

# Sistemas Inteligentes

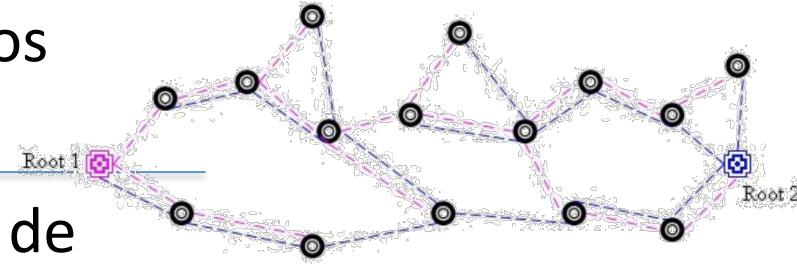


## Tema 1 Problemas de Satisfacción de Restricciones

Dra. Elisa Guerrero Vázquez  
Dpto. Ingeniería Informática  
Universidad de Cádiz

# Real World Problems

Camino más corto entre varios puntos



Enrutamiento óptimo de un paquete de datos en Internet

Plan de mínimo coste para repartir mercancías a clientes



Secuencia óptima de procesos de trabajos en una cadena de producción

Asignación óptima de trabajadores a tareas a realizar



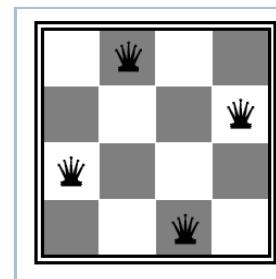
Distribución de tripulaciones de aviones con mínimo coste

## 1.1 Introducción a PSR

■ **Problemas de Satisfacción de Restricciones:** PSR (Constraint Satisfaction Problem: CSP): problemas en los que se busca la asignación correcta de valores a un conjunto de variables de acuerdo a una serie de restricciones

■ Ejemplos:

- N-Reinas
- Coloreado de Mapas
- Criptoaritmética
- Asignación de Tareas



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

## 1.2 Definición de PSR

Conjunto de Variables X<sub>1</sub>...X<sub>n</sub>

- cuyos valores pertenecen a un dominio D<sub>i</sub> de valores posibles

Conjunto de Restricciones C<sub>1</sub>...C<sub>m</sub>

- que definen el conjunto aceptable de valores y propiedades que cada variable puede tomar

Estado:

- se define como una asignación de valores a una o más variables

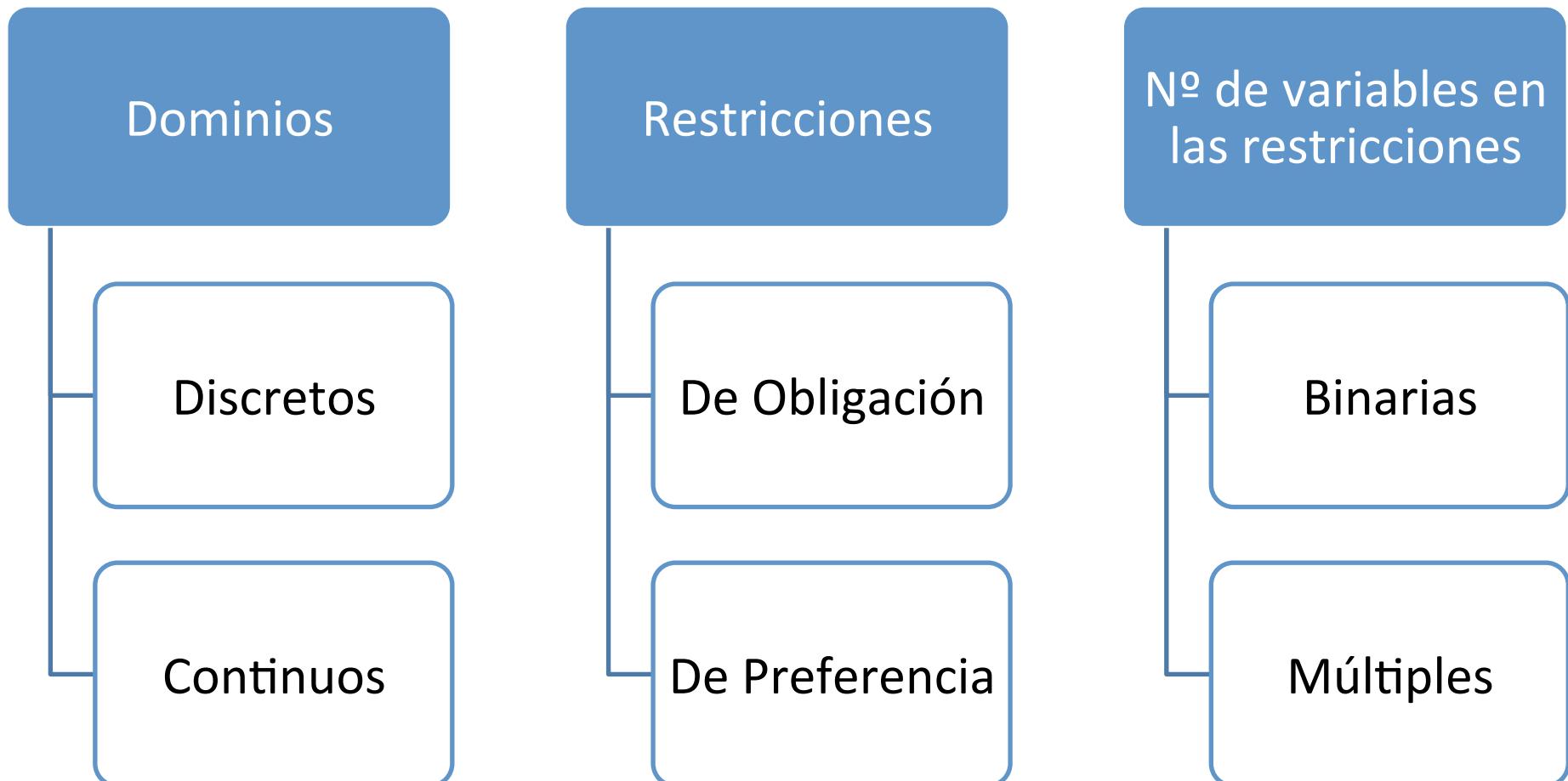
Solución:

- asignación completa que satisface todas las restricciones

## 1.2 Definición PSR



## 1.4 Clasificación de PSR



## 1.3 Ejemplos de PSR: Coloreado de Mapas

- **OBJETIVO:** Asignar colores a cada región de forma que ninguna de las regiones vecinas tengan el mismo color
  - **Variables:**  $R_1 \dots R_7$  representa cada región
  - **Dominio** de cada variable: gama de colores disponible {rojo, verde, azul} en este caso coincide el mismo dominio para todas las variables
  - **Restricciones:**
    - Dos regiones vecinas no pueden tener el mismo color
      - $R_i \neq R_j$  Si  $R_i$  y  $R_j$  son regiones vecinas
  - **Estado:** Cualquier asignación,  $R_1=\text{rojo}$
  - **Solución:** Una asignación completa y consistente



$\{R_1=\text{rojo}, R_2=\text{verde}, R_3=\text{rojo}, R_4=\text{azul},$   
 $R_5=\text{verde}, R_6=\text{rojo}, R_7=\text{verde}\}$

## 1.4 Ejemplo en el Coloreado de Mapas

- **OBJETIVO:** Asignar colores a cada región de forma que ninguna de las regiones vecinas tengan el mismo color
  - **Variables:**  $R_1 \dots R_7$  representa cada región
  - **Dominio** de cada variable: gama de colores
  - **Restricciones:**
    - Dos regiones vecinas no pueden tener el mismo color
      - $R_i \neq R_j$  Si  $R_i$  y  $R_j$  son regiones vecinas
  - **Estado:** Cualquier asignación
  - **Solución:** Una asignación que cumple las restricciones

**Dominio Discreto**

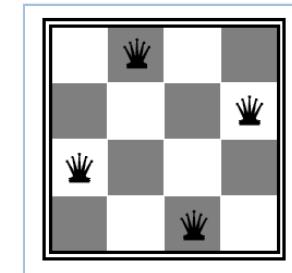
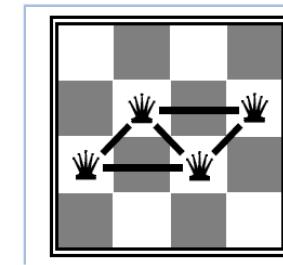
**Restricciones de Obligación**

