

Razonamiento con Restricciones

Tutorial CAEPIA 2003

**Javier Larrosa
Dep. LSI, UPC, Barcelona**

**Pedro Meseguer
IIIA, CSIC, Bellaterra**

Esquema Global

1. Introducción

- Definiciones
- Ejemplos

2. Métodos de Resolución

- Búsqueda
- Inferencia
- Métodos híbridos

3. Modelización

- Primal / dual
- Restricciones globales
- Programación con restricciones

4. Restricciones Blandas

- Modelos
- Algoritmos

Satisfacción de Restricciones

Definición: red de restricciones $P = (X, D, C)$

$X = \{X_1, X_2, \dots, X_n\}$ variables

$D = \{D_1, D_2, \dots, D_n\}$ dominios

$C = \{C_1, C_2, \dots, C_e\}$ restricciones

Dada una restricción C_i ,

$\text{var}(C_i) = \{X_{i1}, \dots, X_{ik}\}$ relaciona k variables
(restricción k -aria)

$\text{rel}(C_i) \subseteq D_{i1} \times D_{i2} \times \dots \times D_{ik}$ tuplas de valores
permitidos

CSP: problema de resolver la red de restricciones

Solución: asignación de valores a variables
satisfaciendo todas las restricciones

Complejidad: NP-completo
algoritmos exponenciales (caso peor)

Interés de los CSP

Relevancia:





- Problemas reales como CSPs:
 - Car sequencing problem
 - Asignación de recursos (*scheduling*)
 - Diseño de bloques (*BIBDs*)
- Para la IA:
 - Restricciones: formal general de representación del conocimiento
 - Satisfacción de restricciones: razonamiento automático
 - Ejemplos:
 - » SAT
 - » razonamiento temporal
 - » razonamiento basado en modelos

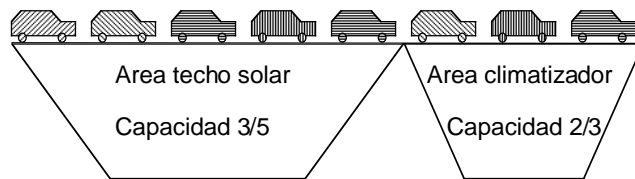
Especialización:

- tipo de dominios:
 - discretos / continuos
 - finitos / infinitos
- tipo de restricciones:
 - binarias / n-arias

Car sequencing problem

Cadena de montaje de coches

				
FAROS ANTINEBLA	X		X	
TECHO SOLAR	X	X		X
CLIMATIZADOR		X	X	



Formulación:

- variables: n coches a producir
- dominios: modelos de coche
- restricciones: capacidad de las áreas

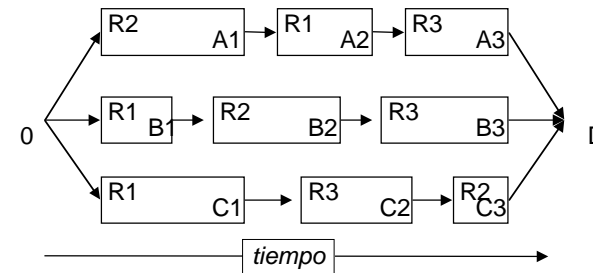
Características:

- CSP no binario, discreto y finito

Asignación de recursos

Job-shop scheduling

- n jobs, cada uno con m operaciones,
 - m recursos, cada operación necesita un recurso de forma exclusiva durante un tiempo
 - precedencia entre las operaciones de cada job,
- ¿Se pueden realizar los n jobs en tiempo D ?



Formulación:

- variables: operaciones
- dominios: tiempos de inicio de cada operación
- restricciones:
 - precedencia entre las operaciones de un job
 - exclusividad de cada recurso en el tiempo

Características:

- CSP binario, discreto y finito (acotado por D)

Restricciones binarias

CSP binario:

$X = \{ X_1, X_2, \dots, X_n \}$ variables

$D = \{ D_1, D_2, \dots, D_n \}$ dominios discretos y finitos

$C = \{ C_{ij} \}$ restricciones binarias

$var(C_{ij}) = \{ X_i, X_j \}$

$rel(C_{ij}) = \{ \text{valores permitidos para } X_i \text{ y } X_j \} = R_{ij}$

$n = |X|$; $d = \max_i |D_i|$; $e = |C|$

Solución: asignación de valores a variables satisfaciendo todas las restricciones.

