Sistemas Inteligentes I

Tema 4. Satisfacción Restricciones

José A. Montenegro Montes
monte@lcc.uma.es

Resumen

- Introducción
- Definición del Problema y Ejemplos
- Restricciones y Consistencia
- Propagación de Restricciones
- Vuelta Atrás
- Búsqueda Local
- Conclusiones

Introducción

Representaciones factorizadas

- En los temas anteriores exploramos problemas que pueden resolverse buscando en un espacio de estados
- Cada **estado era atómico**, es decir, era una caja negra sin estructura interna
- Aquí emplearemos una representación factorizada para cada estado
  - Un conjunto de variables, cada una con su valor
  - Un problema está resuelto cuando cada variable tiene un valor que satisface todas las restricciones sobre dicha variable

Definición del Problema

y ejemplos

Definición (I)

- Un problema de satisfacción de restricciones (*constraint satisfaction problem*, CSP) está formado por tres componentes:
  - Un conjunto de **variables**, $X=\{X_1,...,X_n\}$
  - Un conjunto de **dominios**, uno para cada variable: $D=\{D_1,..., D_n\}$
  - Un conjunto de **restricciones** que especifican combinaciones permitidas de valores, C
- Cada dominio D_i es el conjunto de valores posibles $\{v_1,..., v_k\}$ para la variable v_i

Definición (II)

- Cada **restricción** C_i es un par <*scope*,*rel*> donde *scope* es la tupla de variables que intervienen en la restricción y *rel* es una relación que define los valores que dichas variables pueden tomar.

  - Por ejemplo, si las variables X_3 y X_5 deben tener valores distintos, podemos escribir esta restricción como $< (X_3,X_5), X_3 \neq X_5 >$

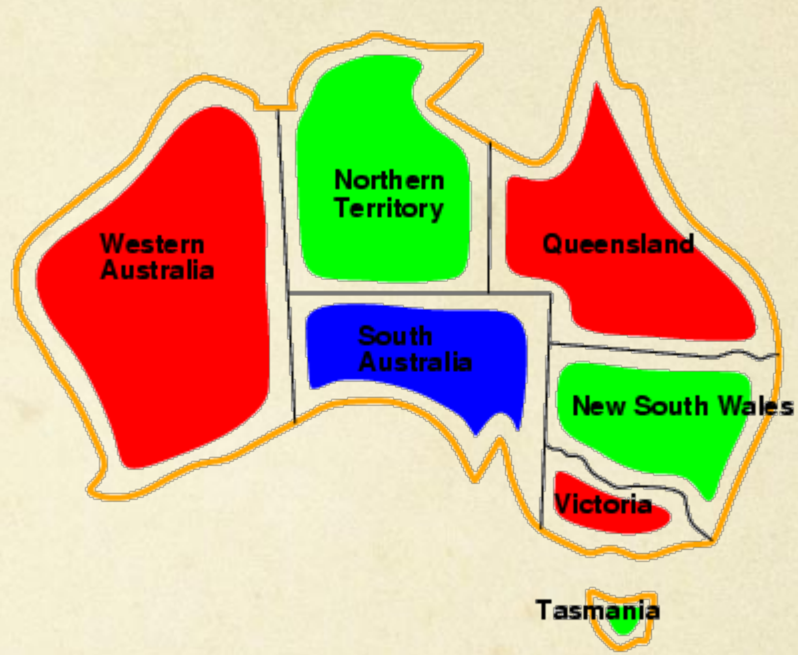
Definición (III)

- Cada estado de un CSP se define como una asignación de valores a algunas o a todas las variables, $\{X_i=v_i, X_j=v_j,\ldots\}$
- Una asignación que no viola ninguna restricción se llama asignación consistente o legal
- Si tenemos una asignación en la cual todas las variables están asignadas, la llamamos asignación completa.
  - En otro caso, la llamamos asignación parcial.
- Una solución de un CSP es una asignación consistente y completa

Ejemplo 1: Coloreado de mapas

- La tarea consiste en colorear cada región de un mapa de tal manera que no haya regiones adyacentes que tengan el mismo color
- Para el mapa de Australia (siguiente transparencia), definimos las variables como X={WA, NT, Q, NSW, V, SA, T}
- El dominio de cada variable es D_i={*red*, *green*, *blue*}
- Hay nueve restricciones: C={$SA \neq WA$, $SA \neq NT$, $SA \neq Q$, $SA \neq NSW$, $SA \neq V$, $SA \neq V$, $WA \neq NT$, $NT \neq Q$, $Q \neq NSW$, $NSW \neq V$}

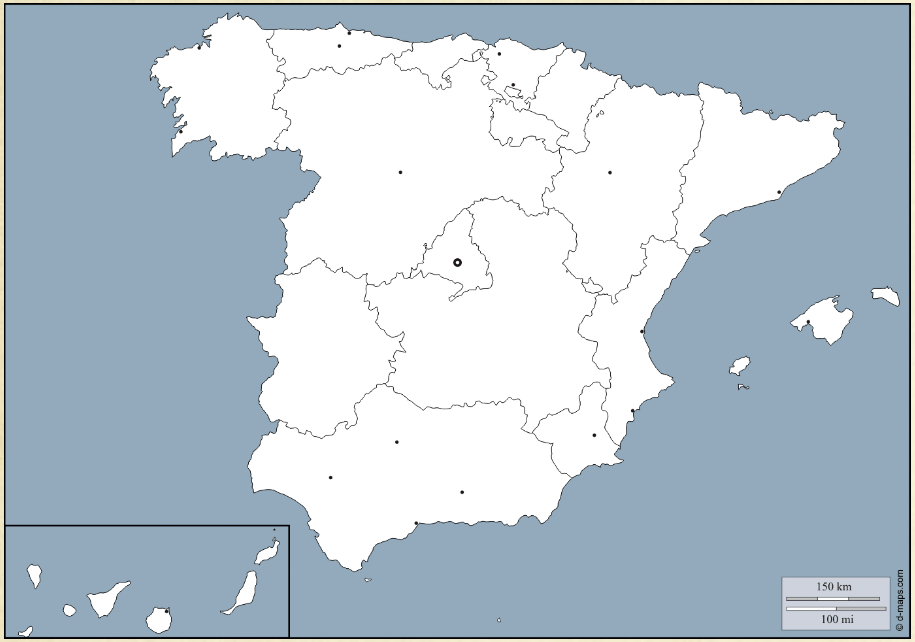
Ejemplo 1: Coloreado de mapas

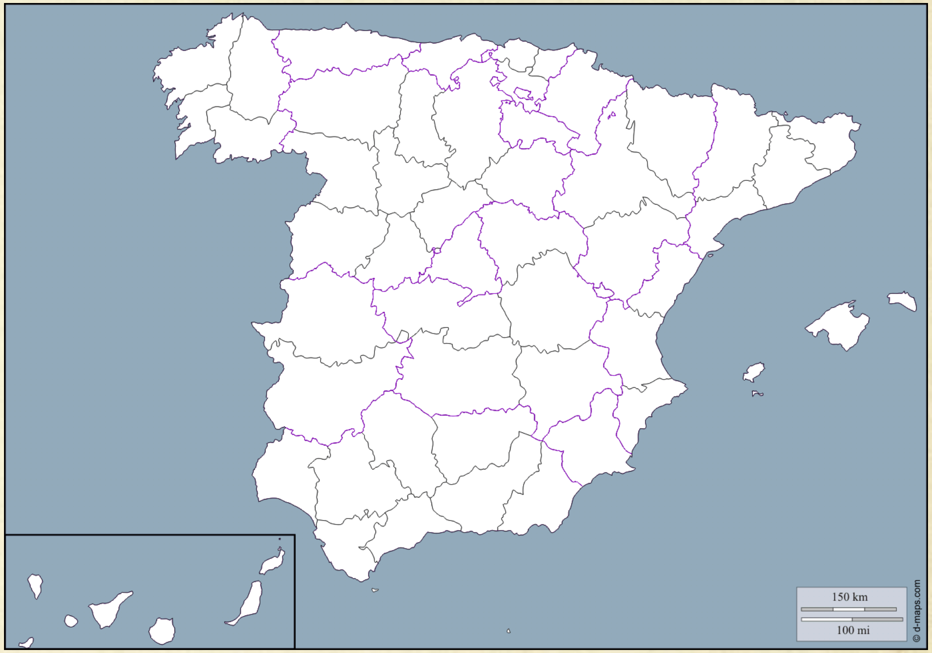


- Ejemplo de solución completa y consistente:

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

Ejemplo 1: Coloreado de mapas





Ejemplo 2: problemas criptoaritméticos

- En un acertijo criptoaritmético, cada letra representa a un dígito distinto (0-9)

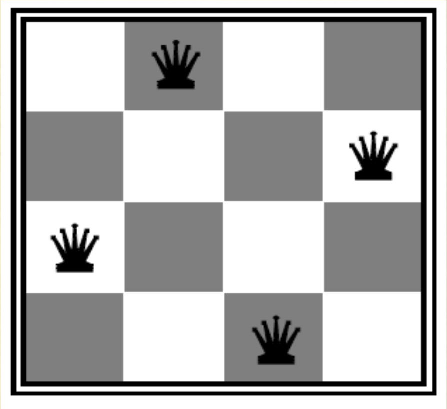
$$
\begin{array}{r}
T\ W\ O \\
+\ T\ W\ O \\
\hline
F\ O\ U\ R
\end{array}
$$

