Artificial Intelligence

Constraint Satisfaction Problems

Marc Toussaint
University of Stuttgart
Winter 2019/20

(slides based on Stuart Russell's AI course)

Motivation:

Here is a little cut in the lecture series. Instead of focussing on sequential decision problems we turn to problems where there exist many coupled variables. The problem is to find values (or, later, probability distributions) for these variables that are consistent with their coupling. This is such a generic problem setting that it applies to many problems, not only map colouring and sudoku. In fact, many computational problems can be reduced to Constraint Satisfaction Problems or their probabilistic analogue, Probabilistic Graphical Models. This also includes sequential decision problems, as I mentioned in some extra lecture. Further, the methods used to solve CSPs are very closely related to descrete optimization.

From my perspective, the main motivation to introduce CSPs is as a precursor to introduce their probabilistic version, graphical models. These are a central language to formulate probabilistic models in Machine Learning, Robotics, AI, etc. Markov Decision Processes, Hidden Markov Models, and many other problem settings we can't discuss in this lecture are special cases of graphical models. In both settings, CSPs and graphical models, the core is to understand what it means to do inference. Tree search, constraint propagation and belief propagation are the most important methods in this context.

In this lecture we first define the CSP problem, then introduce basic methods: sequential assignment with some heuristics, backtracking, and constraint propagation.

Problem Formulation & Examples

Inference

- The core topic of the following lectures is

  Inference: *Given some pieces of information on some things (observed variabes, prior, knowledge base) what is the implication (the implied information, the posterior) on other things (non-observed variables, sentence)*

- Decision-Making and Learning can be viewed as Inference:
  - given pieces of information: about the world/game, collected data, assumed model class, *prior* over model parameters
  - make decisions about actions, classifier, model parameters, etc
- In this lecture:
  - "Deterministic" inference in CSPs
  - Probabilistic inference in graphical models variabels)
  - Logic inference in propositional & FO logic

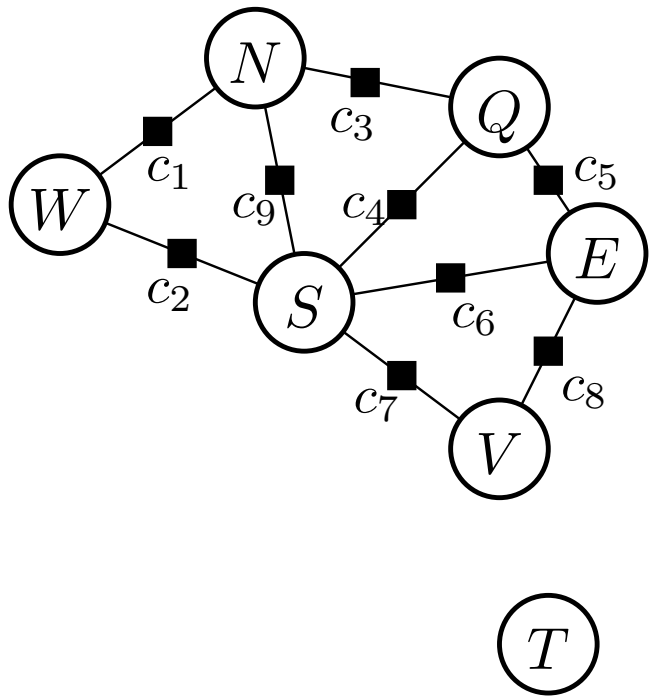
Constraint satisfaction problems (CSPs)

- In previous lectures we considered sequential decision problems CSPs are *not* sequential decision problems. However, the basic methods address them by sequentially making partial decisions

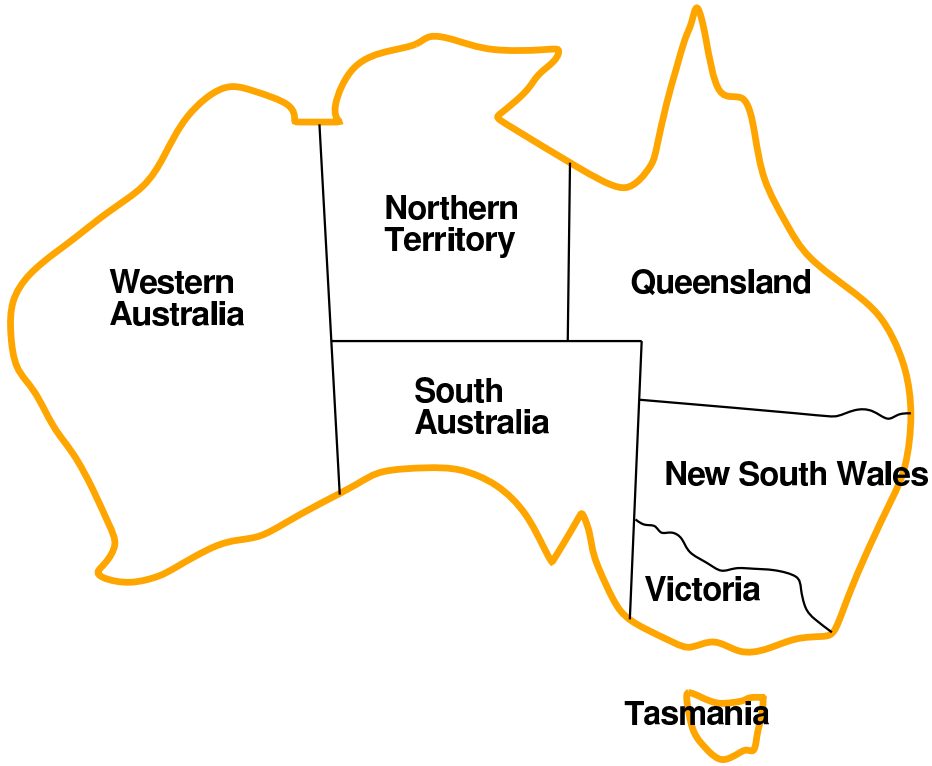
- CSP:
  - We have n variables x_i, each with domain D_i, $x_i \in D_i$
  - We have K constraints C_k, each of which determines the feasible configurations of a subset of variables
  - The goal is to find a configuration $X = (X_1, .., X_n)$ of all variables that satisfies all constraints
- Formally, a constraint is defined as $C_k = (I_k, c_k)$, where $I_k \subseteq \{1, .., n\}$ determines the subset of variables, and $c_k : D_{I_k} \to \{0, 1\}$ determines whether a configuration $x_{I_k} \in D_{I_k}$ of this subset of variables is feasible

