CHAPTER III

Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems (CSP)

Definition

Constraint Programming (CSP: Constraint Satisfaction Problems) is situated at the intersection of Artificial Intelligence and Operations Research. It focuses on problems defined in terms of constraints of time, space, etc. or more generally resources:

Applications :

- **Planning and scheduling problems:** planning production, managing rail traffic, etc.
- **Resource allocation problems:** establishing a timetable, allocating memory space and CPU time (by an operating system), assigning persons to tasks, warehouses to goods, etc.
- **Optimization problems:** routing problems in telecommunications networks, ...

Constraint Satisfaction Problems (CSP)

Definition

Solving a CSP problem can be seen as a special case of heuristic search:

- The internal structure of the nodes has a particular representation:
  - A node is a set of **variables** with corresponding **values**
  - Transitions between nodes take into account **constraints** on the possible **values** of **variables**
- We use **general heuristics** rather than application-specific heuristics:
  - In a CSP problem We eliminate the difficulty of defining a specific heuristic **h** for our application

Example

- **Resources allocation**
  - EXP: Establishing a timetable
  - **Variables:** The different time slots for all premises (Classrooms, Amphitheaters, Labs, …)
  - **Constraints:** only one class is assigned to the same classroom at a given time, no group has two classes at the same time,…

Constraint Satisfaction Problems (CSP)

Formal definition

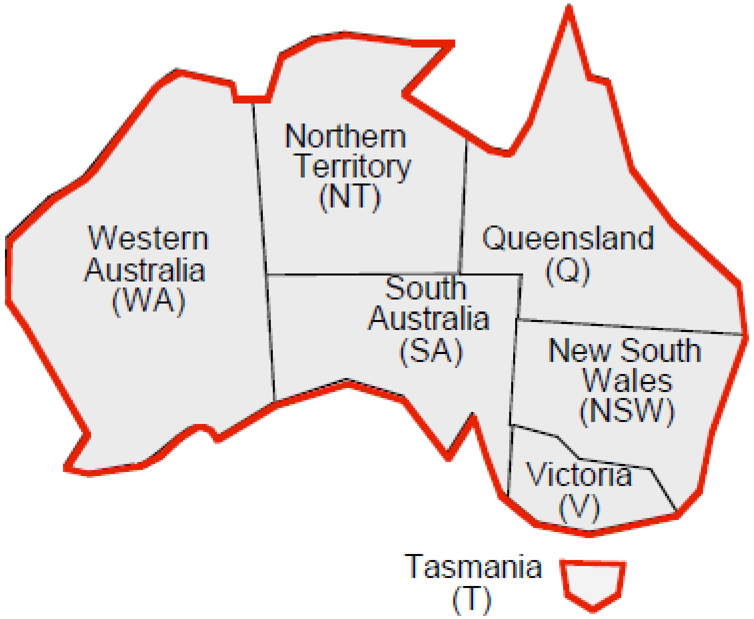
- A finite set of variables $\mathbf{V = \{X_1,…,X_N\}}$
  - Each variable $\mathbf{X_i}$ has a domain $\mathbf{D_i}$ of possible values
- A finite set of contraintes: $\mathbf{C = \{C_1,…,C_M\}}$
- A **state** of a CSP problem is defined by an **assignment** of values $\mathbf{\{X_i=v_i, X_j=v_j, …\}}$
  - An assignment that **does not violate** any constraint is said to be **consistent**
  - An assignment is said to be **complete** if it assigns **values to all variables**
  - A CSP solution is a **complete** and **consistent** assignment

Example 1

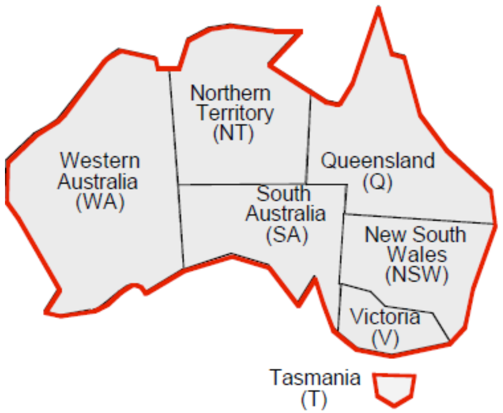
- Consider the following CSP problem:
  - V = {X1,X2,X3}
  - D1 = D2 = D3 = {1,2,3}
  - Constraint: X1+X2 = X3
- Three possible solutions (Complete and Consistent assignments) :
  - {X1=1, X2=1, X3=2}
  - {X1=1, X2=2, X3=3}
  - {X1=2, X2=1, X3=3}

Example 2: Map coloring

- Consider the Australia map coloring problem:
  - We have to use only three colors (Red, Green, Blue) so that two border states never have the same colors