Ejemplo 2: problemas criptoaritméticos

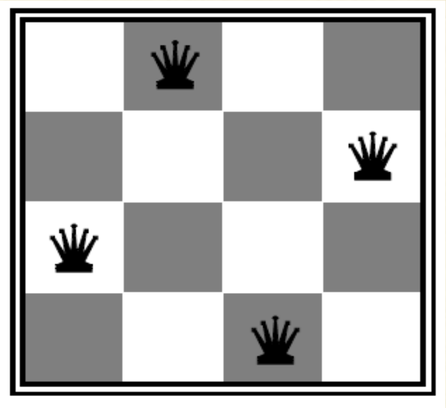
- El requisito de que todas las variables han de tomar diferentes valores se corresponde con la restricción $AllDiff(F,T,U,W,R,O)$
- Introducimos tres variables auxiliares C_{10}, C_{100} y C_{1000}, que representan los dígitos acarreados a las columnas de las decenas, las centenas y los millares, respectivamente
- De esta manera el resto de las restricciones son:
  - $O+O=R+10 \cdot C_{10}$
  - $C_{10}+W+W=U+10 \cdot C_{100}$
  - $C_{100}+T+T=O+10 \cdot C_{1000}$
  - $C_{1000}=F$

Ejemplo 3: N-Reinas

- Variables:
- Dominios:
- Restricciones:



Ejemplo 3: N-Reinas



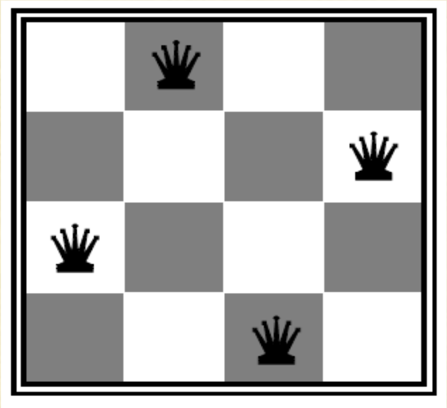
- Variables: X_{ij}
- Dominios: {0,1}
- Restricciones:
  - $\forall i,j\ (X_{ij},X_{ik}) \in \{(0,0),(0,1),(1,0)\}$ $\quad X_{ij} + X_{ik} \leq 1$
  - $\forall\ i,j\ (X_{ij},X_{kj}) \in \{(0,0),(0,1),(1,0)\}$ $\quad X_{ij} + X_{kj} \leq 1$
  - $\forall\ i,j,k\ (X_{ij},X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$ $\quad X_{ij} + X_{i+k,j+k} \leq 1$
  - $\forall ,j,k\ (X_{ij},X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$ $\quad X_{ij} + X_{i+k,j-k} \leq 1$

$$\sum_{i,j} X_{ij} = N$$

Ejemplo 3: N-Reinas

- Variables: Q_k
- Dominios: {1,2,3,...,N}
- Restricciones:
  - Implícito $\forall$ i,j no ataca (Q_i,Q_j)
  - Explícito $(Q_1,Q_2) \in \{(1,3),(1,4),...\}$

Q_1
Q_2
Q_3
Q_4

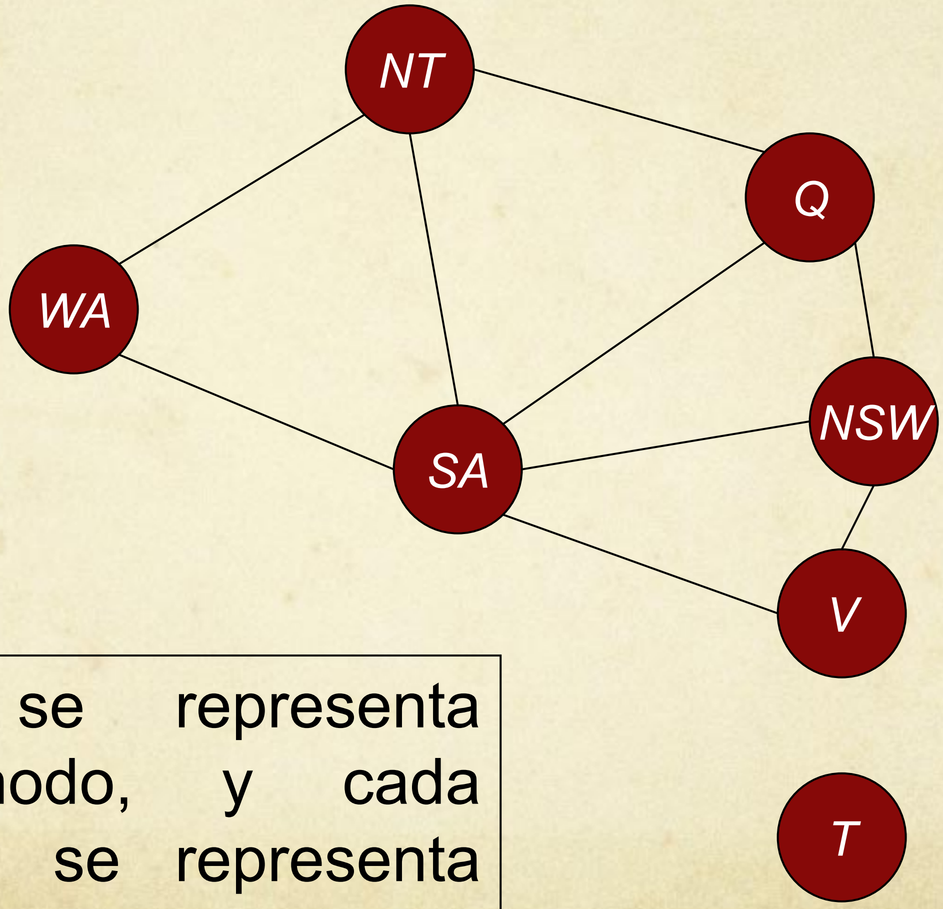


Restricciones y Consistencia

Tipos de restricciones

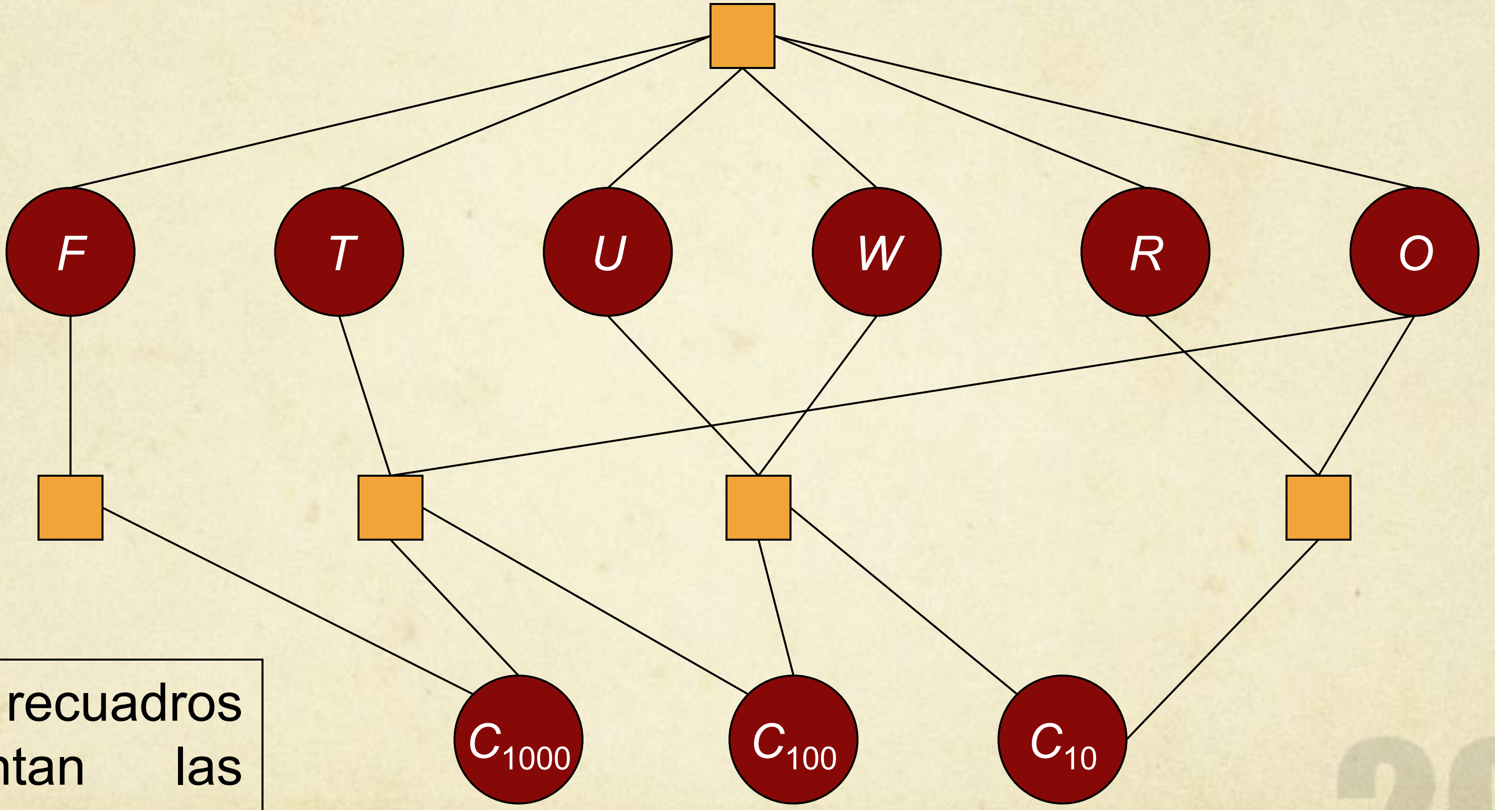
- Una **restricción unaria** restringe el valor de una sola variable
  - Por ejemplo, para imponer que *South Australia* no se coloree de verde escribimos $<(SA), SA \neq green>$
- Una **restricción binaria** relaciona dos variables
  - Por ejemplo, $SA \neq NSW$
- Una restricción en la que participa un número arbitrario de variables se llama **restricción global**
  - Por ejemplo, $AllDiff(F,T,U,W,R,O)$

Grafo de restricciones



Cada variable se representa mediante un nodo, y cada restricción binaria se representa con un arco

Hipergrafo de restricciones



Los recuadros representan las restricciones globales

Propagación de Restricciones

Inferencia en CSPs

Inferencia en CSPs

- Los problemas de búsqueda en el espacio de estados, solamente podemos buscar.
- CSPs tenemos dos posibilidades:
  1. Podemos realizar una búsqueda, escoger una asignación entre varias posibilidades.
  2. Realizar una tipo de inferencia, denominada propagación de restricciones, utilizando las restricciones para reducir el número de valores legales.