Constraint graph

- A CSP corresponds to a *constraint graph*:
  A constraint graph is a *bi-partite graph* where nodes are variables and boxes are constraints
  Constraints may constrain several (or one) variables ($|I_k| \neq 2$)

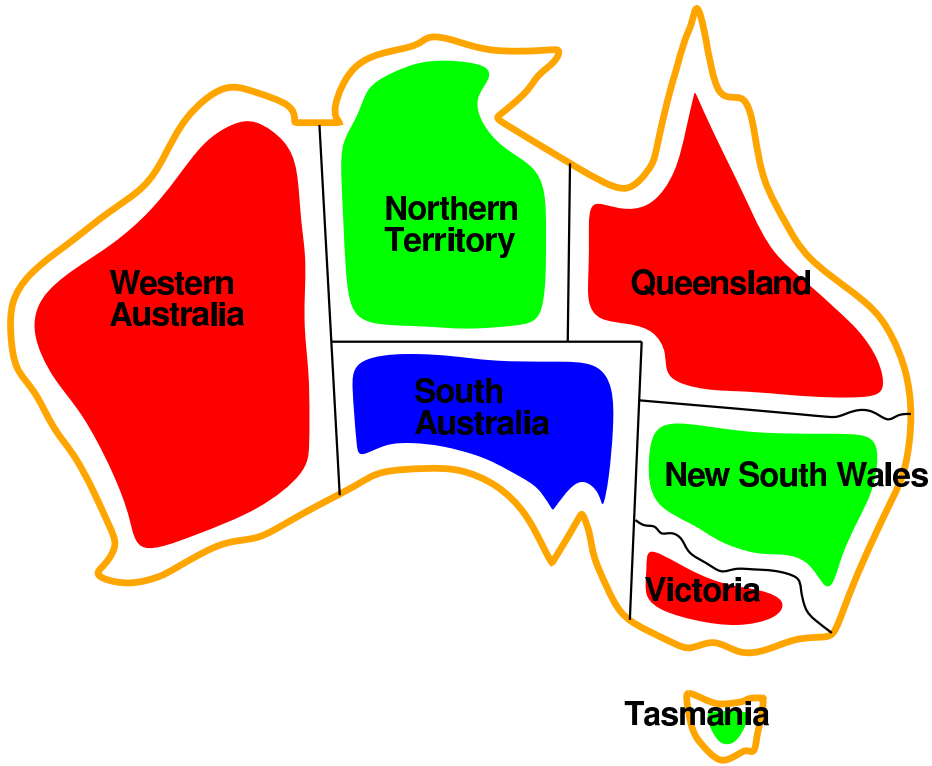


Example: Map-Coloring



- Variables: W, N, Q, E, V, S, T (E = New South Wales)
- Domains: $D_i = \{red, green, blue\}$ for all variables
- Constraints: adjacent regions must have different colors, e.g., $W \neq N$, $(W, N) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.



- *Solutions* are assignments satisfying all constraints, e.g.,

$\{W = red, N = green, Q = red, E = green, V = red, S = blue, T = green\}$

Varieties of CSPs

- Discrete variables: finite domains; each D_i of size $|D_i| = d \Rightarrow O(d^n)$ complete assignments
  - e.g., Boolean CSPs, incl. Boolean satisfiability infinite domains (integers, strings, etc.)
  - e.g., job scheduling, variables are start/end days for each job
  - constraints solvable, nonlinear undecidable
- Continuous variables
  - e.g., start/end times for Hubble Telescope observations
  - linear constraints solvable in poly time by LP methods
- Real-world examples
  - Assignment problems, e.g. who teaches what class?
  - Timetabling problems, e.g. which class is offered when and where?
  - Hardware configuration
  - Transportation/Factory scheduling

Varieties of constraints

Unary constraints involve a single variable, $|I_k| = 1$

e.g., $S \neq green$

Pair-wise constraints involve pairs of variables, $|I_k| = 2$

e.g., $S \neq W$

Higher-order constraints involve 3 or more variables, $|I_k| > 2$

e.g., Sudoku

Methods for solving CSPs

Sequential assignment approach

- Let's start with the straightforward but inefficient approach, then fix it

- We assign variable values **sequentially**; the solver state is defined by the values assigned so far. The solver's decision process is defined by:
  - Initial state: the empty assignment, $\{\}$
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment $\Rightarrow$ fail if no feasible assignments (not fixable!)
  - Goal test: the current assignment is complete

- Every solution appears at depth n with n variables $\Rightarrow$ use depth-first search
- $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!