Example 2 : Map coloring

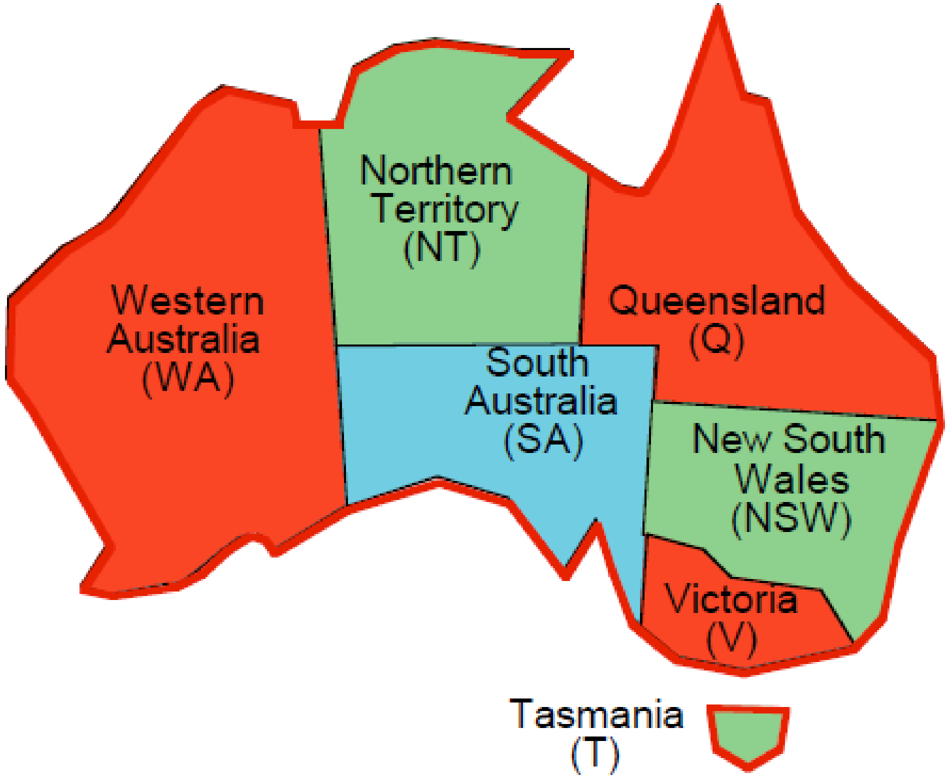


- **CSP Formularization :**
  - The variables are the states of Australia:
    - V = {WA, NT, Q, NSW, V, SA, T}
  - The domain of each variable is the set of three colors:
    - D = {R, G, B}

  - The constraint: Two border states have to be colored with different colors:
    - WA ≠ NT, WA ≠ SA, NT ≠ Q, …

Example 2: Map coloring

- **Complete and consistent solution:**
  - WA = R, NT = G, SA = B, Q = R, NSW = G, V = R, T = G



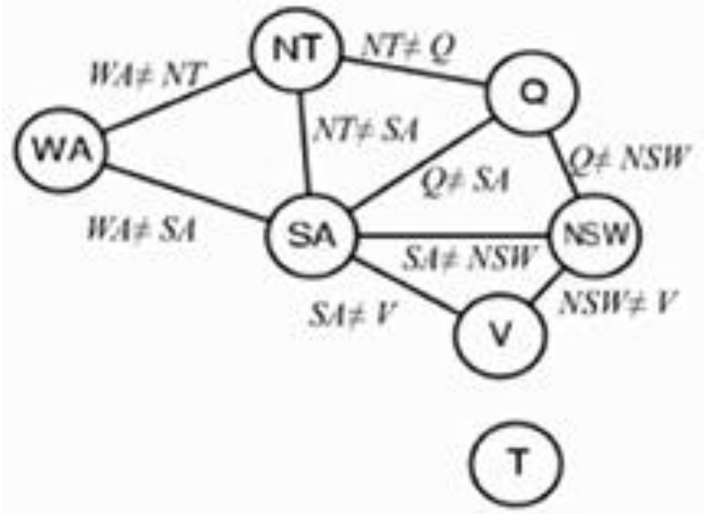
Types of constraints

A constraint is characterized by its arity (number of variables it involves):

- **Unary:** Constraints only concern one variable.
- **Binary:** Constraints involve two variables.
- **Multiple:** Constraints involve 3 or more variables.

For problems with **binary** constraints, we can visualize the CSP problem by a constraint graph:

- The nodes are the variables (node = variable)
- Arcs are the constraints between two variables



Depth-First Search for CSP

- **Search parameters:**
  - A state is an assignment
  - Initial state : empty assignment {}
  - Transition function: assigns a value to a variable not yet assigned
  - Goal function: Returns True if the assignment is complete and consistent

- This algorithm is general and can be applied to all CSP problems
- The solution must be complete (it appears at a depth N)

Depth-First Search for CSP

- **Limitations :**
  - **Level 1** of the tree : **N*D** branches (each variable can take D values)
  - **Level 2 : (N-1)*D** branches for each node (so on until level N)
  - **Result : N!*D^N** nodes for only **D^N** complete assignments
- Solution 1 : Consider **only one variable** to assign at each level
- Solution 2 : **Backtrack** when no new consistent assignment is possible (no point continuing to assign variables if there is already a constraint violation)

Solution 1 + Solution 2 = Backtracking search Algorithm

Constraint Satisfaction Problems (CSP)

Backtracking Search Algorithm

Algorithm Backtracking-search(csp)

Variables, domains, constraints

    Return **Backtrack**({},csp)

Backtrack(assignment,csp){

Assignment of variables

1. If assignment is complete, return assignment
2. Else X=**Non-Assigned-Var**(assignment,csp)

   Choosing the next variable

3. For each **v** in **Ordered-Values**(X,assignment,csp)

   Order of values to try

   1. If **consistent** ((X=v), assignment, csp)
      1. Add (X=v) to assignment
      2. csp* = csp but where **Domain**(X, csp) is {v}
      3. csp*, ok = **Inference**(csp*)

         Try to simplify the CSP problem

         If it detects a conflict ok = false

      4. If ok = true
         1. Result = **Backtrack**(assignment, csp*)
         2. If Result ≠ false, return Result
      5. Else Remove (X=v) from assignment
4. Return false

}

Backtracking Search Algorithm

Description

- The variables of our problem are instantiated with their domain values, in a specific order, until one of these choices does not satisfy a constraint. In this case, we must question the last instantiation carried out. A new value is tried for the last variable instantiated (which we call the current variable).