Inferencia en CSPs

- La propagación de restricciones se pueden intercalar con la búsqueda o como paso previo a la búsqueda.
- Dependiendo del problema con la propagación de restricciones es suficiente para encontrar una solución y no es necesario realizar la búsqueda.
- La idea es mantener la consistencia en el grafo de restricciones.

Consistencia de nodos

- Una variable es **nodo-consistente** si todos los valores del dominio de la variable satisfacen las **restricciones unarias** sobre dicha variable
    - Por ejemplo, si tenemos la restricción unaria <(*SA*),*SA* ≠ *green*>, podemos hacer *SA* nodo-consistente quitando *green* de su dominio, lo que deja a *SA* con el dominio reducido {*red*, *blue*}
- Siempre es posible eliminar todas las restricciones unarias de un CSP ejecutando la consistencia de nodos

Consistencia de arcos

- Una variable X_i es arco-consistente si, para cada valor del dominio de X_i y cada **restricción binaria** (X_i, X_j), podemos encontrar al menos un valor en el dominio de X_j que satisface la restricción
- Una red es arco-consistente si toda variable es arco-consistente con las demás variables
- El siguiente algoritmo asegura la arco-consistencia de la variable X_i con respecto a otra variable X_j; devuelve *true* si y sólo si se ha revisado el dominio de X_i

Algoritmo de revisión de dominios

function REVISE(csp, X_i, X_j) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  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
      $revised \leftarrow true$
  return $revised$

Algoritmo AC-3 (I)

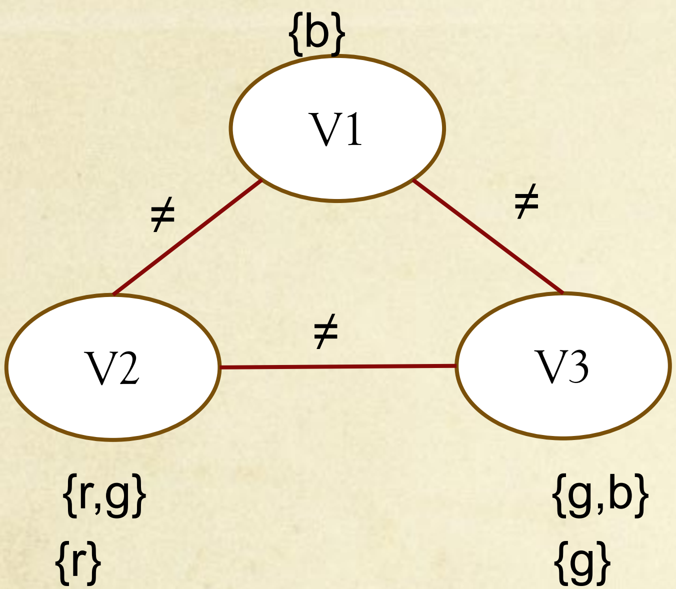
- El algoritmo más popular para asegurar la **arco-consistencia en una red** se llama AC-3
- Inicialmente el conjunto contiene todos los arcos del CSP
- A continuación extrae un arco cualquiera (X_i, X_j) del conjunto y hace X_i arco-consistente con respecto a X_j
  - Si esto reduce el dominio D_i, entonces añadimos al conjunto todos los arcos (X_k, X_i), tales que X_k es vecino de X_i
- Si un dominio se reduce a nada, entonces el CSP original no tenía solución, y devolvemos *false*. En otro caso obtenemos un CSP en el que es más fácil buscar

Algoritmo AC-3 (II)

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components $(X,\ D,\ C)$
  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 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 $true$

Algoritmo AC-3 (III)



