CS3011: Introduction to Artificial Intelligence

Constraint Satisfaction Problems (CSP)

Instructors: Dr. Durgesh Singh

CSE Discipline, PDPM IIITDM, Jabalpur -482005

Definition

- A **constraint satisfaction problem** (CSP) consists of
  - a set of variables,
  - a domain for each variable, and
  - a set of constraints.
- The aim is to choose a value for each variable so that the resulting possible world satisfies all the constraints.

Constraint Satisfaction Problems

A constraint satisfaction problem consists of three components, X,D, and C:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

- **A solution is an assignment of a value to each variables X_i such that every constraints satisfied.**

CSP Examples: Map Coloring

Map of Australia: We are given the task of coloring each region either red, green, or blue in such a way that no two neighboring regions have the same color.

Example: Map Coloring

- Variables: WA, NT, Q, NSW, V, SA, T

- Domains: $D = \{\text{red}, \text{green}, \text{blue}\}$

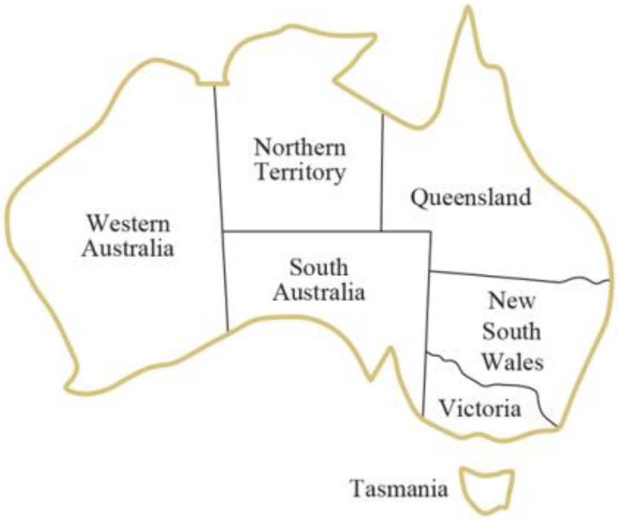
- Constraints: adjacent regions must have different colors

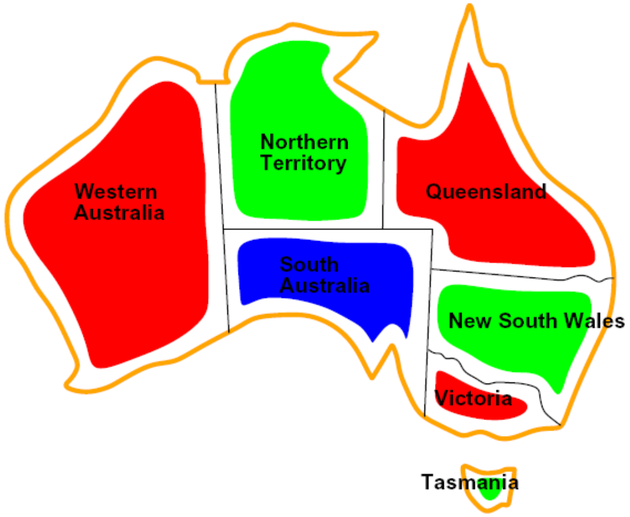
  Implicit: $WA \neq NT$

  Explicit: $(WA, NT) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), \ldots\}$

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

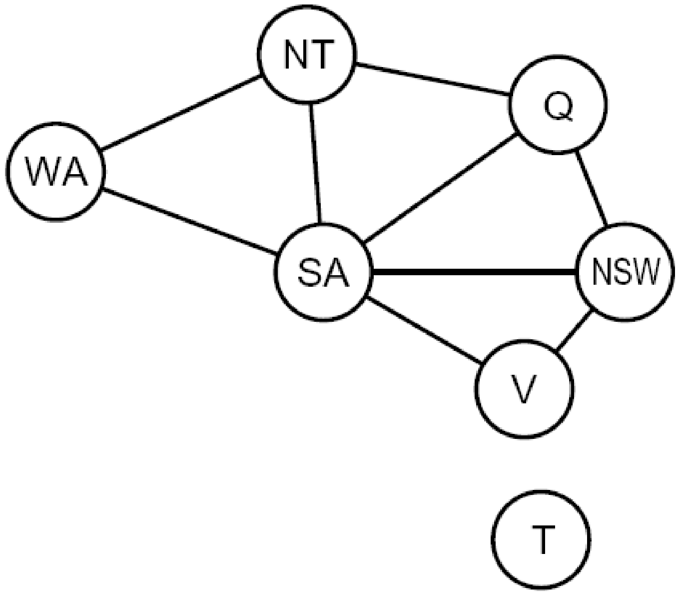
  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}