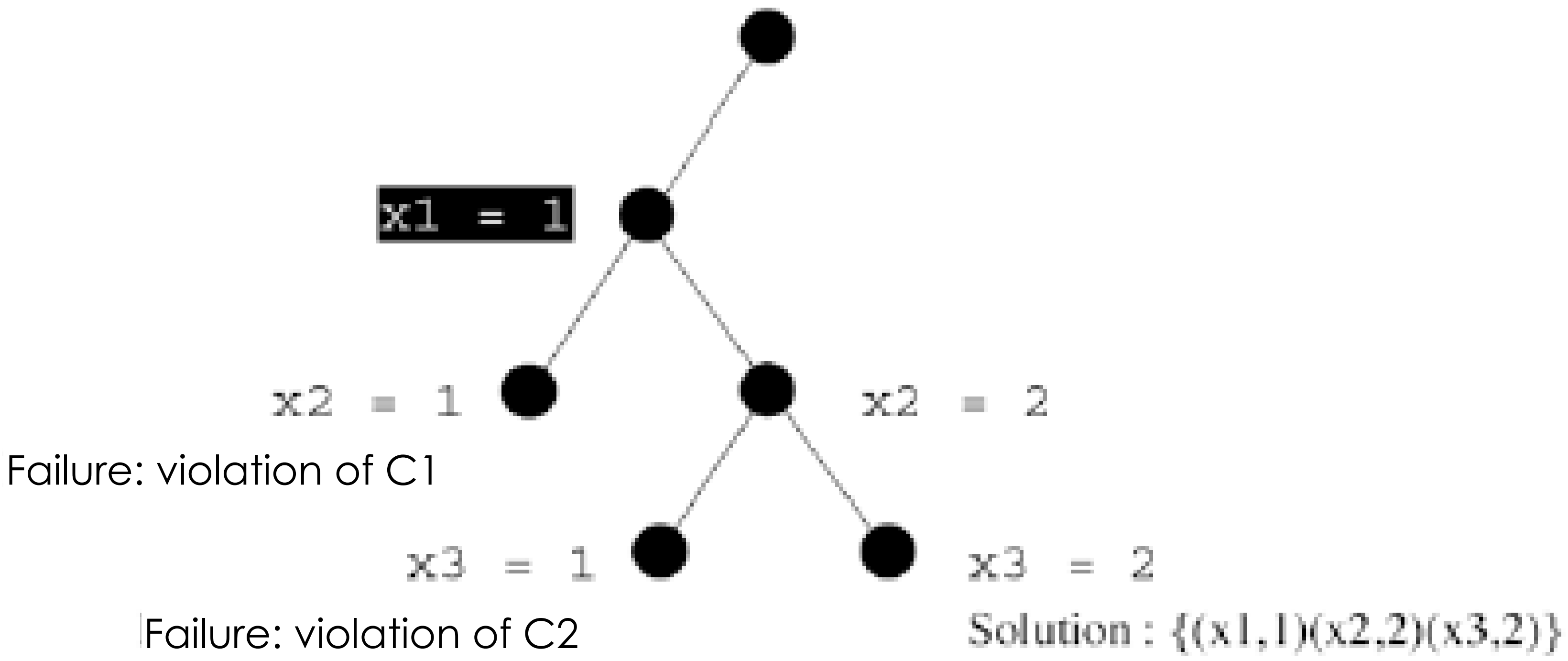
- If all the values in the domain of this variable have been tested without success, we must carry out a backtrack: We choose another value for the variable immediately preceding the current variable. We repeat this process until we obtain a solution (that is to say an instantiation of all the variables).

- If we have gone through the entire search tree without finding it, then we have proven that the problem has no solution.