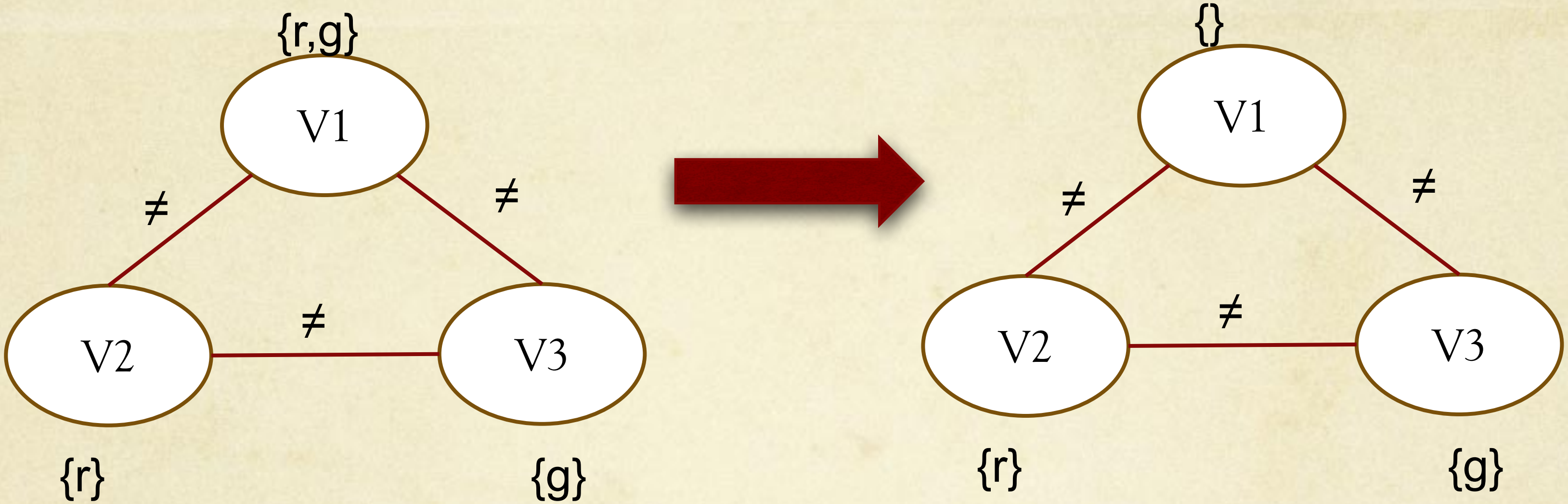

```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

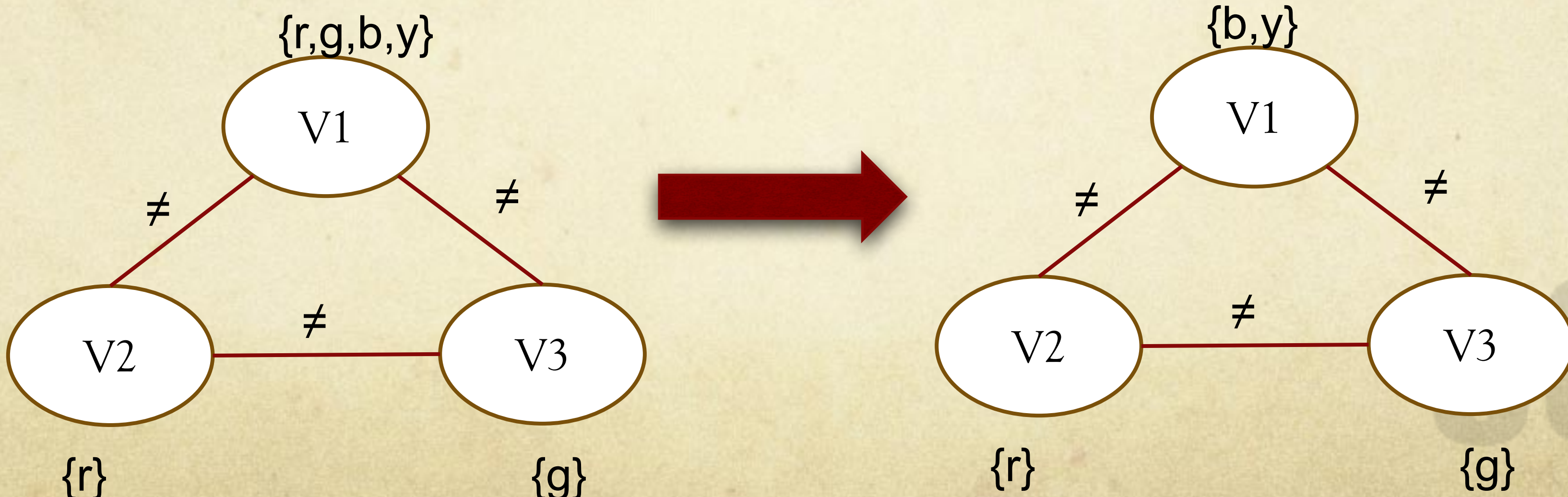
```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(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
```

Arcos	REVISE
(V1,V2)	FALSE
(V1,V3)	FALSE
(V2,V1)	FALSE
(V2,V3)	FALSE
(V3,V1)	TRUE (V3,b) eliminado Añado ~~(V1,V3)~~ (V2,V3)
(V3,V2)	FALSE
~~(V1,V3)~~	~~FALSE~~
(V2,V3)	TRUE (V2,g) eliminado Añado (V1,V2) ~~(V3,V2)~~
(V1,V2)	FALSE
~~(V3,V2)~~	~~FALSE~~

Algoritmo AC-3 (IV)





Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

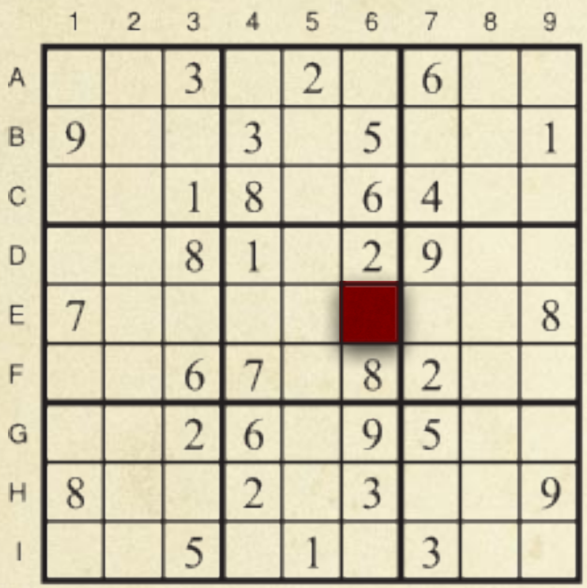
Sudoku

- Un sudoku es un CSP con 81 variables, una para cada casilla.
- Las casillas vacías tienen un dominio {1,2,3,4,5,6,7,8,9}
    - Las casillas rellenas tienen un dominio con un único valor.
- Hay 27 Alldiff restricciones: una para cada fila, columna y casillas de 9 esquinas.
    - Alldiff (A1,A2,A3,A4,A5,A6,A7,A8,A9)
    - Alldiff (B1,B2,B3,B4,B5,B6,B7,B8,B9)
    - Alldiff (A1,A2,A3,B1,B2,B3,C1,C2,C3)
    - ....

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

Sudoku



(a)

- Consideramos variable E6.
  - Casillas: Podemos eliminar 1,7,2,8 del dominio. $D_{casillas}=\{3,4,5,6,9\}$
  - Columna: Podemos eliminar 5,6,2,8,9,3 del dominio. $D_{Columna}=\{1,4,7\}$
  - Fila: Podemos eliminar 7 y 8 del dominio $D_{Fila}=\{1,2,3,4,5,6,9\}$
- $\text{Domino}_{E6}=\{4\}$

Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

- Consideramos variable I9.
  - Casillas: Podemos eliminar 3,5,9 del dominio. D_{I9}={1,2,4,6,7,8,9}
  - Columna: Podemos eliminar 1,8,9 del dominio. D_{I9}={2,4,6,7}
  - Fila: Podemos eliminar 1,5,3 del dominio D_{I9}={2,4,6,7}
- $Domino_{I9}$ ={2,4,6,7}

Sat. Restr.

Vuelta Atrás

Algoritmo básico (I)

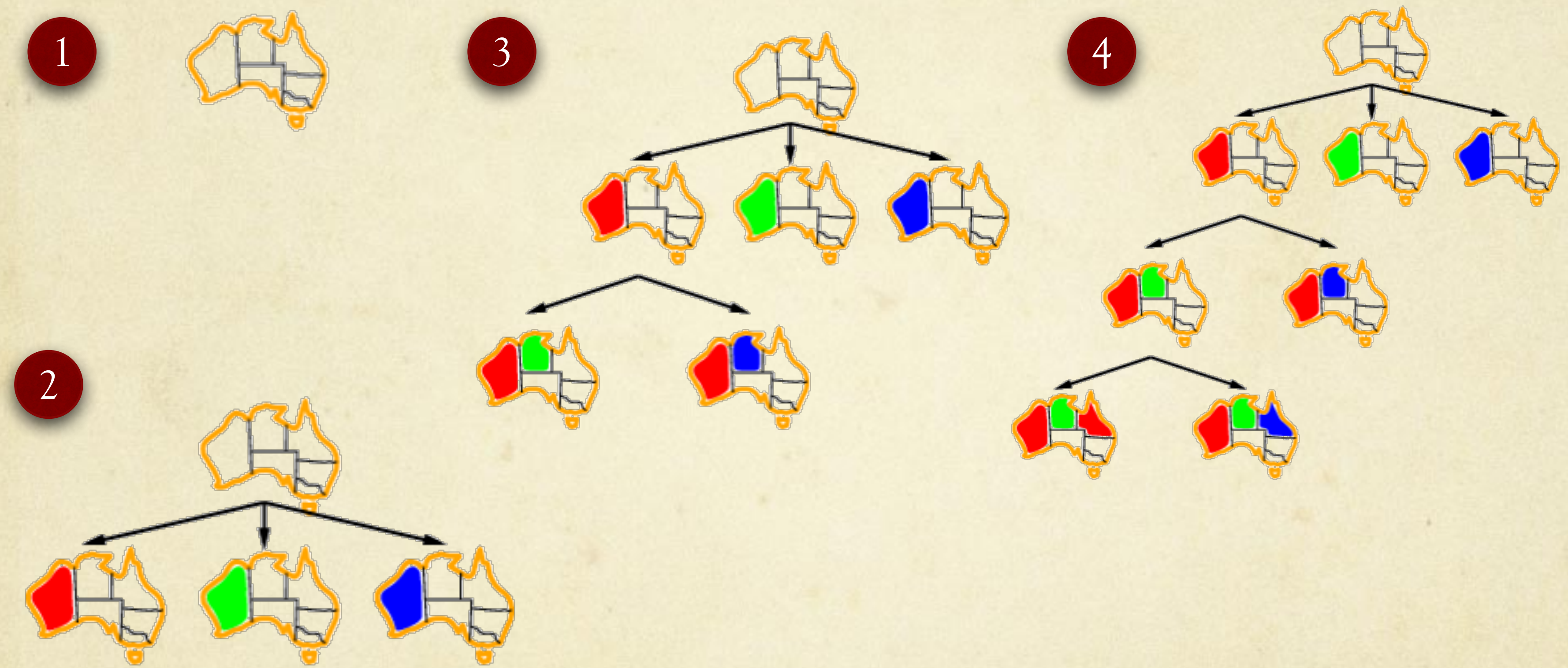
- Muchos CSPs no se pueden resolver solamente por inferencia sobre las restricciones; llega un momento en el que hay que buscar una solución
- La búsqueda por vuelta atrás es una búsqueda primero en profundidad que en cada momento elige un valor para una sola variable,
  - y vuelve atrás cuando una variable no tiene ningún valor legal que quede por probar
- Debemos considerar las siguientes preguntas:
  - ¿Qué variable debería ser asignada a continuación?
  - ¿En qué orden deberíamos probar sus valores?
  - ¿Qué inferencias deberían realizarse en cada paso?

Algoritmo básico (II)

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
   return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
         add {var = value} to assignment
         inferences ← INFERENCE(csp, var, value)
         if inferences ≠ failure then
            add inferences to assignment
            result ← BACKTRACK(assignment, csp)
            if result ≠ failure then
               return result
      remove {var = value} and inferences from assignment
   return failure
```

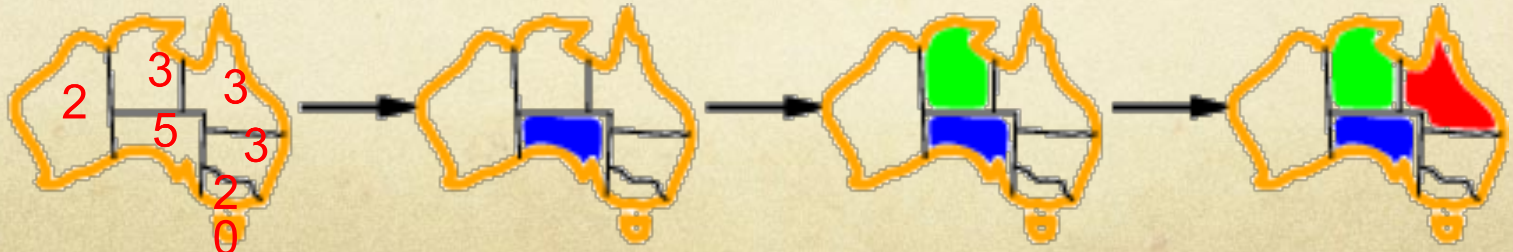
Ejemplo Algoritmo



Ordenación de las variables

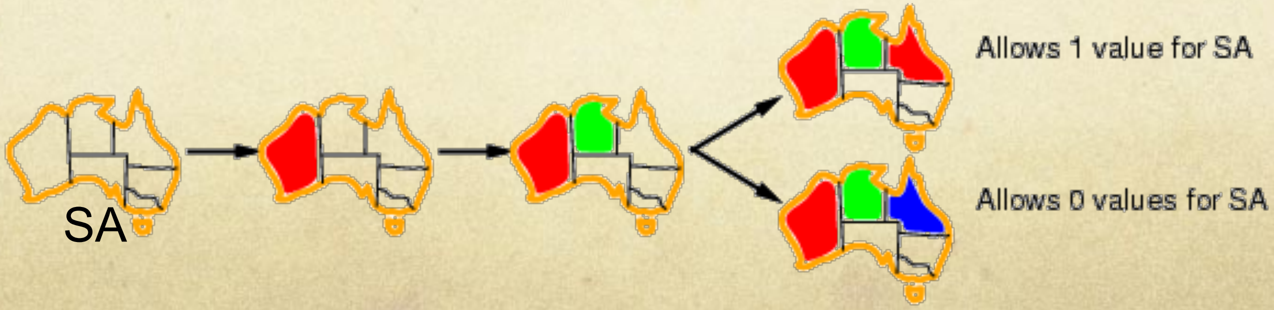
- Habitualmente se elige la variable que tenga el menor número de valores legales. Esto es lo que se llama el heurístico del mínimo número de valores restantes (*minimum remaining values heuristic*, MRV)

