Intelligent Systems

Lecture 1
Constraint Satisfaction Problems

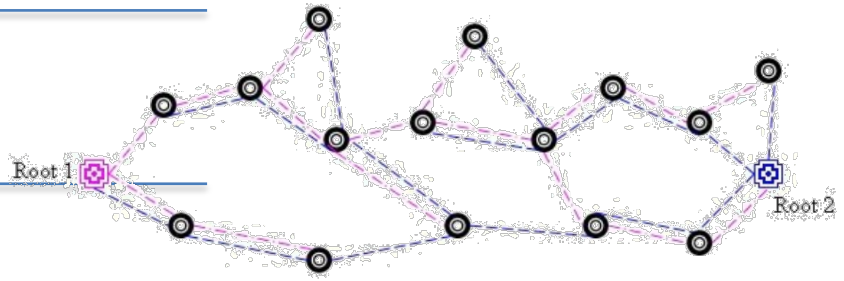
Dra. Elisa Guerrero Vázquez
Dpto. Ingeniería Informática
University of Cadiz

Overview

1. Introduction to Constraint Satisfaction Problems
2. Problem Formulation
3. Backtracking for CSP
    1. MRV
    2. LCV
    3. Forward Checking
4. AC3
5. Local Search

1.1 Real World CSP Problems

Shortest path between several points

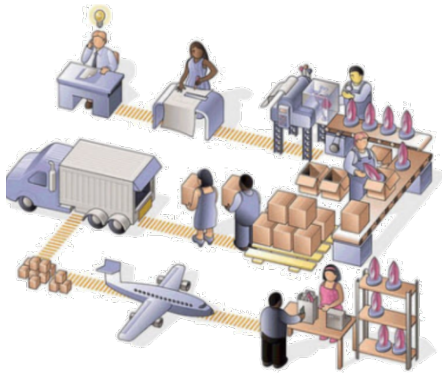


Optimal routing of data in Internet

Minimal cost planning for product shipping

Optimal sequencing in process manufacturing

Task scheduling

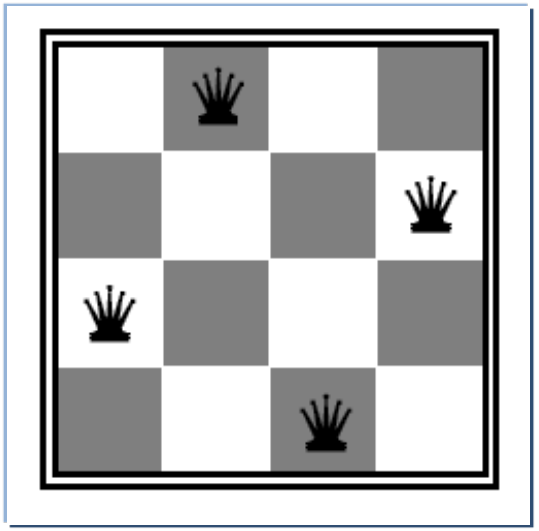


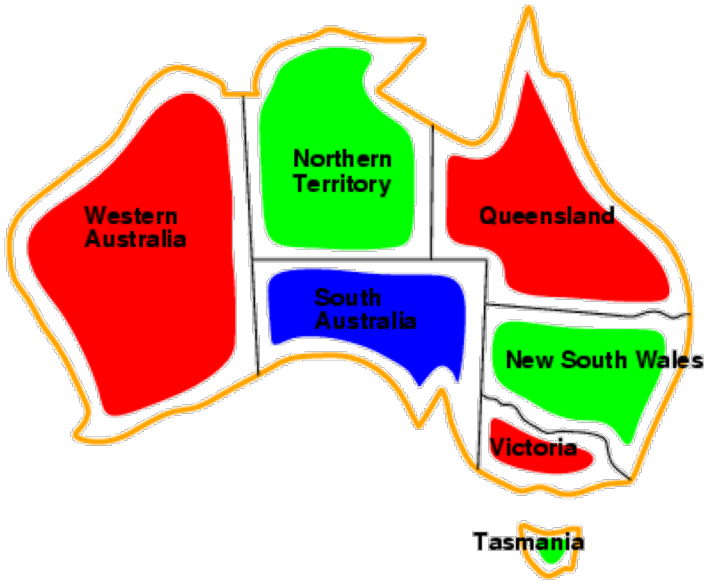
Optimal aircrew selection

https://www.youtube.com/watch?v=y4RAYQjKb5Y

1.1 CSP problems

- N-Queens
- Map Coulouring
- Cryptography






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

1.1 CSP definition

Constraint Satisfaction Problem (CSP)

the solution is a correct assignment of values to each variable according to a set of constraints that must be satisfied

1.2 CSP formulation

Set of Variables $X_1 \ldots X_n$

- whose values belong to a domain Di

Set of Constraints $C_1 \ldots C_m$

- the set of allowed values
- rules or properties that each variable must satisfied

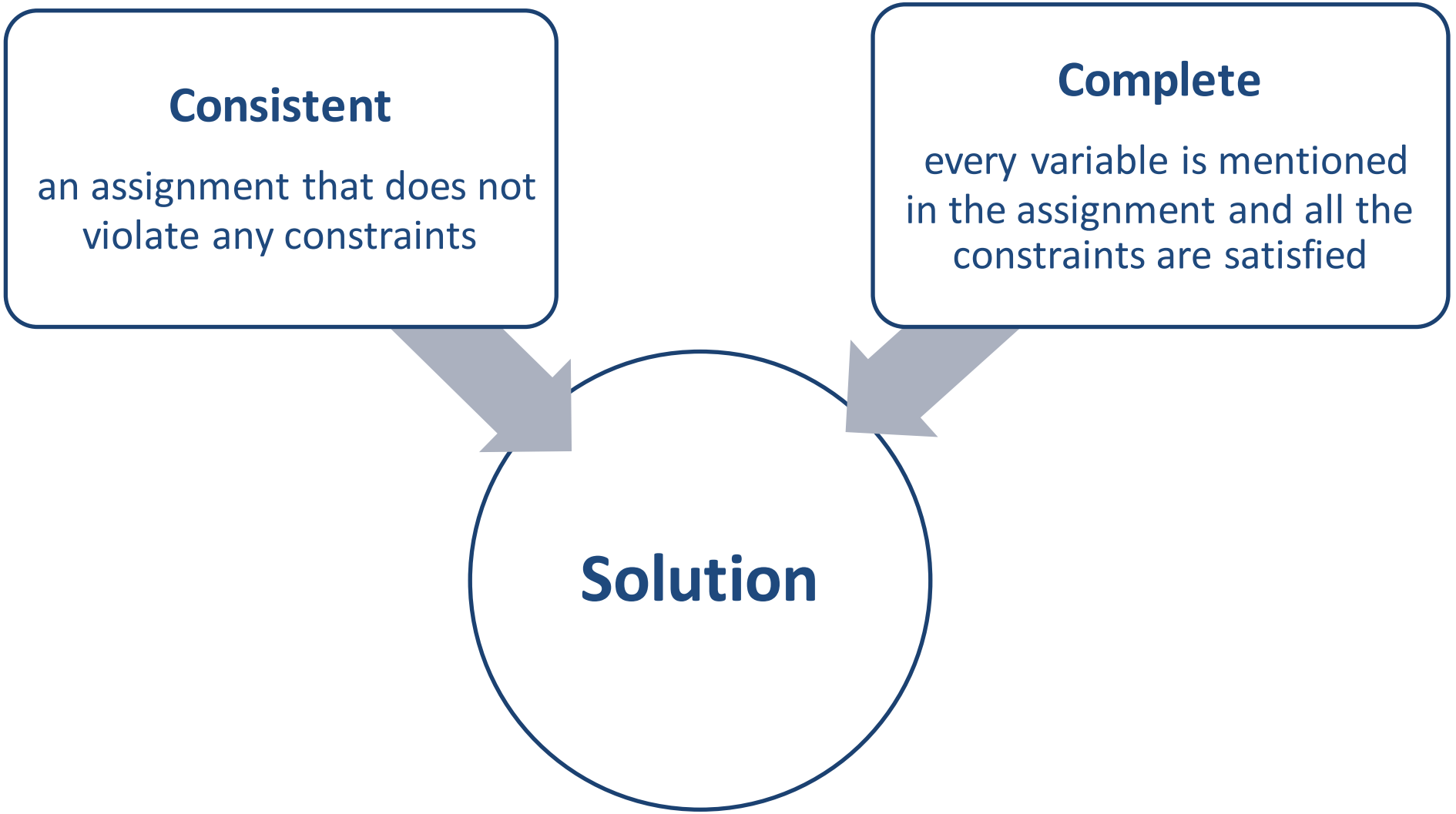
State:

- assignment of values to variables

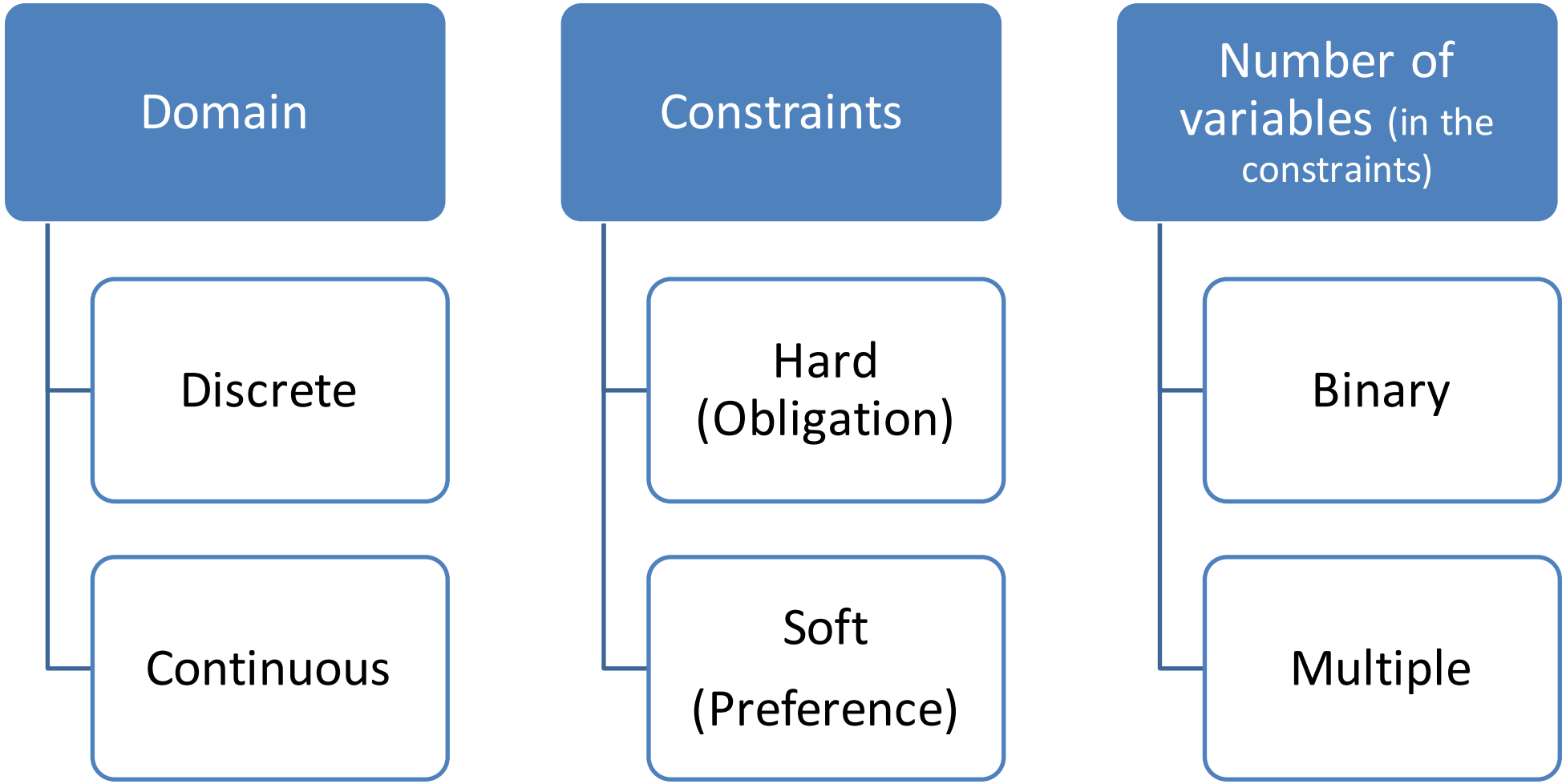
Solution:

