9

Smart Searching Using Constraints

Claudia Chirita

School of Informatics, University of Edinburgh

THE UNIVERSITY *of* EDINBURGH
informatics

Based on slides by: Jacques Fleuriot, Michael Rovatsos, Michael Herrmann, Vaishak Belle

9.a

Constraint Satisfaction Problems

CONSTRAINT SATISFACTION PROBLEMS (CSP)

STANDARD SEARCH PROBLEM

state a *black box* – any data structure that supports a successor function, a heuristic function and a goal test

CSP

state a set of **variables**, each of which has a value

solution when each variable has a value that satisfies all its constraints

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

IDEA eliminate large portions of the search space by identifying variable/value combinations that violate the constraints.

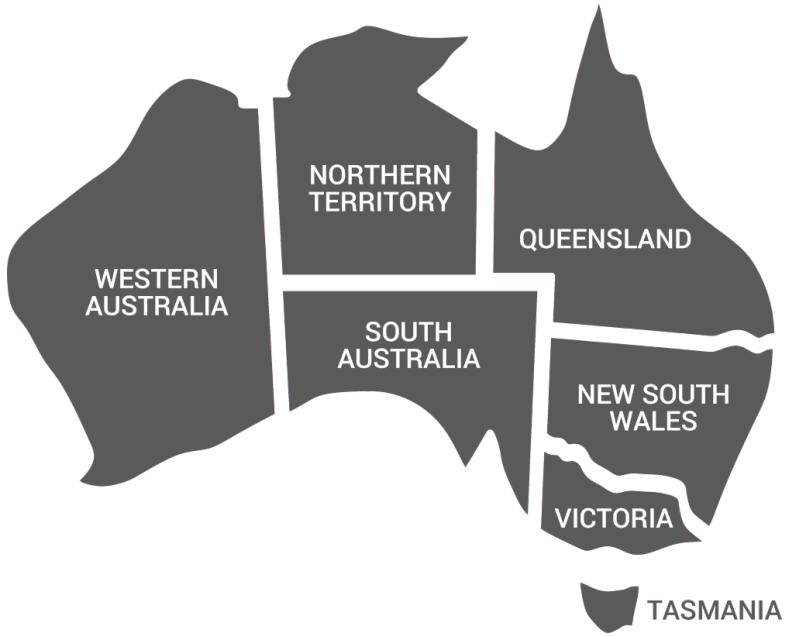
CONSTRAINT SATISFACTION PROBLEMS (CSP)

CSP consist of:

- a set $X = \{X_1, ..., X_n\}$ of **variables**
- a set $D = \{D_1, ..., D_n\}$ of **domains**;
  each domain D_i is a set of possible values for variable X_i
- a set C of **constraints** that specify accepted combinations of values.

  A constraint $c \in C$ consists of
  - a **scope** – tuple of variables involved in the constraint
  - a **relation** that defines the values that the variables can take

EXAMPLE · MAP-COLOURING



Variables $\{\mathrm{WA}, \mathrm{NT}, \mathrm{Q}, \mathrm{NSW}, \mathrm{V}, \mathrm{SA}, \mathrm{T}\}$

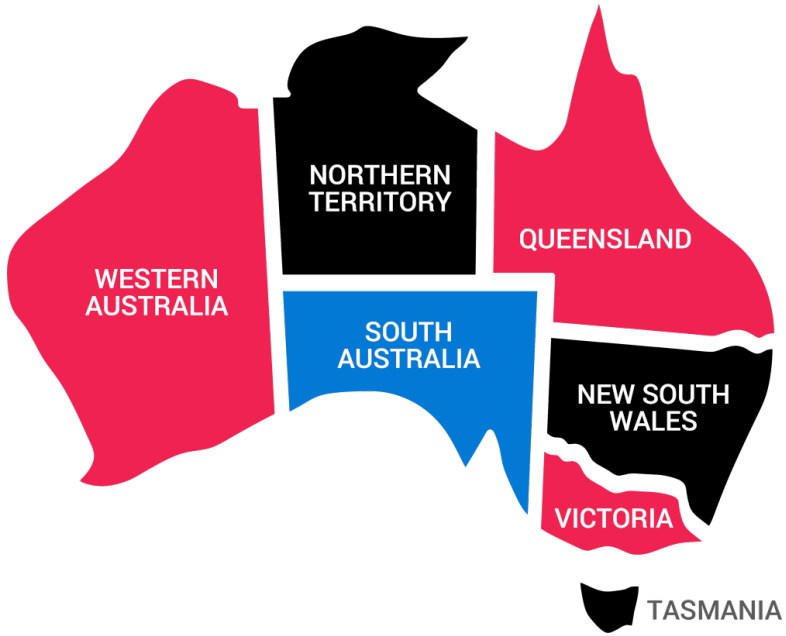
Domains $D_i = \{\text{red}, \text{black}, \text{blue}\}$