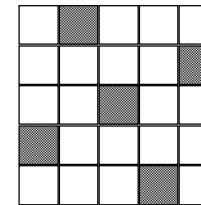
Generalidad: todo problema n-ario se puede reformular como binario [Rossi et al, 90]

Ejemplos:

- Coloreado de grafos
- Satisfacibilidad booleana (SAT)
- N-reinas, crucigramas, criptoaritmética

N-reinas

Definición: posicionar n reinas en un tablero de ajedrez $n \times n$, de forma que no se ataquen.



$n = 5$

Formulación: 1 reina por fila

- variables: reinas, X_i reina en la fila i -ésima
- dominios: columnas posibles $\{1, 2, \dots, n\}$
- restricciones: no colocar dos reinas en
 - la misma columna
 - la misma diagonal
$$R_{ij} = \{(a,b) \mid a \neq b \wedge |i-j| \neq |a-b|\}$$

Características:

- CSP binario, discreto y finito
- existe un método de solución constructivo

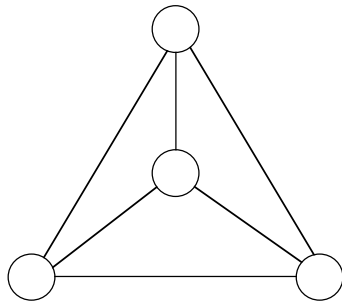
Coloreado de grafos

Definición: Dado un grafo,

- n nodos
- m colores,

asignar un color a cada nodo de forma que no haya dos nodos adyacentes con el mismo color.

Colores



Formulación:

- variables: nodos
- dominios: colores posibles
- restricciones: \neq nodos adyacentes

Características:

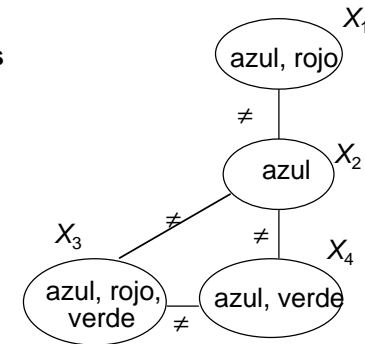
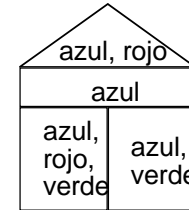
- CSP binario, discreto y finito

Grafo de restricciones

Grafo de restricciones:

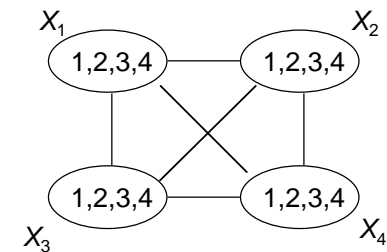
- $\{X_i\}$ nodos
- $\{D_i\}$ dominios en los nodos
- $\{C_{ij}\}$ arcos etiquetados

Coloreado de mapas



4-reinas

	1	2	3	4
X_1				
X_2				
X_3				
X_4				



Esquema Global

1. Introducción

- Definiciones
- Ejemplos

2. Métodos de Resolución

- Búsqueda ←
- Inferencia
- Métodos híbridos

3. Modelización

- Primal / dual
- Restricciones globales
- Programación con restricciones

4. Restricciones Blandas

- Modelos
- Algoritmos

Métodos de resolución: Búsqueda

Búsqueda:

- Explora el *espacio de estados* del problema (configuraciones posibles)
- Termina cuando:
 - Encuentra una solución
 - Demuestra que no hay solución
 - Agota los recursos computacionales

Búsqueda sistemática (BS):

- Visita *todos* los estados que podrían ser solución
- Algoritmos *completos*:
 - Si hay solución, la encuentran
 - Si no hay, demuestran que no existe
- Complejidad *exponencial* (caso peor)

