Intelligent Systems

UCA Universidad de Cádiz

Lecture 1
Constraint Satisfaction Problems

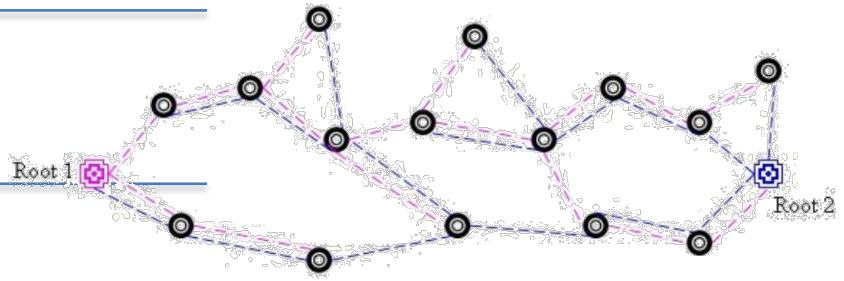
Dra. Elisa Guerrero Vázquez
Dpto. Ingeniería Informática
University of Cadiz

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

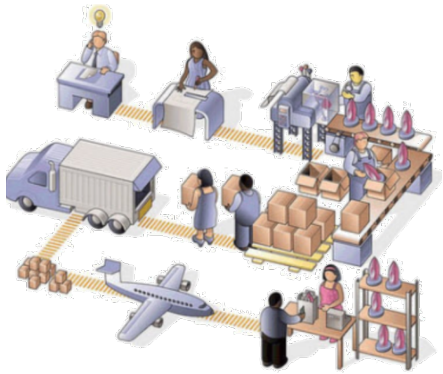


Optimal routing of data in Internet