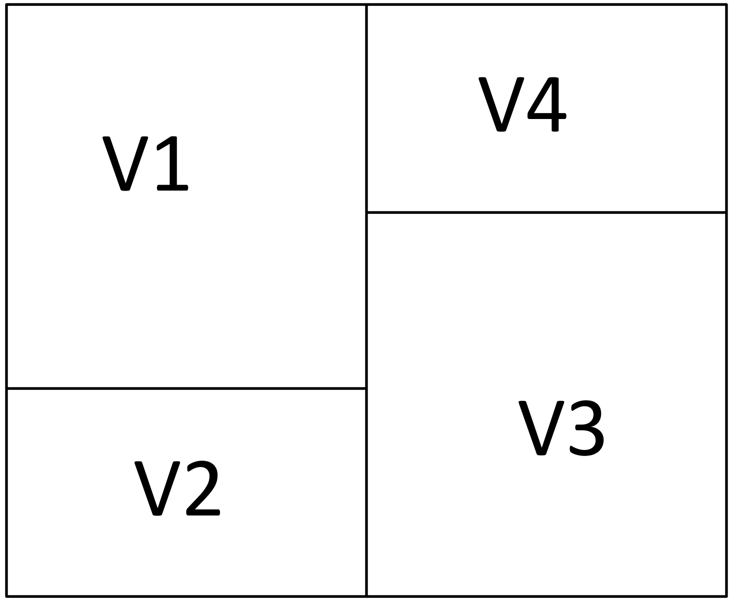
Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints



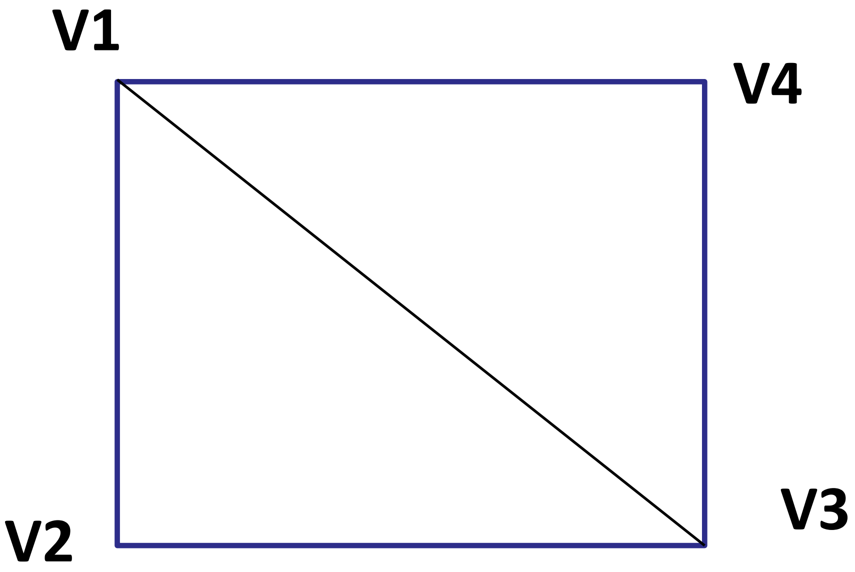
Example: Map Colouring

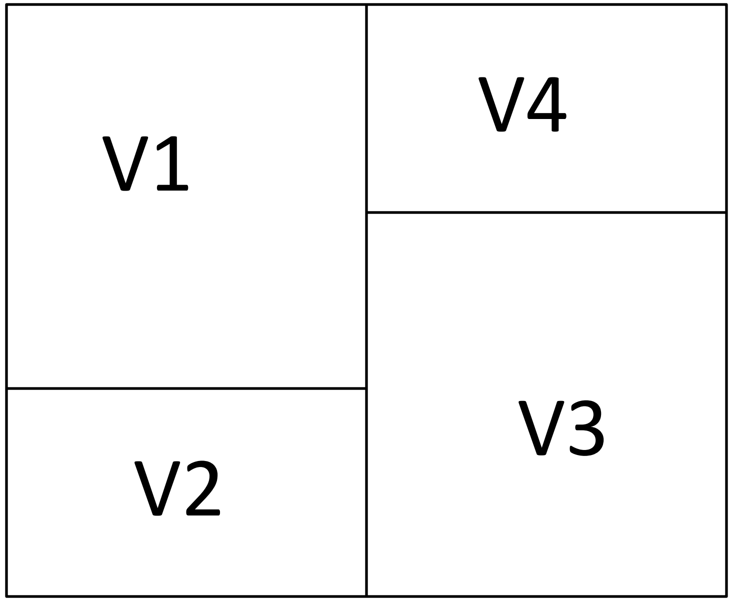
- Can we colour all 4 regions with 3 colours (Red, Green, Blue)so that no two adjacent regions are the same colour?