- complete assignment that satisfies all the constraints

1.3 Assignments



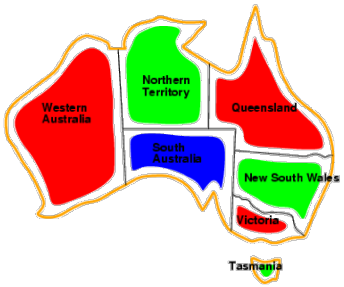
1.4 CSP classification



1.5 CSP formulation example: Graph-colouring

- **GOAL**: Assign different colours to adjacent regions
- **Variables**: R_1 .. R_7 each region
  - **Domain** of each variable: set of colours {red, green, blue}
  - **Constraints**:
    - Two adjacent regions must have different colours
      - $R_i \neq R_j$ If R_i and R_j are adjacent
  - **State**: any assignment, R_1=red
  - **Solution**: Consistent and complete assignment

$$\{R_1=\text{red}, R_2=\text{green}, R_3=\text{red}, R_4=\text{blue},$$
$$R_5=\text{green}, R_6=\text{red}, R_7=\text{green}\}$$



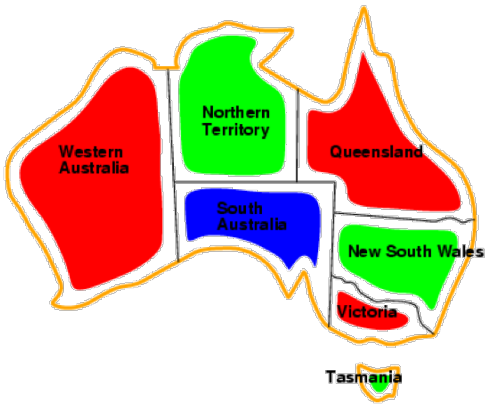
1.5 CSP formulation example: Graph-colouring

- GOAL: assign different colours to adjacent regions
  - **Variables**: R_1 .. R_7 each region
  - **Domain** of each variable: set of colours {red, green, blue}
  - **Constraints**:
    - Two adjacent regions must have
      - $R_i \neq R_j$ If R_i and R_j are adjace
  - **State**: any assignment, R_1=red
  - **Solution**: Consistent and complete

Discrete Domain

Hard Constraints

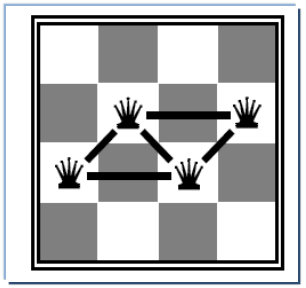
Binary Constraints

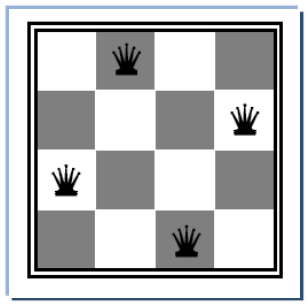


{R_1=red, R_2=green, R_3=red, R_4=blue,
R_5=green, R_6=red, R_7=green}

1.5 CSP formulation example: N-Queens

- GOAL: Place N queens on an NxN chess board so that no queen can attack any other queen
  - **Variables:** Q_1, Q_2, Q_3, Q_4 representing each queen position (Queen 1 is always in column 1, Queen 2 in column 2, …)
  - **Domain:** row numbers {1, 2, 3 y 4}
  - **Constraints**:
    - Different Row: $Q_i \neq Q_j$
    - Different Diagonal: $|Q_i - Q_j| \neq |i - j|$
  - **State**: Any assignment
  - **Solution**: Q_1=3 Q_2=1 Q_3=4 Q_4=2





Discrete Domains, Hard Constraints, Binary Constraints

1.5 CSP formulation example: Criptoarithmetic

- GOAL: assign different digits to the letters
  - **Variables**: each letter is a different variable, and two more variables are needed:

    $$\{T, W, O, F, U, R, X_1, X_2\}$$

  - **Domain** for the letters: values from 0 to 9, for the carrying variables, values from 0 to 1
  - **Constraints**:
    - Sum1: $O+O=R + 10* X_1$
    - Sum2: $X_1 + W + W = U + 10 * X_2$
    - Sum3: $X_2 + T + T = O + 10*F$