Constraints adjacent regions must have different colours

e.g. $\mathrm{WA} \neq \mathrm{NT}$ or

$(\mathrm{WA}, \mathrm{NT}) \in \{(\text{red}, \text{black}), (\text{red}, \text{blue}), (\text{black}, \text{red}),$
$(\text{black}, \text{blue}), (\text{blue}, \text{red}), (\text{blue}, \text{black})\}$

EXAMPLE · MAP-COLOURING



Solutions are complete and consistent assignments.

e.g. $\mathrm{WA} \mapsto \mathrm{red}$, $\mathrm{NT} \mapsto \mathrm{black}$, $\mathrm{Q} \mapsto \mathrm{red}$, $\mathrm{NSW} \mapsto \mathrm{black}$,
$\mathrm{V} \mapsto \mathrm{red}$, $\mathrm{SA} \mapsto \mathrm{blue}$, $\mathrm{T} \mapsto \mathrm{black}$.

CONSTRAINT GRAPHS

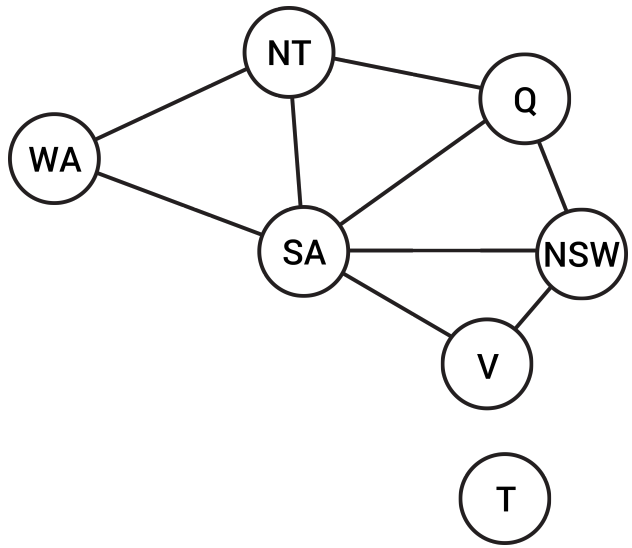
BINARY CSP

Each constraint relates two variables.

CONSTRAINT GRAPH

nodes are variables

arcs (edges) represent constraints



CSP · VARIETIES

A. DISCRETE VARIABLES

finite domains

n variables, domain size d, $O(d^n)$ complete assignments

e.g. Boolean CSPs, including Boolean satisfiability (NP-complete)

infinite domains

integers, strings, etc.

e.g. job scheduling

- variables are start/end days for each job
- we need a constraint language to express
- $\text{StartJob}_1 + 5 \leqslant \text{StartJob}_3$

B. CONTINUOUS VARIABLES

e.g. start/end times for Hubble Space Telescope observations

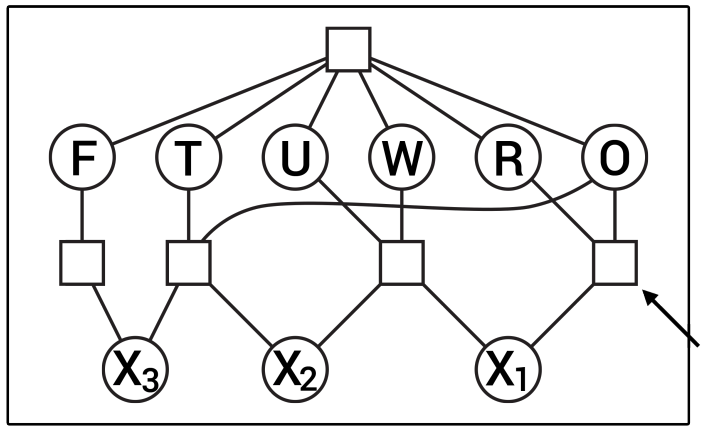
linear constraints solvable in polynomial time by linear programming