Example: Map Colouring as CSP

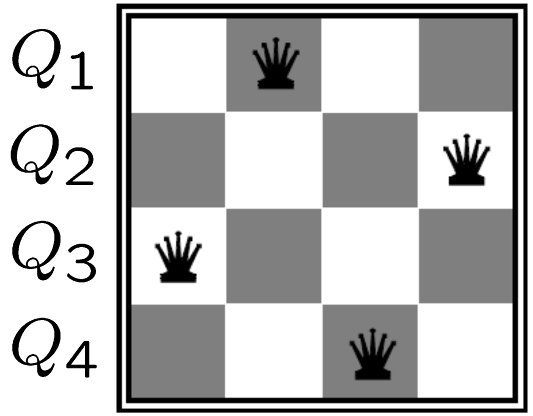
- X= { V1, V2, V3, V4}
- Di = { Red, Green, Blue}
- Constraints: adjacent regions must have different colors
- Solution?





Example: N-Queens

- Formulation:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

$\cdots$

Example: Cryptarithmetic

- Variables:

  F T U W R O C_1 C_2 C_3

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- Constraints:

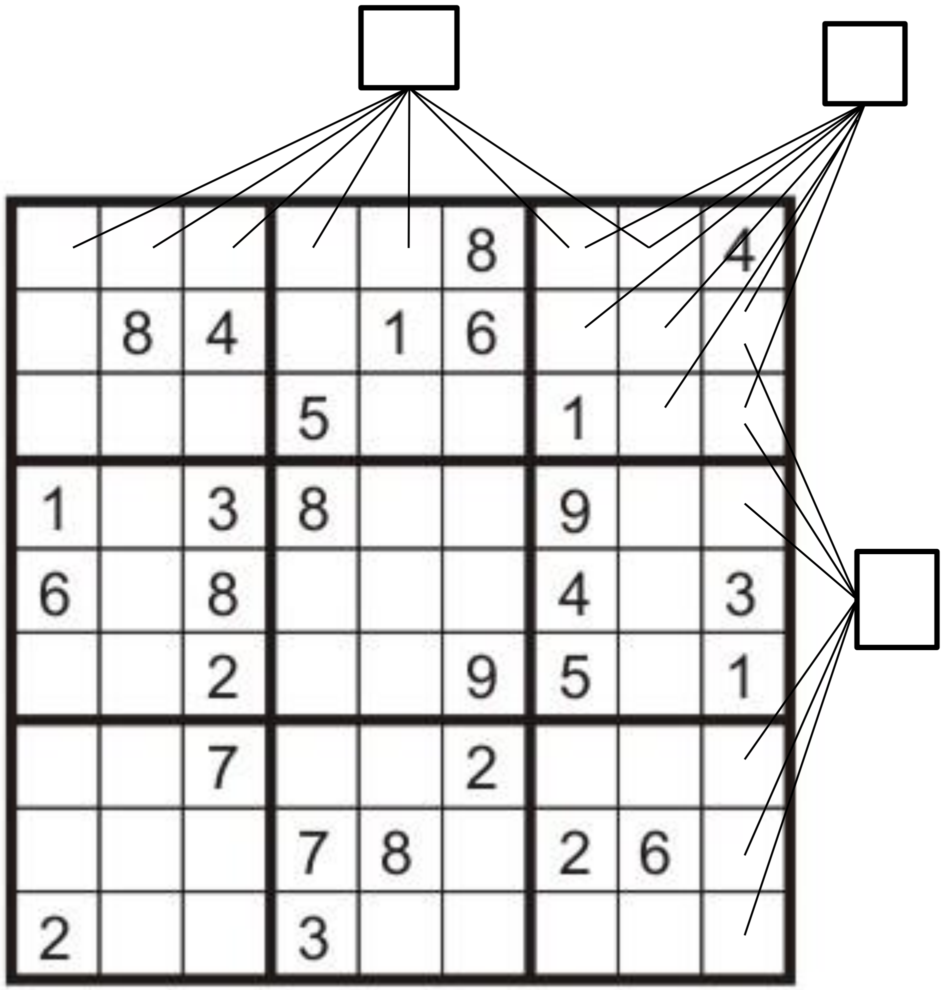
  $\text{alldiff}(F, T, U, W, R, O)$

$$O + O = R + 10 \cdot C_1$$
$$C_1 + W + W = U + 10 \cdot C_2$$
$$C_2 + T + T = O + 10 \cdot C_3$$
$$C_3 = F \; ,$$

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

A cryptarithmetic problem : Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed.

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

9-way alldiff for each row

9-way alldiff for each region

Varieties of Constraints

- **Unary constraints** involve a single variable (equivalent to reducing domains), 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., cryptarithmetic column constraints