Backtracking sequential assignment

- Two variable assignment decisions are commutative, i.e.,
  $[W = red$ then $N = green]$ same as $[N = green$ then $W = red]$
- We can fix a single next variable to assign a value to at each node!
  This drastically reduces the branching factor of the search tree.
- This does not compromise completeness (ability to find the solution)
  $\Rightarrow$ $b = d$ and there are d^n leaves

- *Depth-first search for CSPs with single-variable assignments is called* ***backtracking*** *search*
- Backtracking search is the basic uninformed algorithm for CSPs

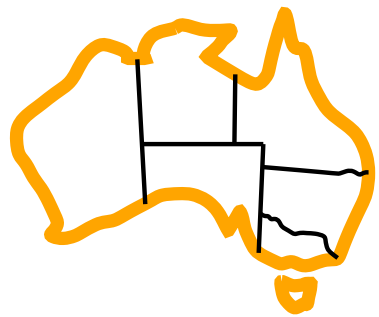
  Can solve n-queens for $n \approx 25$

Backtracking search

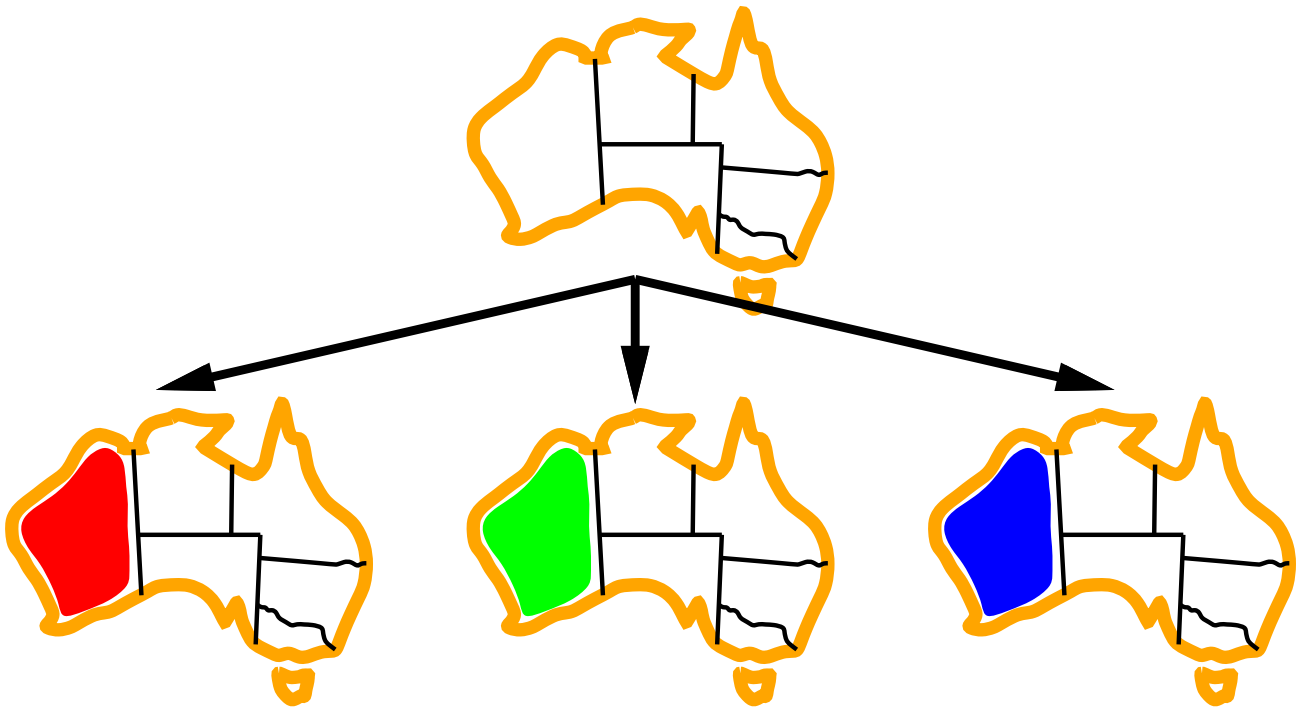
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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 ORDERED-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
```

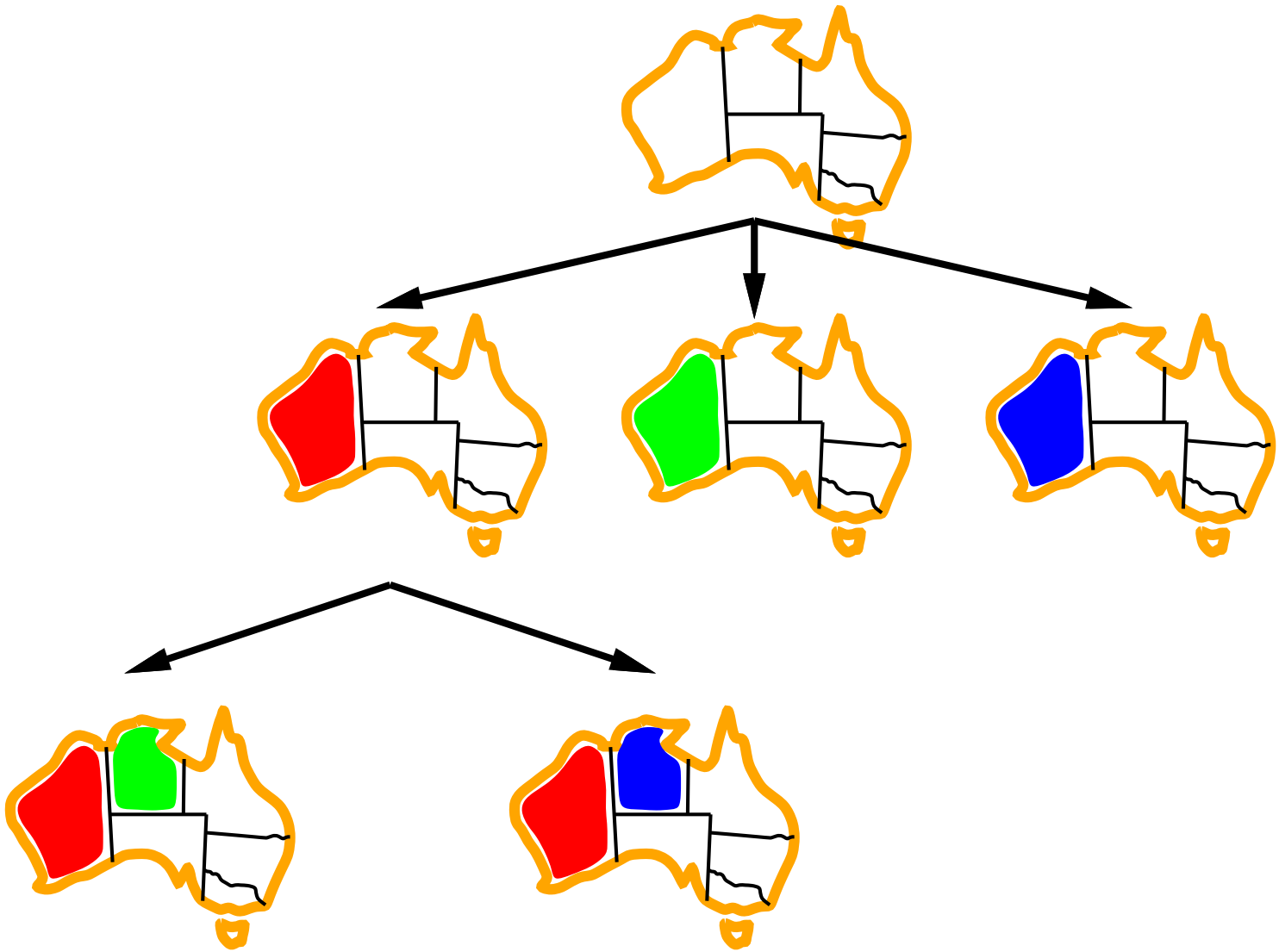
Backtracking example