Búsqueda local (BL):

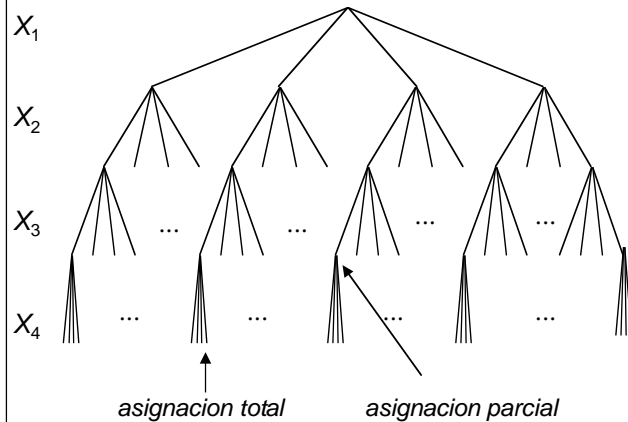
- Visita estados de forma heurística:
 - *No garantiza* visitar todos los estados
 - Puede *repetir* visitas al mismo estado
- Algoritmos *incompletos*:
 - Puede haber solución y no encontrarla
- Complejidad: acotada por los recursos

Búsqueda sistemática: árbol de búsqueda

Espacio de estados: representable por un árbol

- raíz: asignación vacía
- a cada nivel asociamos una variable
- sucesores: valores de la variable del nivel
- rama: define una asignación

4-reinas:



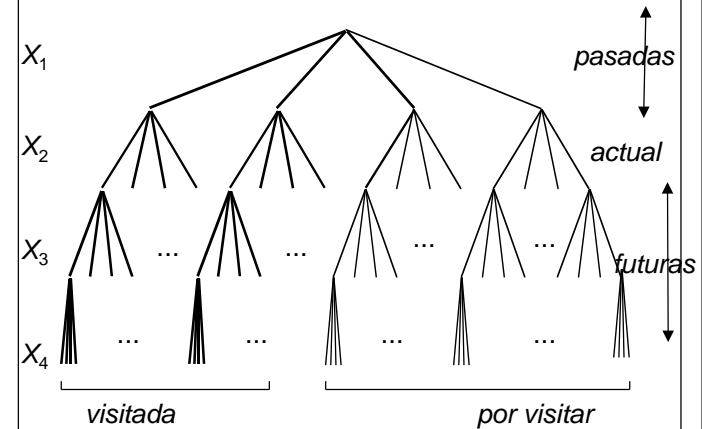
Arbol de búsqueda:

- contiene *todos* los estados
- recorrido *exhaustivo* → método *completo*

Recorrido del árbol de búsqueda

Recorrido primero en profundidad: preorden

- en cada nivel se asigna una nueva variable
 - variable *actual*
- variables
 - *pasadas* (asignadas), P
 - *futuras* (sin asignar), F



Si nodo actual inconsistente:

- subárbol sucesor *no* contiene soluciones
- no se visita → se *poda*

Backtracking

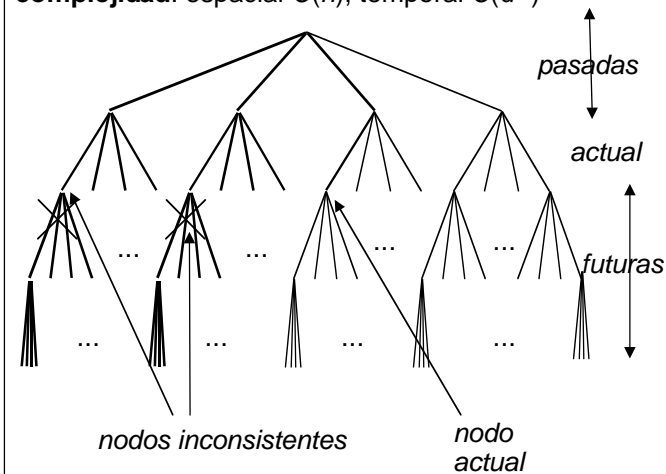
Búsqueda: primero en profundidad (DFS)