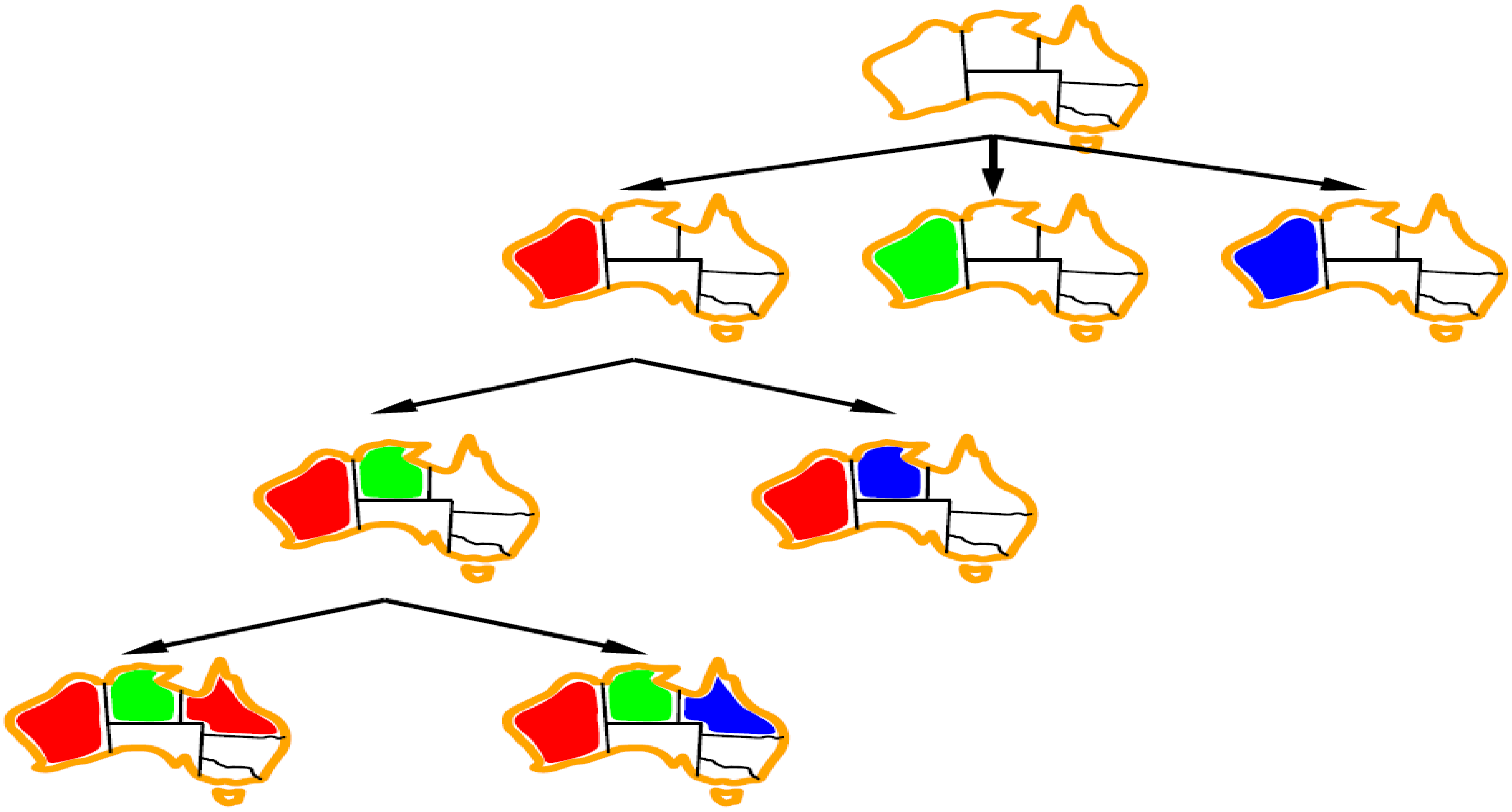
Solving CSPs

Standard Search Formulation

- States defined by the values assigned so far (partial assignments)
  - **Initial state:** the empty assignment, {}
  - **Successor function:** assign a value to an unassigned variable
  - **Goal test:** the current assignment is complete and satisfies all constraints

- We'll start with the straightforward, naïve approach, then improve it
  - Depth-first search for CSPs with single-variable assignments is called backtracking search
  - Backtracking search is the basic uninformed algorithm for CSPs

Backtracking Example

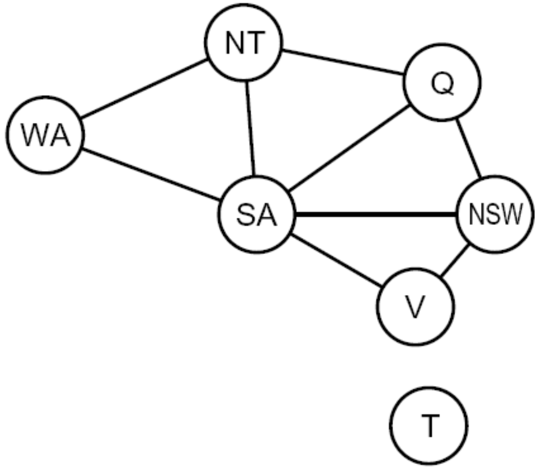


Improving Backtracking

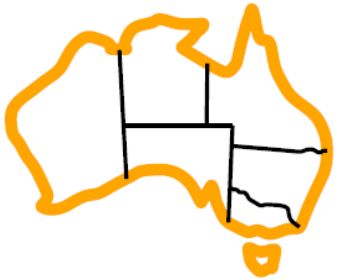
- Really important how to choose the next variable
  - Minimum Remaining Values (MRV) heuristic
  - Degree heuristic(involved in most constraints)
  - Least-constraining-value heuristic

- Propagating information through constraints
  - Forward Checking
  - Arc consistency