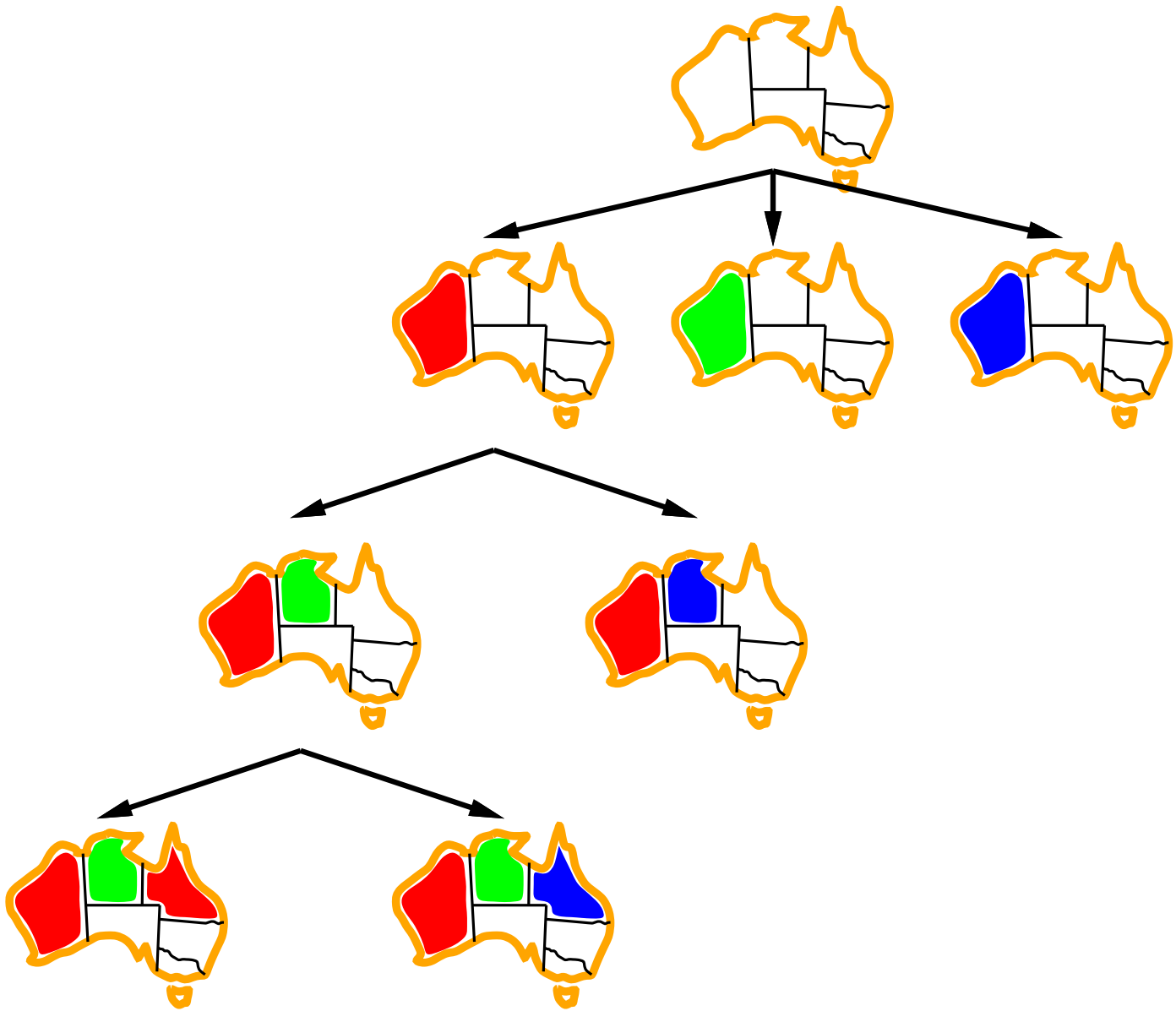
Backtracking example



Backtracking example



Backtracking example



Improving backtracking efficiency

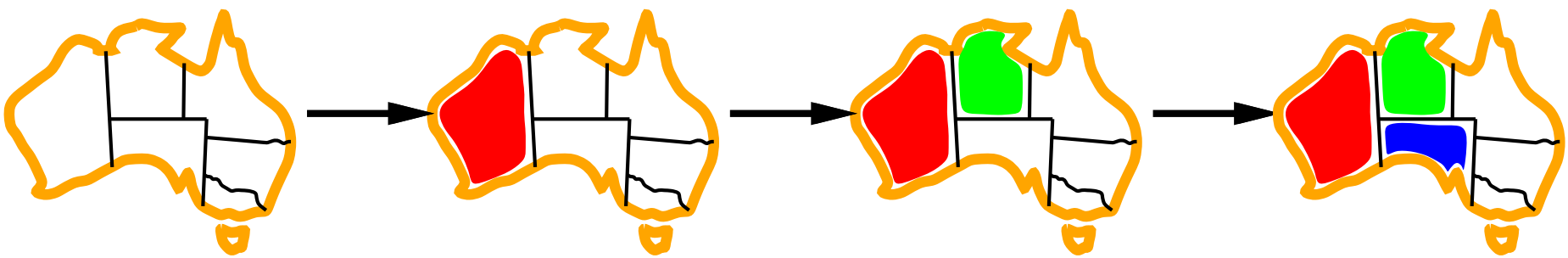
Simple heuristics can give huge gains in speed:

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?

Variable order: Minimum remaining values

Minimum remaining values (MRV):

choose the variable with the fewest legal values

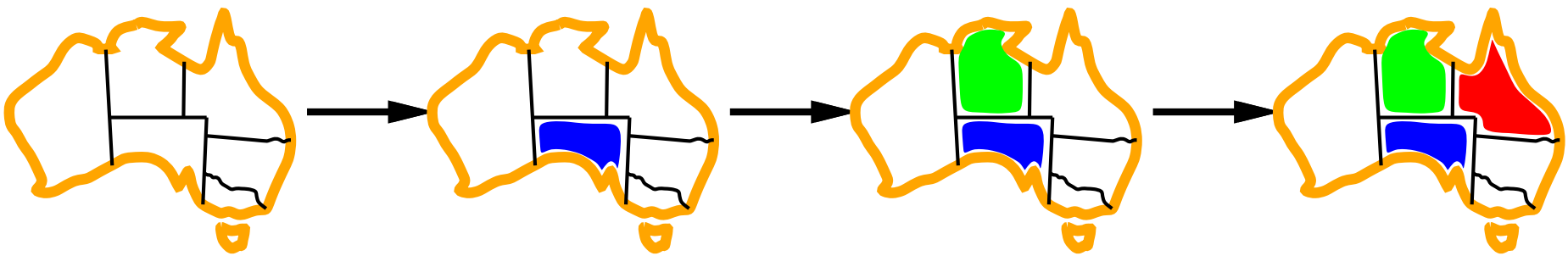


Variable order: Degree heuristic

Tie-breaker among MRV variables

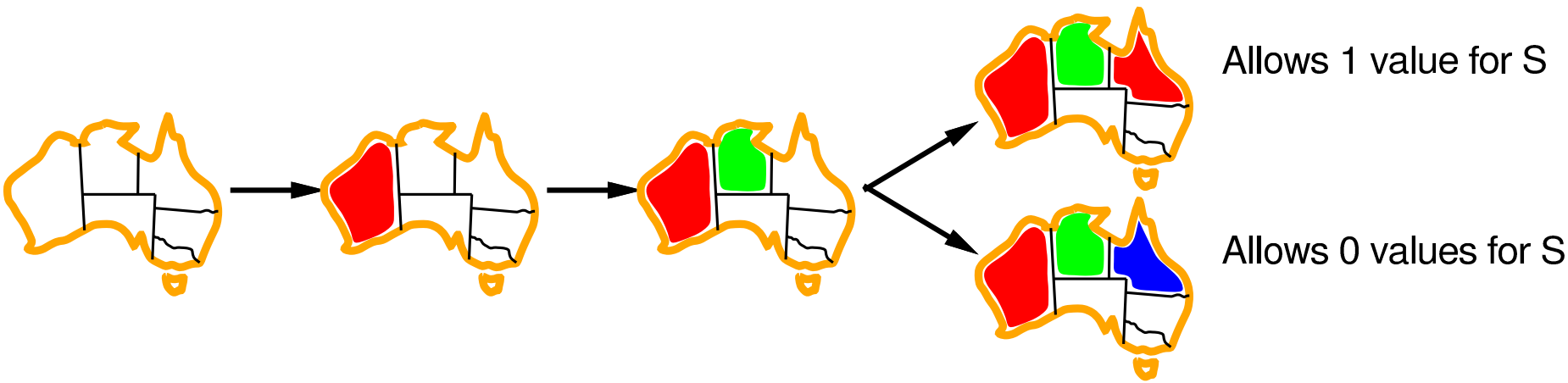
Degree heuristic:

choose the variable with the most constraints on remaining variables



Value order: 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

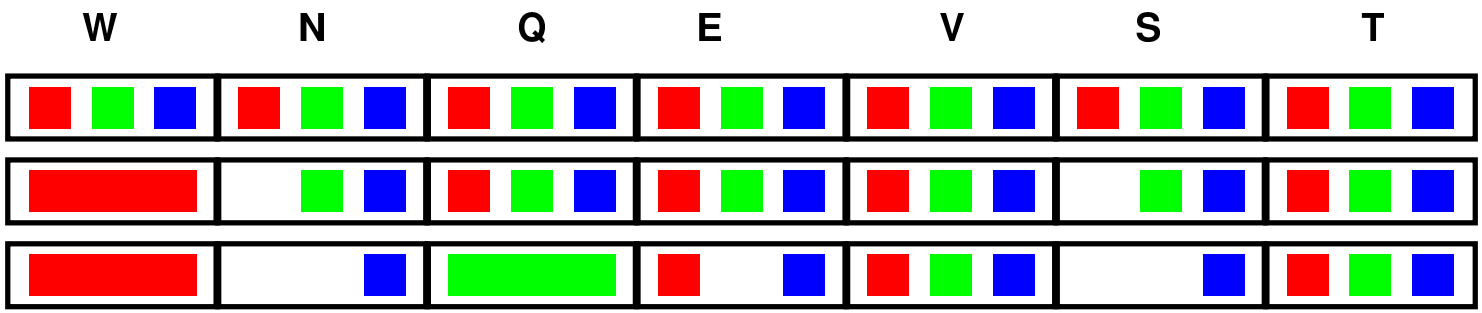
Constraint propagation

- After each decision (assigning a value to one variable) we can compute what are the remaining feasible values for all variables.