VARIETIES OF CONSTRAINTS

UNARY CONSTRAINTS	involve a single variable e.g. SA $\neq$ black
BINARY CONSTRAINTS	involve pairs of variables e.g. SA $\neq$ WA
HIGHER-ORDER	involve 3 or more variables e.g. crypt-arithmetic column constraints
GLOBAL CONSTRAINTS	involve an arbitrary number of variables

EXAMPLE · CRYPT-ARITHMETIC

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



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 $\text{Alldiff}(F, T, U, W, R, O)$ ⟵ **global constraint**

$$O + O = R + 10 \cdot X_1$$

$$X_1 + W + W = U + 10 \cdot X_2$$

$$X_2 + T + T = O + 10 \cdot X_3$$

$$X_3 = F, \quad T \neq 0, \quad F \neq 0$$

EXAMPLES · REAL-WORLD CSP

Assignment problems

e.g. who teaches what class

Timetabling problems

e.g. which class is offered when and where

Transportation scheduling

Factory scheduling

Many real-world problems involve real-valued variables.

9.b

Backtracking search

STANDARD SEARCH FORMULATION

Let's start with the straightforward approach, then adapt it.

States are defined by the values assigned so far.

Initial state	the empty assignment $\{\}$
Successor function	assign a value to an unassigned variable that does not conflict with the current assignment $\longrightarrow$ *fail* if no legal assignments
Goal test	the current assignment is complete

For CSPs with with n variables, any solution appears at depth n

$\Rightarrow$ use depth-first search.

BACKTRACKING SEARCH

Variable assignments are **commutative**.

e.g. $[\text{WA} \mapsto \text{red}$ then $\text{NT} \mapsto \text{black}]$

is the same as

$[\text{NT} \mapsto \text{black}$ then $\text{WA} \mapsto \text{red}]$

We only need to consider assignments to a single variable at each node.
Thus, $b = d$, and there are d^n leaves.

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

Backtracking search: the basic uninformed algorithm for CSP.

Can solve the n-queens problem for $n \approx 25$.

BACKTRACKING SEARCH

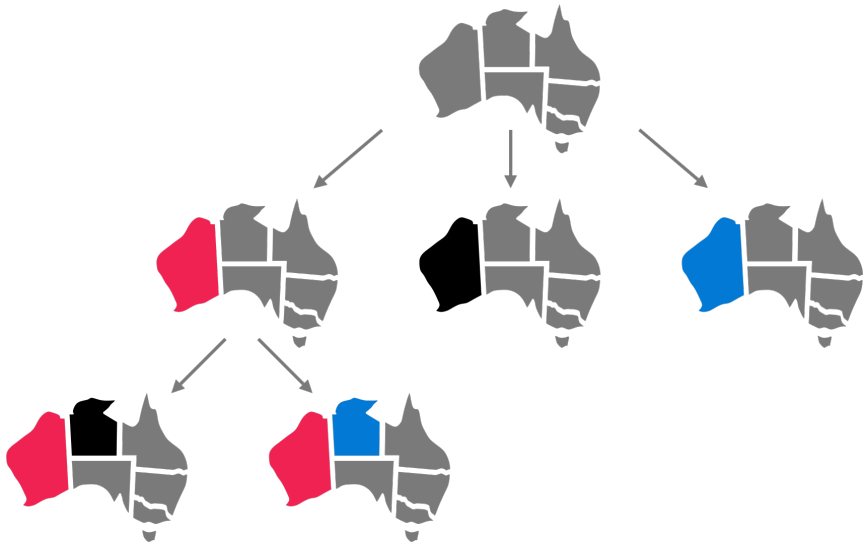
```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
    remove {var = value} and inferences from assignment
  return failure
```

Optional; can be used to impose arc-consistency (more on this later)