**Restricciones Binarias**



## 1.3 Ejemplos de PSR: N-Reinas

- **OBJETIVO:** Colocar N-Reinas en un tablero de ajedrez de forma que ninguna esté amenazada
  - **Variables:**  $R_1, R_2, R_3, R_4$  representan las posiciones de cada reina en las columnas 1, 2, 3 y 4 (*cada reina fija en una columna*)
  - **Dominio** de cada variable: filas a las que se puede mover cada reina  $\{1, 2, 3 \text{ y } 4\}$
  - **Restricciones:**
    - Distinta Fila:  $R_i \neq R_j$
    - Distinta Diagonal:  $|R_i - R_j| \neq |i - j|$
  - **Estado:** Cualquier asignación
  - **Solución:**  $R_1=3 \ R_2=1 \ R_3=4 \ R_4=2$



*Dominios Discretos y Restricciones Binarias y de Obligación*

## 1.3 Ejemplos de PSR: Criptoaritmética

- **OBJETIVO:** Asignar a cada letra un valor numérico diferente
  - **Variables:** cada una de las letras del problema, más los dos acarreos:

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

- **Dominio** de cada variable: valores numéricos del 0 al 9 y valores {0,1} para  $X_1$  y  $X_2$

- **Restricciones:**

- Suma1:  $O+O=R + 10* X_1$
- Suma2:  $X_1 + W + W = U + 10 * X_2$
- Suma3:  $X_2 + T + T = O + 10*F$

$$\begin{array}{r}
 & X_2 & X_1 \\
 & T & W & O & + \\
 & T & W & O \\
 \hline
 F & O & U & R
 \end{array}$$

*Restricciones Múltiples y de Obligación*

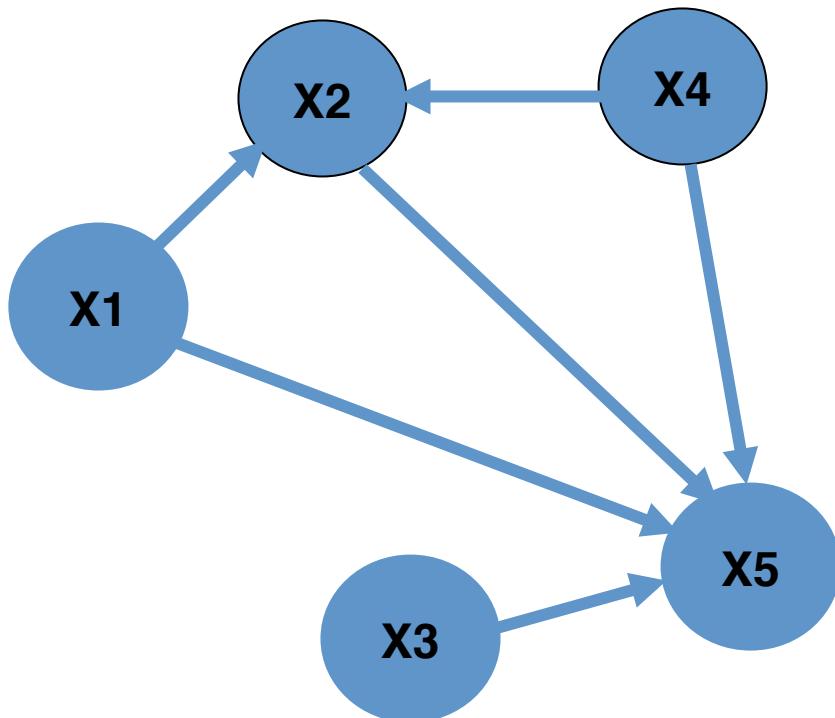
## 1.3 Ejemplos de PSR: Trabajadores y Tareas

- Supongamos que  $n$  trabajadores tienen que realizar  $n$  tareas, y que conocemos el tiempo  $c_{ij}$  de realización por parte del trabajador  $i$ -ésimo ( $t_i$ ) de la tarea  $j$ -ésima ( $T_j$ )
- El problema consiste en asignar a cada trabajador una y sólo una tarea, de manera que se realicen todas las tareas en un tiempo total “mínimo”

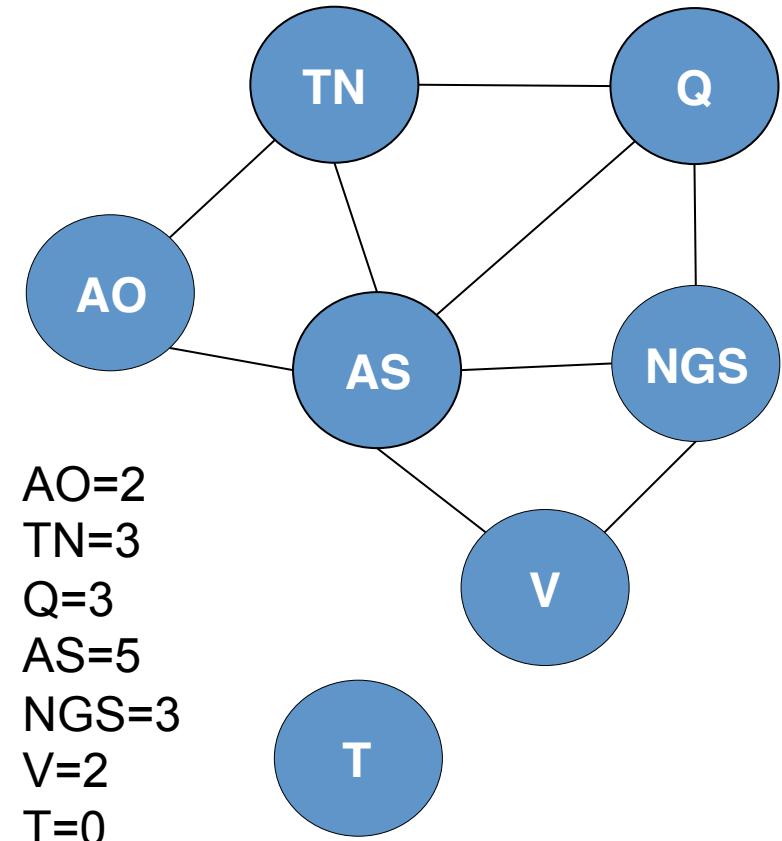
Trabajadores \ Tareas	Tarea 1	Tarea 2	Tarea 3	Tarea 4
Trabajador 1	12	43	15	7
Trabajador 2	9	10	6	4
Trabajador 3	5	13	29	2
Trabajador 4	4	11	17	9

## 1.5 Representación de PSR

- Un PSR binario se suele representar mediante un grafo:
  - **Nodos o Vértices:** Variables
  - **Arcos o Aristas:** Relaciones binarias entre las variables