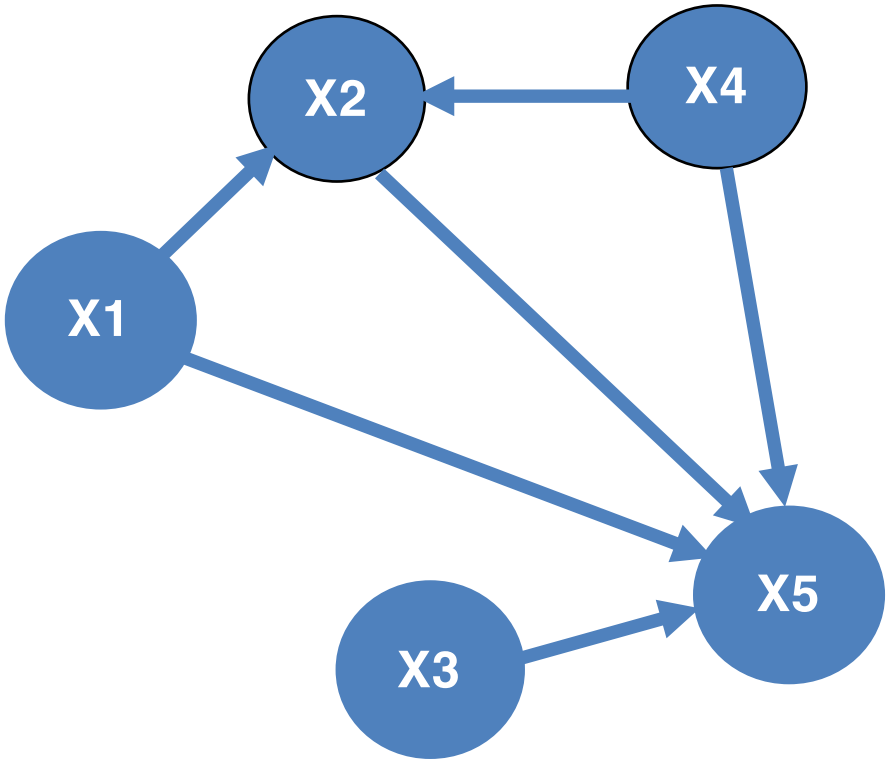
Hard and Multiple Constraints

	X_2	X_1		
	T	W	O	
	T	W	O	+
F	O	U	R	

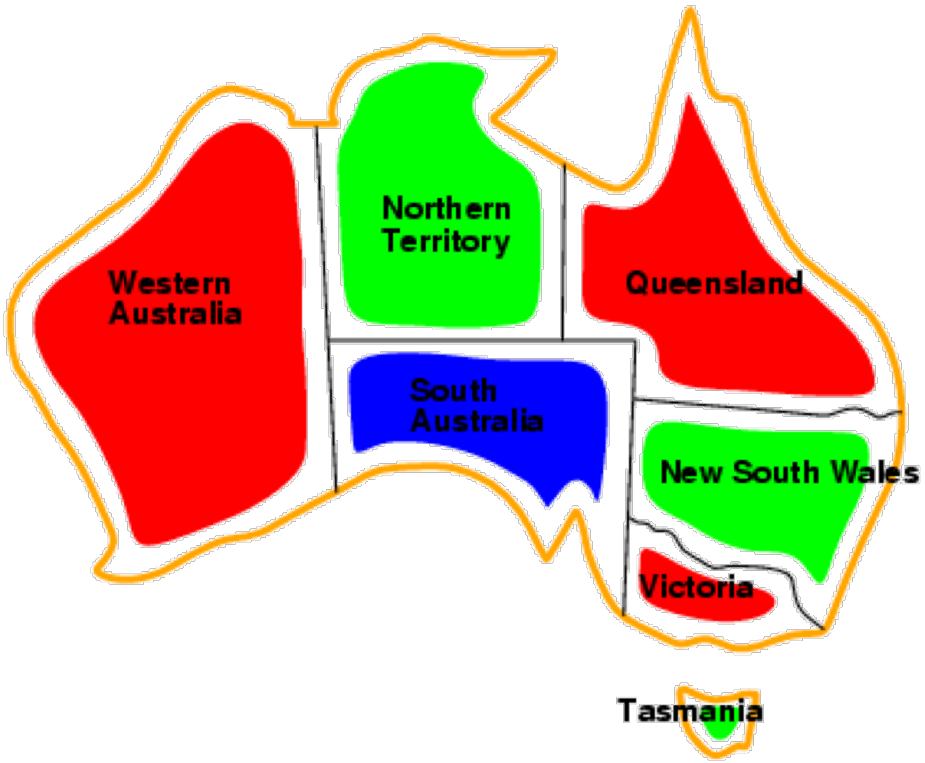
	0	1		
	8	3	6	
	8	3	6	+
1	6	7	2	

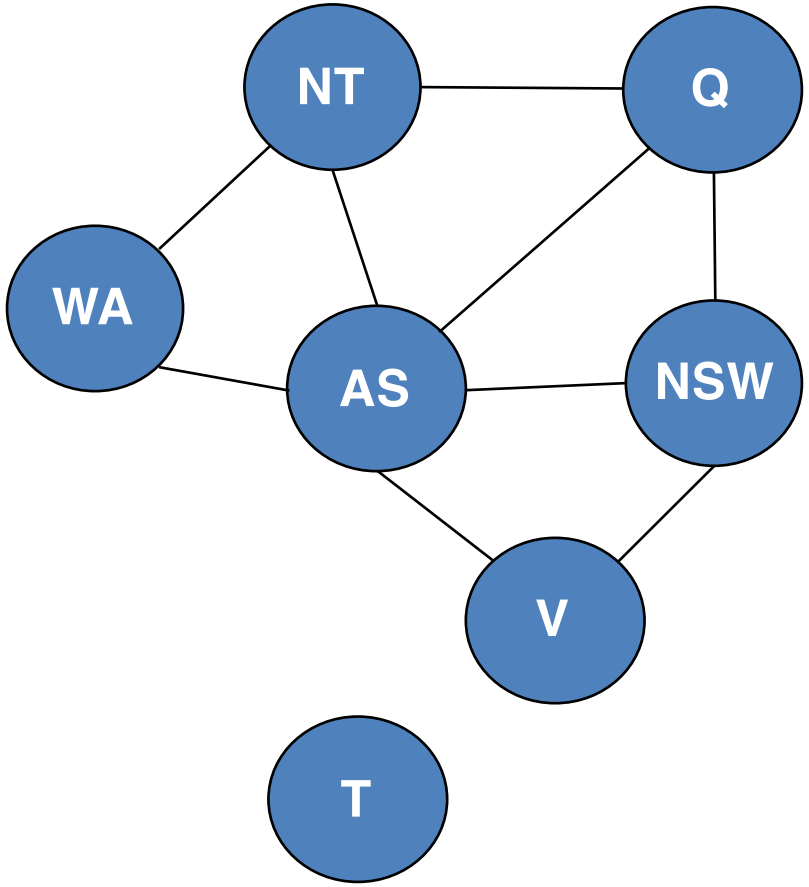
1.6 CSP representation

- Binary constraint graph:
  - **Nodes :** Variables
  - **Arcs or edges:** Binary relations between variables

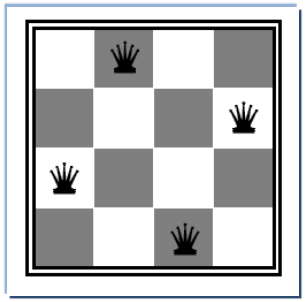


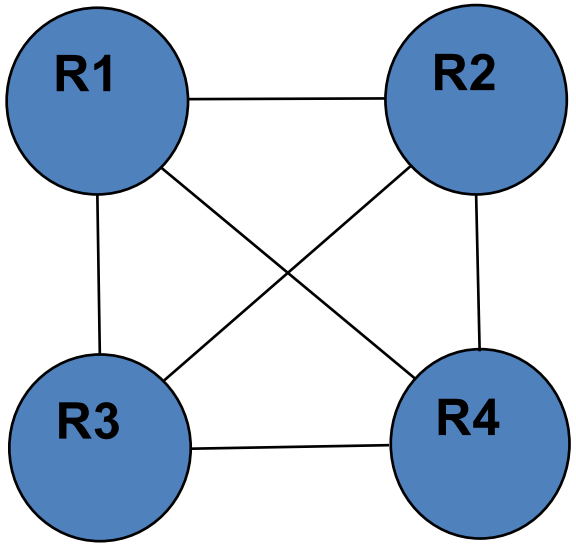
1.6 Binary constraint graph





1.7 N-Queens Example





2. CSP solving

- **Search strategies**

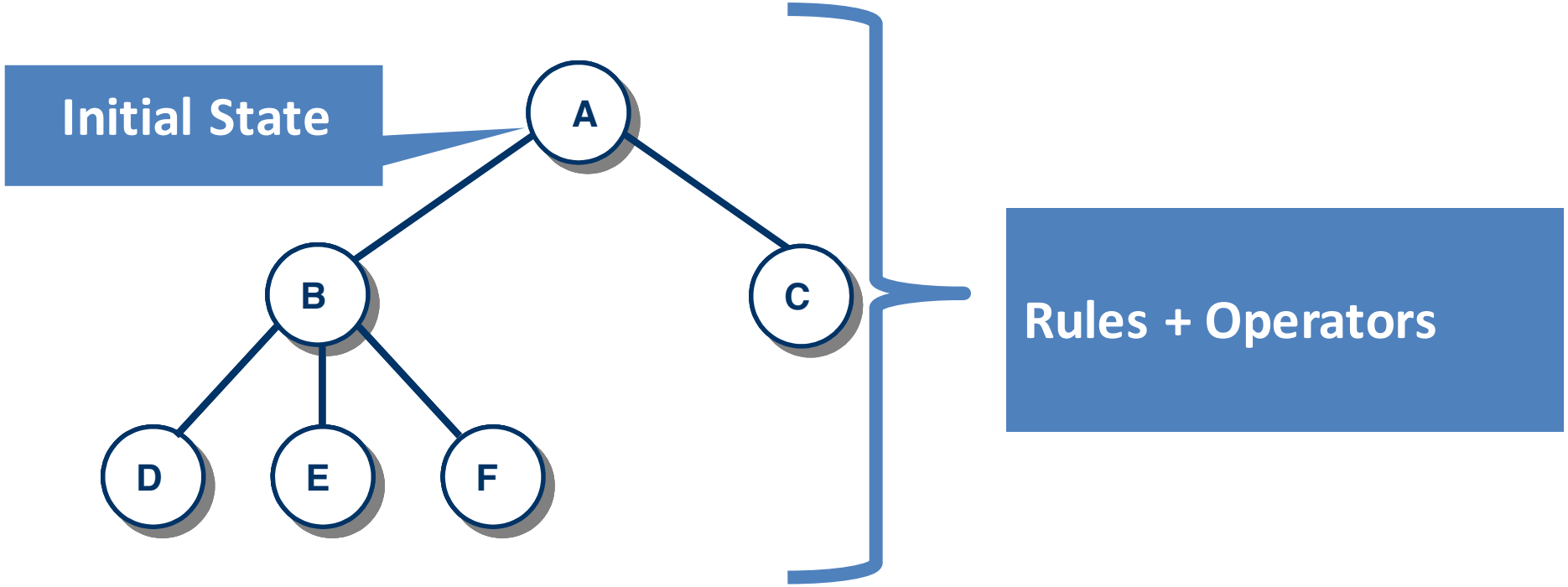
  Systematic search: Exploration of the state space

- **Consistency approaches**

  Inconsistent values are removed from variables domains

  Help to reduce the state space

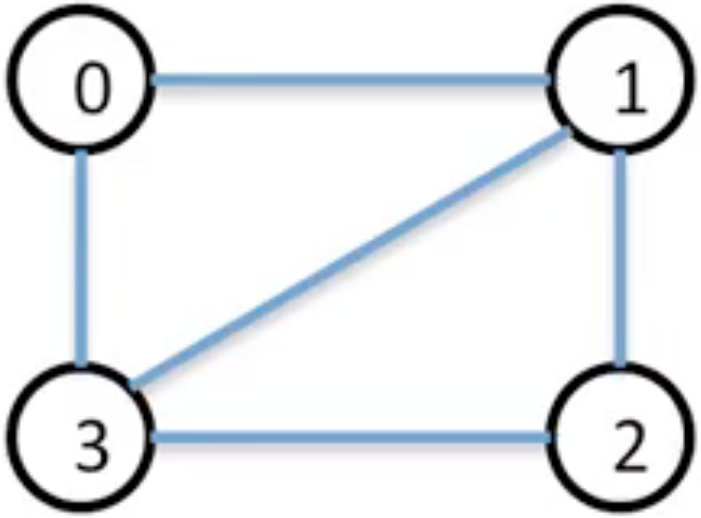
State Space

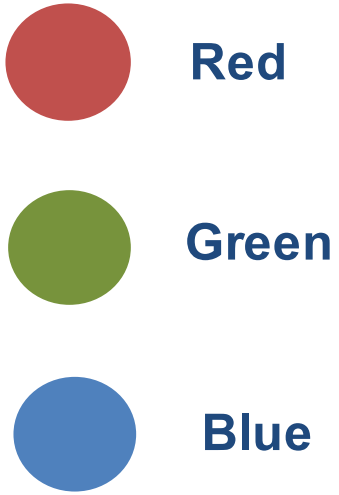


- Goal Test
- Path or Solution
- Solution Cost

2.1 Example

- Graph colouring problem





2.1 CSP as state space search

Incremental formulation as a standard search problem:

- **Initial State**: empty assignment

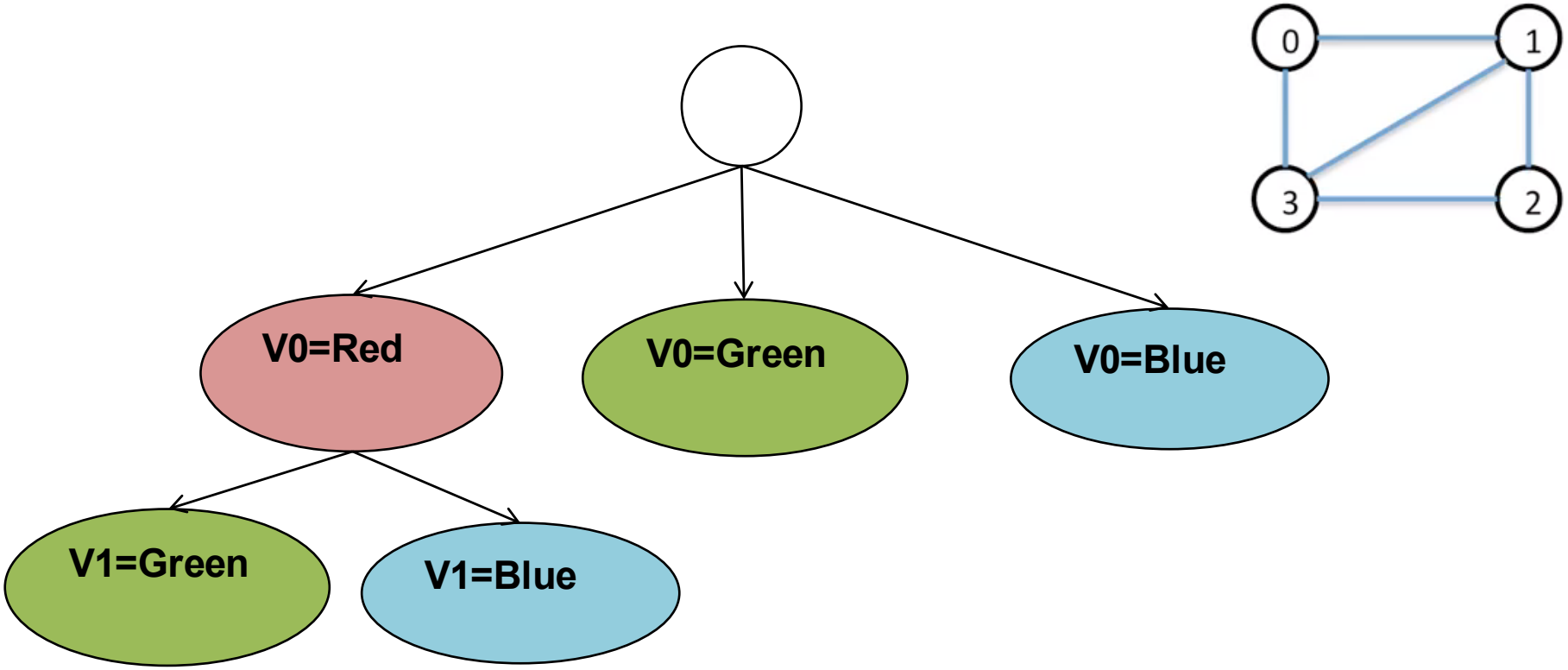
  { V0, V1, V2, V3}

  { , , , }
- **List of Actions:** Assign a color: Red, Green or Blue
- **Successor Function**: assignment of a value *v* to an unassigned variable when this action does not conflict with previous assignments
  - **isSafe** function to guarantee consistent assignments

  {**Red** , , , }
  { Red , **Green**, , }
- **Goal Test**: the current assignment is complete

2.1 CSP as state space search

Incremental formulation as a standard search problem:



¿Depth or Breadth Search?

2.2 Some considerations

- FINITE DEPTH: the number of variables determines the solution depth
- CONMUTATIVITY: assignment order is irrelevant
  - Depth-first search for CSPs with single-variable assignments is called backtracking search
- CONTROL OF REPEATED STATES is unnecessary

2.3 Backtracking

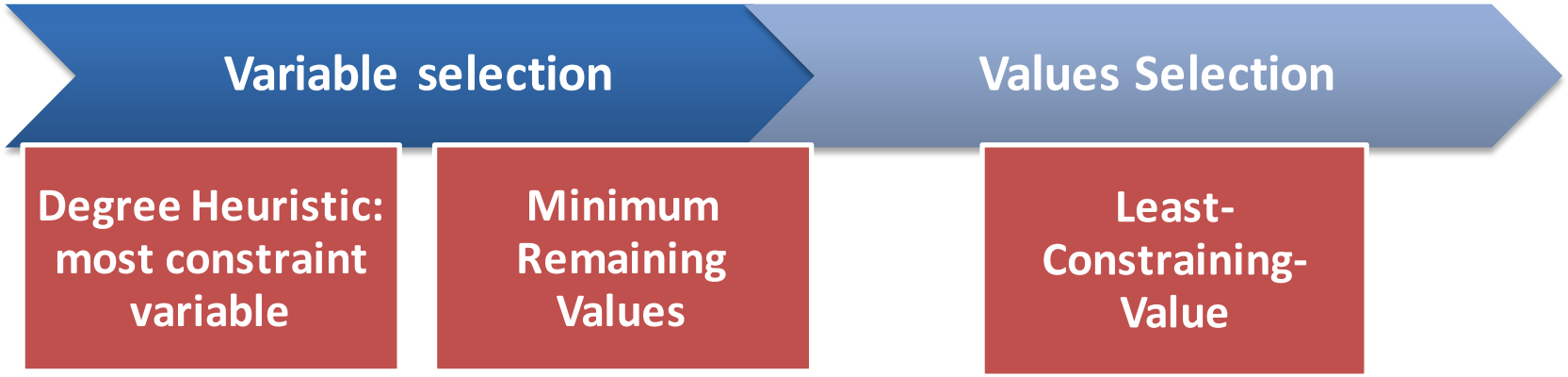
Special depth search …

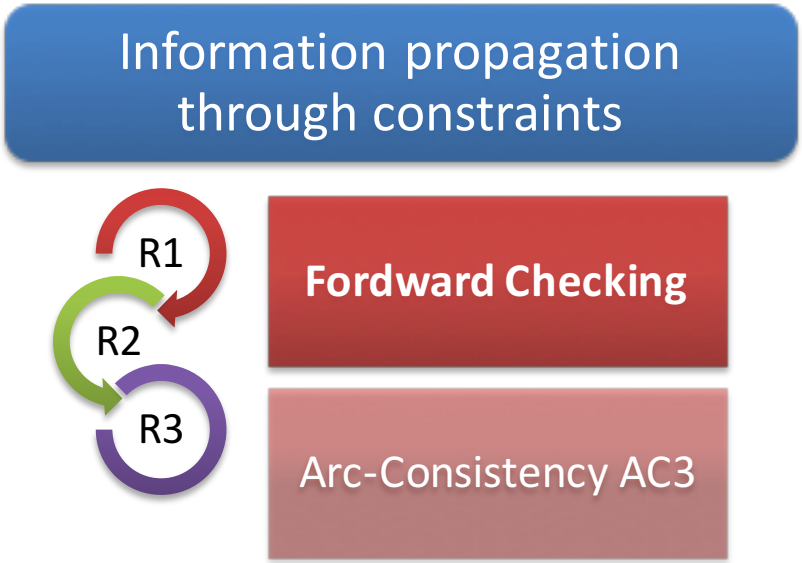
- Consider the variables in some order
- Pick an unassigned variable and give it a provisional value such that it is consistent with all of the constraints
  - If no such assignment can be made, we've reached a dead end and we need to backtrack to the previous variable
- Continue this process until a solution is found or all possible assignments have been exhausted

2.3 Backtracking Algorithm

```
function BacktrackiNSWearch(csp) returns solution or failure
                    return  Recursive-Backtracking(csp)
function [csp]= Recursive-Backtracking(csp)
 //returns  solution/failure and the assignment
          var←SELECT-UNASSIGNED-VARIABLE(csp)
          if Notempty(var)
                    values_list←ORDER-DOMAIN-VALUES(var,csp)
                    while  Not complete(csp.assignment)   & Not empty(values_list)
                              value←next(values_list)
                              if consistent (csp,value,vari)  then
                                        add {var←value} to csp.assignment
                                         [csp]=Recursive_Backtracking(csp)
                                         if  Not complete(csp.assignment)
                                                  undo(csp.assignment)
                                        end
                              end
                    end %while
          end
end %function
```

3 General purpose heuristcs





3.A Variable selection

Variable Selection

var←SELECT-UNASSIGNED-VARIABLE(csp)

Degree Heuristic:

selects the variable that is involved in the largest number of constraints of other unassigned variables

- useful as a tie-breaker or at the beginning of the search process

Minimum Remaining Values (MRV):

Selects the variable with less legal values

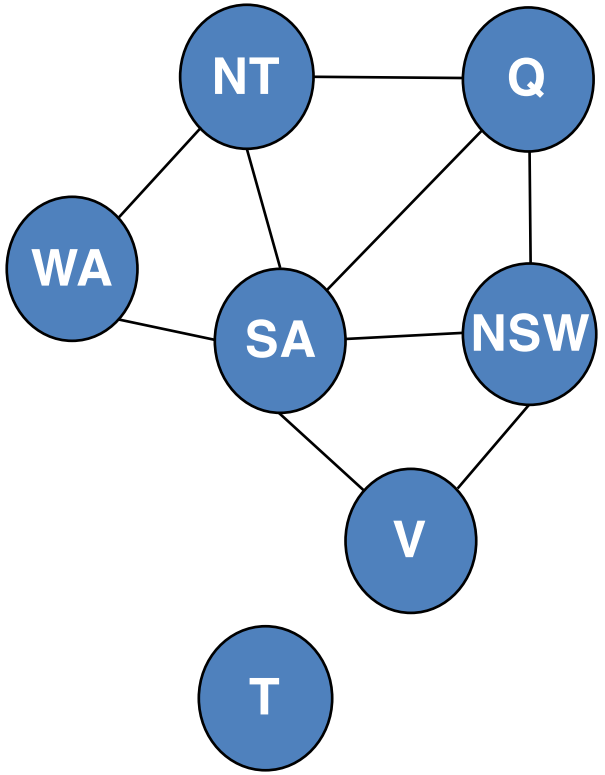
- **To increase the probability of pruning**

3.A Example of Degree Heuristic

Number of Constraints:

WA=2
NT=3
Q=3
SA=5
NSW=3
V=2
T=0

Wich variable goes next?



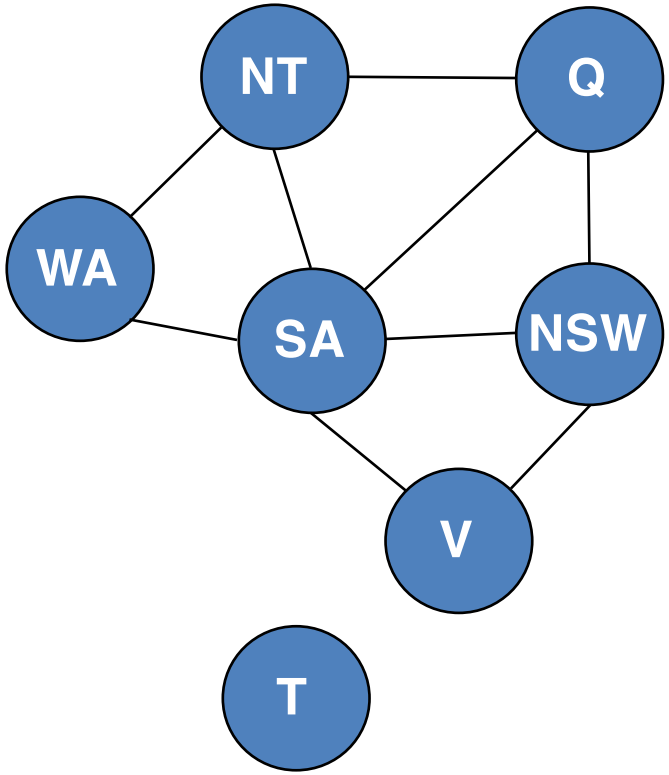
3.A Example of Degree Heuristic

SA=red

Degree Heuristic selects AS

Number of Constraints:

WA=2
NT=3
Q=3
SA=5
NSW=3
V=2
T=0



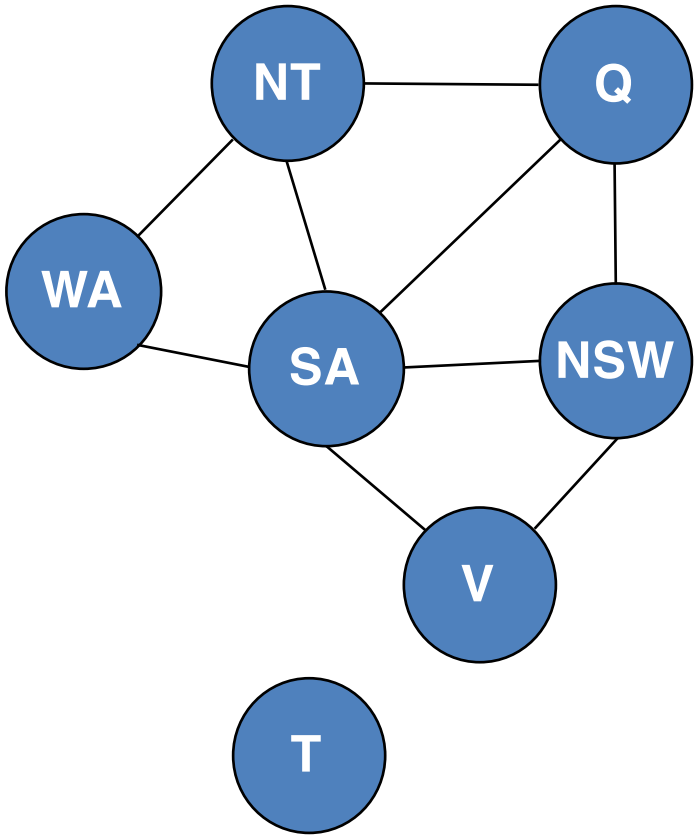
3.A Example of MRV (Minimum Remaining Values)

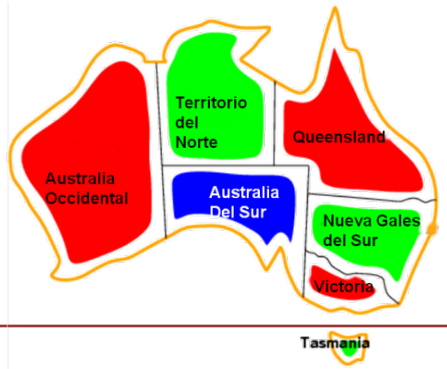
WA=red

WA=red
NT=green

Which variable goes next?

Possible values:
SA={blue}
Q={red, blue}





3.A Example of MRV (Minimum Remaining Values)

WA=red

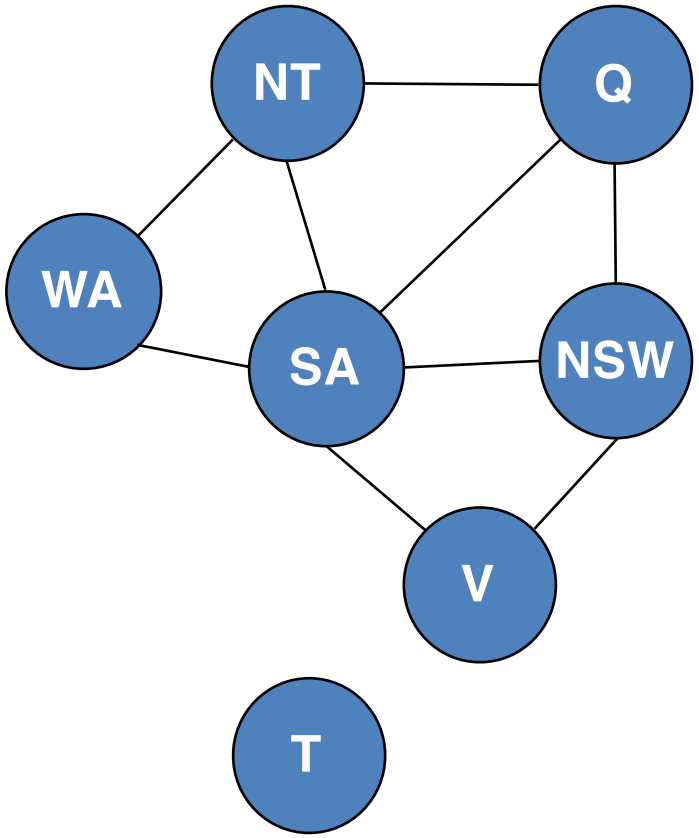
WA=red
NT=green

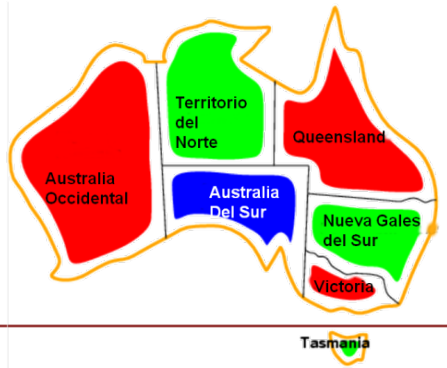
WA=red
NT=green
SA=blue

Which variable goes next?

Possible values:
SA={blue}
Q={red, blue}

MRV selects SA





3.B Values order heuristics

Once a variable has been selected, the algorithm must decide on the order in which to examine its values

Order of values

values_list←ORDER-DOMAIN-VALUES(var,csp)

Least-Constraining-Value (LCV)

Selects the value that appears in fewer constraints
e.g., the most free value

- Tries to leave as many options for the rest of the variables to be assigned

3.B Example of LCV

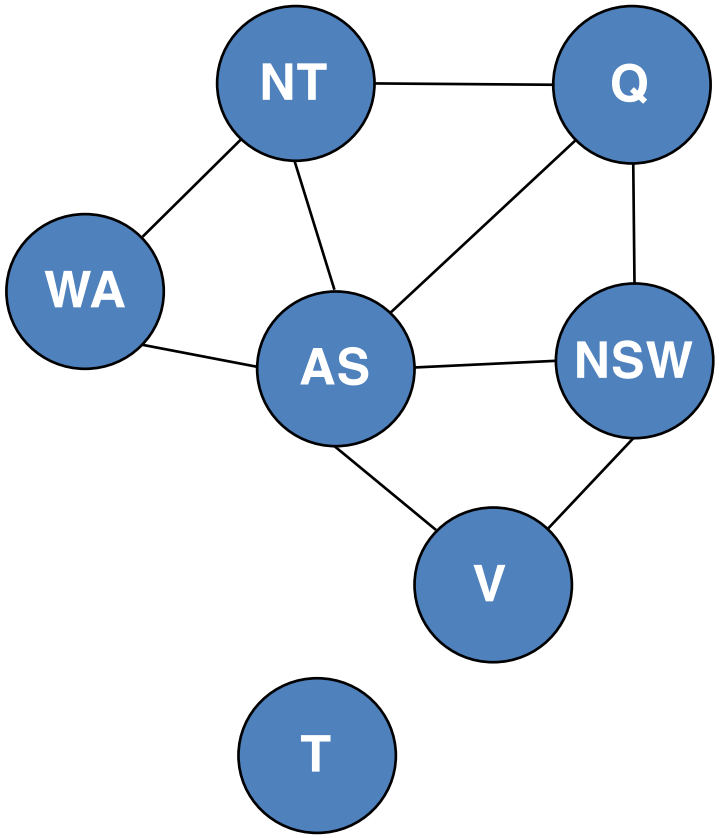
WA=red

WA=red
NT=green

WA=red
NT=green
Q= ?