- Initially, every variable has the full domain D_i. Constraint propagation reduces these domains, deleting entries that are inconsistent with the new decision.
- These dependencies are recursive: Deleting a value from the domain of one variable might imply infeasibility of some value of another variable $\rightarrow$ contraint *propagation*. We update domains until they're all consistent with the constraints.

This is Inference

Constraint propagation

- Example: quick failure detection after 2 decisions



N and S cannot both be blue!

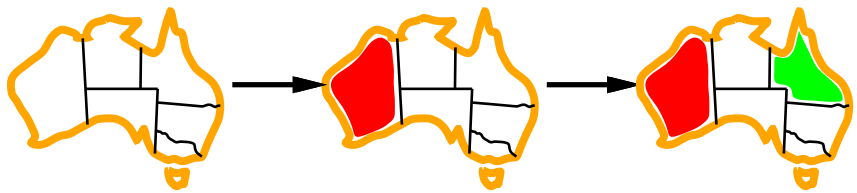
- Constraint Propagation: propagate the implied constraints serveral steps to reduce remaining domains and detect failures early.

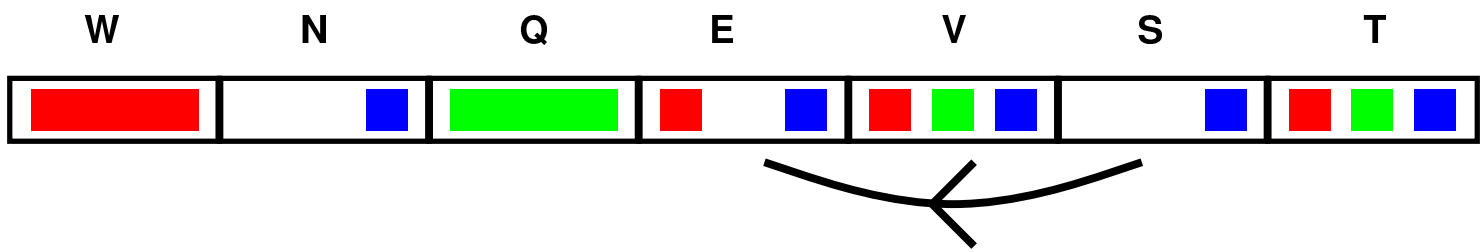
Constraint propagation

- Constraint propagation generally loops through the set of constraint, considers each constraint *separately*, and deletes inconsistent values from its adjacent domains.
- As it considers constraints separately, it does not compute a final solution, as backtracking search does.