## 1.5 Representación mediante grafo



# Resolución de un PSR

## ■ Algoritmos de búsqueda

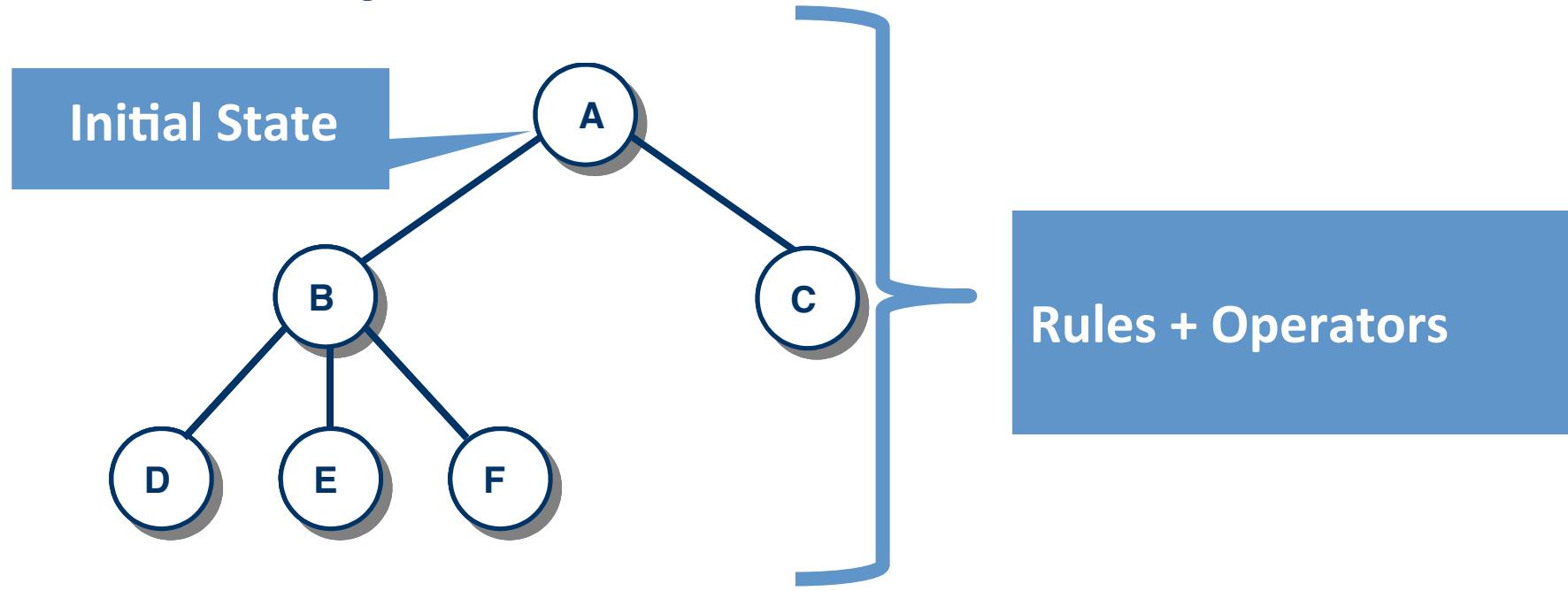
Exploración sistemática del espacio de estados hasta encontrar una solución o probar que no existe ninguna

## ■ Técnicas de consistencia

Eliminación de valores inconsistentes de los dominios de las variables. Suelen combinarse con las técnicas de búsqueda, ya que reducen el espacio de soluciones

## 2. Search Problem Formalization

### Space states



- Goal Test
- Path or Solution
- Solution Cost

## 2.1 PSR como problema de espacio de estados

Formulación *incremental*, como en un problema de búsqueda estándar:

- **Estado inicial:** la asignación vacía { }, en que todas las  $n$  variables no están asignadas
- **Función sucesor:** asignación de un valor  $d$  a cualquier variable no asignada, a condición de que no suponga conflicto con variables ya asignadas
- **Test objetivo:** la asignación actual es completa
- **Coste del camino:** un coste constante (p.e., 1) para cada paso

*¿Profundidad o Anchura?*

## 2.2 Consideraciones

- PROFUNDIDAD FINITA: El número de variables del problema establece la profundidad de la solución
- CONMUTATIVIDAD: En PSR el **orden de aplicación de las asignaciones de valores a variables** es irrelevante
  - Por lo tanto, *los algoritmos de búsqueda para PSR suelen generar los sucesores considerando asignaciones posibles para sólo una variable en cada nodo del árbol de búsqueda*
- CAMINO A LA SOLUCIÓN es irrelevante
- CONTROL DE LOS ESTADOS REPETIDOS es innecesario
  - No hay que realizar comprobaciones para evitar repeticiones
  - No hace falta la lista de CERRADOS

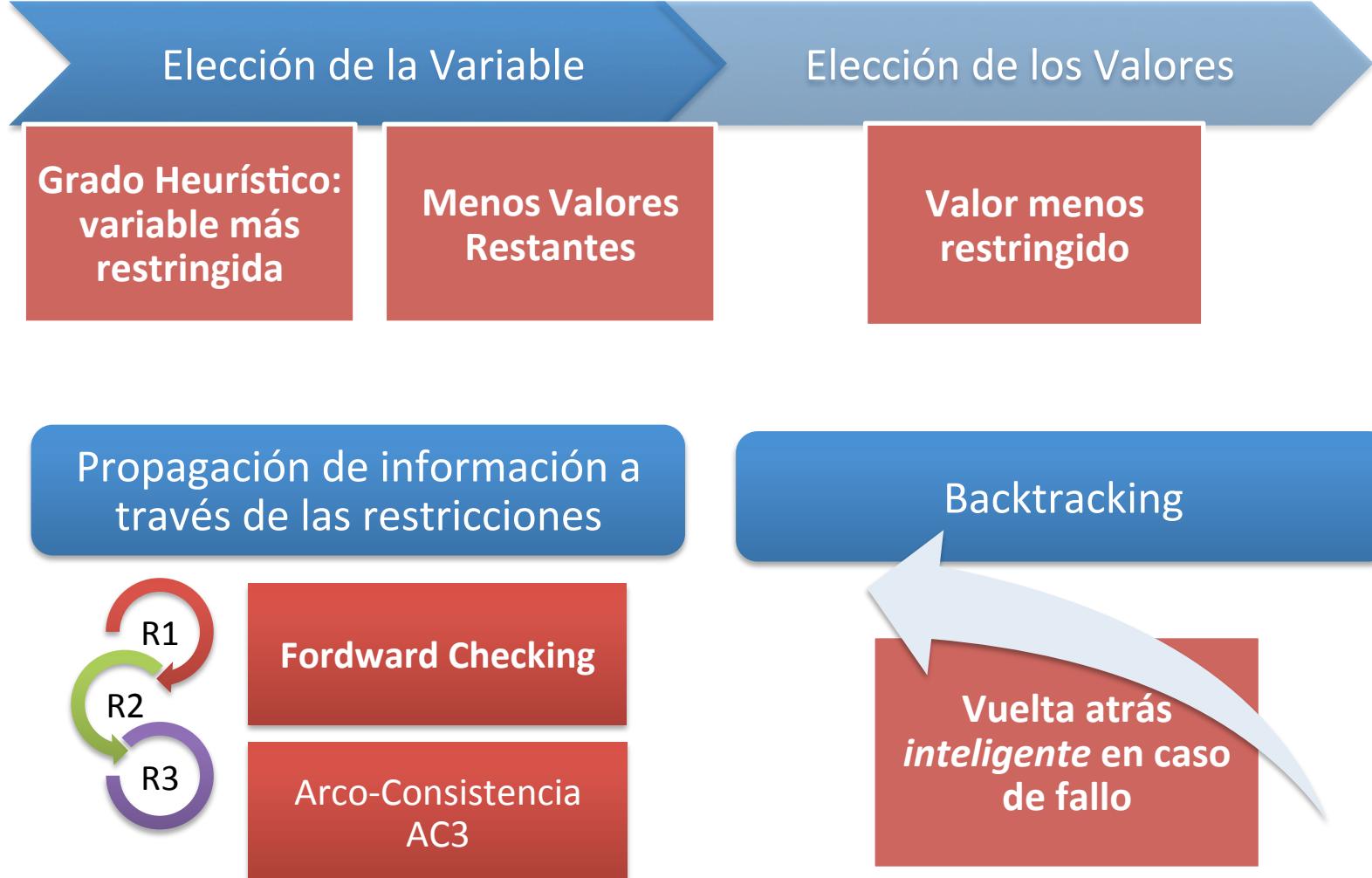
## 2.3 Backtracking: Búsqueda con vuelta atrás

- Búsqueda en profundidad que:
  - elige valores para una variable a la vez
  - vuelve atrás cuando una variable no tiene ningún valor legal para asignarle
  
- El algoritmo
  - genera los sucesores **incrementalmente**, uno a la vez
  - extiende la asignación actual para generar un sucesor, más que volver a copiarlo

## 2.3 Backtracking Algorithm

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

# 3 Heurísticas de propósito general



## 3.A Selección de la siguiente variable

### Elección de la Variable

**var←SELECT-UNASSIGNED-VARIABLE(csp)**

**Grado Heurístico:**  
variable más restringida

**Selecciona la variable que aparezca en el mayor número de restricciones**

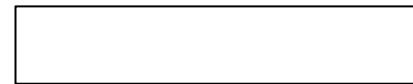
- Útil en caso de empate con MVR o al comienzo del proceso de búsqueda