- También es posible emplear el heurístico del grado (*degree heuristic*, DEG), que elige la variable que interviene en el mayor número de restricciones con otras variables no asignadas
  - Esta manera de elegir las variables intenta obtener un fallo tan pronto como sea posible para podar secciones más grandes del árbol de búsqueda rápidamente



Ordenación de los valores

- En algunos casos el heurístico del valor menos restrictivo (*least constraining value*, LCV) puede resultar útil
- Prefiere el valor que elimina el menor número de opciones para las variables vecinas en el grafo de restricciones
- Este heurístico intenta obtener una solución tan pronto como sea posible eligiendo primero los valores más verosímiles



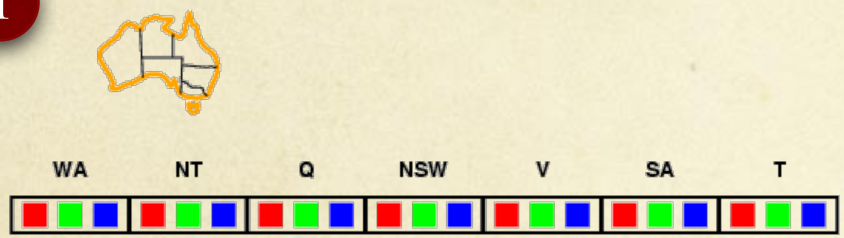
Intercalando búsqueda e inferencia

- Cada vez que hacemos una elección de un valor para una variable, intentamos inferir nuevas reducciones de dominio en las variables vecinas
- Una de las estrategias más sencillas es la comprobación hacia delante (*forward checking*)
  - Cada vez que se asigna una variable *X*, para cada variable no asignada *Y* que está conectada a *X* mediante una restricción, borramos del dominio de *Y* los valores que son inconsistentes con el valor elegido para *X*
- Para muchos problemas la búsqueda es más efectiva si combinamos la heurística MRV con la comprobación hacia delante.

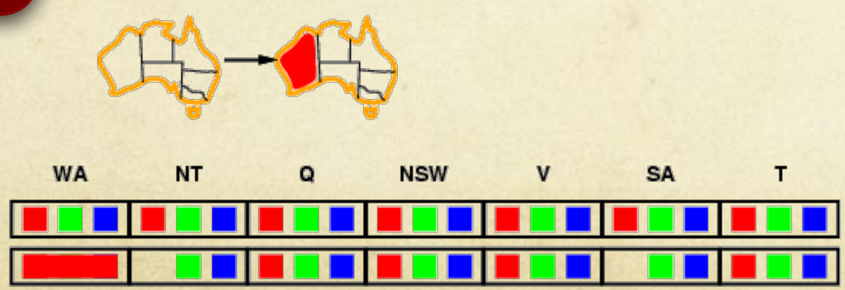
Intercalando búsqueda e inferencia

Forward checking

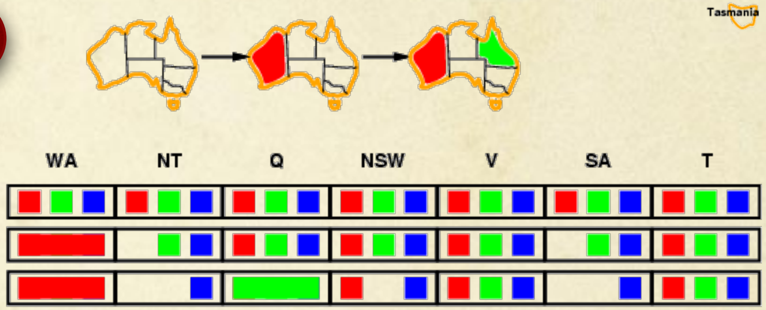
1



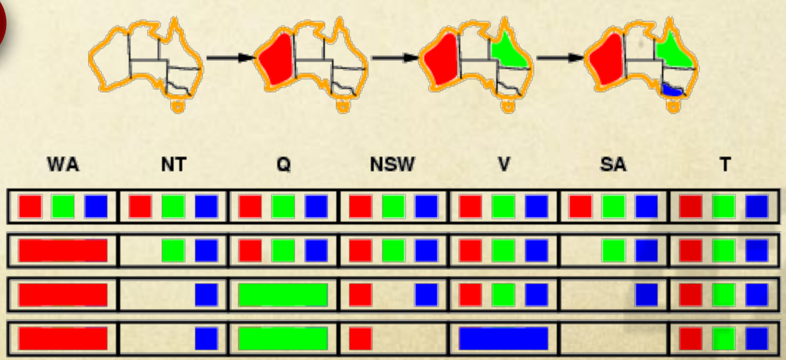
2



3



4



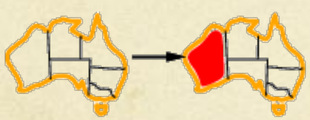
Intercalando búsqueda e inferencia

Forward checking y MRV (heurístico del mínimo número de valores restantes)

1

?