Minimal cost planning for product shipping

Optimal sequencing in process manufacturing

Task scheduling

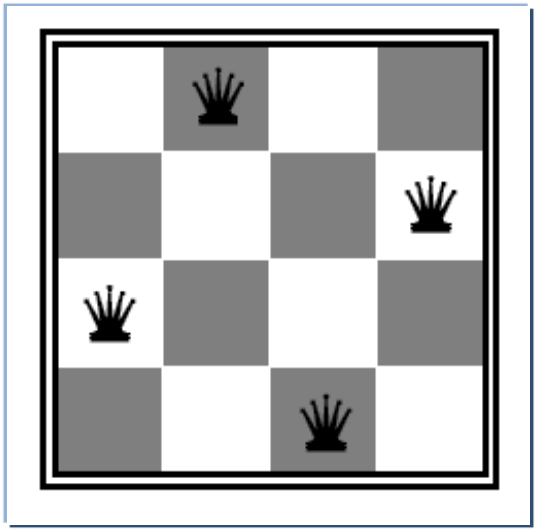


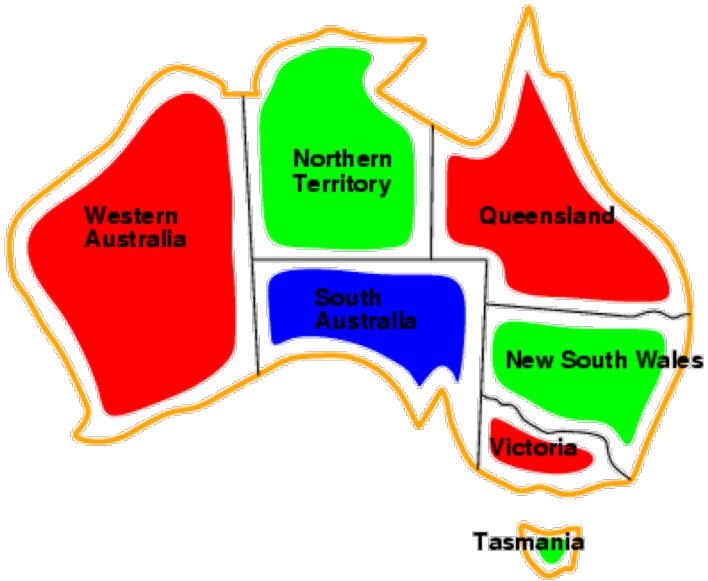
Optimal aircrew selection

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

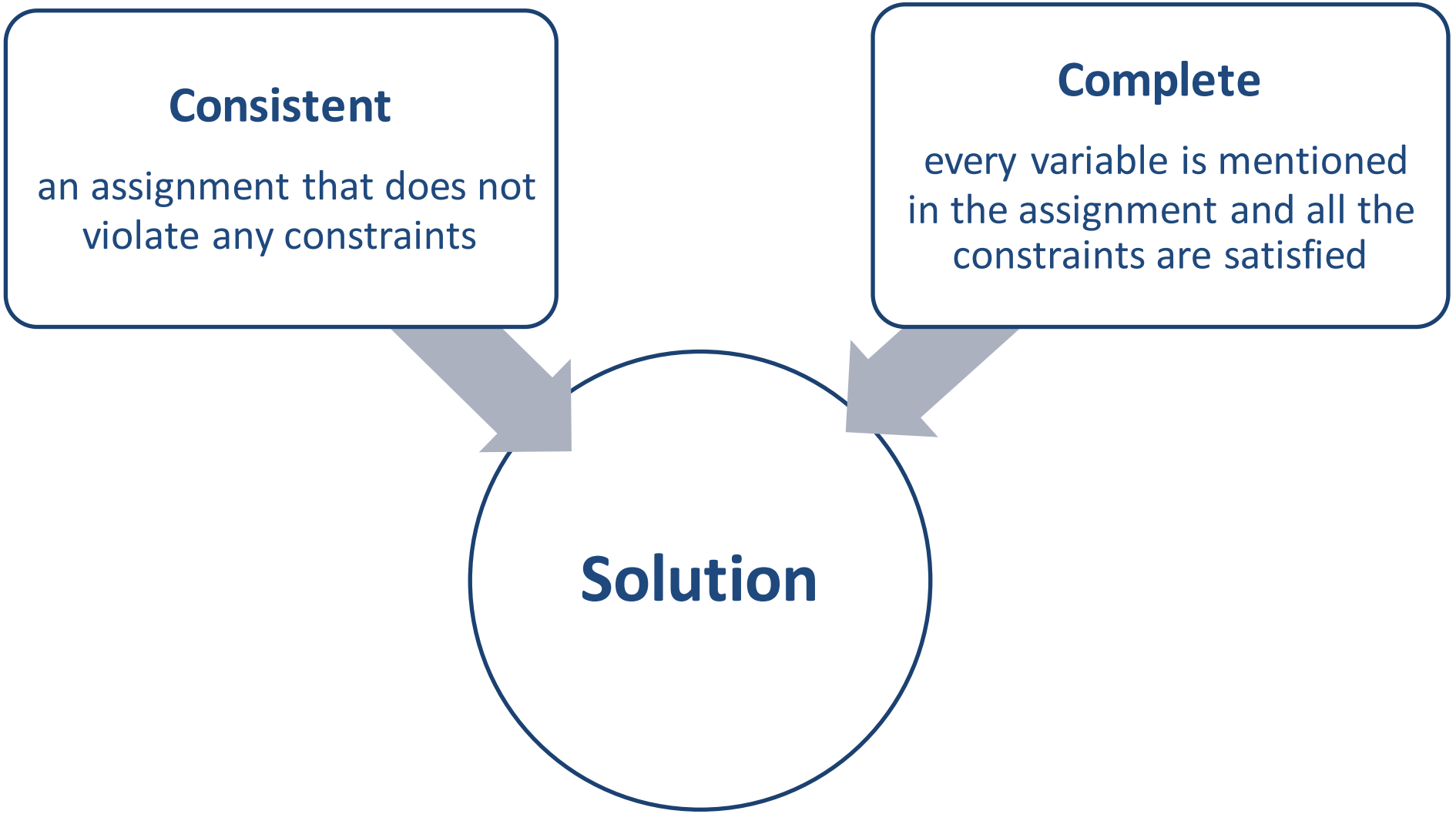
State:

- assignment of values to variables

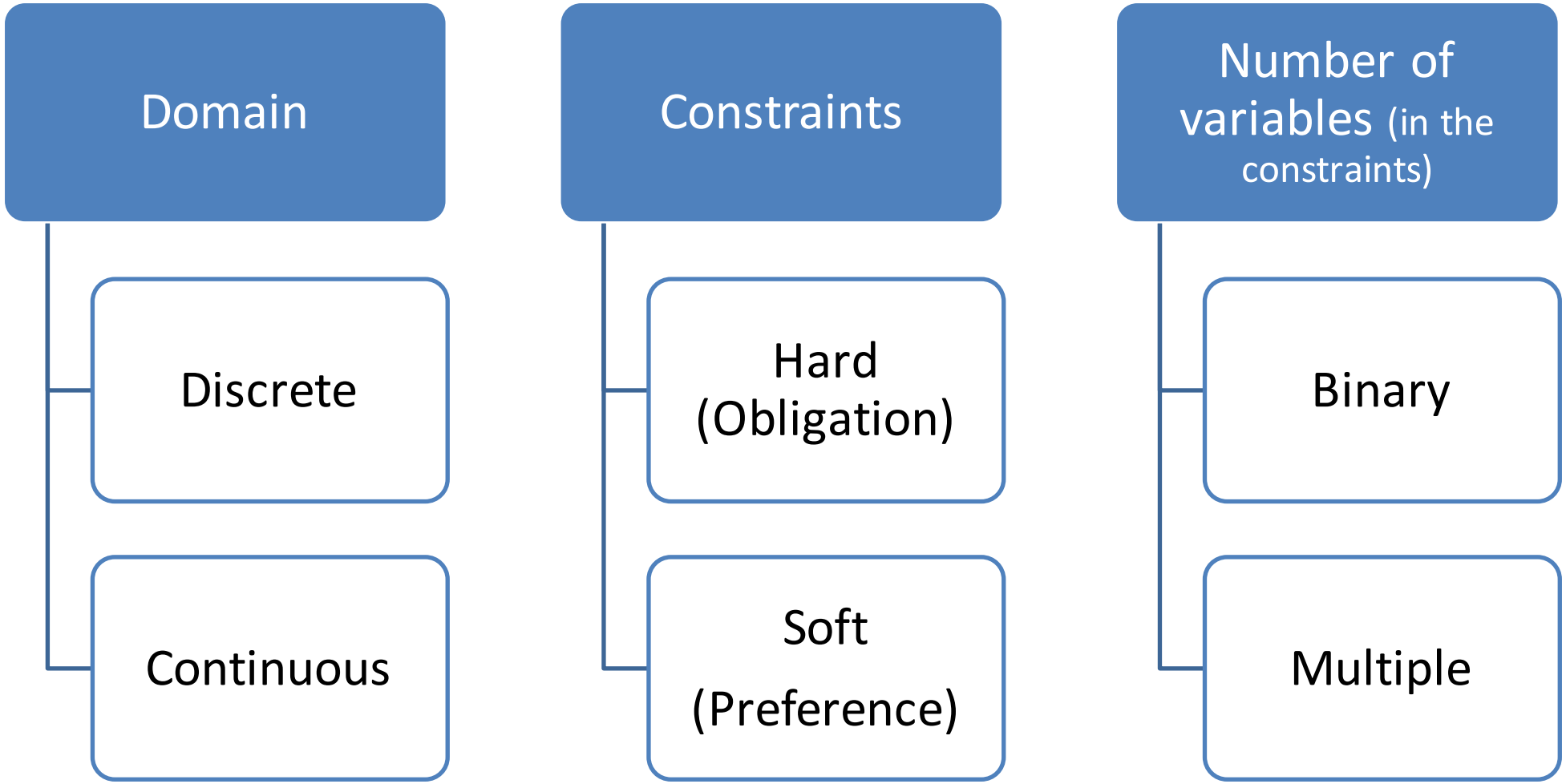
Solution:

- complete assignment that satisfies all the constraints

1.3 Assignments



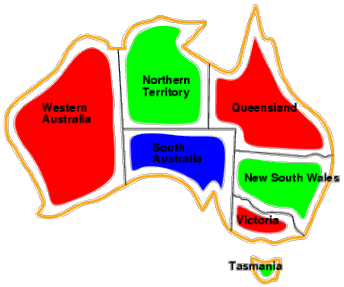
1.4 CSP classification



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

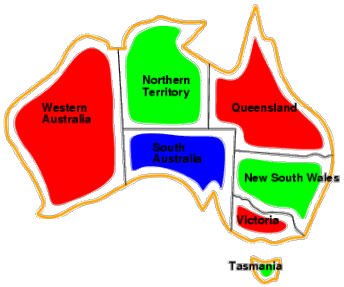
- GOAL: assign different colours to adjacent regions
  - **Variables**: R_1 .. R_7 each region
  - **Domain** of each variable: set of colours {red, green, blue}
  - **Constraints**:
    - Two adjacent regions must have
      - $R_i \neq R_j$ If R_i and R_j are adjace
  - **State**: any assignment, R_1=red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

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

Discrete Domain

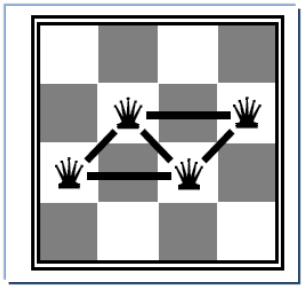
Hard Constraints

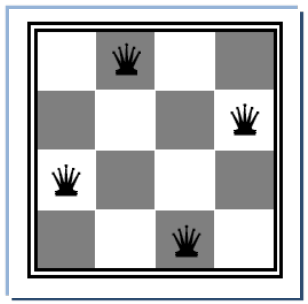
Binary Constraints



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: $X_2 + T + T = O + 10*F$

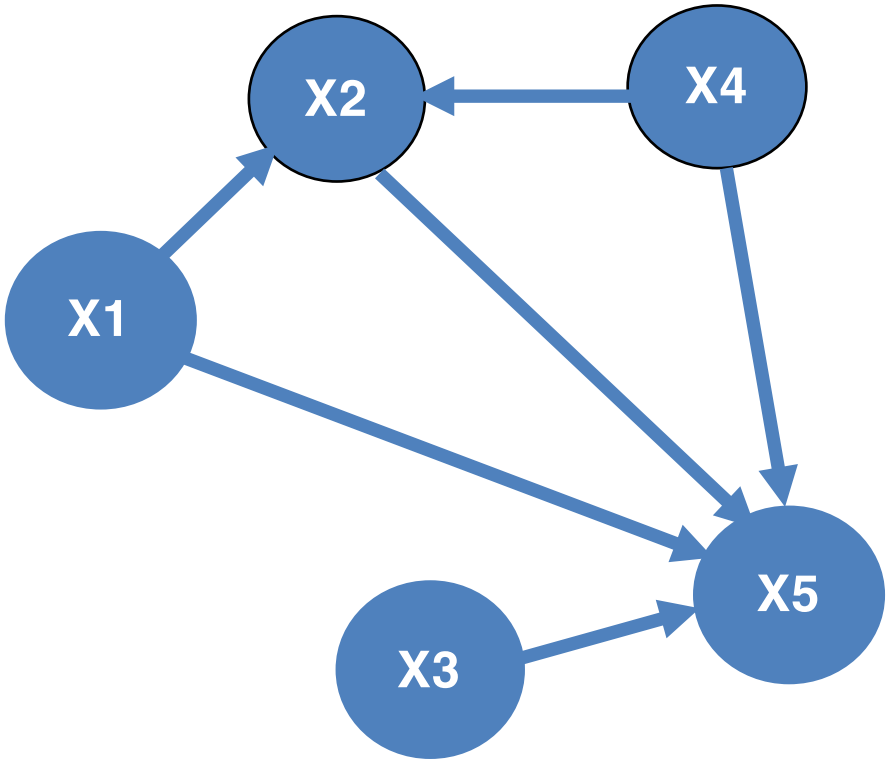
Hard and Multiple Constraints