Assuming that Q is the next variable …

Possible values:
Q={red, blue}



3.B Example of LCV

WA=red

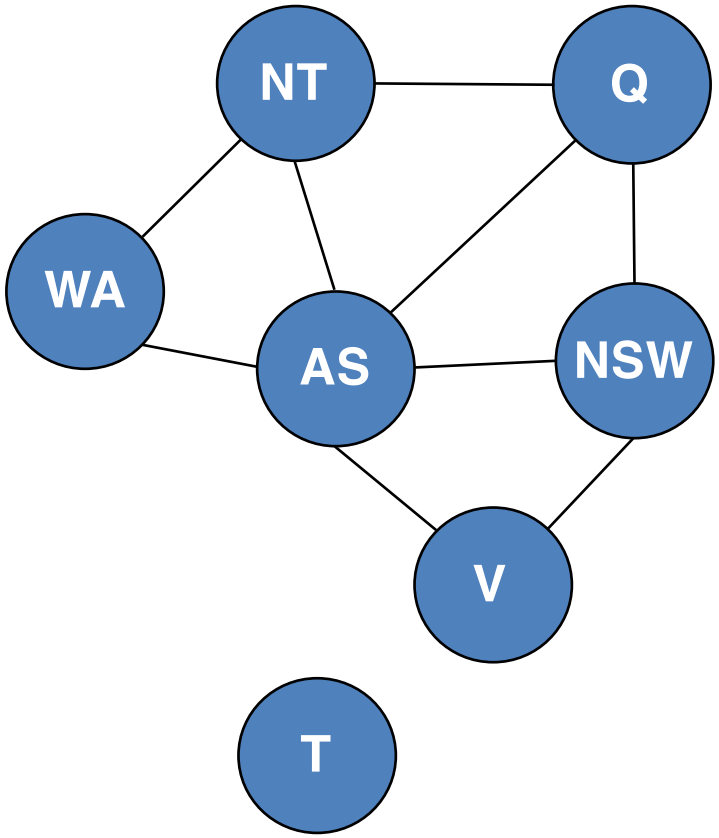
WA=red
NT=green

Assuming that Q is the next variable …

Possible values:
SA={blue}
Q={red, blue}

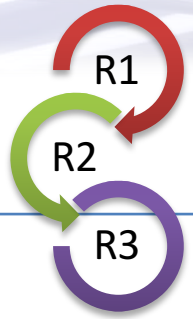
WA=red
NT=green
Q= red

red appears just once and blue twice



3.C Propagating information through constraints

Fordward Checking



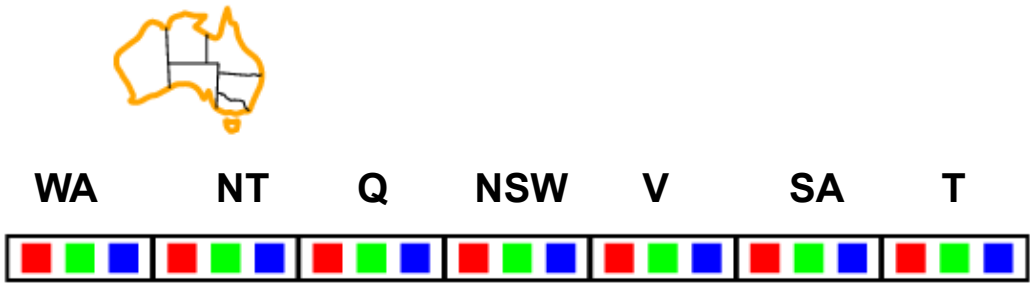
- When X is assigned a value …

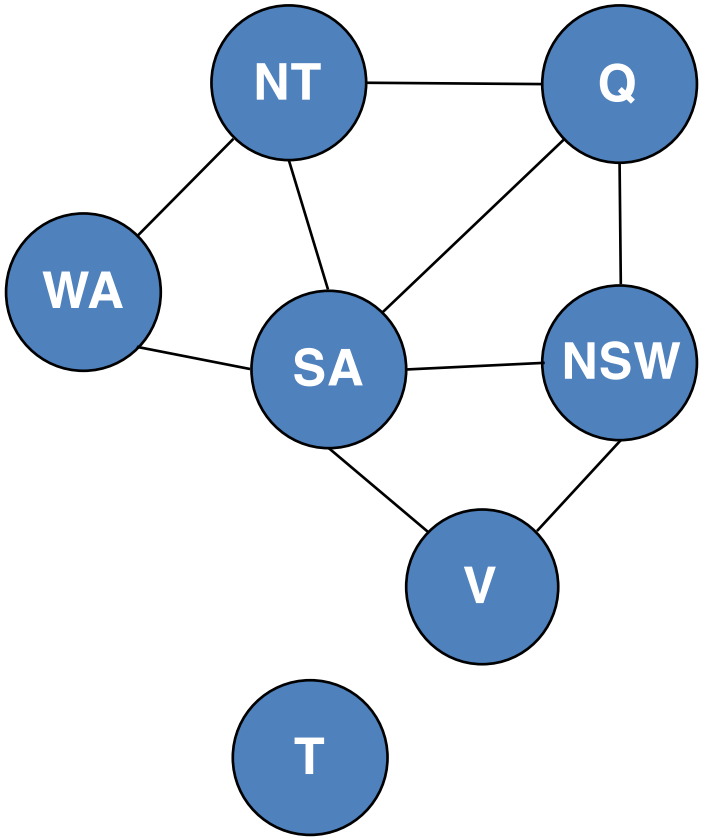
  FC looks at each unassigned variable Y that is connected to X by a constraint and deletes from Y ’s domain any value that is inconsistent with the value chosen for X

  - Each node in the search tree must contain the state and the list of possible values
  - MRV is an obvious partner of FC