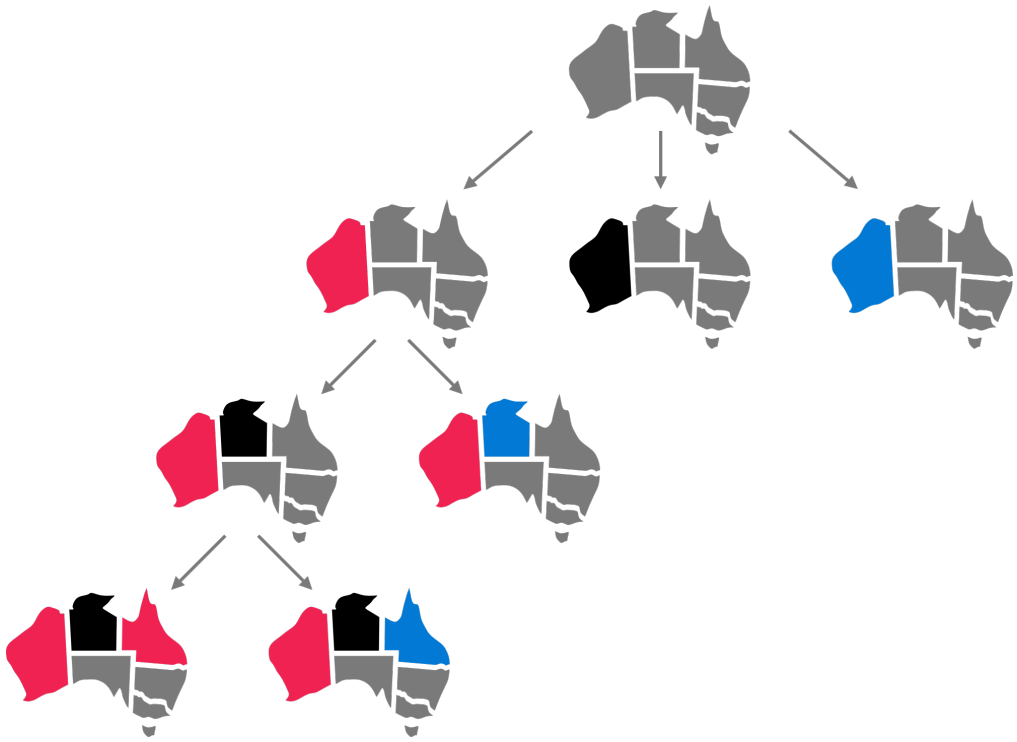
BACKTRACKING · EXAMPLE

BACKTRACKING · EXAMPLE

BACKTRACKING · EXAMPLE



BACKTRACKING · EXAMPLE



9.c

Efficiency matters

IMPROVING BACKTRACKING EFFICIENCY

General-purpose methods can give huge gains in speed.

Which **variable** should be assigned next?

SELECT-UNASSIGNED-VARIABLE

Then, in what order should its **values** be tried?

ORDER-DOMAIN-VALUES

What inferences should be performed at each search step?

INFERENCE

Can we detect inevitable failure early?

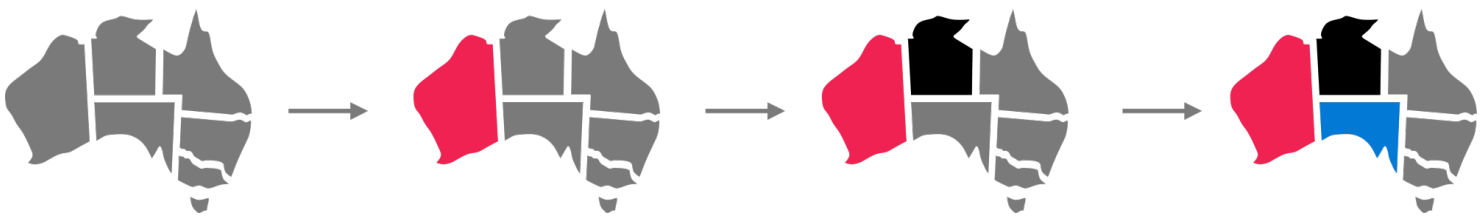
MOST CONSTRAINED VARIABLE

var $\leftarrow$ SELECT-UNASSIGNED-VARIABLE(csp)