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

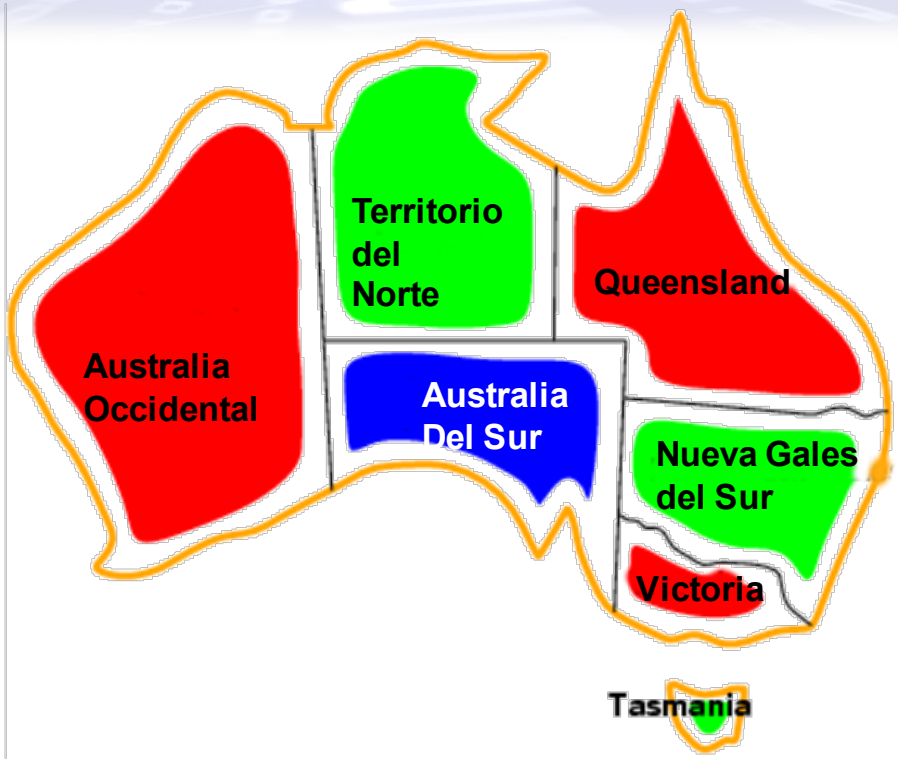
	0	1		
	8	3	6	
	8	3	6	+
1	6	7	2	

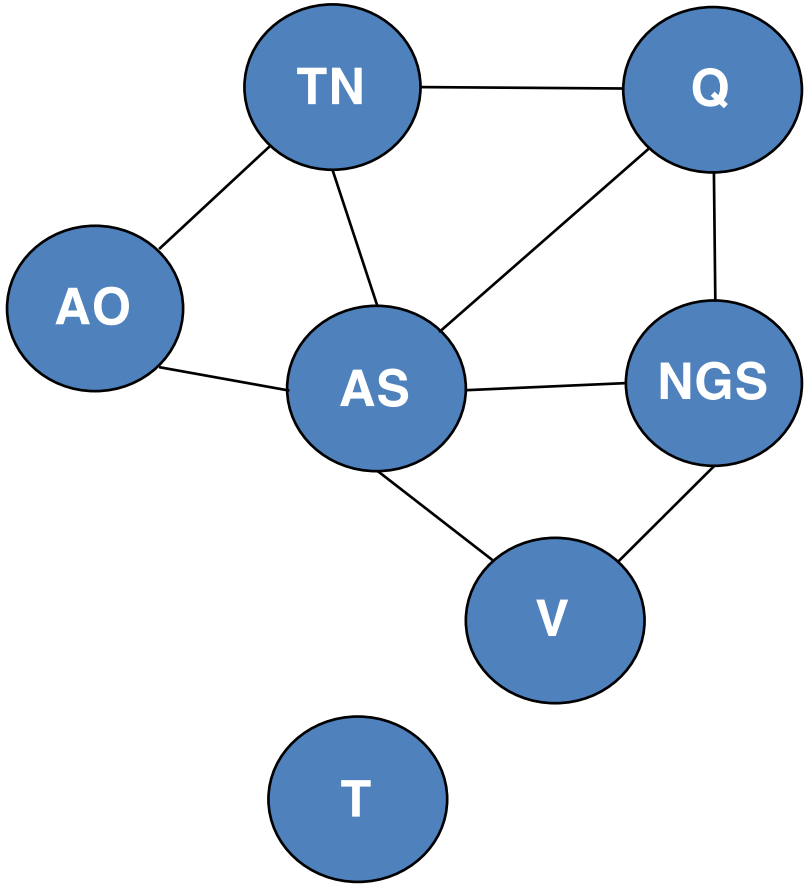
1.6 CSP representation

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

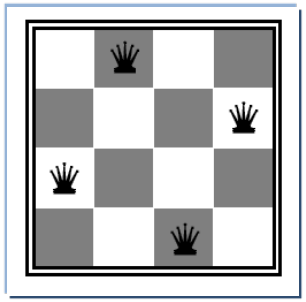


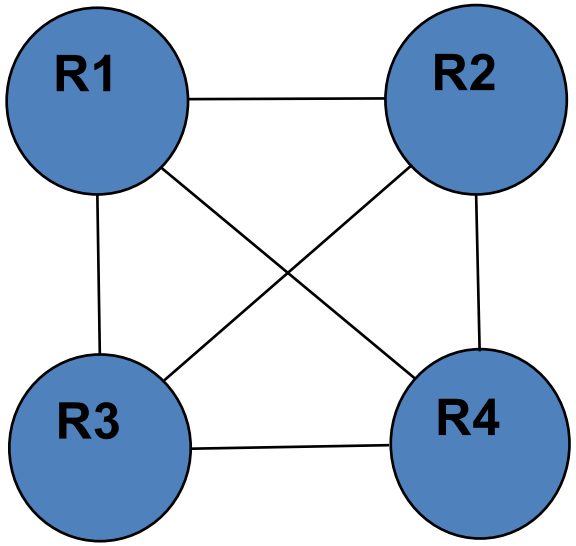
1.6 Binary constraint graph





1.7 N-Queens Example





2. CSP solving

- **Search strategies**

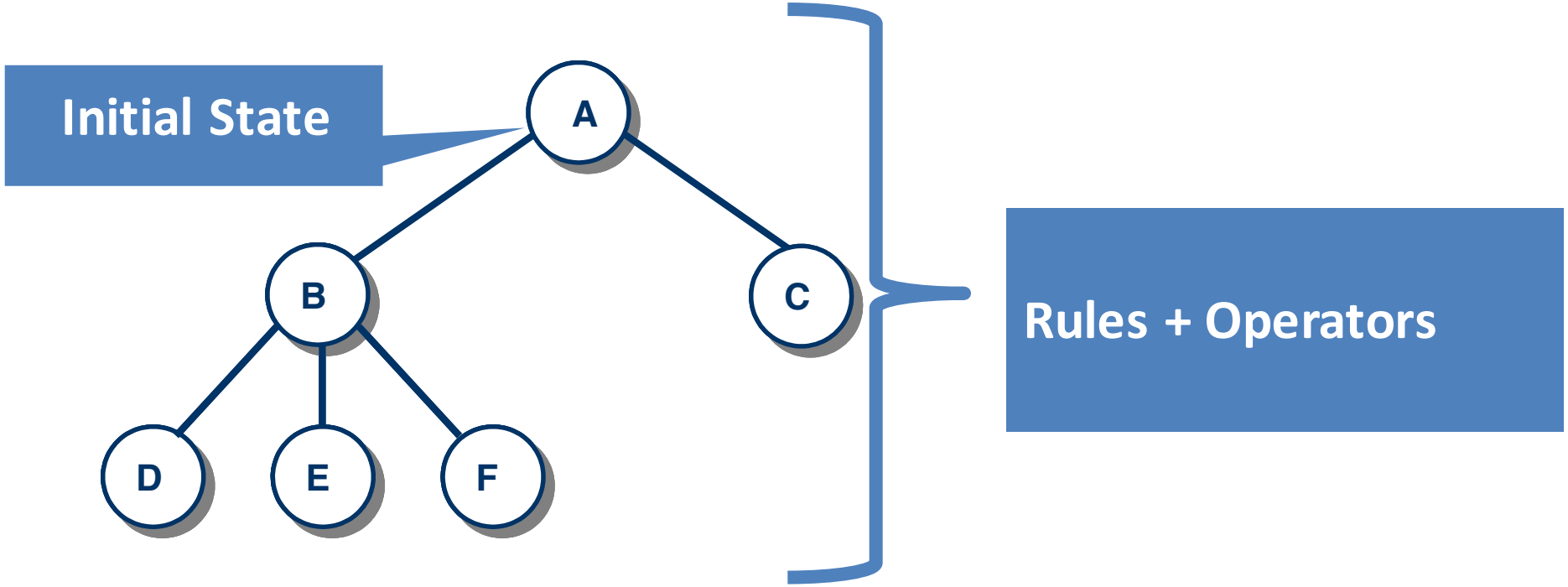