**Menos Valores Restantes (MVR):**

**MVR escoge la variable con menos valores *legales***

- Mayor probabilidad de poda

## 3.A Ejemplo de Grado Heurístico



**¿Siguiente Variable a Asignar?**

AS=rojo

**Grado Heurístico selecciona AS**

**Número de restricciones:**

AO=2

TN=3

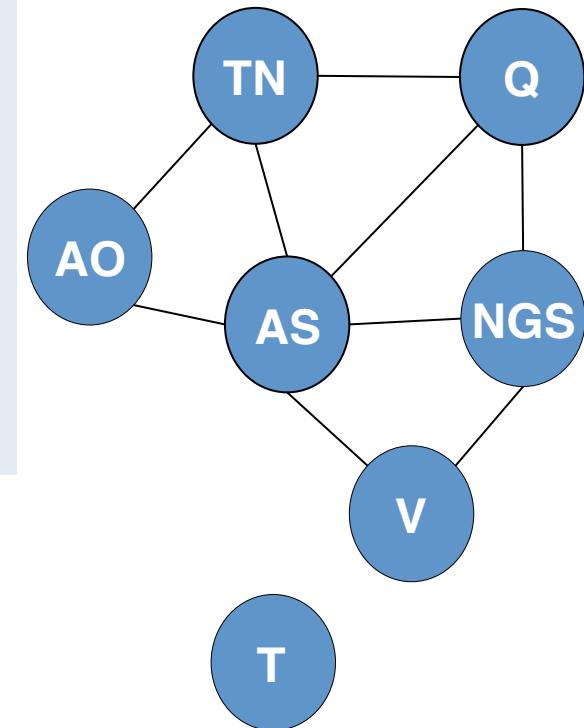
Q=3

AS=5

NGS=3

V=2

T=0



## 3.A Ejemplo de MVR (Mínimos Valores Restantes)

AO=rojo

AO=rojo  
TN=verde

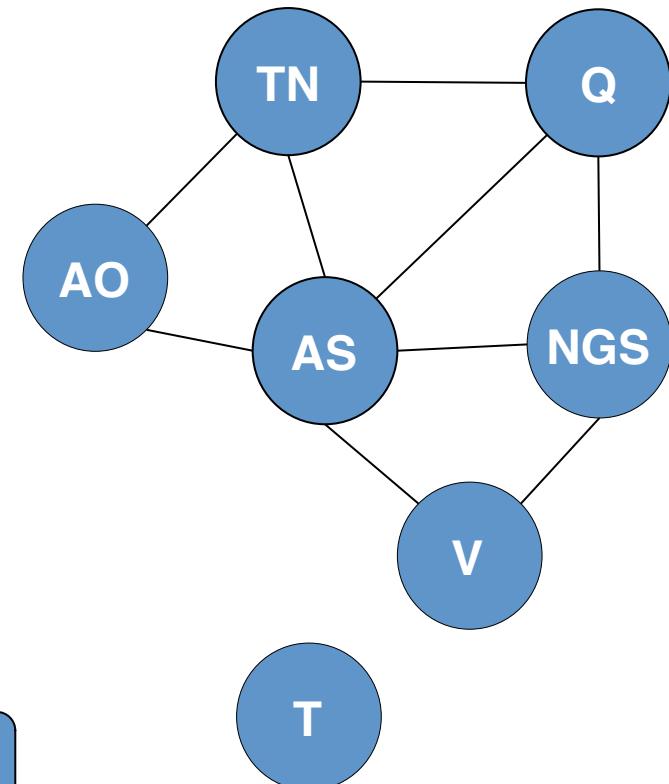
AO=rojo  
TN=verde  
AS=azul



**¿Siguiente Variable a Asignar?**

**Valores posibles:**  
 $AS=\{\text{azul}\}$   
 $Q=\{\text{rojo, azul}\}$

**MVR selecciona AS**



## 3.A Heurística para decidir el orden de valores

### Orden de los valores

**values\_list←ORDER-DOMAIN-VALUES(var,csp)**

Valor Menos Restringido

**Elige el valor que participa en el menor número de restricciones o sea, el valor más libre**

- Trata de dejar el mayor número de opciones para el resto de variables por asignar

## 3.B Ejemplo de VMR (valor menos restringido)

AO=rojo

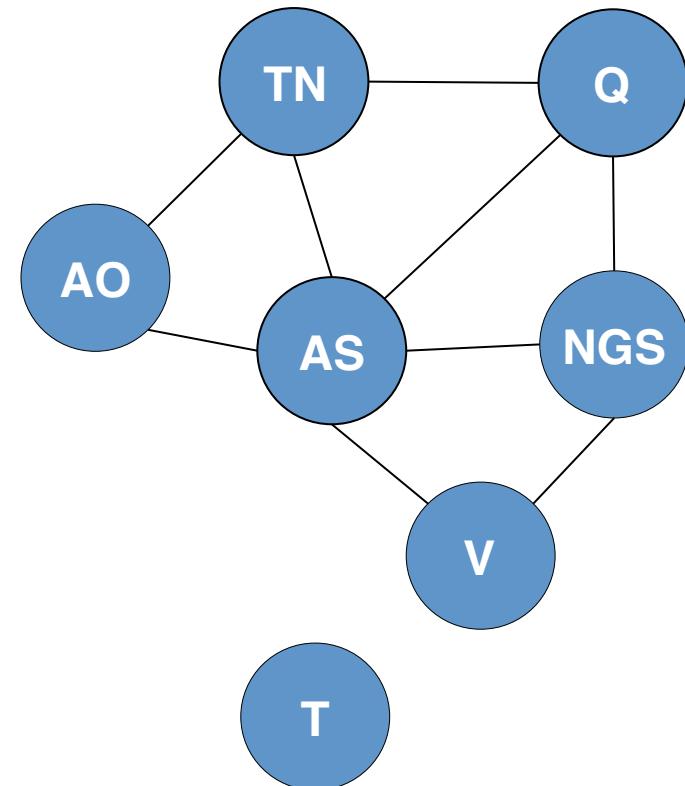
AO=rojo  
TN=verde

AO=rojo  
TN=verde  
Q= rojo

rojo aparece en  
menos restricciones

Suponiendo que la  
siguiente variable es Q  
¿qué valor asignar?

Valores posibles:  
 $AS=\{\text{azul}\}$   
 $Q=\{\text{rojo, azul}\}$



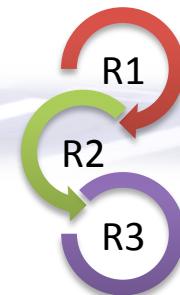
## 3.C Propagación de la información

### Fordward Checking

#### Comprobación hacia delante

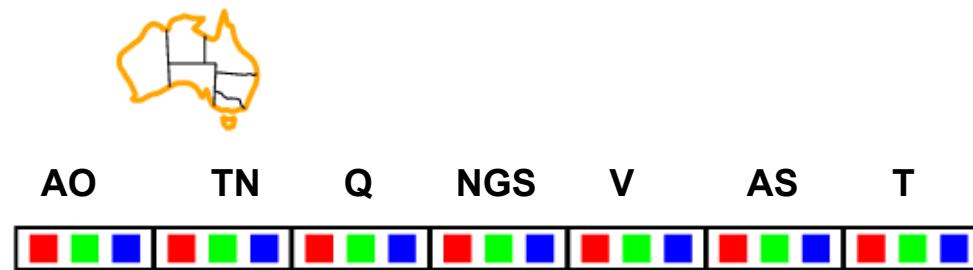
Cuando se asigna a X un nuevo valor, se elimina del dominio de Y aquellos valores que sean inconsistentes con el asignado a X (Y debe estar relacionada con X por una restricción)

- Cada nodo del árbol de búsqueda debe contener el estado y la lista de valores posibles
- MVR es el complemento ideal para Forward checking