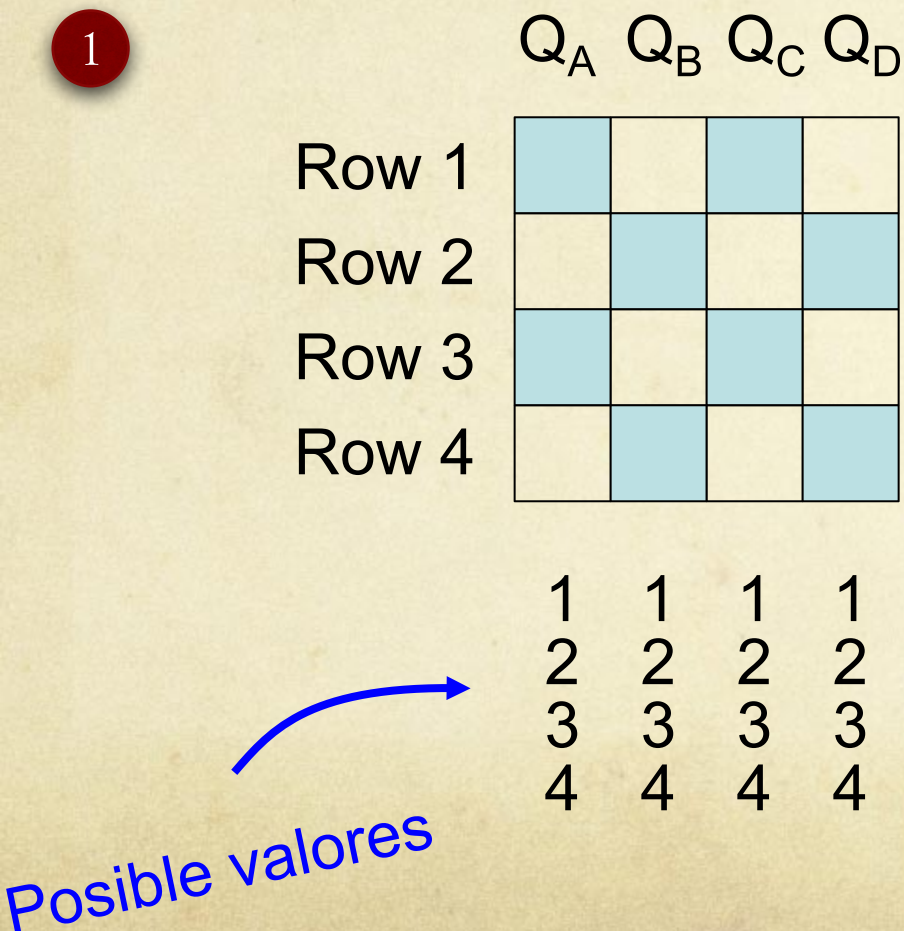
2



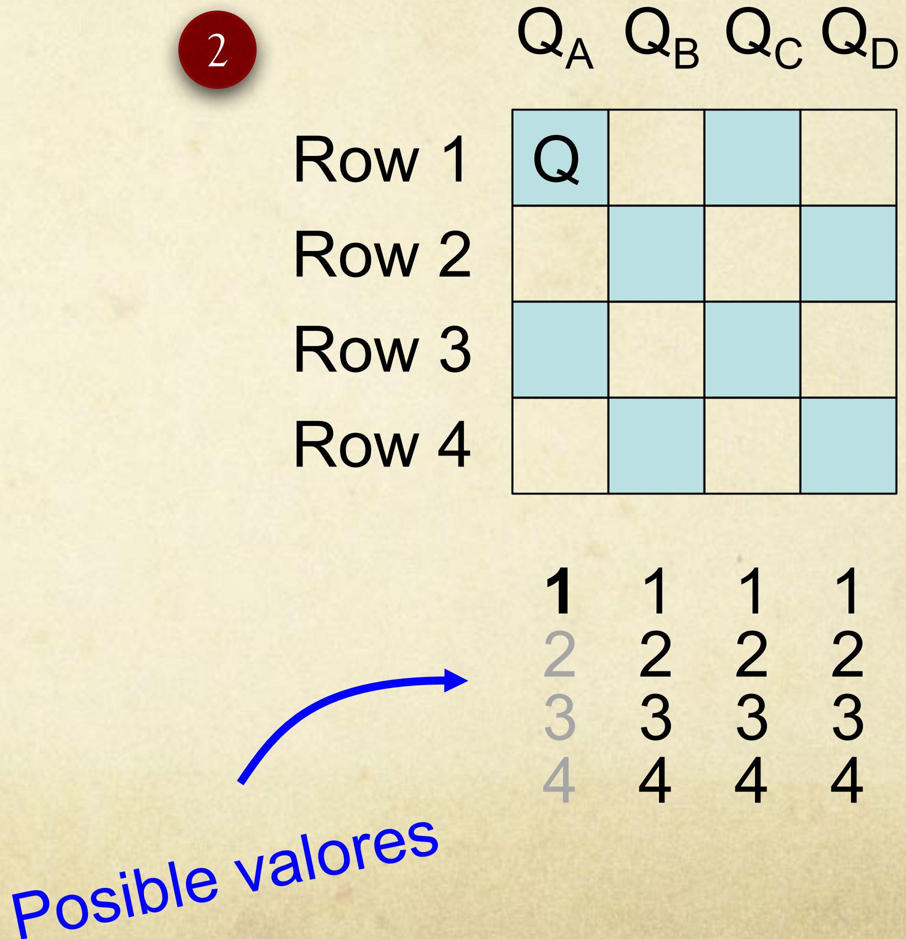
Intercalando búsqueda e inferencia

Forward checking N Reinas

1



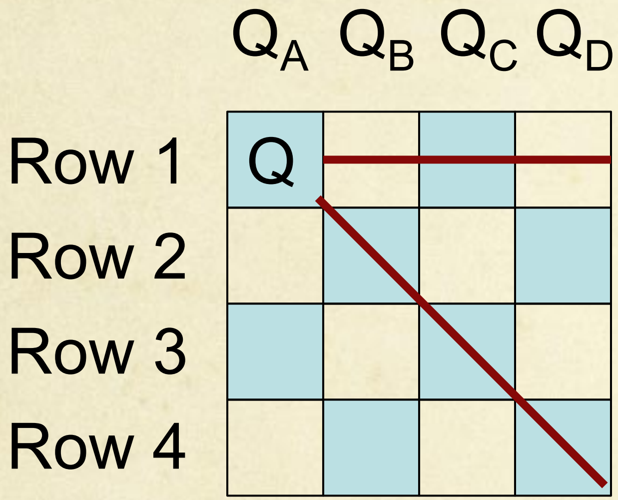
2



Intercalando búsqueda e inferencia

Forward checking N Reinas

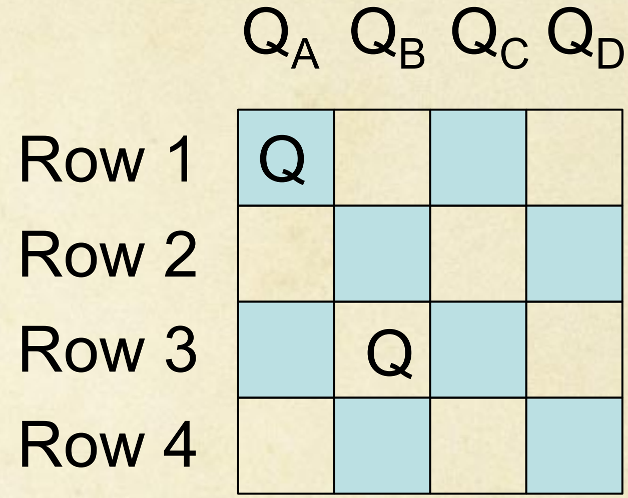
3



1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Posible valores

4

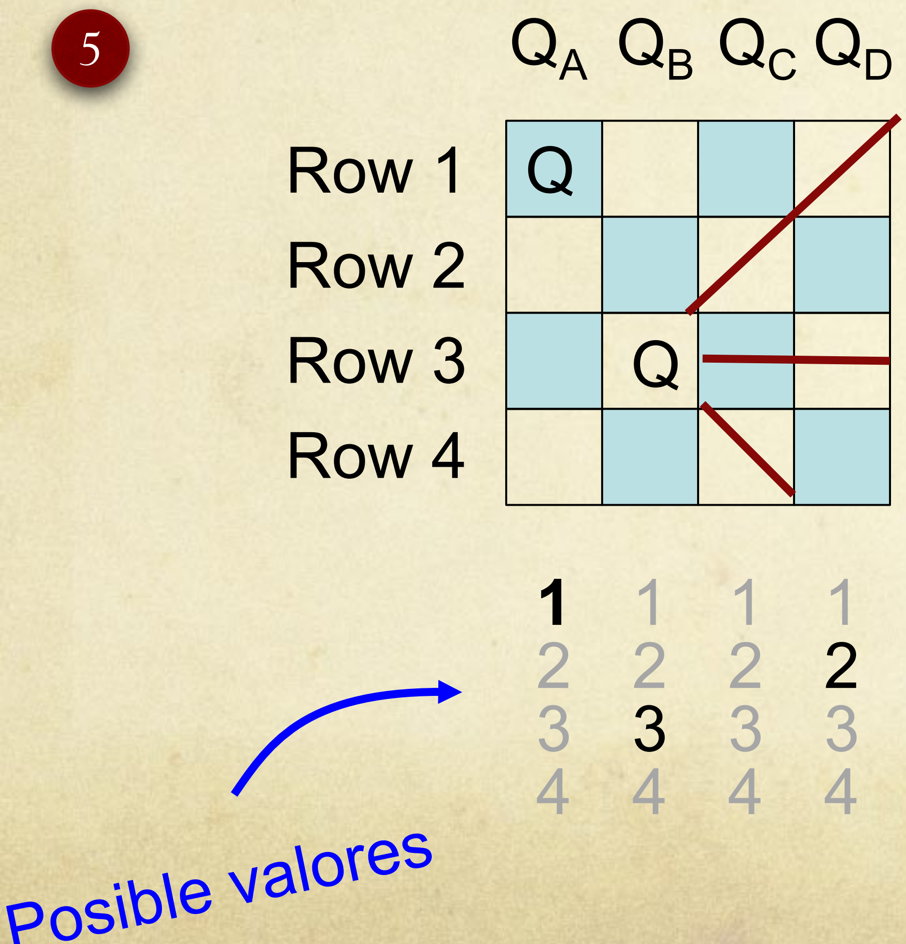


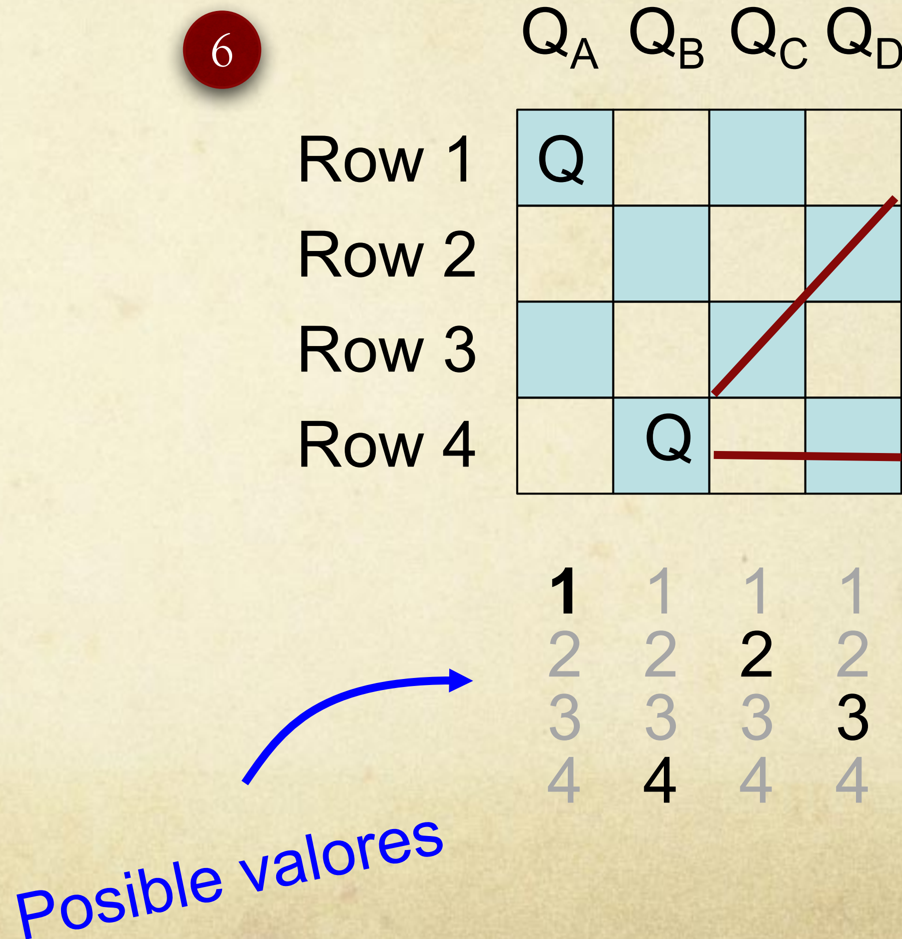
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Posible valores