  Systematic search: Exploration of the state space

- **Consistency approaches**

  Inconsistent values are removed from variables domains

  Help to reduce the state space

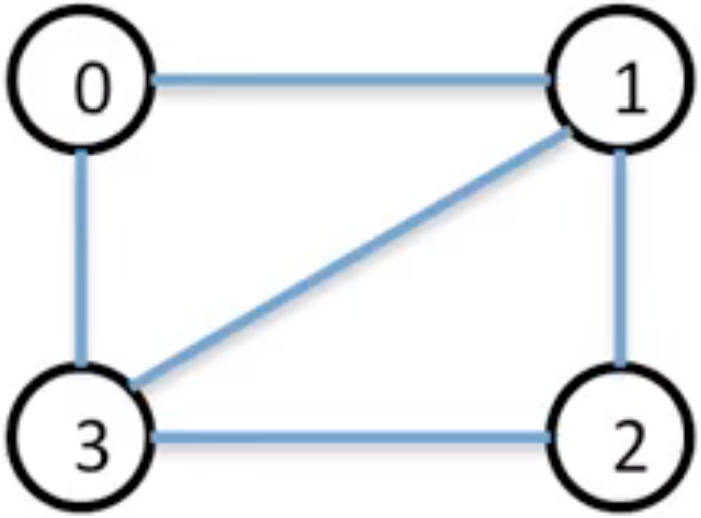
State Space

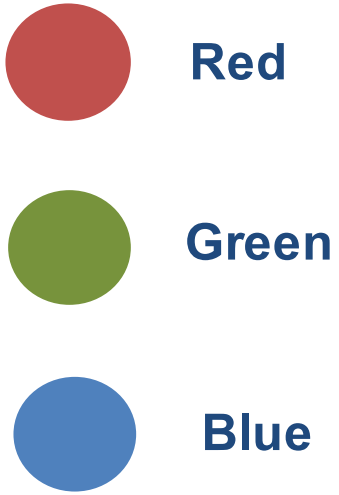


- Goal Test
- Path or Solution
- Solution Cost

2.1 Example

- Graph colouring problem





2.1 CSP as state space search

Incremental formulation as a standard search problem:

- **Initial State**: empty assignment

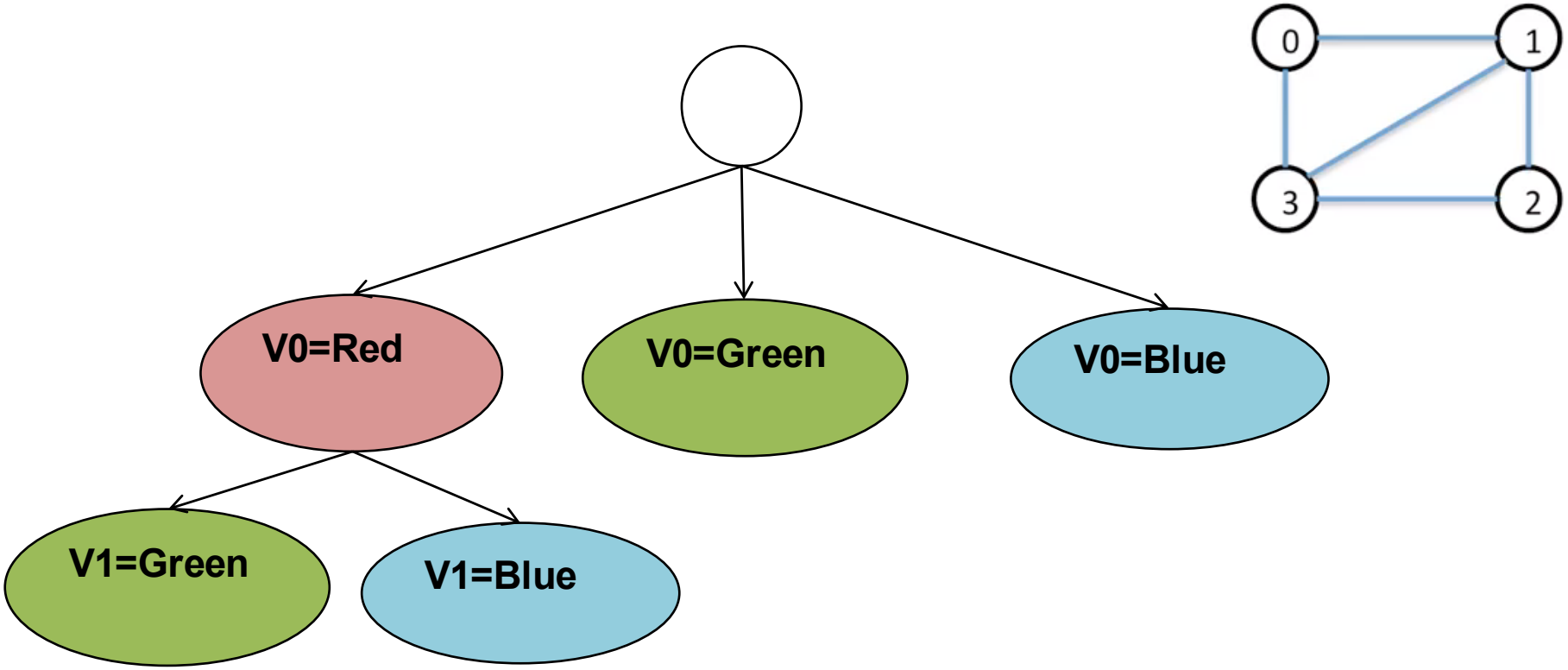
  { V0, V1, V2, V3}

  { , , , }
- **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 BacktrackingSearch(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)
                              end
                    end %while
          end
end %function
```