Arc consistency (=constraint propagation for pair-wise constraints)

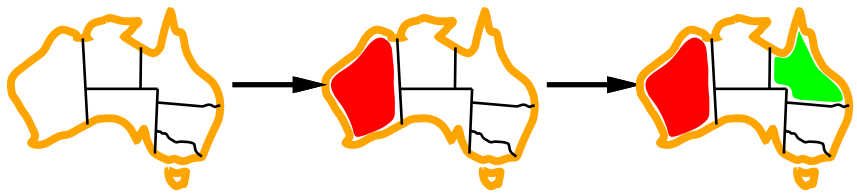
- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
  for *every* value x of X there is *some* allowed y

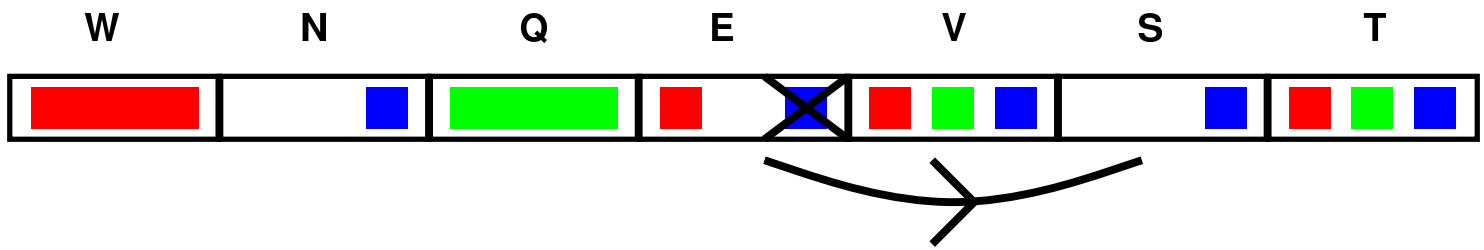




Arc consistency (=constraint propagation for pair-wise constraints)

- 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

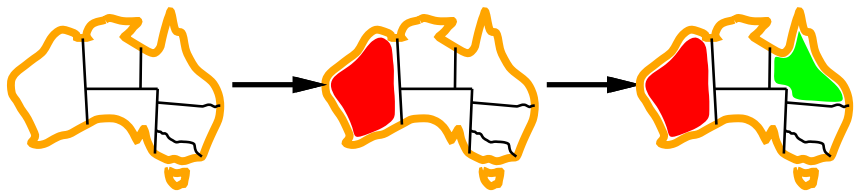


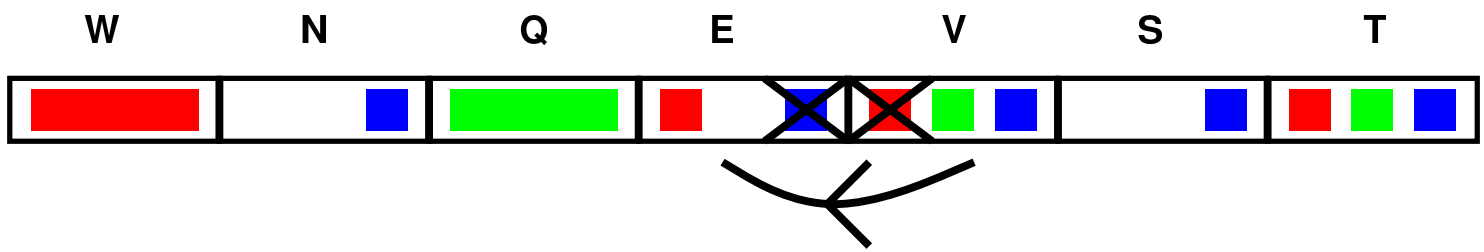


- If X loses a value, neighbors of X need to be rechecked

Arc consistency (=constraint propagation for pair-wise constraints)

- Simplest form of propagation makes each arc consistent
- $X \to Y$ is consistent iff
  for *every* value x of X there is *some* allowed y

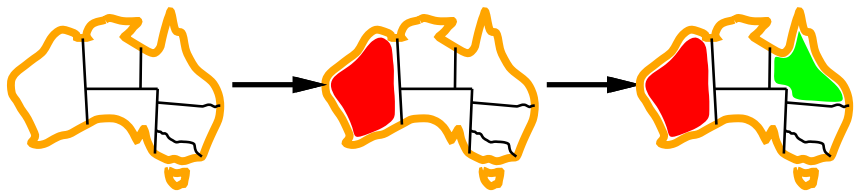


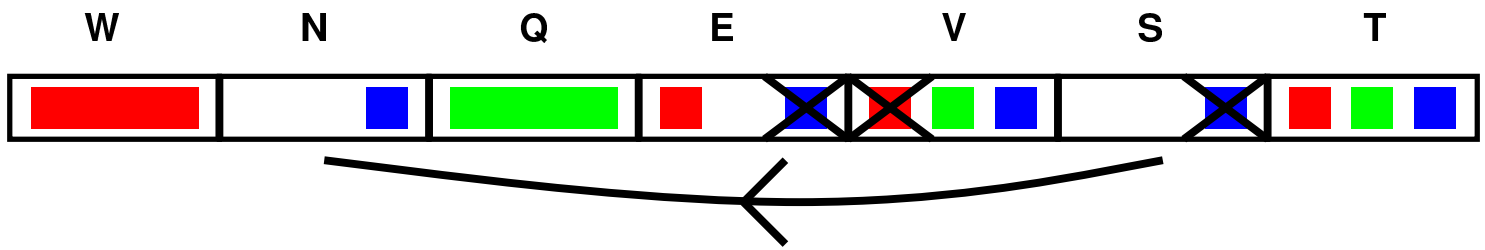


- If X loses a value, neighbors of X need to be rechecked
  Arc consistency detects failure earlier than forward checking