Backtracking Search Algorithm

- **Illustration 1:**

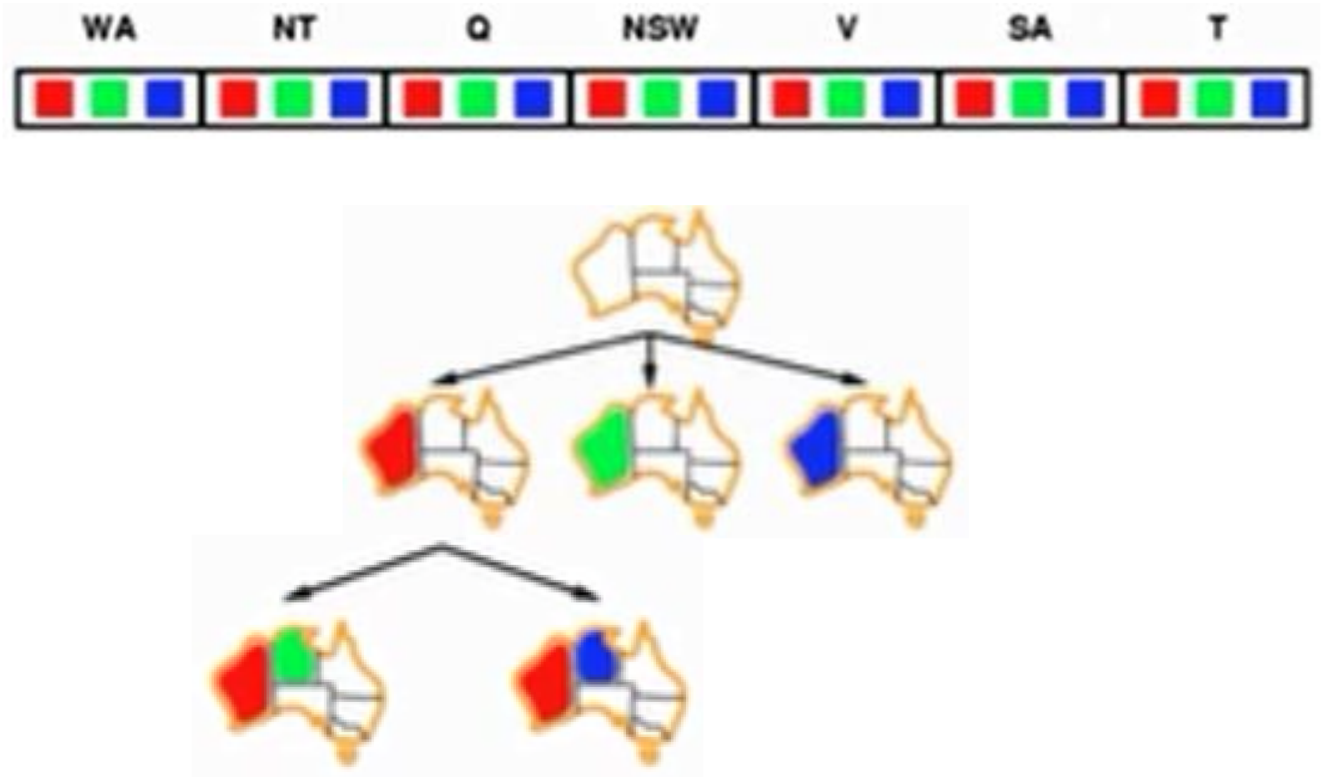
  – Consider x1, x2 and x3 three variables,
  – Consider D(x1) = D(x2) = D(x3) = {1, 2, 3} their domains.
  – We put the following constraints: C1 = [x1< x2] and C2 = [x2 = x3].
  – We assume that we instantiate the variables in ascending order of indices, by choosing the smallest value first. x1 = 1



Constraint Satisfaction Problems (CSP)

Backtracking Search Algorithm

- **Illustration 2:**



Exercise 1:

- Consider the following CSP problem :
  - V = {X1,X2,X3}
  - D1 = D2 = D3 = {1,2,3}
  - Constraint: X1+X2 = X3

Exercise 2:

We want to solve the 4-Queens problem using the Backtracking-Search Algorithm:

- Trace the Backtracking–Search tree in order to find a solution

(We assume that we instantiate the variables in ascending order of indices, by choosing the smallest value first.)

Backtracking Search Algorithm

- **Improvements:**
  - Filtering and propagation
  - Use of general heuristics
    - Choosing next variable (Non-Assigned-Var)
    - Choosing next value to assign (Ordered-Value)
    - Detect conflicting assignments and reduce domains (Inference)

Constraint Satisfaction Problems (CSP)

Backtracking Search Algorithm

- **Filtering and propagation**
  - Delete the values of variables domains involved in a constraint (Avoid traversing branches which cannot lead to a solution)

- **Example**

– Consider the variables X1, X2 and X3,

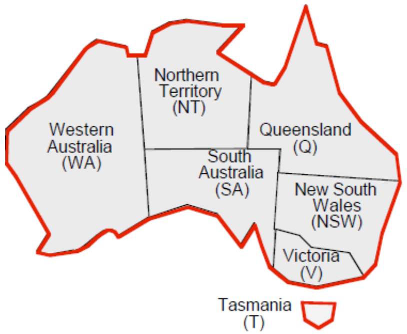
– Consider D(X1) = D(X2) = D(X3) = {1, 2, 3} their domains,

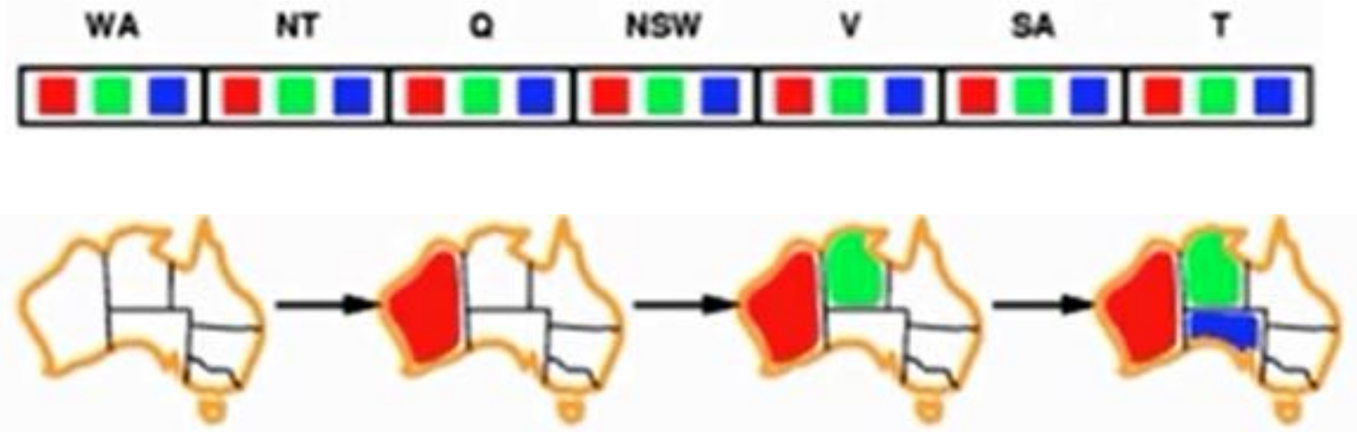
– Consider the constraints C1 = [X1 < X2] and C2 = [X2 = X3]

- **Filtering:** The value **3** can be **deleted** from **D(X1)**, because there is **no value** in **D(X2)** such that **C1** will be satsified if We instantiate **X1** by **3** (and then **1** can be **deleted** from the **D(X2)** with the same way)
- **Propagation:** Filtering **1** from **D(X2)** relatively to **C1** can be propagated to **D(X3)**: if the value **1** is no longer belonging to **X2**, then it can be removed from **D(X3)** because there will not be any solution to **C2** such that **X3** is instantiated with **1**.