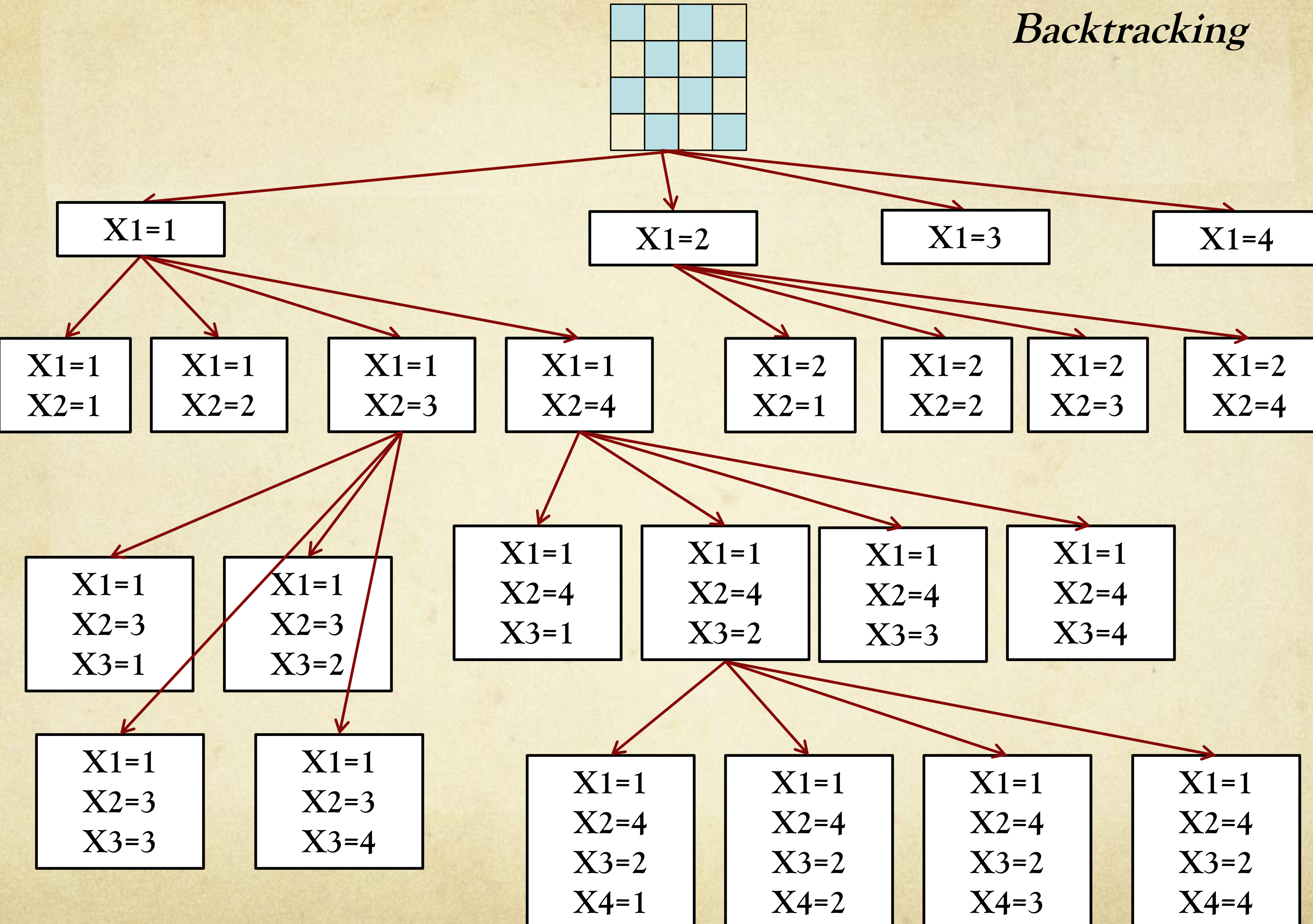
Intercalando búsqueda e inferencia

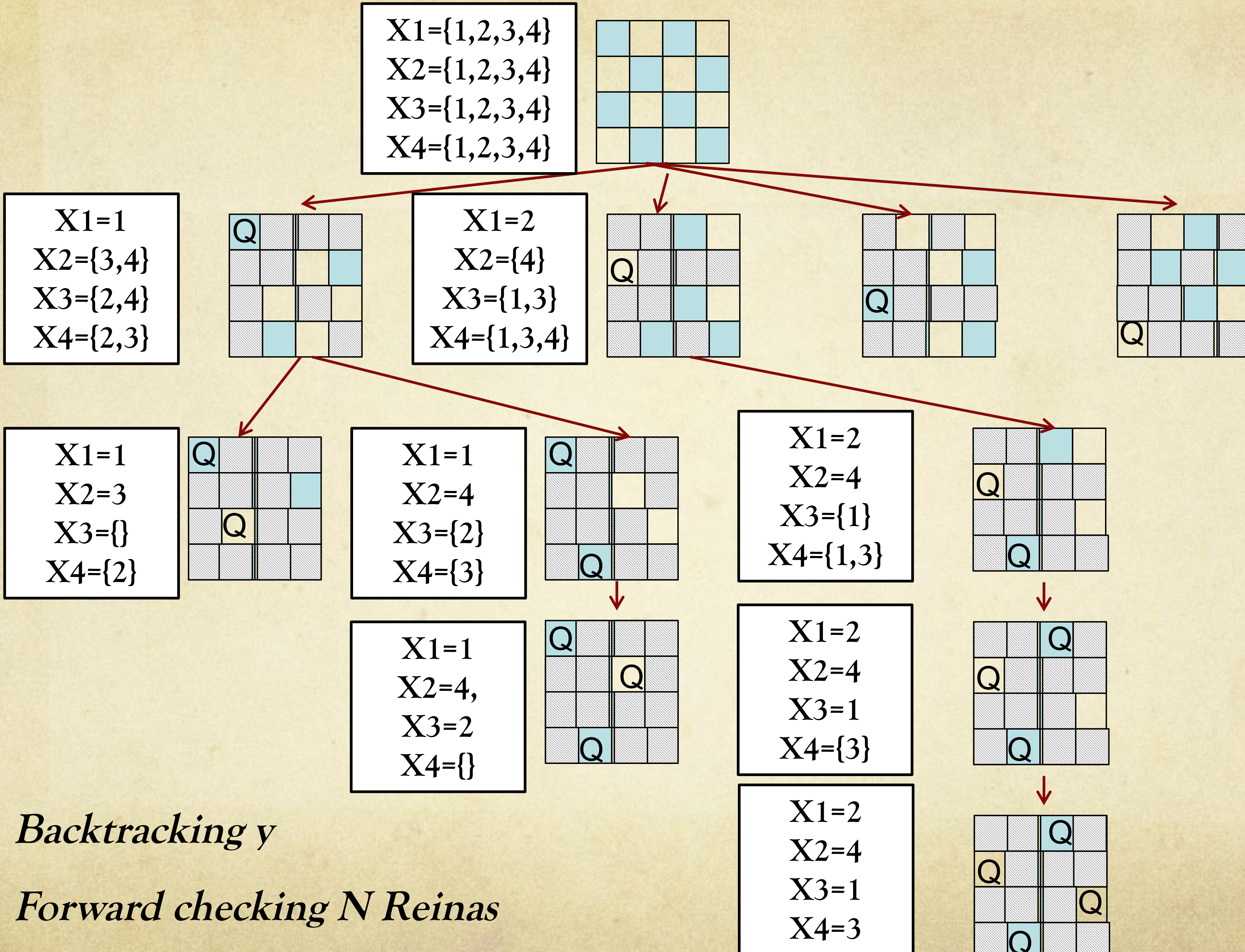
Forward checking N Reinas





Backtracking





Backtracking y

Forward checking N Reinas

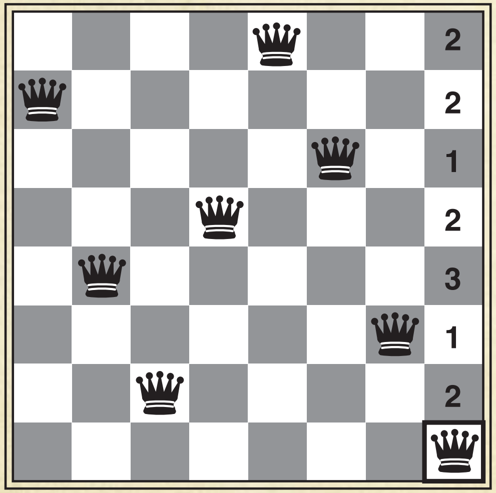
Búsqueda Local

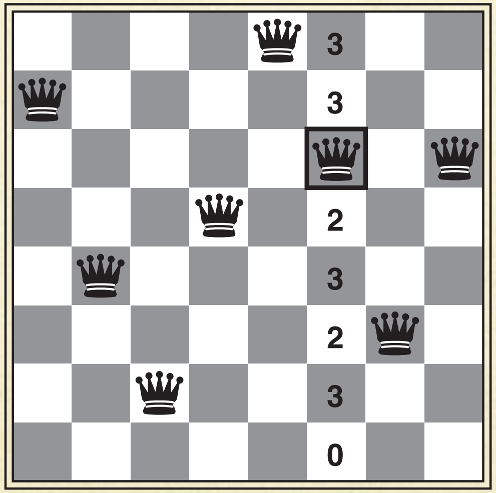
Búsqueda Local

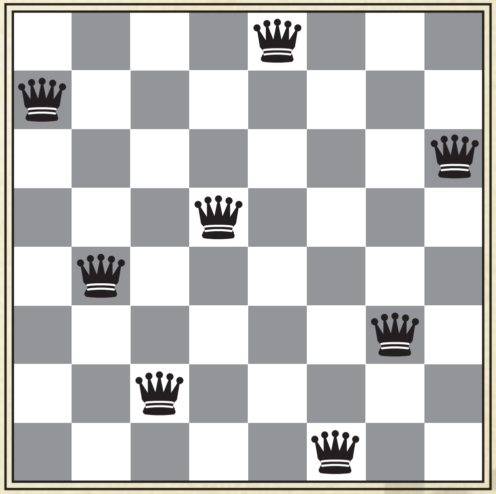
- Los algoritmos de búsqueda local pueden ser efectivos para resolver algunos problemas de CSP.
- El estado inicial establece un valor a cada variable
- Y la búsqueda cambia el valor de una variable cada paso.
- La solución inicial normalmente no cumplirá las restricciones, el objetivo es que se cumplan las restricciones.

Búsqueda Local

- Para escoger variable, la mejor heurística es seleccionar aquella que tiene un menor número de conflictos con otras variables (Min-conflicts heurístic)







Búsqueda Local

function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS($var, v, current, csp$)
    set $var = value$ in $current$
  return $failure$

Conclusiones

Sumario

- Los problemas de satisfacción de restricciones representan un estado mediante un conjunto de pares variable/valor y representan las condiciones que debe cumplir la solución mediante un conjunto de restricciones sobre las variables
- Las técnicas de inferencia usan las restricciones para inferir qué pares variable/valor son consistentes
- Habitualmente se emplea la búsqueda por vuelta atrás para encontrar una solución

Sistemas Inteligentes

José A. Montenegro Montes
monte@lcc.uma.es

Ejercicios

Ejercicio 1

Diseño de un horario:

El director de un instituto tiene que diseñar el horario de una clase.

Para simplificar vamos a suponer que el mismo horario se aplica a todos los días.

Hay seis horas, que deben rellenarse con estas asignaturas: Lengua, Inglés, Matemáticas, Física, Filosofía y Biología.

Ejercicio 1

- Hay algunas restricciones:
  - Las horas de Lengua e Inglés no pueden estar adyacentes, para que los alumnos no mezclen ambos idiomas.
  - Las Matemáticas no deben enseñarse a primera ni a última hora, para que los estudiantes estén lo suficientemente atentos.
  - La Física debe ir después de las Matemáticas.
  - La Filosofía sólo se puede asignar a la segunda o a la cuarta hora.

Ejercicio 1

- *Variables:*

Lengua, *Inglés*, *Matemáticas*, *Física*, *Filosofía* y *Biología* (una variable por asignatura).

- *Dominios:*

Lengua, *Inglés*, *Matemáticas*, *Física*, *Filosofía*, *Biología* ∈ {1, ..., 6}, donde los números representan las horas del horario.