En cada nodo: consistencia entre variables asignadas

consistencia($P, actual$): si consistente, continua DFS, sino, backtracking

consistencia($P, actual$): es suficiente comprobar que *actual* es consistente con P

complejidad: espacial $O(n)$, temporal $O(d^n)$

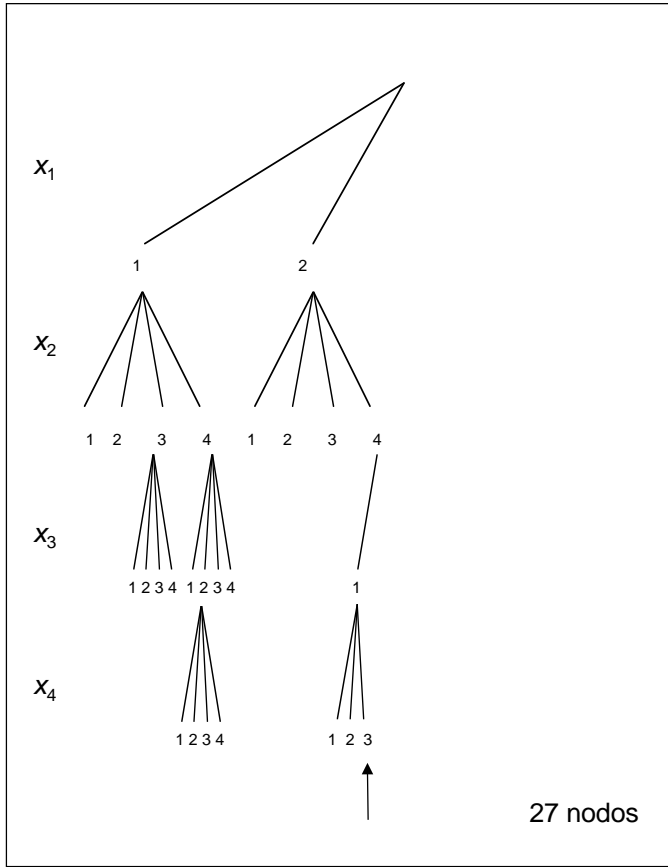


BT: Código

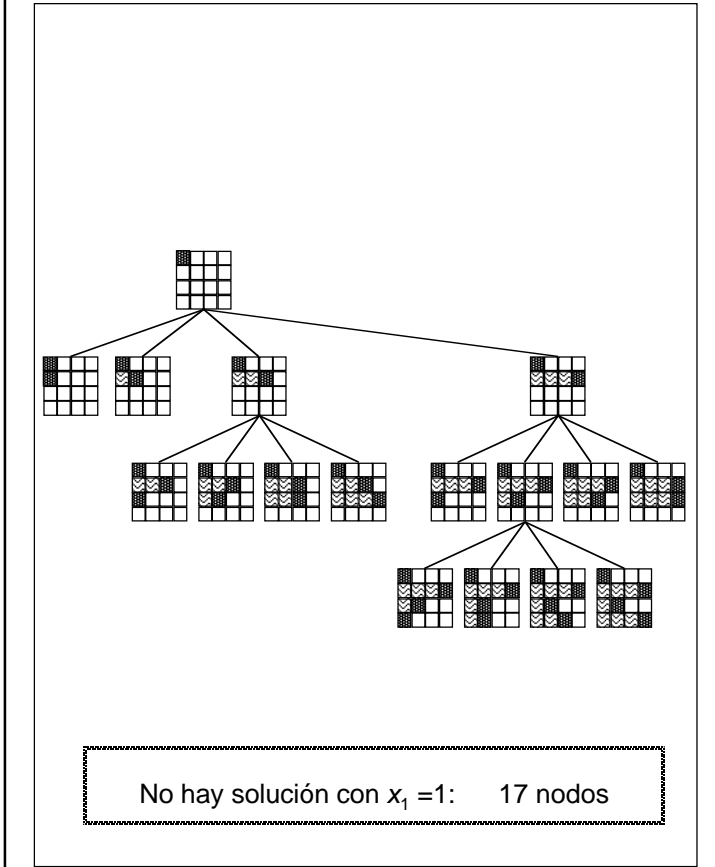
```
funcion test(i, a, pasadas):booleano
  para todo  $x_j \in pasadas$  hacer
    si  $(a, valor(x_j)) \notin R_{ij}$  retorna FALSO;
  retorna CIERTO;
```

```
funcion BT(i, pasadas): booleano
  para todo  $a \in D_i$  hacer
     $x_i := a$ ;
    si test(i, a, pasadas) entonces
      si  $i = n$  retorna CIERTO;
      sino si BT(i+1, pasadas  $\cup \{x_i\}$ )
        retorna CIERTO;
  retorna FALSO;
```