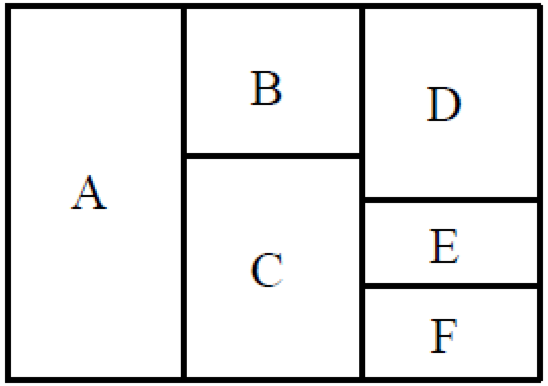
Backtracking Search Algorithm

- **Heuristic 1: Choosing the order of variable assignment**
  - **Minimum Remaining Value (MRV) heuristic**
    - At each step, choose the variable with the fewest remaining consistent values





- **Heuristic 1: Choosing the order of variable assignment**
  - **Minimum Remaining Value (MRV) heuristic**
    - **Exercise:** Use MRV to color the zones in the following shape:



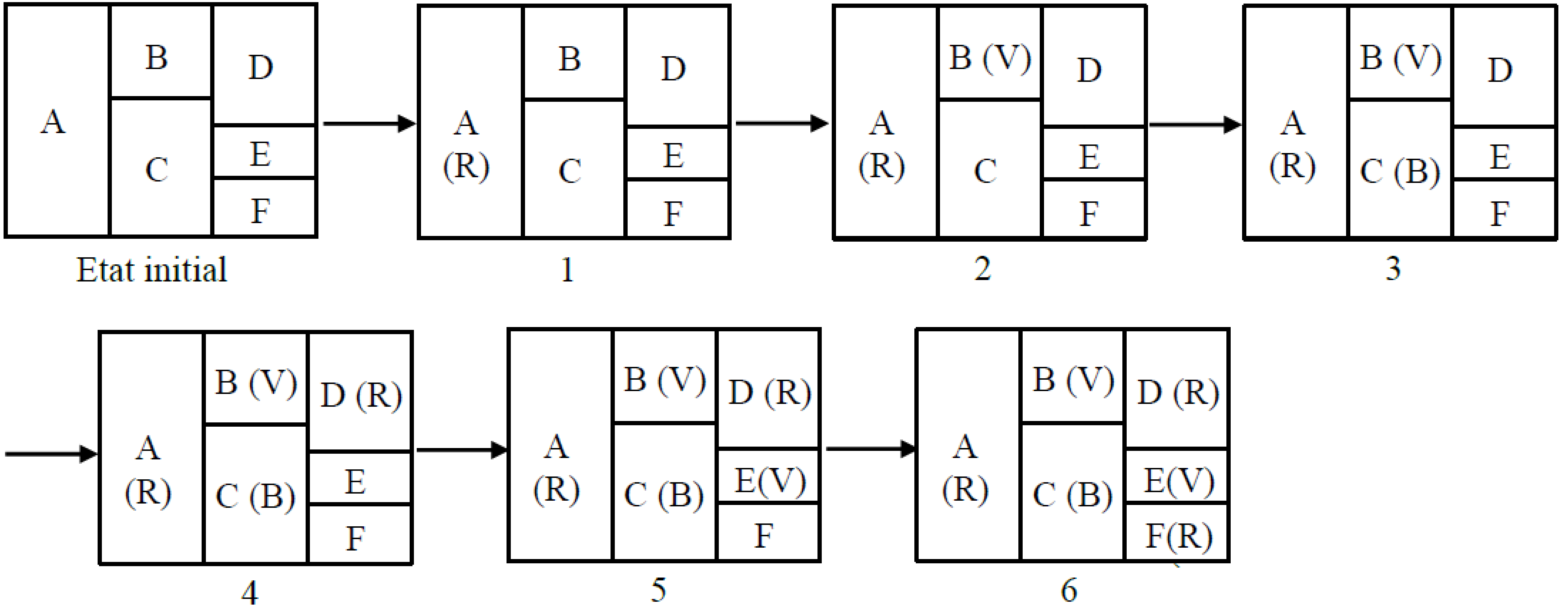
    - Constraint: Using only three colors (R,G,B), two neighboring zones never have the same color.
    - In case of conflict, choose by order of color indices and alphabetocal order of zones.