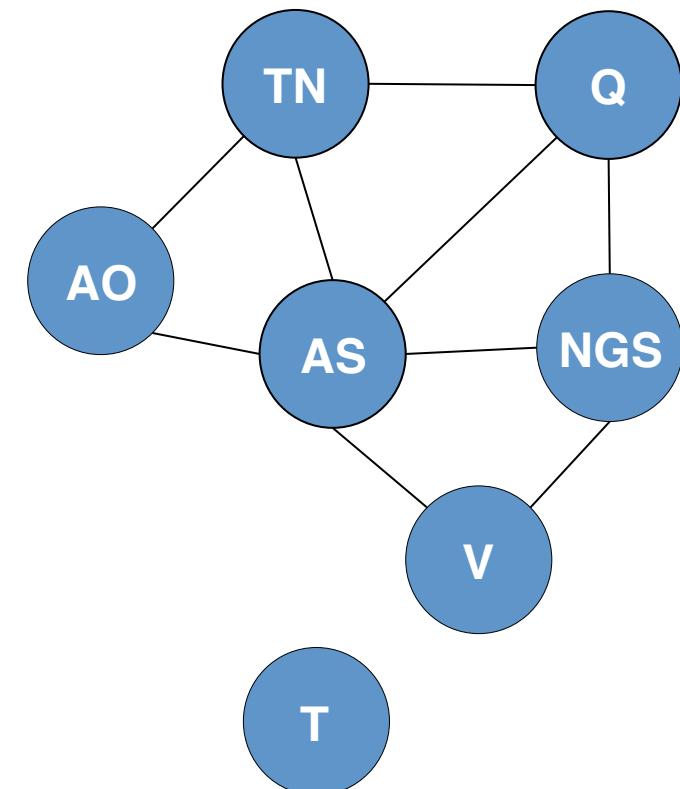
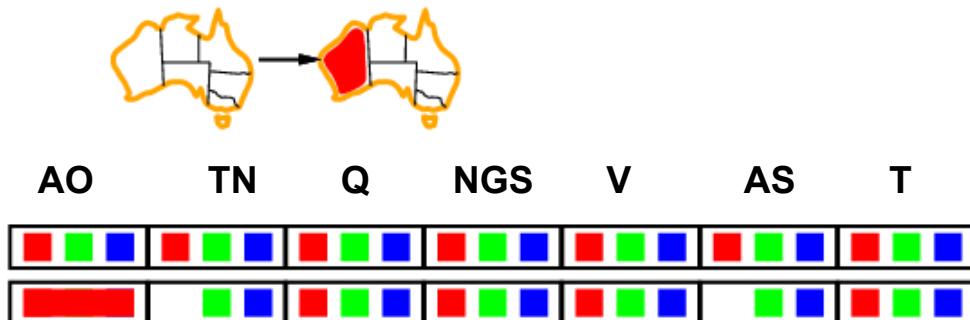


## 3.C Ejemplo de forward checking

- Inicialmente



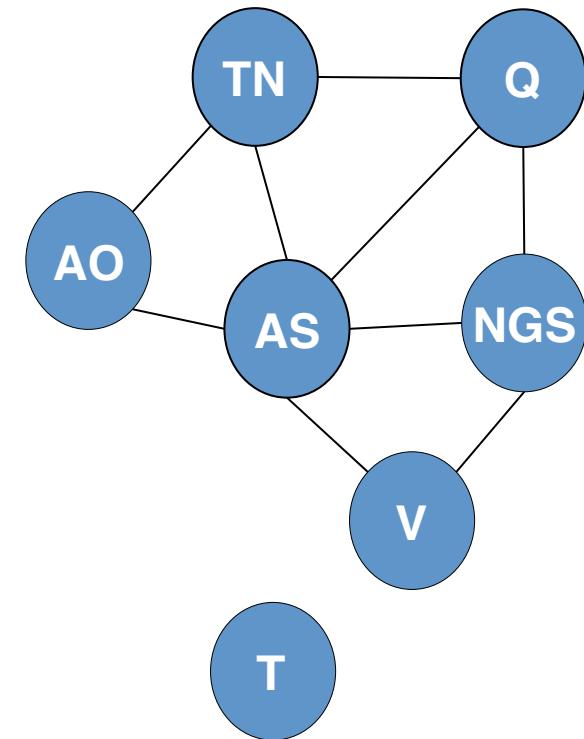
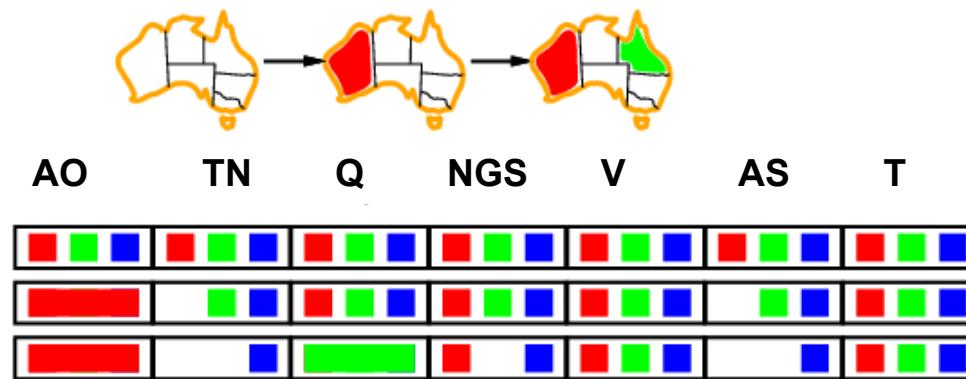
- AO=rojo



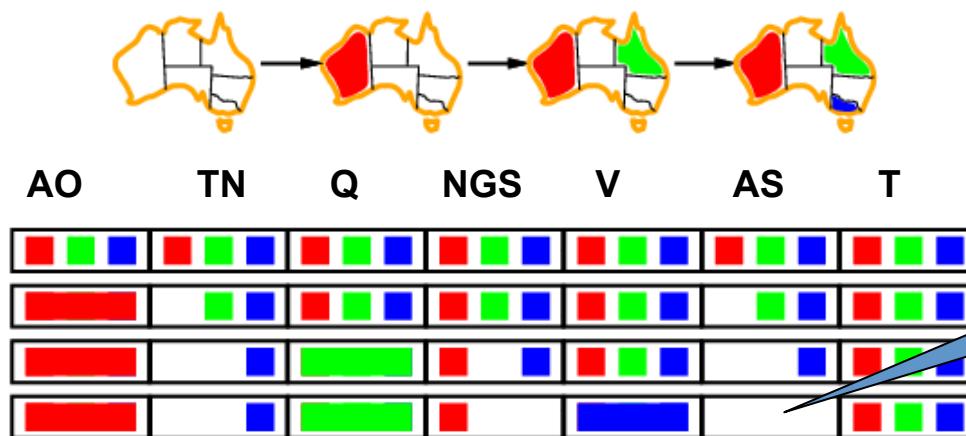
Se elimina el valor rojo de TN y AS

### 3.C Ejemplo de forward checking

- AO=rojo Q=verde



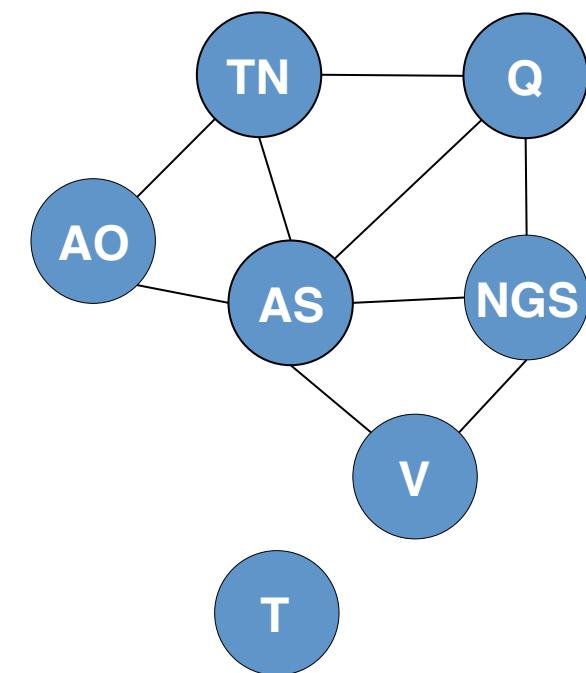
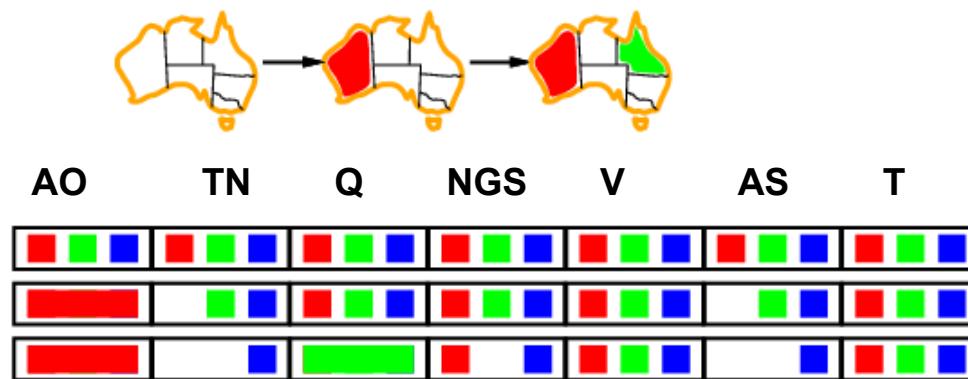
- AO=rojo Q=verde V=azul