  Can be run as a preprocessor or after each assignment

Arc consistency (=constraint propagation for pair-wise constraints)

- 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 pair-wise 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 DOM$[X_i]$ changed
  changed $\leftarrow$ *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$[X_i]$; *changed* $\leftarrow$ *true*
  return *changed*

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$

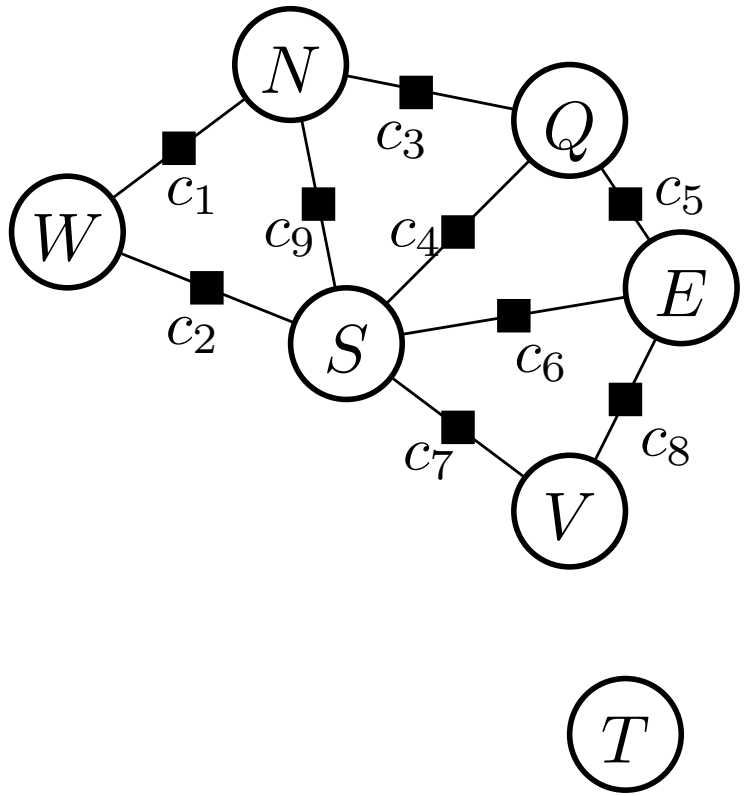
Constraint propagation

- Very closely related to message passing in probabilistic models

- In practice: design approximate constraint propagation for specific problem

  E.g.: Sudoku: If X_i is assigned, delete this value from all peers

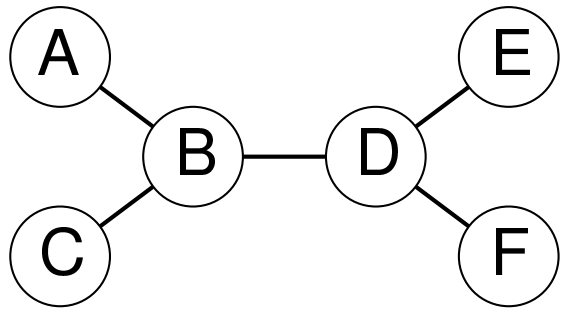
Problem structure



Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

Tree-structured CSPs

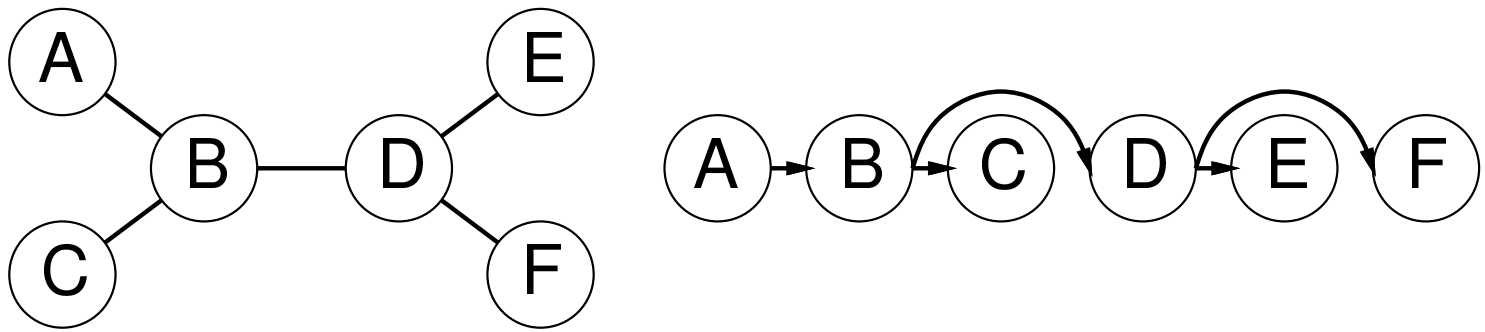


Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time