Most constrained variable heuristic

choose the variable with the fewest legal values

a.k.a. minimum-remaining-values (MRV) heuristic



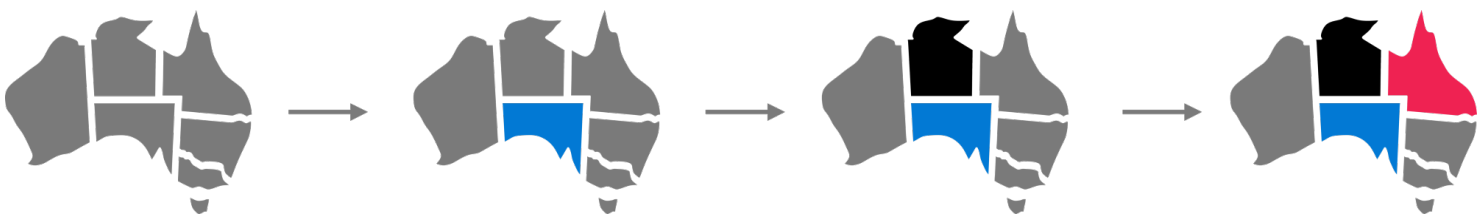
MOST CONSTRAINING VARIABLE

Tie-breaker among most constrained variables.

Most constraining variable heuristic

choose the variable with the most constraints on remaining variables, thus reducing branching

a.k.a. degree heuristic



LEAST CONSTRAINING VALUE

ORDER-DOMAIN-VALUES

Given a variable, choose the **least constraining value**

the one that rules out the fewest values in the remaining variables

Combining these heuristics: n-queens feasible for $n \approx 1000$.

INFERENCE · FORWARD CHECKING

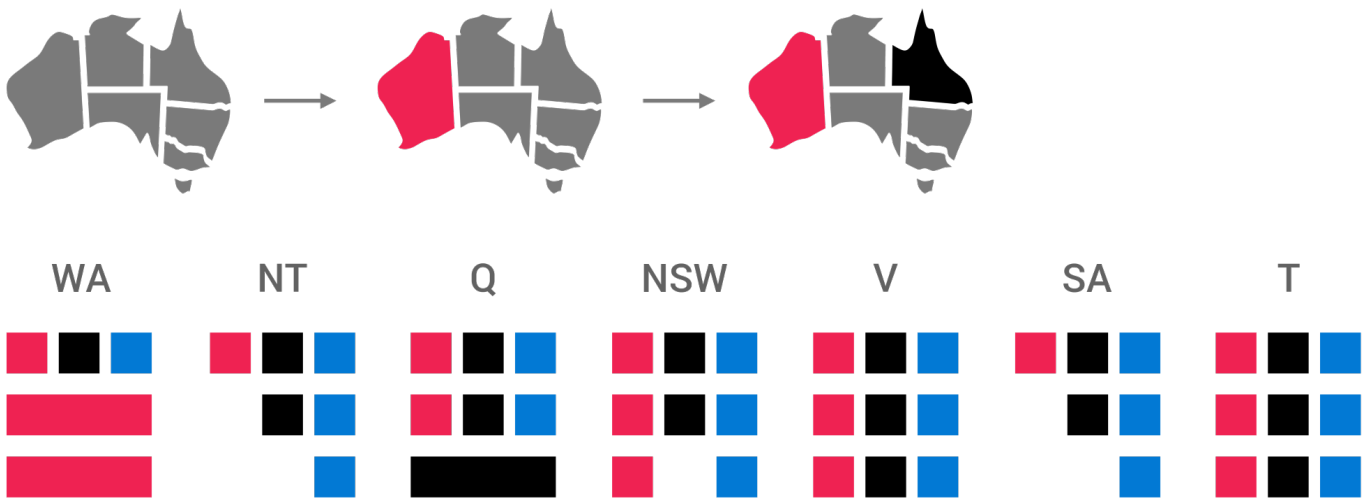
IDEA Keep track of remaining legal values for unassigned variables.
Terminate the search when a variable has no more legal values.

INFERENCE · FORWARD CHECKING

IDEA Keep track of remaining legal values for unassigned variables.
Terminate the search when a variable has no more legal values.

