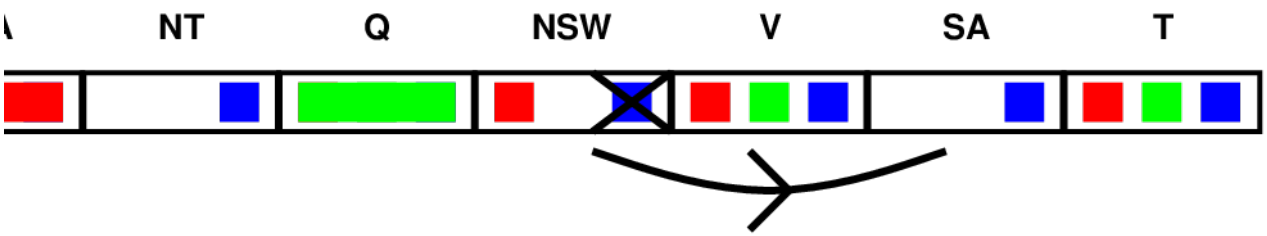
$X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y



Arc consistency

Simplest form of propagation makes each arc consistent

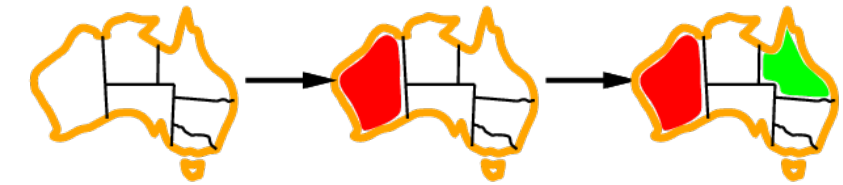
$X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y