Minimum Remaining Values Heuristic

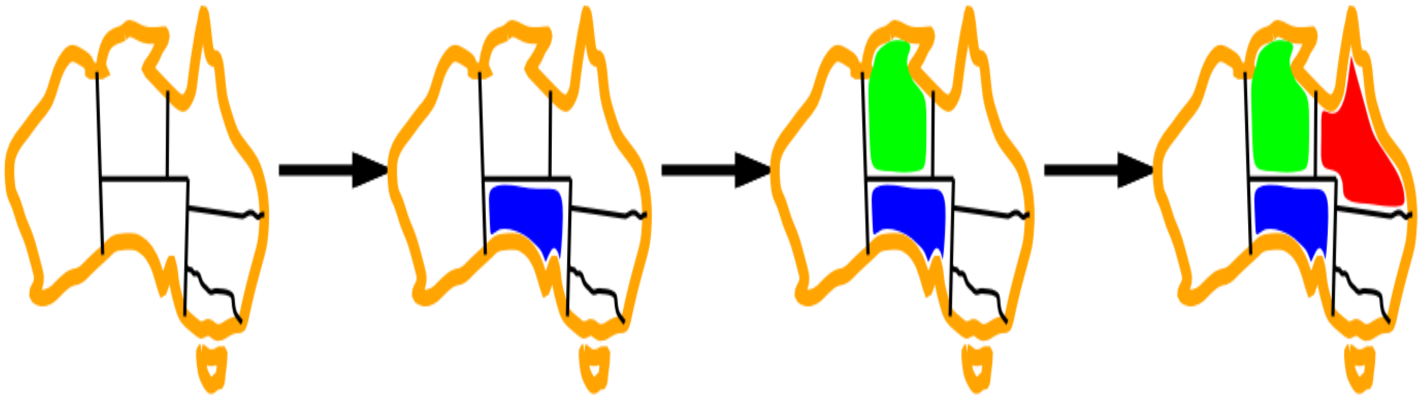


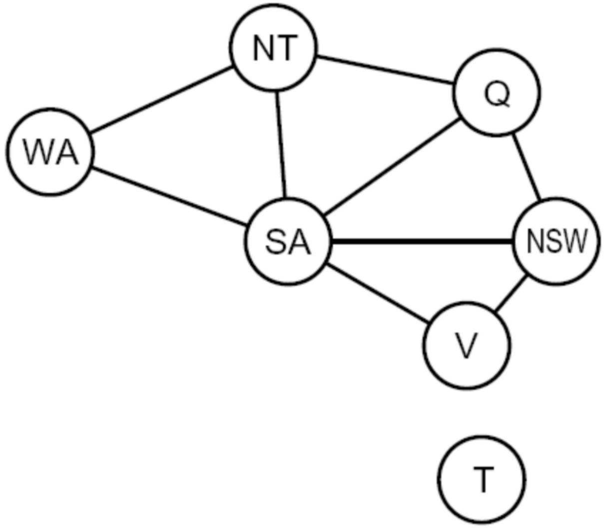
- Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain



Degree heuristic

- Degree heuristic:
  - choose the variable that is involved in the largest number of constraints on other unassigned variables



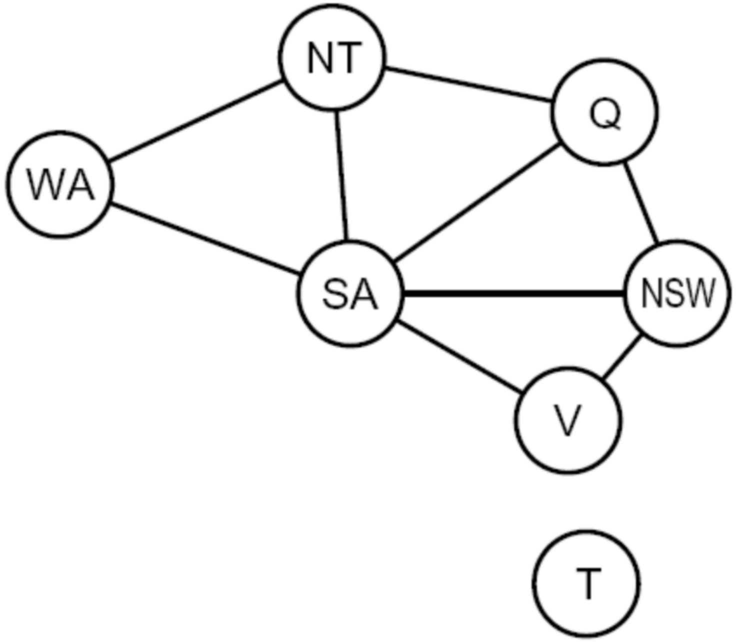


Least-constraining-value **(LCV)** heuristic