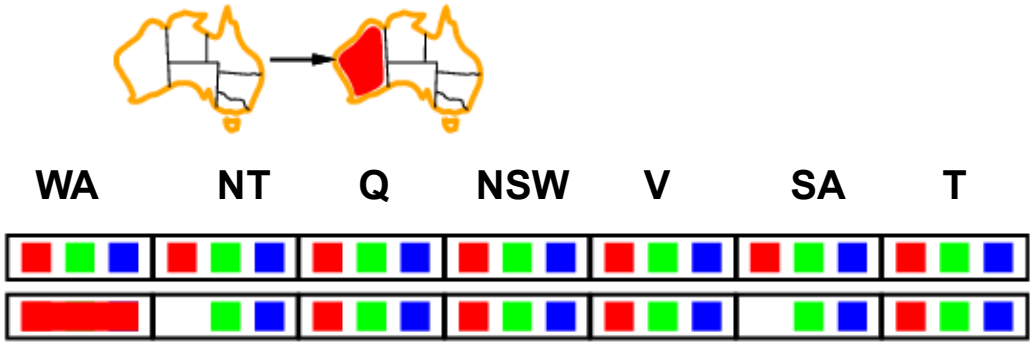
3.C Example of FC

- Initially





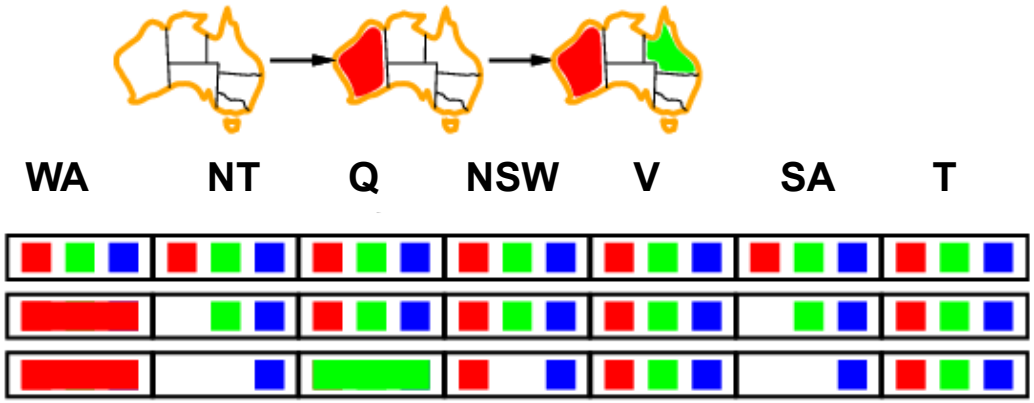
- WA=red

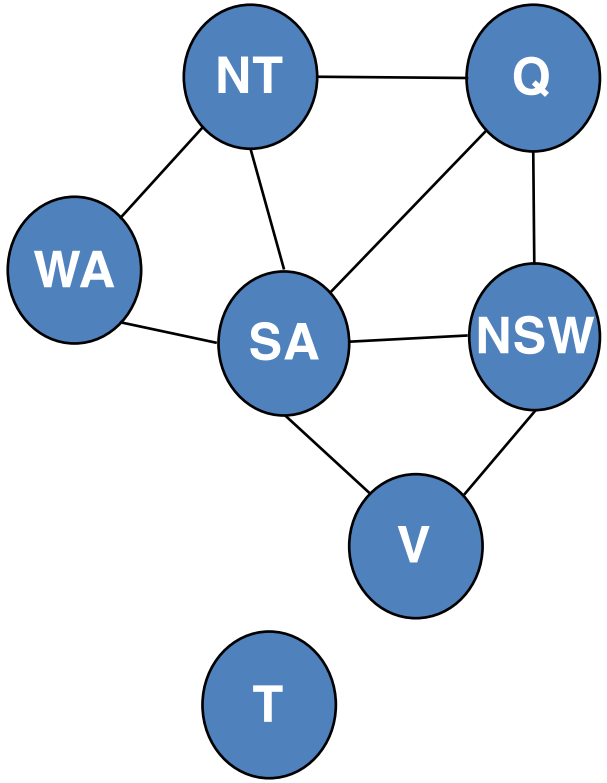


Red value is removed from NT and SA domains

3.C Example of FC

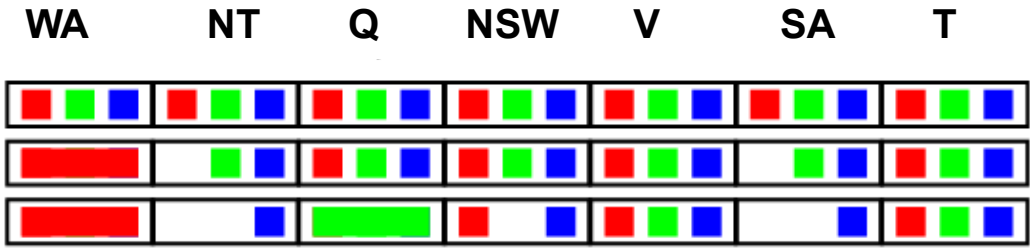
- WA=red Q=green

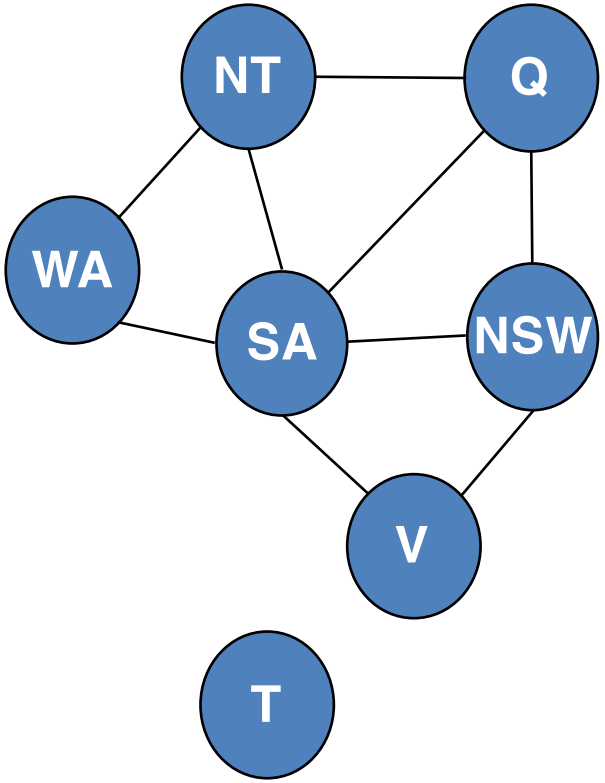




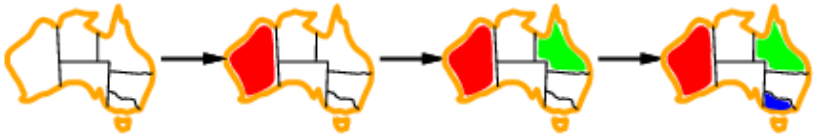
3.C Example of FC

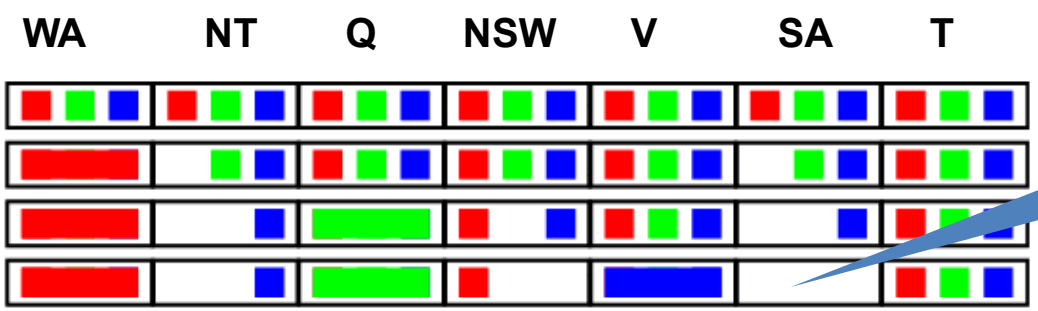
- WA=red Q=green





- WA=red Q=green V=blue

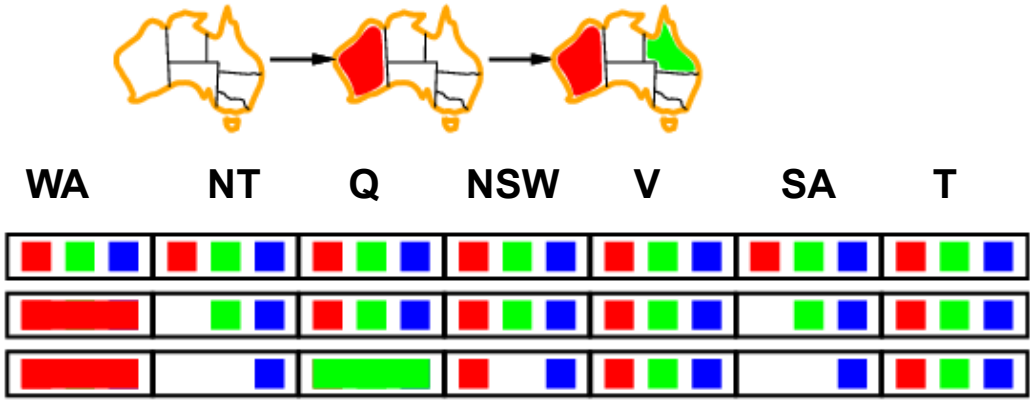


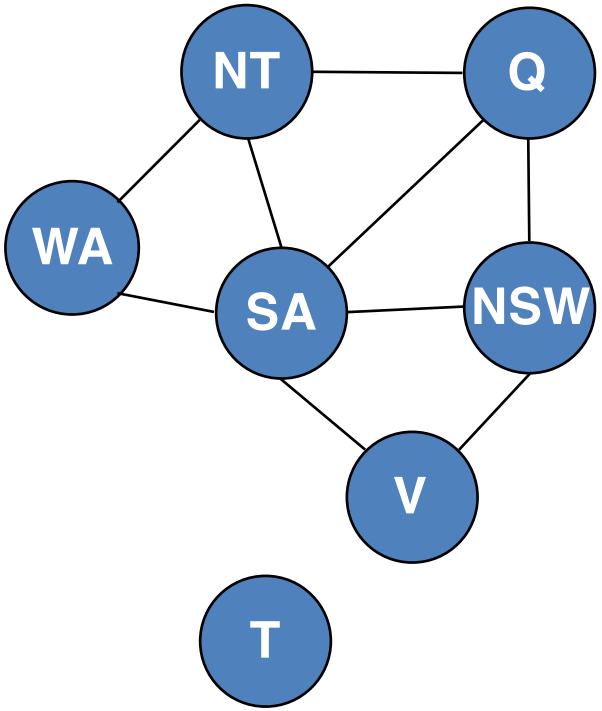


SA domain is empty !!

3.C Example of FC

- LIMITATION: Although forward checking detects many inconsistencies, it does not detect all of them





- NT and SA are both blue !!

3. Exercise

- Repeat forward checking applying MRV, Degree Heuristic y LCV.

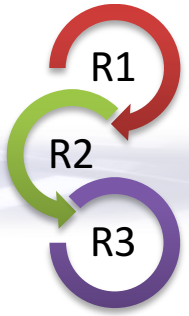
Algoritmo forward checking

- Forward checking can be seen as the application of a simple step of arc-consistency between the variable that has been assigned a value and each of the variables that remain to be instantiated:
  1. Select *xi*.
  2. Assign $xi \leftarrow aj : aj \in Di$.
  3. REPEAT:
     1. forward-check:

        Remove from the domains of the variables (*xi*+1.. *xn*), those values that are inconsistent with respect to the assignment (*xi, aj*), according to the set of constraints.

        Increment i
  4. UNTIL i>n
  5. If there exists a unassigned variable, and its domain is empty then retract assignment $xi \leftarrow aj$. Do:
     - Try with other values of *Di*, go to step(2).
     - If Di is empty:
       - If $i > 1$, decrement *i (try with previous variable)* and go back to step (2).
       - If $i = 1$, exit (No Solution).

3.C Limitations of Forward Checking



Forward Checking

Propagation of constraints

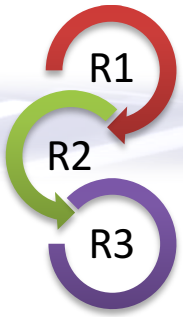
NO COMPLETE

Speed is required to be efficient

ALTERNATIVE:

Arc-consistent VARIABLES

3.C Constraint Propagation



arc-consistency AC3

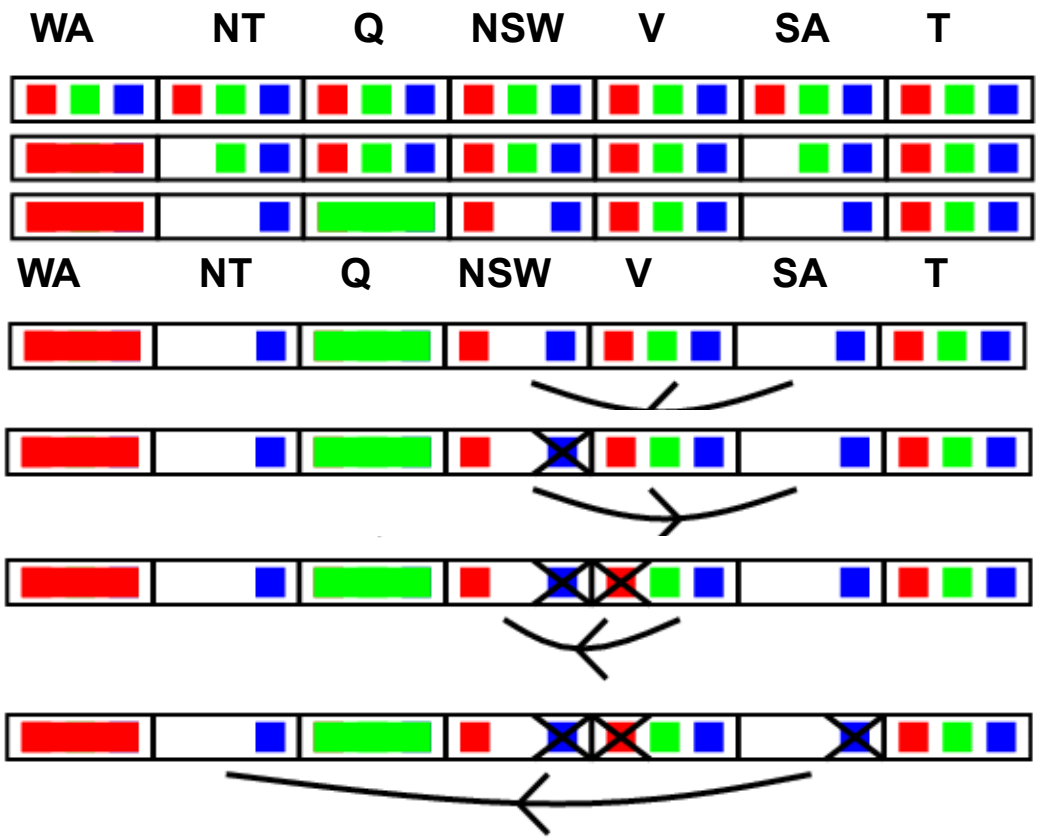
- **Arc-Consistent:** when for each pair of variables variables (X, Y) and for each value x_i of D_x there exists a value y_j of D_y such as Constraints are satisfied
  - *Current domains must be consistent with all the constraints*

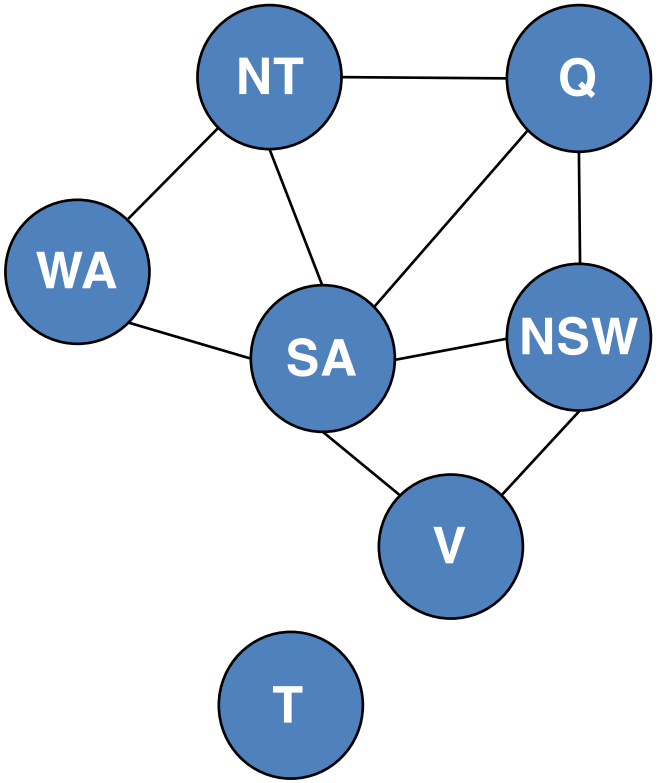
3.C ARC-CONSISTENCY

(X Y) Is consistent if and only if:

- For each value x_i of X there exists some allowed value y_j

- WA=red Q=green





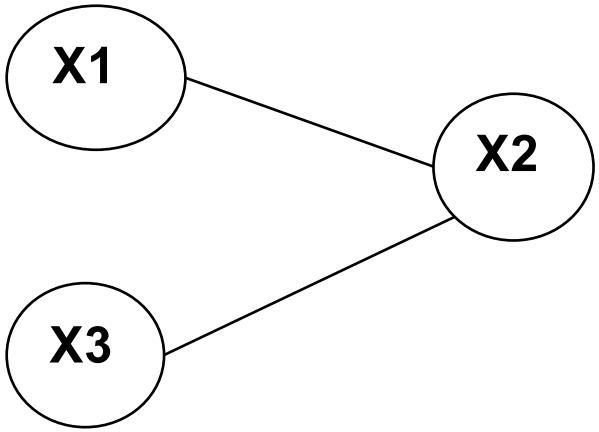
3.C Arc-consistency

D1={1..10}

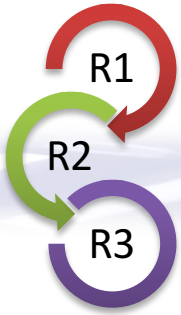
D2={5..15}

D3={8..15}

- X1>X2
- X2>X3



3.C Algoritmo AC3



Procedure (intuitive idea):

From initial domains:

- Update domain in each step
- Return a set of updated domains where all arcs are consistent

Domains update:

- If an arc is inconsistent, try to remove from the distinguished variable those values that do not satisfy any constraint