AS se queda sin  
valores legales

### 3.C Ejemplo de forward checking

- INCONVENIENTE: Forward checking propaga la información pero puede que no detecte a tiempo todos los fallos



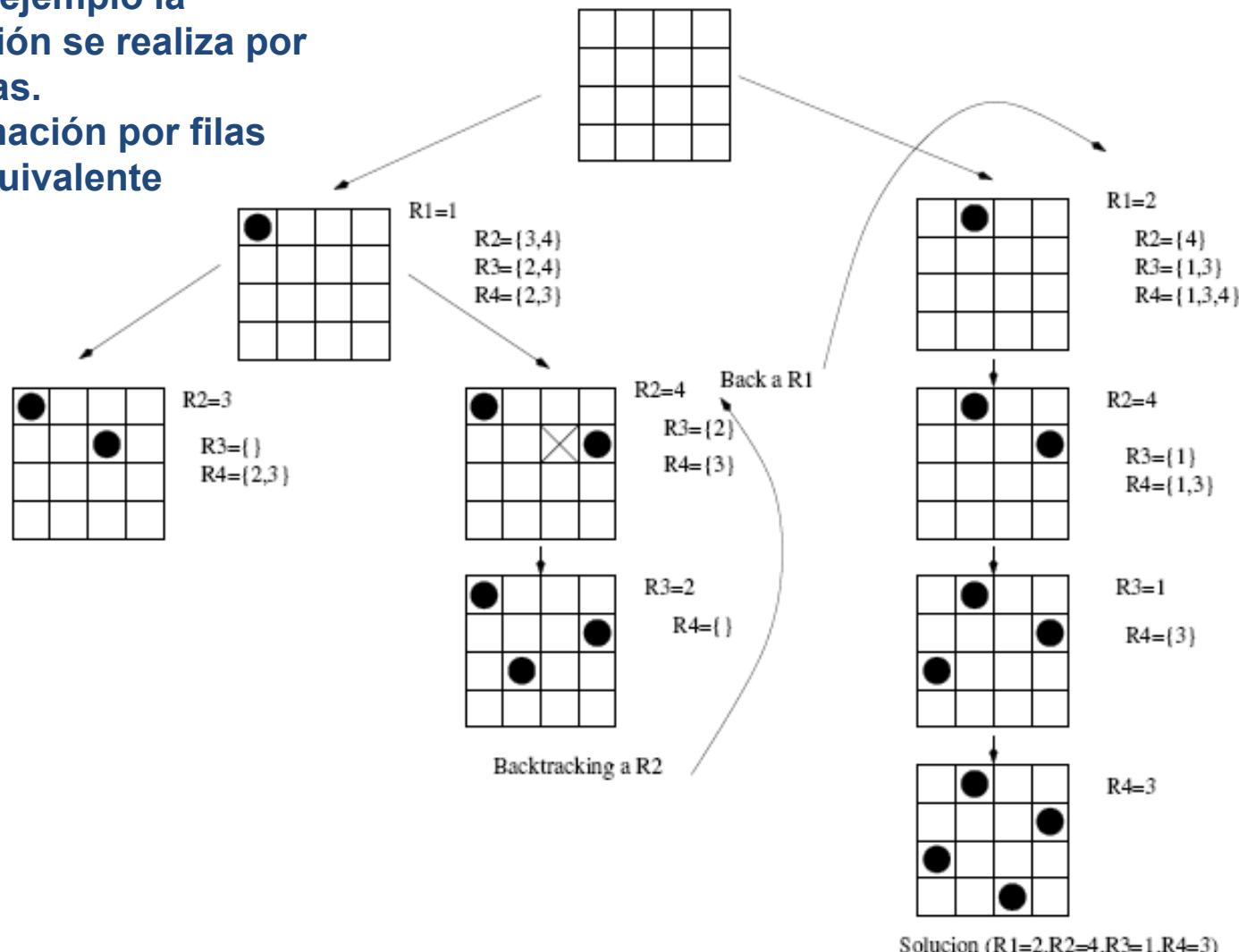
- TN y AS no pueden ser los dos azul !!

### 3. Ejercicio Propuesto

- Repetir el proceso de forward checking aplicando además las estrategias de Mínimos Valores Restantes, Grado heurístico y Valor menos restringido.

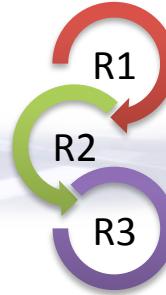
# Ejemplo: 4-reinas mediante forward checking

En este ejemplo la asignación se realiza por columnas.  
 La asignación por filas sería equivalente



Solución (R1=2,R2=4,R3=1,R4=3)

### 3.C Inconvenientes de Forward Checking



#### Forward Checking

Las implicaciones de una restricción se propagan sobre otras variables

NO es completa

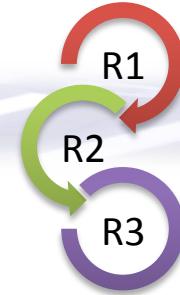
Se requiere rapidez para que sea eficiente

#### ALTERNATIVA:

Todas las variables sean **arco consistentes**

## 3.C Propagación de restricciones

Arco-Consistencia  
AC3

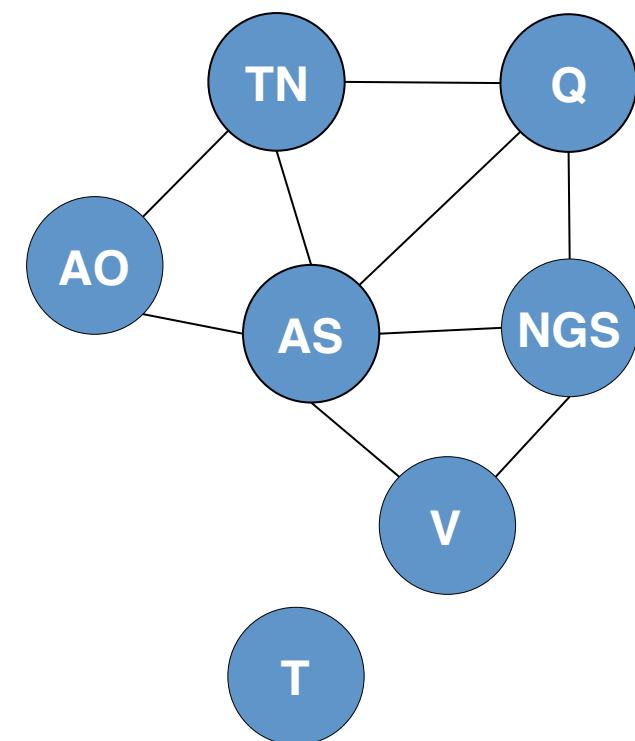
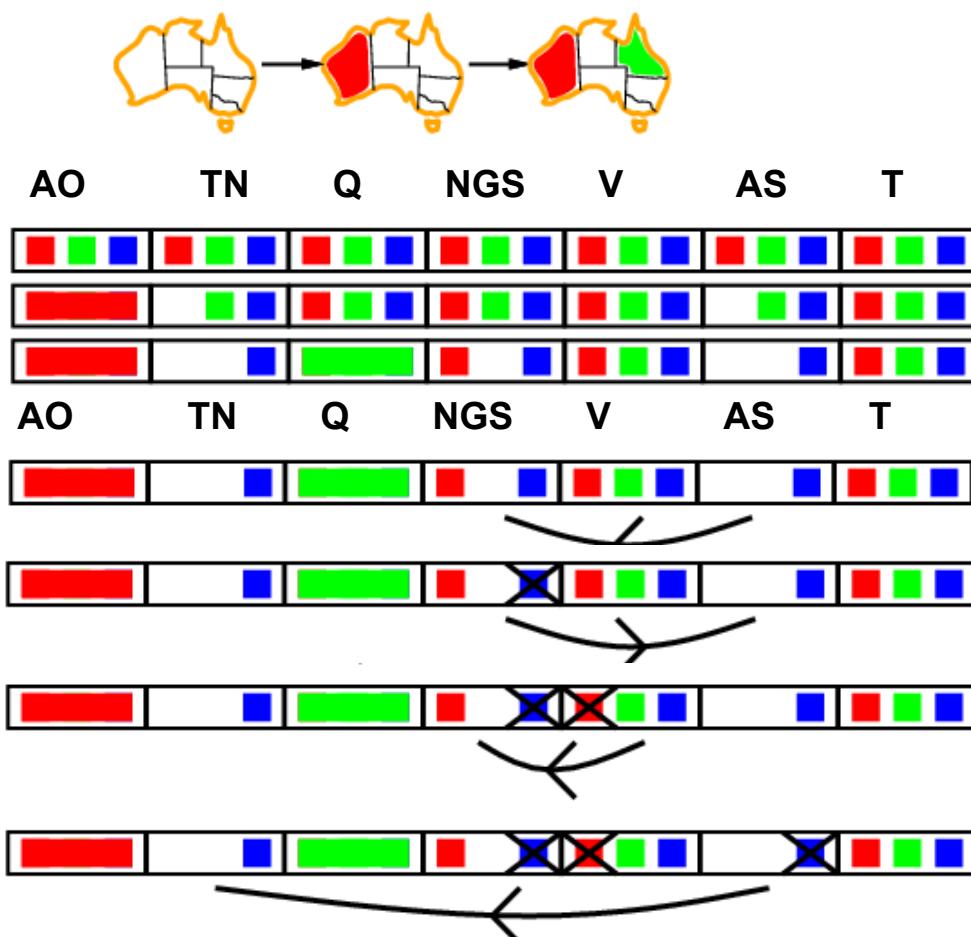