Constraint Satisfaction Problems (CSP)

MRV

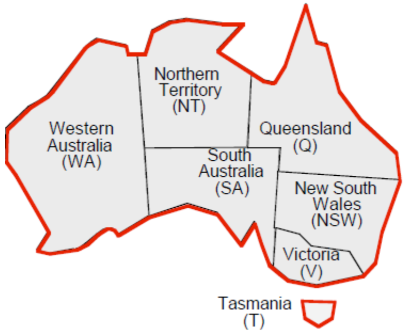
Backtracking Search Algorithm

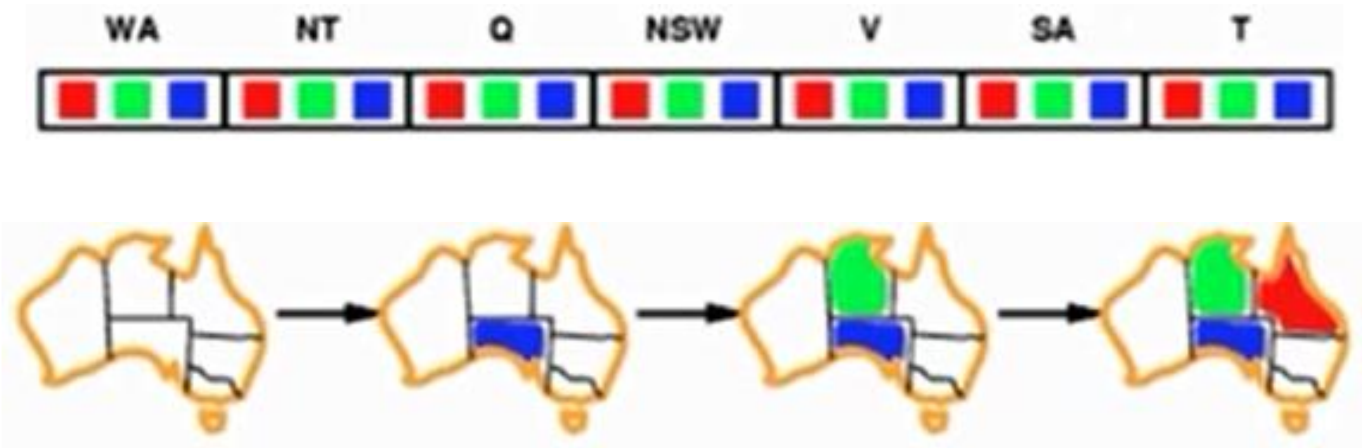
A			B			C			D			E			F		
R	V	B	~~R~~	V	B	~~R~~	~~V~~	B	R	~~V~~	~~B~~	~~R~~	V	~~B~~	R	~~V~~	~~B~~
			1			1	2			2	3	4		3		5	3



Backtracking Search Algorithm

- **Heuristic 1: Choosing the order of variable assignment**
  - **Degree heuristic**
    - Choose the variable with the most constraints involving variables not yet assigned (if the previous heuristic gives the same number of consistent values)

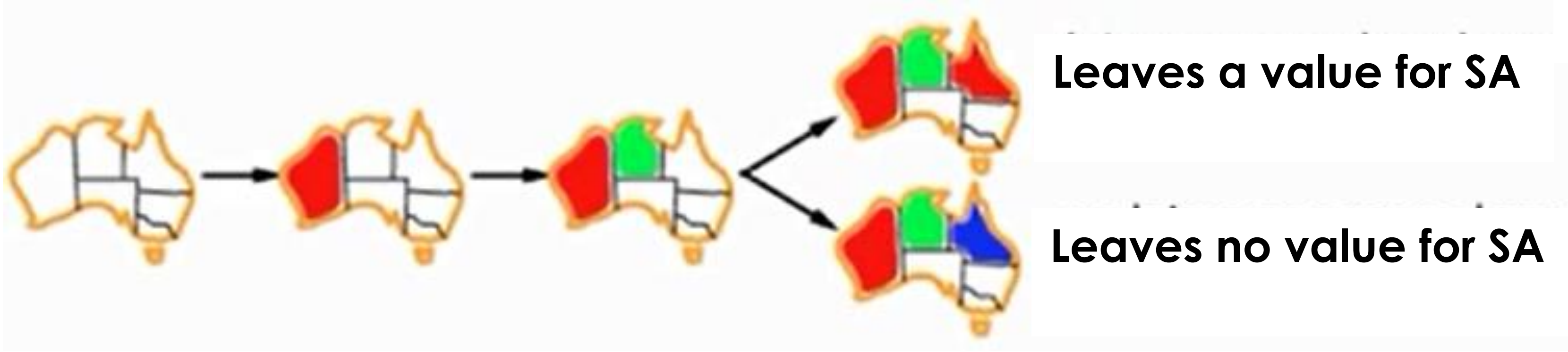


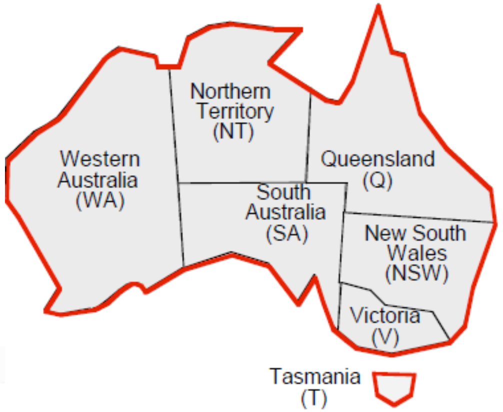


Backtracking Search Algorithm

- **Heuristic 2: Choosing next value to assign**
    - **Least constraining value**

Choose a value that invalidates the fewest possible values for variables not yet assigned
(Choose the value that will remove the fewest choices for neighboring variables)