Stop criteria

- All arcs are consistent
- Inconsistency: A domain is empty

3.C AC3 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 we remove a value**
removed ← false
for each x **in** DOMAIN$[X_i]$ **do**
if no value y **in** DOMAIN$[X_j]$ allows (x,y) to satisfy the constraint between X_i and X_j
then delete x from DOMAIN$[X_i]$; removed ← true
return removed

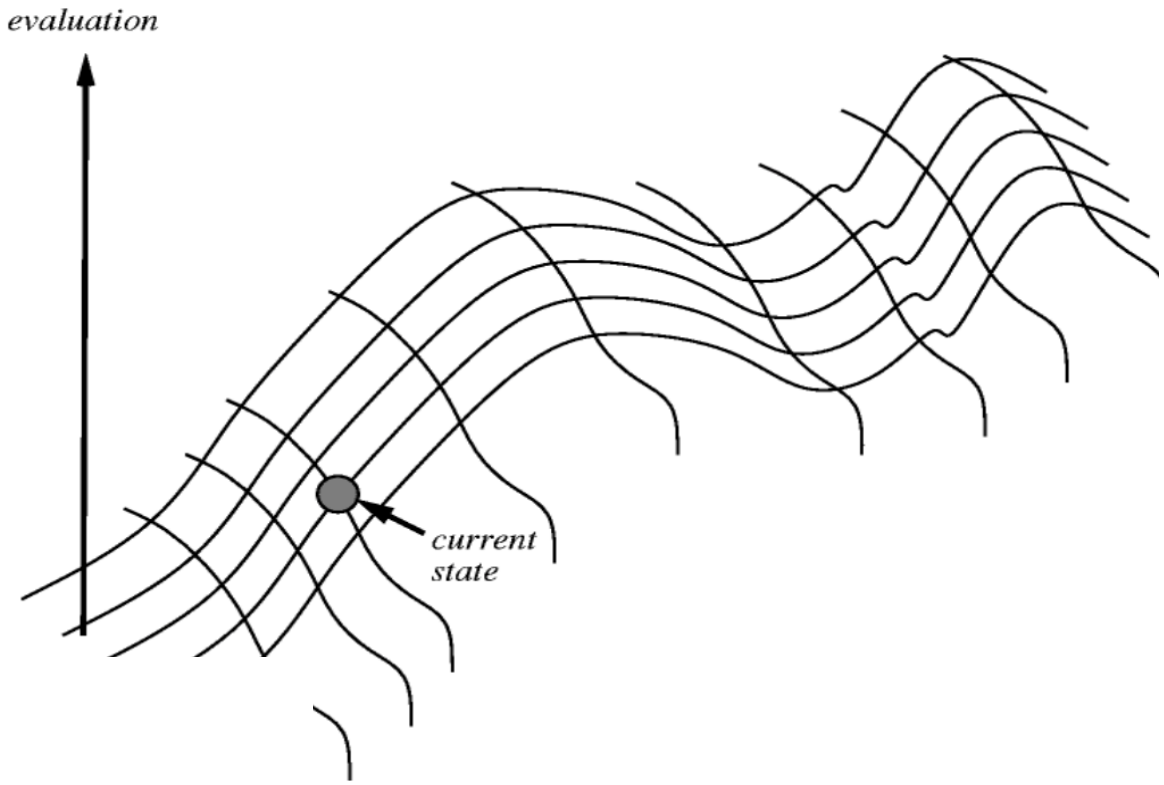
3.C Algoritmo AC3

- From the application of AC3 we can end with:
  - An empty domain: no solution
  - Unique domain: a unique solution
  - At least a domain is not unique: more than one solution might exist

- AC3 can be used in combination with any other search strategy:
  - Backtracking and backjumping
  - Local search and heuristcs of minimum conflicts

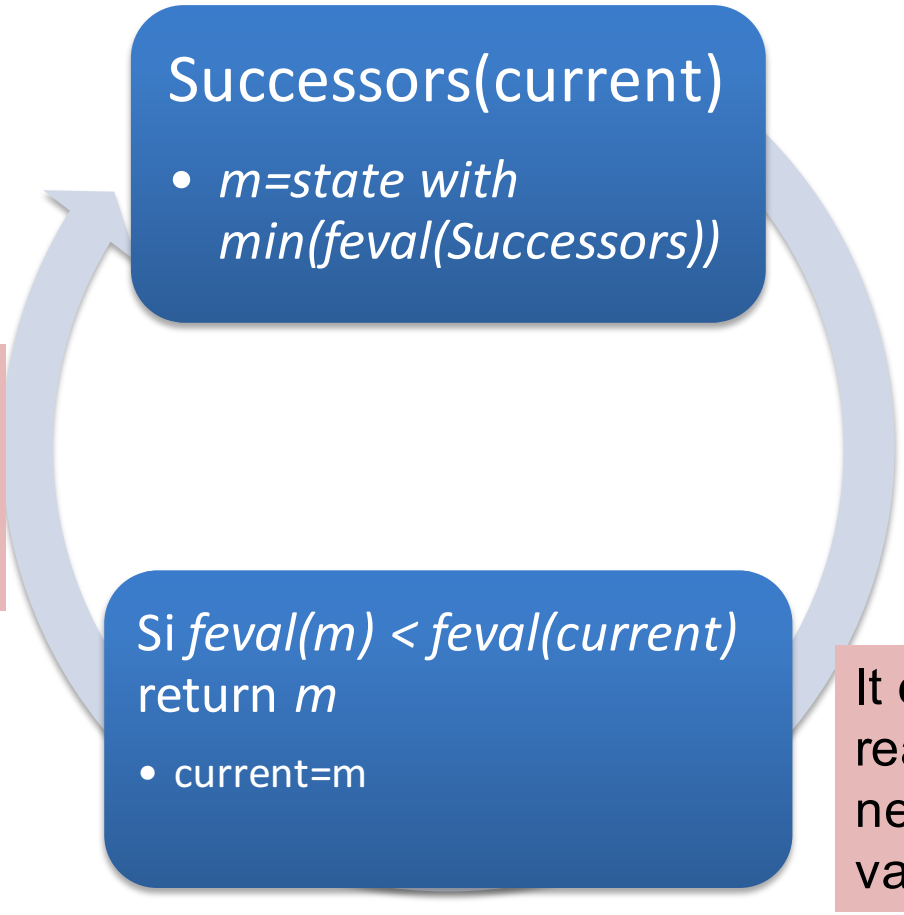
4. Hill-Climbing Algorithms

- Loop that constinously moves forward:
  - Increasing values (if the goal is to maximize the evaluation function)
  - Decreasing values (if the goal is to minimize the evaluation function)



4.1 Hill-climbing search

```
neighbor = successor of actual with min(feval)
if feval(neighbor) < feval(current)
        current = neighbor
```

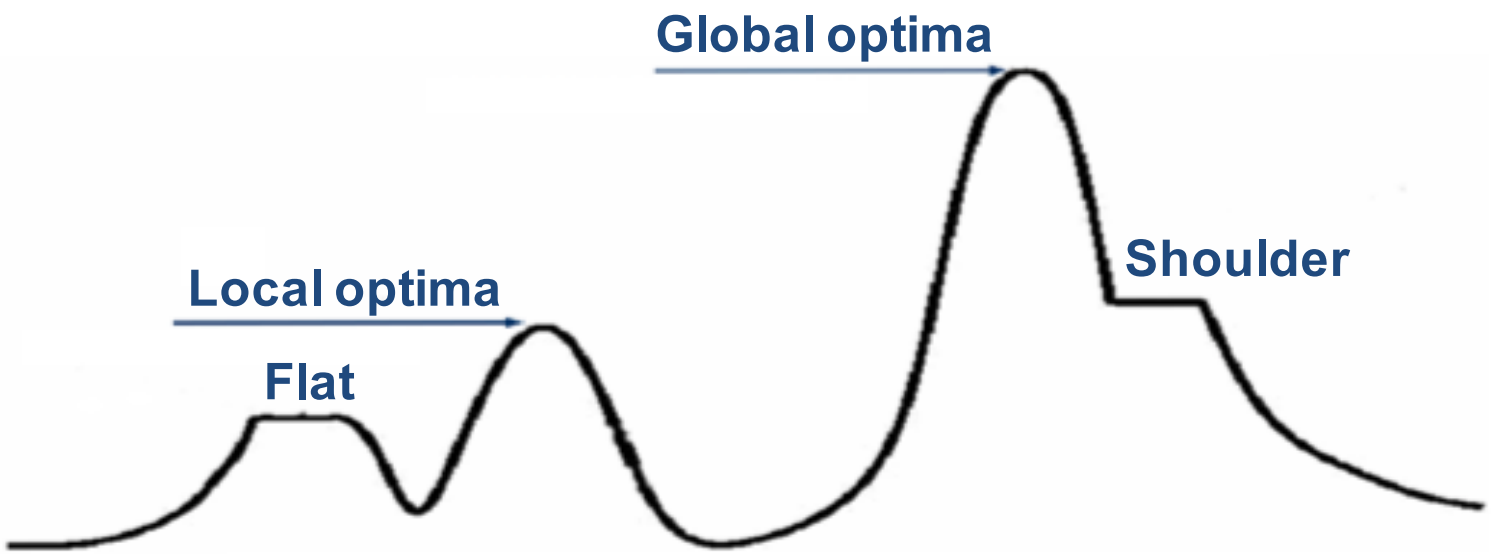


Continue the trace through states that decrease the evaluation function

It ends when a peak is reached: where none neighbor has a lower value of feval

4.2 Problems with local search

It can get stuck in local optima …



4.3 Alternative: Beam local search

- Begin with k random generated states
- Loop until the solution state is found
  - Generate the list of all the successors of the k States.
  - Select the k best states from this list

4.3 Hill-Climbing for CSP

- **States:** They use a complete-state formulation (consistent or inconsistent)
- **Initial State:** random generated
- **Final State:** Solution to CSP
- **Successors:** usually works by changing the value of one variable at a time

5. Summary

When the path to the Solution is irrelevant:

- They keep only one State in memory: the current State
- They move only to the neighboring nodes of the current node
- They are not systematic in the search
- They use little memory
- They can find reasonable solutions in large or infinite spaces of States
- They can get stuck in local maxima/minima

Bibliografía

- Russell, S. y Norvig, P. *Inteligencia Artificial (un enfoque moderno) (Pearson Educación, 2004). Segunda edición.* Cap. 5: "Constraint Satisfaction Problems"
- Nilsson, N.J. *Inteligencia artificial (una nueva síntesis)* (McGraw–Hill, 2000). Cap. 11 "Métodos alternativos of búsqueda y otras aplicaciones"
- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach) (Oxford University Press,* 1998) "Constraint Satisfaction Problems"

Transparencias

- Luigi Ceccaroni – Universidad Politécnica of Cataluña
- José Luis Ruiz Reina et al. – Universidad of Sevilla