- **Arco Consistente:** si para cada par de variables ( $X, Y$ ) y para cualquier valor  $x_i$  de  $D_x$  existe un valor  $y_j$  de  $D_y$  tal que se satisfacen las restricciones
  - *Los dominios actuales de cada variable tienen que ser consistentes con todas las restricciones*

### 3.C Consistencia de Arcos

$(X \ Y)$  es consistente si:

- Para cada valor de  $x_i$  de X hay algún valor permitido  $y_j$
- AO=rojo Q=verde



## 3.C Arco-Consistencia

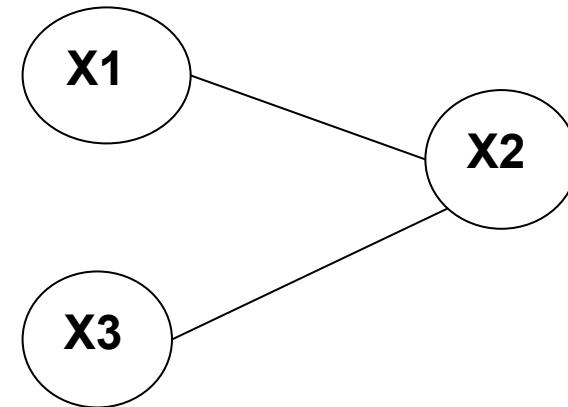
D1={1..10}

D2={5..15}

D3={8..15}

■ X1>X2

■ X2>X3



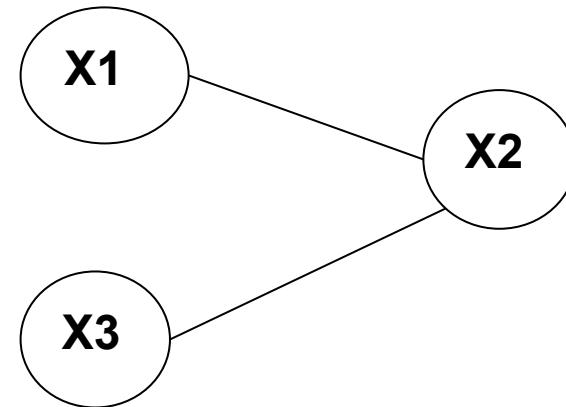
## 3.C Arco-Consistencia

D1={1..10}

D2={5..15}

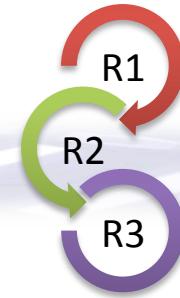
D3={8..15}

- X1>X2
- X2>X3



1. Al hacer c12 arco consistente, se limita d1 = {6..10} y d2 = {5..9}
2. Al hacer c23 arco consistente, se limita d2={9} y d3={8}
3. En el siguiente ciclo, d1={10}, d2={9} y d3={8}

### 3.C Algoritmo AC3



Procedimiento (idea intuitiva):

- a partir de los dominios iniciales de las variables
- en cada paso se realiza la actualización de los dominios
- devolver un conjunto de dominios actualizados tal que todos los arcos del problema sean consistentes

Actualización de dominios:

- Si un arco es inconsistente, podemos hacerlo consistente eliminando del dominio de la variable distinguida aquellos valores para los que no existen valores en los dominios de las restantes variables que satisfagan la restricción

Criterio de Parada

- Todos los arcos son Consistentes
- INCONSISTENCIA: Si un dominio queda vacío

### 3.C Algoritmo AC3

- Arcos: estructura con los campos variable y restricción
  - Funciones de acceso: ARCO-VARIABLE y ARCO-RESTRICCIÓN
- Generación de todos los arcos de un PSR:

```
FUNCION ARCOS()
1. Hacer ARC igual a vacío
2. Para cada RESTRICCIÓN es *RESTRICCIONES*
    2.1 Para cada VARIABLE en PSR-RESTR-VARIABLES (RESTRICCION)
        Añadir a ARC el arco formado por VARIABLE+RESTRICCION
3. Devolver ARC
```

- Cálculo de dominio actualizado:

```
FUNCION ACTUALIZA-DOMINIO(VARIABLE, RESTRICCION, VARIABLES)
1. Hacer DOMINIO-ACTUAL igual al dominio de VARIABLE en VARIABLES
   Hacer NUEVO-DOMINIO igual a vacío
2. Para cada VALOR en DOMINIO-ACTUAL hacer
    2.1 Si para ese VALOR de VARIABLE existe una asignación a las
        restantes variables de RESTRICCION que satisfaga RESTRICCION,
        incluir VALOR en NUEVO-DOMINIO
3. Devolver NUEVO-DOMINIO
```

### 3.C Algoritmo AC3

**FUNCION AC-3 (VARIABLES)**

1. Hacer RED igual a ARCOS()
2. Mientras RED no sea vacío
  - 2.1 Hacer ACTUAL el primero de RED y RED el resto de RED
  - 2.2 Hacer VARIABLE igual a ARCO-VARIABLE(ACTUAL) y RESTRICCION igual ARCO-RESTRICCION(ACTUAL)
  - 2.3 Hacer DOMINIO-ACTUAL igual al dominio de VARIABLE en VARIABLES
  - 2.4 Hacer DOMINIO-NUEVO igual a ACTUALIZA-DOMINIO(VARIABLE, RESTRICCION, VARIABLES)
  - 2.5 Si DOMINIO-NUEVO y DOMINIO-ACTUAL son distintos, entonces
    - 2.5.1 Actualizar (destructivamente) el dominio de ACTUAL en VARIABLES con NUEVO-DOMINIO
    - 2.5.2 Incluir en RED todos los arcos de ARCOS() en los que aparezca VARIABLE y ésta NO sea la variable distinguida del arco
  - 2.6 Devolver VARIABLES

- Entrada: una lista de variables+dominios
- Salida: la lista de entrada, con los dominios actualizados (destructivamente) de manera que todos los arcos del PSR son consistentes

### 3.C Algoritmo AC3

- De la aplicación de AC3 se puede llegar a:
  - Al menos un dominio vacío: no hay solución posible
  - Todos los dominios son unitarios: solución encontrada
  - Al menos un dominio no es unitario: posibilidad de múltiples soluciones
- AC3 puede utilizarse en combinación con alguna estrategia de búsqueda:
  - Backtracking y salto atrás dirigido por conflicto
  - Búsqueda local y heurística de mínimos conflictos