- Given a choice variable, choose the least constraining value i.e., the one that rules out the fewest values in the remaining variables

Example:

- The partial assignment with WA =Red and NT= green and that our next choice is for Q.
- Blue would be a bad choice because it eliminates the last legal value left for Q's neighbour, SA.
- The least-constraining value heuristic therefore prefers red to blue.

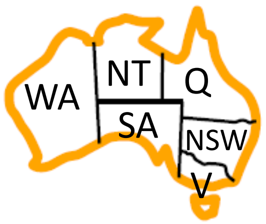


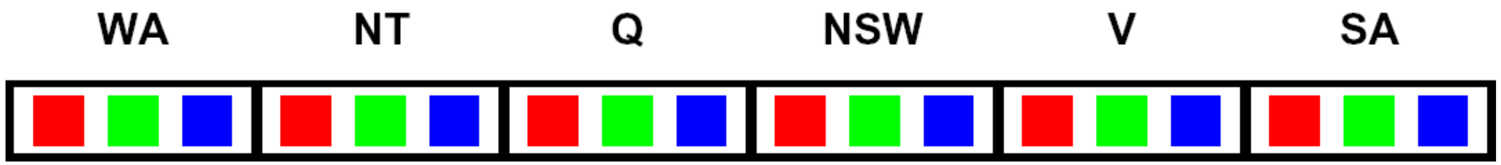
Propagating information through constraints