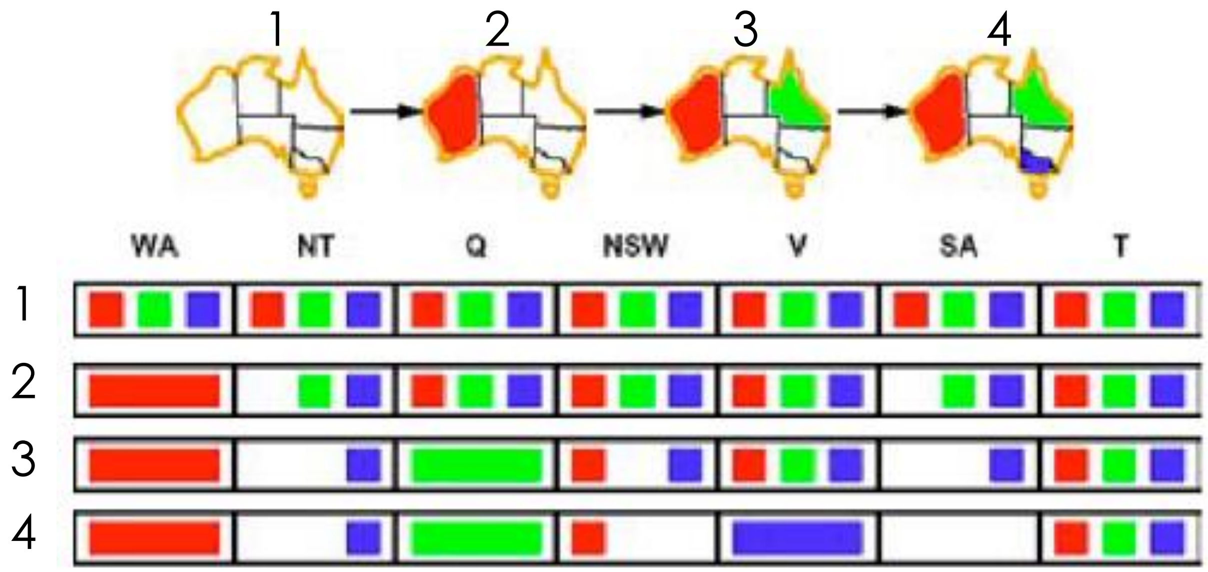
Constraint Satisfaction Problems (CSP)

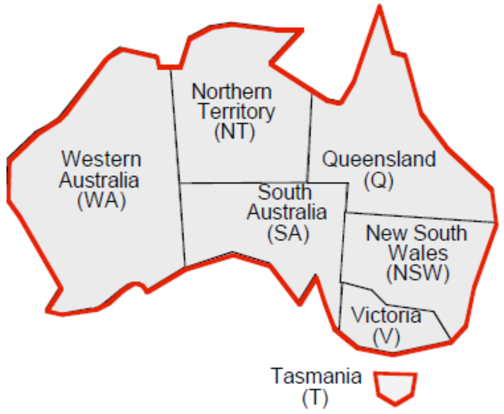
Forward checking Algorithm

- **Heuristic 3: Detect conflicting assigments**

Forward checking:

- Before assigning a value **v** to a variable, check that **v** is consistent with the following variables, i.e. that there exists at least one value for each following variable that is consistent with **v**. (unlike **backtrack**, which checks that the value of the current variable is consistent with the values assigned to previous variables).





Forward checking Algorithm

Algorithm Forward-Checking(X,csp)

For each X_k in neighbors(X,csp)

    Changed, csp = Revise(X_k,X,csp)

    if Changed and Domain(X_k,csp) is empty **Then** return(void,false)

    else return(csp,true)

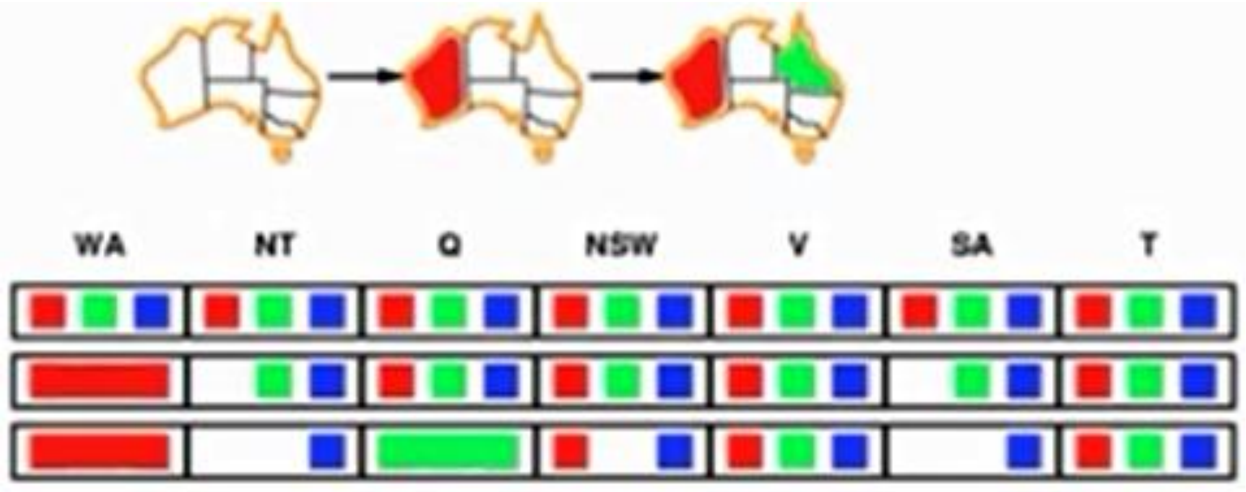
Revise(X_i,X_j,csp){ *//reduce the domain of X_i depending of the domain of X_j*