### 3.D Backtracking dirigido por conflicto

- Vuelta atrás hasta el conjunto de variables que originaron el fracaso: **conjunto conflicto**
- Conjunto conflicto para una variable X es el conjunto de variables previamente asignadas que están relacionadas con X por las restricciones
- No evita cometer los mismos errores en otra rama

## 4. Algoritmos de Escalada (Hill-Climbing)

- Bucle que continuamente se mueve en la dirección del valor:
  - creciente (si se trata de maximizar una función de coste)
  - decreciente (si se trata de minimizar la función de coste)

## 4.1 Búsqueda en Escalada

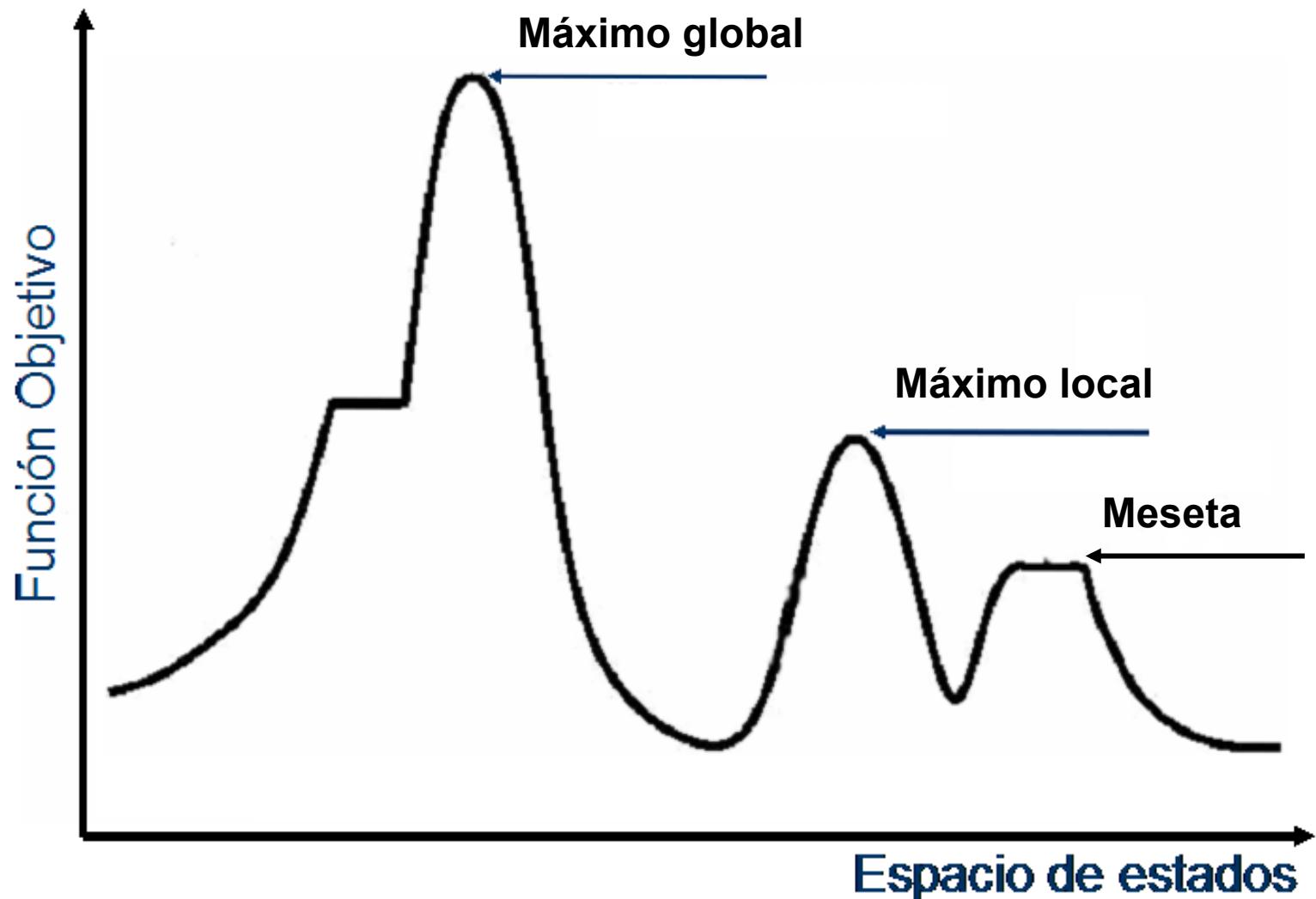
- Se generan los sucesores de un estado  $n$  y sea  $m$ , el de mayor  $f(m)$ . Si  $f(m) > f(n)$  devuelve  $m$  como expansión.
- Sigue el recorrido a través de los nodos en los que el valor de dicha función sea máximo (cuesta arriba).
- No mantiene un árbol de búsqueda, tan solo una estructura con el estado y el valor de la función objetivo.
- La B. en Escalada sólo mira a los vecinos inmediatos al estado actual.
- Termina cuando alcanza un “pico” donde ningún vecino tiene un valor más alto.

## 4.1 Algoritmo B. en Escalada

- La lista de ABIERTOS sólo mantendría un único estado después de aplicar los operadores a actual

```
vecino:=sucesor de actual con f mayor
si f(vecino) <= f(actual)
entonces ABIERTOS:= actual
si no    ABIERTOS:= vecino
```

## 4.1 Problemas



## 4.2 Búsqueda por Haz Local

- Comienza con estados generados aleatoriamente
- Guarda la pista de  $k$  estados.
- Mientras alguno no sea objetivo
  - Se generan todos los sucesores de los  $k$  estados.
  - Se seleccionan los  $k$  mejores sucesores de la lista completa y se repite el proceso

## 4.3 Búsqueda Local para PSR

- Estados: asignaciones completas (consistentes o inconsistentes)
- Estado inicial: escogido aleatoriamente
- Estado final: solución al PSR
- Generación de un sucesor: elegir una variable y cambiar el valor que tiene asignado (heurística y aleatoriedad)
  
- Heurística de **Mínimos Conflictos**:
  - Seleccionar **variable** distinta de la última modificada que participa en más restricciones NO satisfechas en el estado
  - Seleccionar **valor** que cause el mínimo número de conflictos con otras variables (Valor Menos Restringido)

## 4.3 Algoritmo

FUNCION MIN-CONFLICTOS(MAX-ITERACIONES)

1. Hacer ACTUAL igual a un nodo con una asignación generada aleatoriamente.
2. Para I desde 1 hasta MAX-ITERACIONES hacer
  - 2.1 Si la asignación de ACTUAL es una solución, devolverla y terminar.
  - 2.2 Hacer VARIABLE igual a una variable (distinta de la última modificada) escogida aleatoriamente de entre aquellas que participan en el mayor número de restricciones incumplidas.
  - 2.3 Hacer SUCESORES igual a la lista de nodos obtenidos cambiando en la asignación ACTUAL el valor de VARIABLE por cada uno de los posibles valores del dominio de VARIABLE en \*VARIABLES\* (excepto el correspondiente al valor que tenía en ACTUAL)
  - 2.4 Hacer ACTUAL el nodo de SUCESORES escogido aleatoriamente de entre aquellos cuya asignación incumple el menor número de restricciones
3. Devolver FALLO

## 5. Conclusiones

Cuando el camino a la solución es irrelevante:

- Guardan sólo un estado en memoria: el estado actual
- Se mueven sólo a los nodos vecinos del nodo actual
- No son sistemáticos en la búsqueda
- Utilizan poca memoria
- Pueden encontrar soluciones razonables en espacios de estados grandes o infinitos
- Pueden quedar atrapados en máximos/mínimos locales

## Bibliografía

- Russell, S. y Norvig, P. *Inteligencia Artificial (un enfoque moderno)* (Pearson Educación, 2004). Segunda edición. Cap. 5: “Problemas de Satisfacción de Restricciones”
- Nilsson, N.J. *Inteligencia artificial (una nueva síntesis)* (McGraw–Hill, 2000). Cap. 11 “Métodos alternativos de 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 de Cataluña
- José Luis Ruiz Reina et al. – Universidad de Sevilla