INFERENCE · FORWARD CHECKING

IDEA Keep track of remaining legal values for unassigned variables.
Terminate the search when a variable has no more legal values.

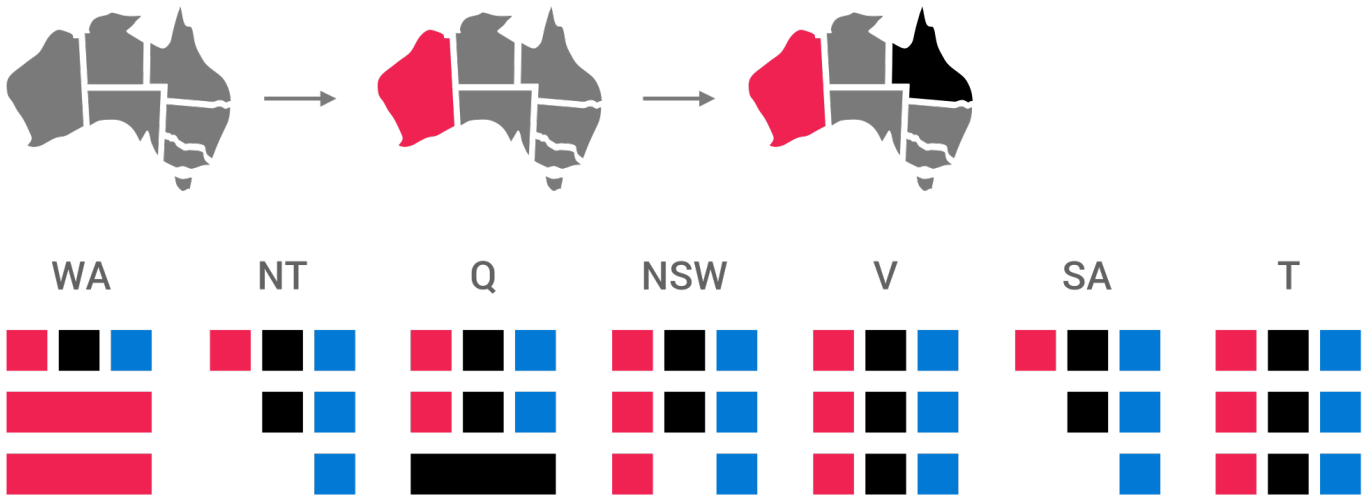


INFERENCE · FORWARD CHECKING

IDEA Keep track of remaining legal values for unassigned variables.
Terminate the search when a variable has no more 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 in the domain of X

there is **some** allowed value y in the domain of Y

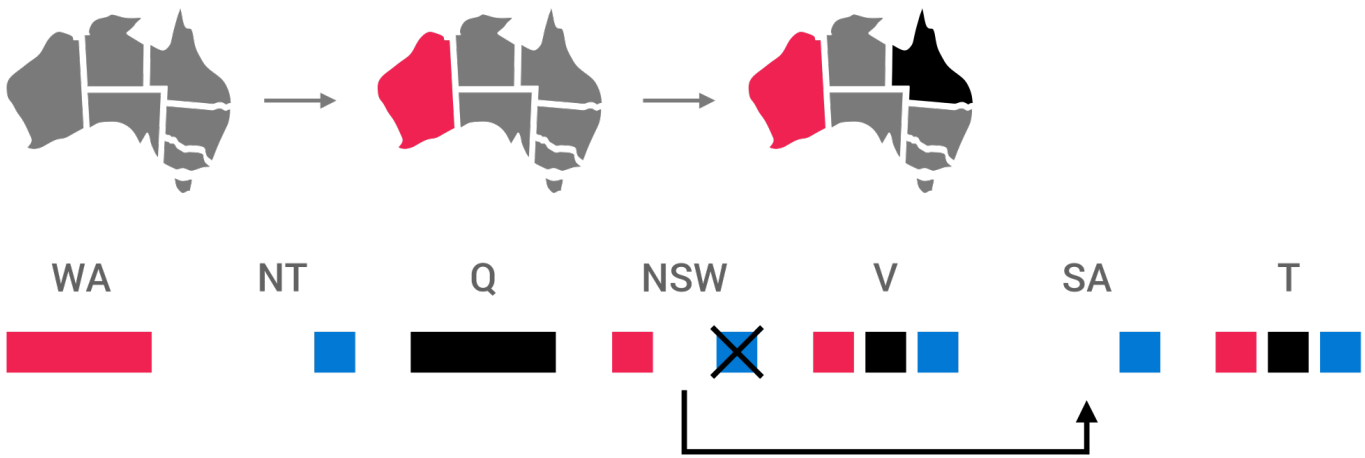
ARC CONSISTENCY

Simplest form of propagation makes each arc **consistent**.