Ejercicio 1

Restricciones:

- Dos asignaturas no se pueden enseñar al mismo tiempo a la misma clase, así que tendremos:

    AllDiff(*Lengua*, *Inglés*, *Matemáticas*, *Física*, *Filosofía*, *Biología*)

- También tendremos las restricciones correspondientes al enunciado del problema:

    a) $|Lengua - Inglés| > 1$

    b) $Matemáticas \in \{1,6\}$

    c) $Física > Matemáticas$

    d) $Filosofía \in \{2,4\}$

Ejercicio 1

Backtracking + FW + MRV

Filosofía={2,4}
Matemáticas ={3,4,5}
Física ={4,5,6}
Lengua ={1,3,4,5,6}
Inglés ={1,3,4,5,6}
Biología ={1,3,4,5,6}

Filosofía=2
Matemáticas ={3,4,5}
Física ={4,5,6}
Lengua ={1,3,4,5,6}
Inglés ={1,3,4,5,6}
Biología ={1,3,4,5,6}

Filosofía= 2
Matemáticas = 3
Física ={4,5,6}
Lengua ={1,4,5,6}
Inglés ={1,4,5,6}
Biología ={1,4,5,6}

Filosofía= 2
Matemáticas = 3
Física = 4
Lengua = {1,5,6}
Inglés ={1,5,6}
Biología ={1,5,6}

Filosofía= 2
Matemáticas = 3
Física = 4
Lengua = 1
Inglés ={5,6}
Biología ={5,6}

Filosofía= 2
Matemáticas = 3
Física = 4
Lengua = 1
Inglés = 5
Biología = {6}

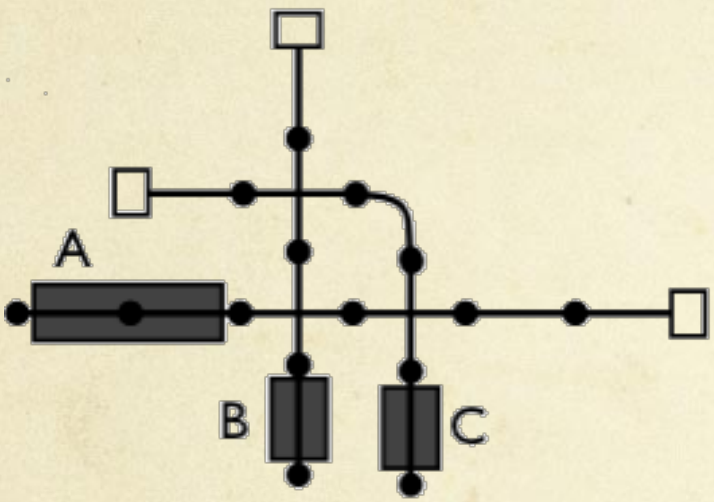
Filosofía= 2
Matemáticas = 3
Física = 4
Lengua = 1
Inglés = 5
Biología = 6

1. *AllDiff(Lengua, Inglés, Matemáticas, Física, Filosofía, Biología)*
2. $|Lengua - Inglés| > 1$
3. *Matemáticas* ∈ {1,6}
4. *Física*>*Matemáticas*
5. *Filosofía* ∈ {2,4}

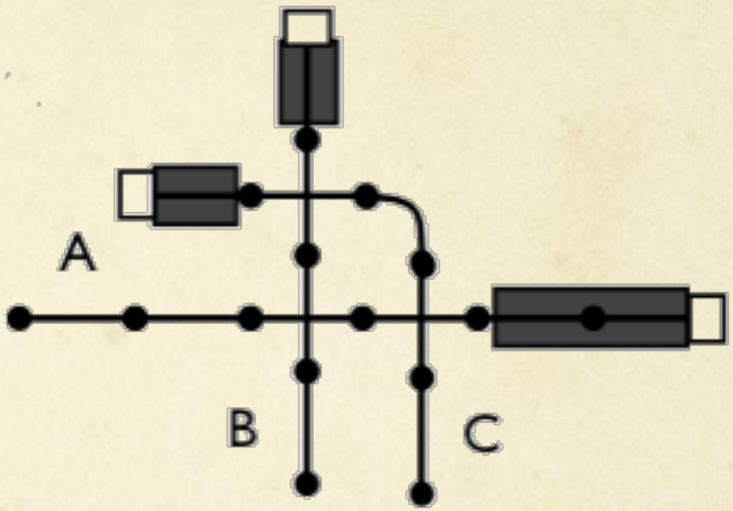
Ejercicio 2

- Un encargado de estación debe decidir cuando los trenes A, B y C deben partir. Una vez que los trenes han partido, se moverán un "hueco" en su vía a la hora hasta que llegue al destino.
  - Cada tren puede salir a las 1, 2 o 3 de la tarde.
  - Existen dos restricciones:
    - Todos los trenes deben salir en horas distintas y
    - Dos trenes no pueden ocupar a la vez los cruces de vías hasta que no pase una hora.
- Nótese que el tren A ocupa dos huecos. Además la restricción de colisión es impuesta solamente a la conclusión de cada hora, ya que consideramos en este problema el tiempo como variable discreta.

Ejercicio 2



Situación inicial



Situación final

Ejercicio 2

- *Variables y dominios:*

  Tendremos una variable por tren, almacenará su hora de salida: A, B, C $\in\{1, 2, 3\}$.

- ***Restricciones***:

  1. Cada tren debe salir a una hora distinta:

  $$AllDiff(A,B,C) \Leftrightarrow (A \neq B) \wedge (B \neq C) \wedge (A \neq C)$$

  2. Restricciones de las intersecciones:

  $$(A+1 \neq B) \wedge (A+1 \neq C) \wedge (A+2 \neq C) \wedge (B \neq C+1)$$