Compare to general CSPs, where worst-case time is $O(d^n)$

This property also applies to logical and probabilistic reasoning!

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

$$\textsc{RemoveInconsistent}(Parent(X_j), X_j)$$

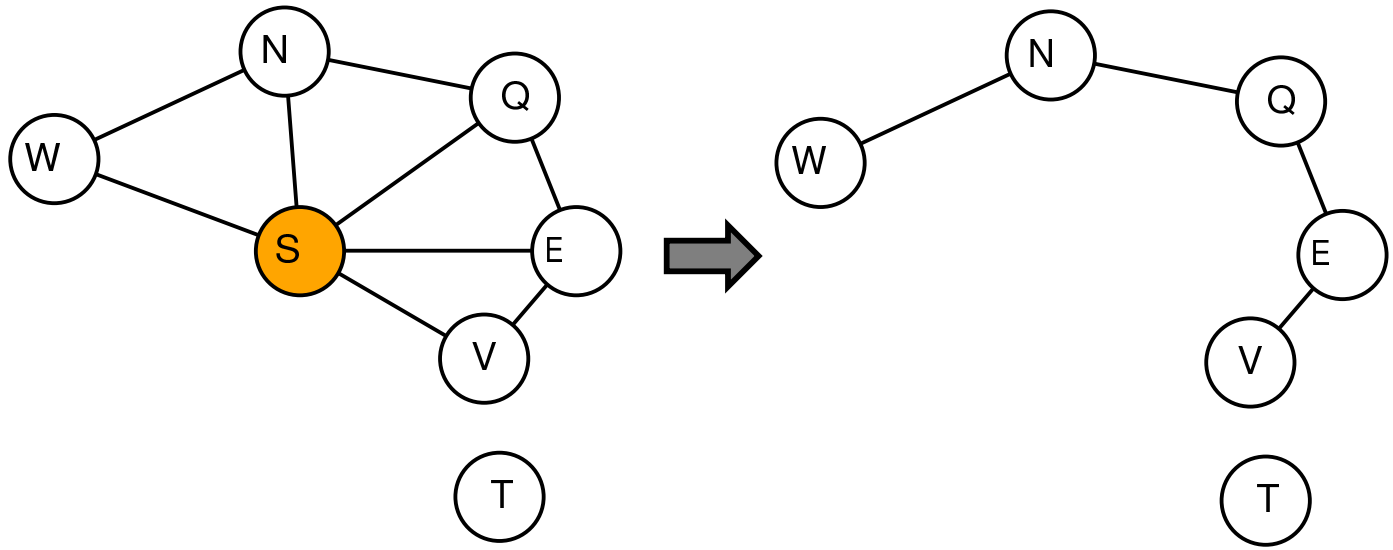
   This is *backward constraint propagation*

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

   This is *forward sequential assignment* (trivial backtracking)

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 $\Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

Summary

- CSPs are a fundamental kind of problem:
  finding a feasible configuration of n variables
  the set of constraints defines the (graph) structure of the problem
- Sequential assignment approach
  Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- 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
  If after assigning some variables, the remaining structure is a tree
  $\rightarrow$ linear time feasibility check by tree CSP