- Forward Checking
- Arc consistency

Forward Checking

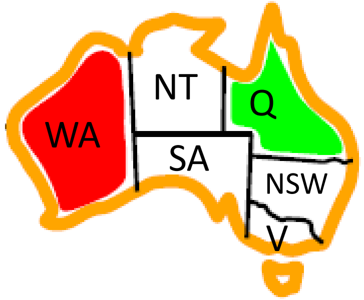
- Idea: Keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values

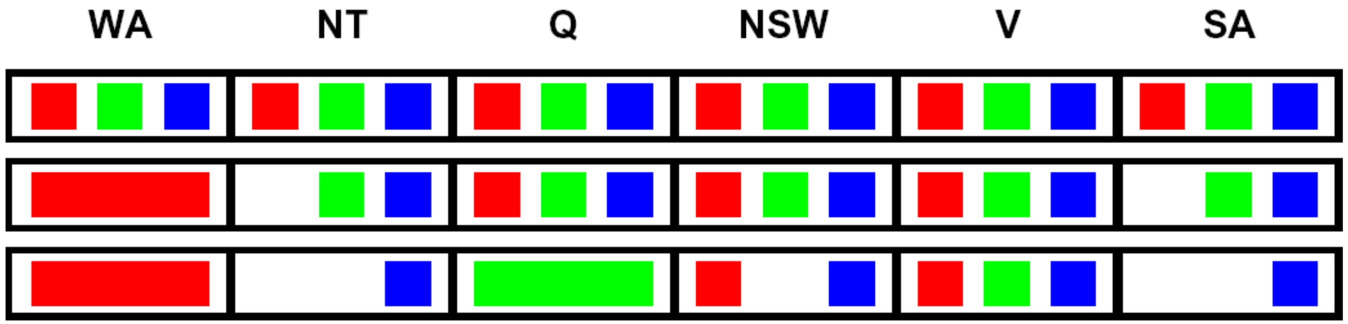




Forward Checking

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





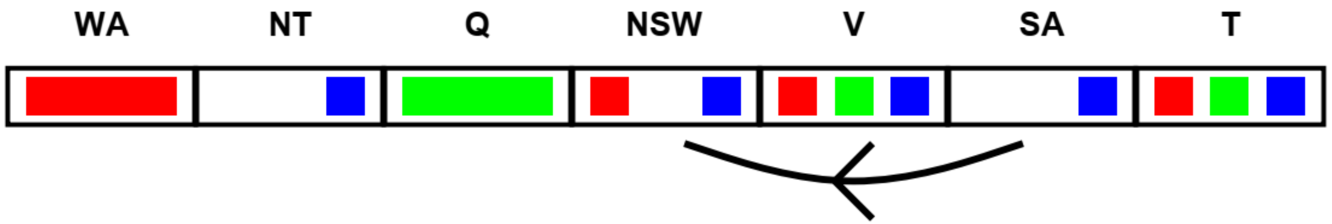
- NT and SA cannot both be blue!