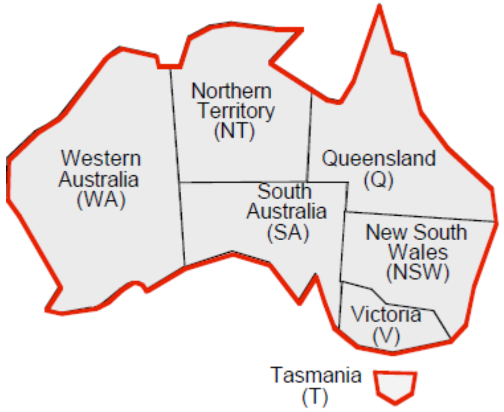
1. Changed = false
2. **For each** x in Domain(X_i,csp)
    1. **if** no y in Domain(X_j,csp) satisfies constraint between X_i and X_j
        1. Remove x from Domain(X_i,csp) *//change the csp*
        2. Changed = true
        3. Return(Changed,csp)

}

AC-3 Algorithm

- ***Forward checking*** propagates information from assigned variables to unassigned variables, but it **does not detect local conflicts** between these variables:

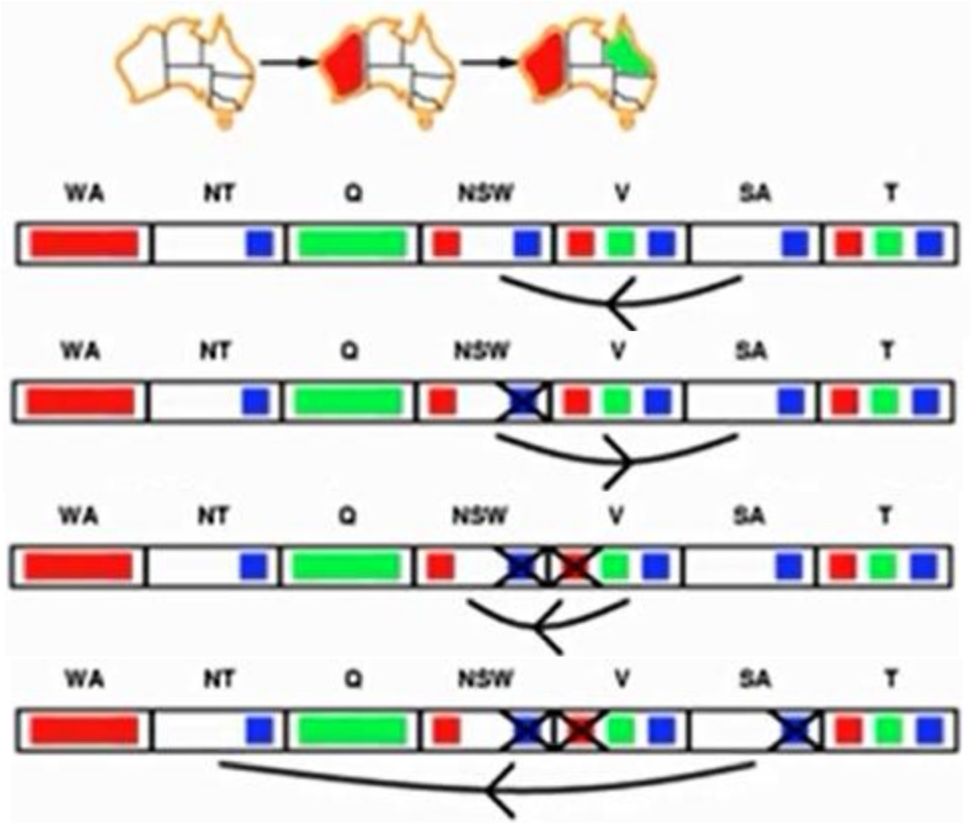


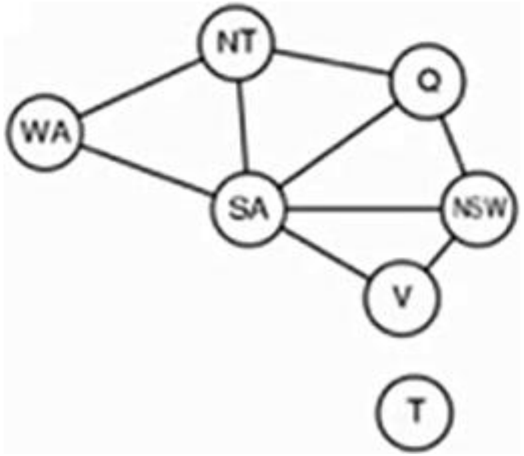


- NT and SA can't be blue together !
- **AC-3** (Arc consistency) allows constraints to be checked locally by using constraint propagation

AC-3 Algorithm

- AC-3 :
  - Check consistency between arcs (consistency of constraints between two variables)
  - The arc X → Y is said to be consistent **only if** : For each value x of X there exists at least a permitted value y of Y





If a variable loses a value, its neighbors must be rechecked

Local search

Local search principle

- The path to the solution is unimportant (e.g.: hill-climbing)
  - We can work with states which are complete assignments (consistent or not)
  - **Disadvantage:** can fall into local optima

- **Min-conflicts algorithm**
  - Objective function: minimize the number of conflicts
  - Looks like hill-climbing but uses stochasticity

Min-conflicts Algorithm

Algorithm Min-conflicts(csp, nb_iterations)

Assignment = a random complete assignment (probably not consistent) of csp

For i = 1 to nb_iterations

1. If assignment is consistent Then return assignment
2. Else X = variable choosen randomly in Variable(csp)
3. v = value in Domain(X,csp) satisfying the most constraints of X
4. Assign (X = v) in assignment

Return false

- Min-conflicts : - Can solve the problem of 1.000.000 Queens in 50 steps

  - Plan Hubble Telescope observations in 10 minutes instead of 3 weeks

Reason for success: there are several possible solutions (scattered) in the space of states