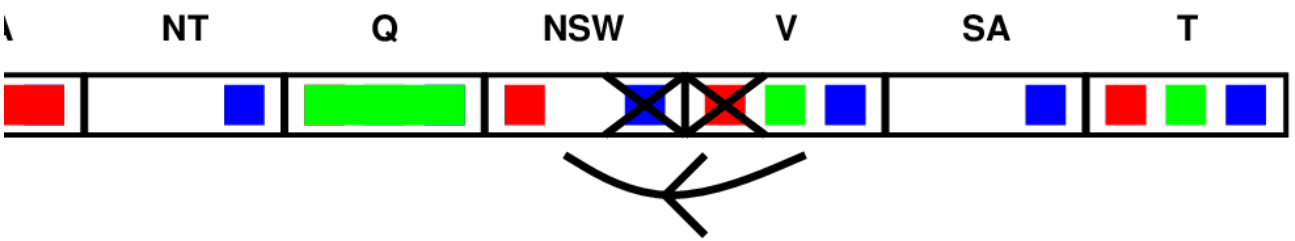


Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y



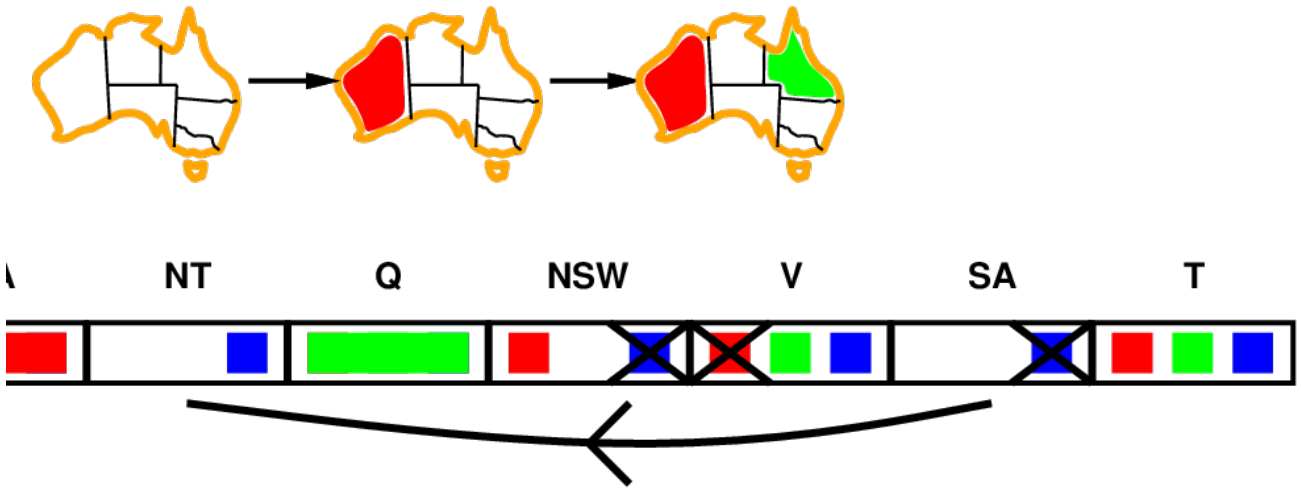


If X loses a value, neighbors of X need to be rechecked

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 y



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

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm

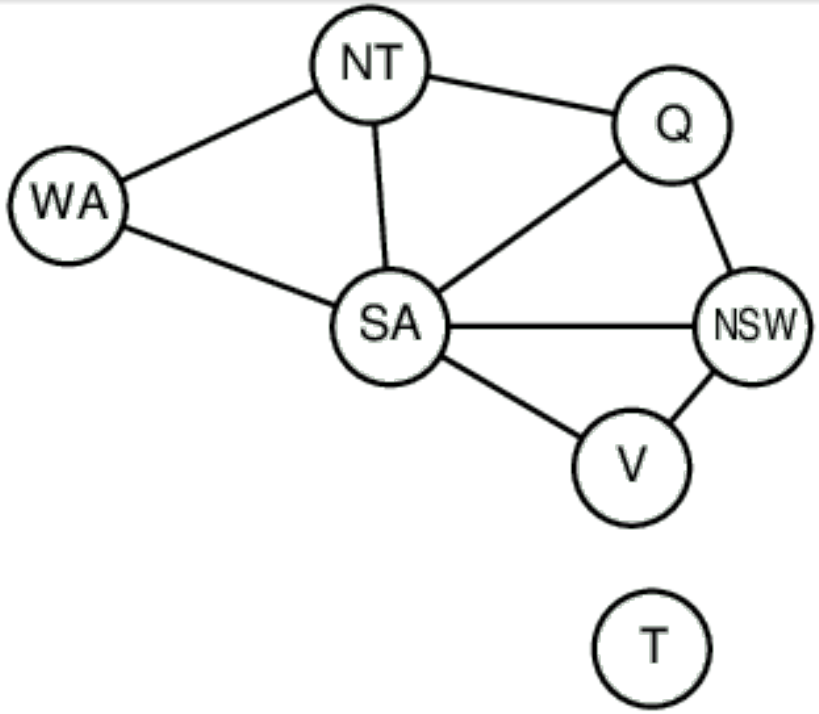
function AC-3(*csp*) **returns** the CSP, possibly with reduced domains
  inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  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*

Problem structure



Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

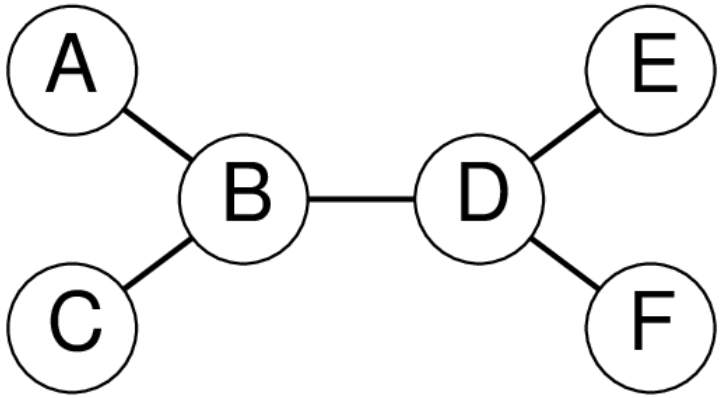
Problem structure contd.

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
$4 \cdot 2^{20}$ = 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 $O(n\,d^2)$ time

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