$X \rightarrow Y$ is consistent iff

for **every** value x in the domain of X

there is **some** allowed value y in the domain of Y



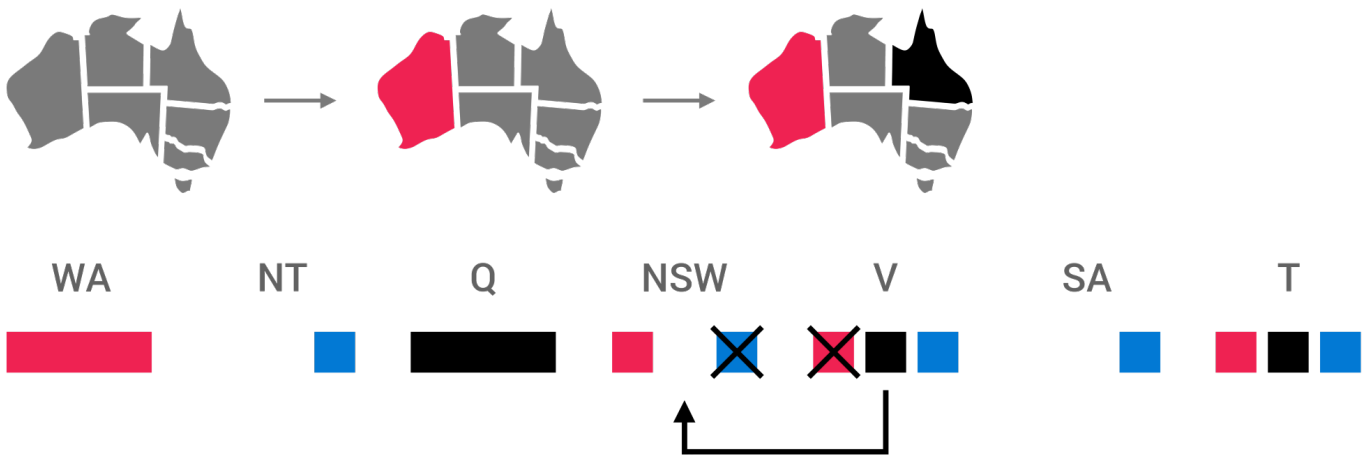
ARC CONSISTENCY

Simplest form of propagation makes each arc **consistent**.

$X \rightarrow Y$ is consistent iff

for **every** value x in the domain of X

there is **some** allowed value y in the domain of Y



If X loses a value, its neighbours need to be rechecked.

ARC CONSISTENCY

Simplest form of propagation makes each arc **consistent**.

$X \rightarrow Y$ is consistent iff

for **every** value x in the domain of X

there is **some** allowed value y in the domain of Y

If X loses a value, its neighbours need to be rechecked.

Detects failure earlier than forward checking.

Can be run as a preprocessor or after each assignment.

ARC CONSISTENCY · ALGORITHM · AC-3

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components $(X,\ D,\ C)$
  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 REVISE(csp, X_i, X_j) **then** ← Make X_i arc-consistent with respect to X_j
      if size of $D_i = 0$ **then return** $false$ ← No consistent value left for X_i so fail
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to $queue$ ← Since revision occurred, add all neighbours of X_j for consideration (or reconsideration)
  return $true$

---

function REVISE(csp, X_i, X_j) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      $revised \leftarrow true$
  return $revised$

d – maximum size of the domains

c – number of binary constraints

Time complexity: $\mathrm{O}(\mathrm{c}\mathrm{d}^3)$ Space complexity: $\mathrm{O}(\mathrm{c})$

CSP IN SHORT

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.

Forward checking prevents assignments that are certain to lead to later failure.

Constraint propagation does additional work to limit values and detect inconsistencies.