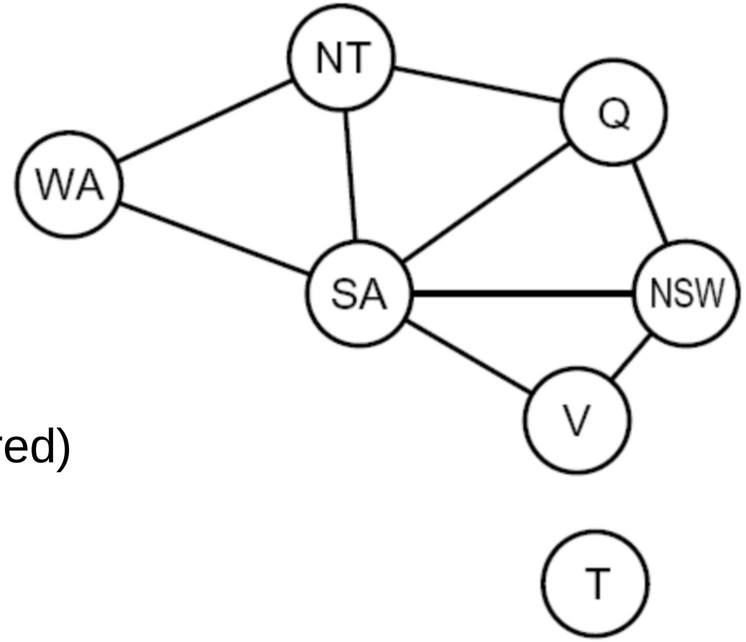
Arc consistency

- Simplest form of propagation makes each arc consistent

  X →Y is consistent iff

  for every value x of X there is some allowed y





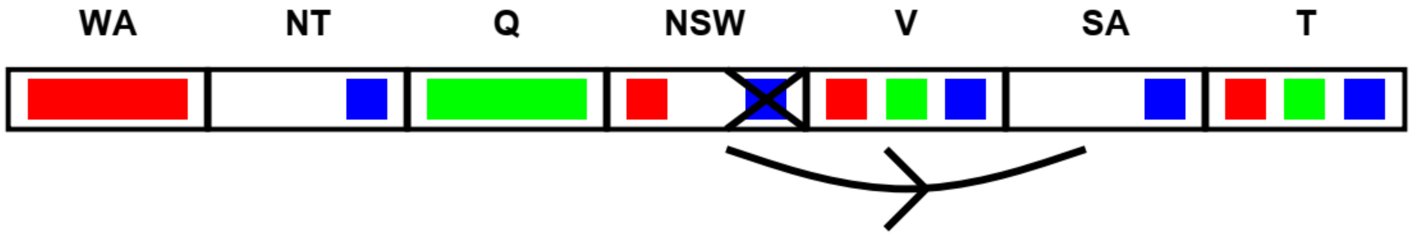
The current domain of SA and NSW are {blue} and {red, blue}.
For SA=blue, there is a consistent assignment for NSW (i.e., NSW=red)
Therefore, arc from SA to NSW is consistent.

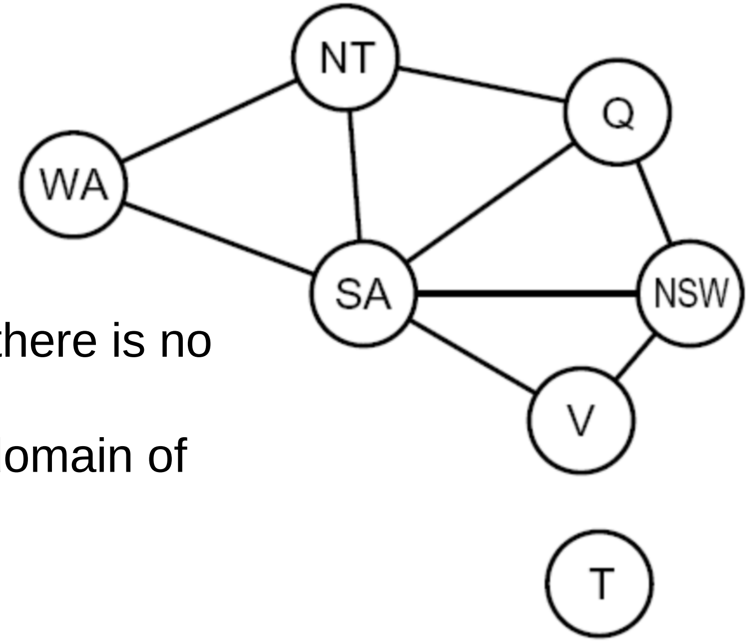
Arc consistency

- Simplest form of propagation makes each arc consistent

  X →Y is consistent iff

  for every value x of X there is some allowed y





- The reverse arc from NSW to SA is not consistent: for NSW ={blue} , there is no consistent assignment for SA.
- The arc can me made consistent by deleting the value blue from the domain of NSW
- Arc consistency detects failure earlier than forward checking

AC -3 Algorithm

- The most popular algorithm for enforcing arc consistency is called AC-3.
- To make every variable arc-consistent, the AC-3 algorithm maintains a queue of arcs to consider.
  - Initially, the queue contains all the arcs in the CSP.
  - Each binary constraint becomes two arcs, one in each direction
- If revised down to nothing, then we know the whole CSP has no consistent solution, and AC-3 can immediately return failure.
- Otherwise, we keep checking, trying to remove values from the domains of variables until no more arcs are in the queue.
    - At that point, we are left with a CSP that is equivalent to the original CSP—they both have the same solutions—but the arc-consistent CSP will be faster to search because its variables have smaller domains.

AC -3 Algorithm

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  queue ← a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ POP(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if size of $D_i = 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add (X_k, X_i) to *queue*
  return *true*

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised ← *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 ← *true*
  return *revised*