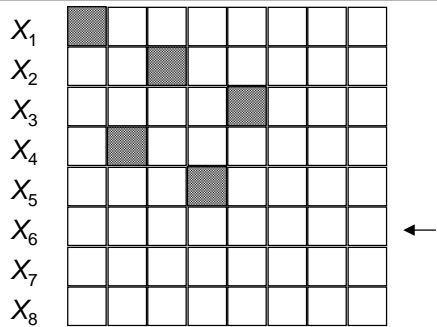

Backtracking: 4-reinas



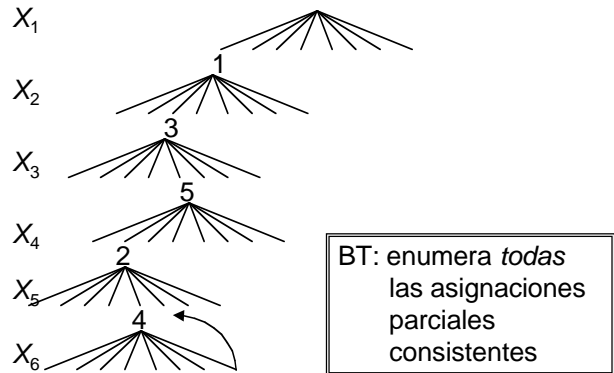
BT: Ejemplo



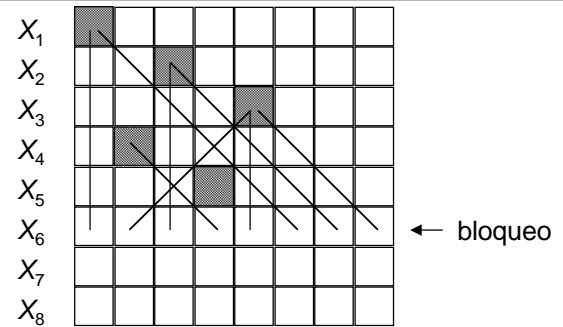
Backtracking cronológico



Arbol de búsqueda:

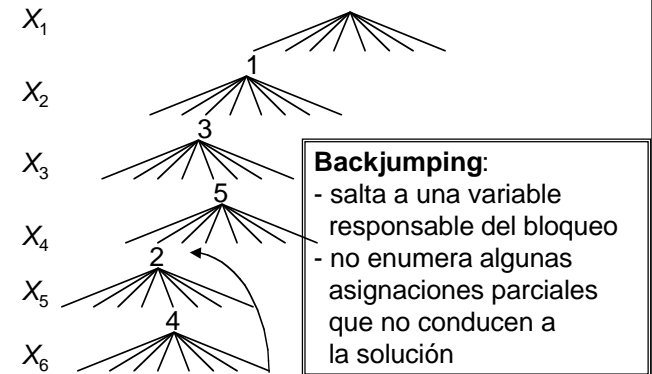


Backjumping



Observación:

- cambiar el valor de X_5 no elimina el bloqueo
- backtracking sobre una variable anterior: X_4



Backjumping dirigido por conflictos

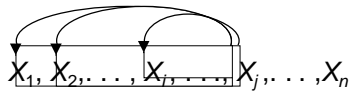
Conjunto conflicto de X_j :

- var pasadas incompatibles con algun valor de X_j

X_1	█																			{}	conjunto conflicto
X_2	1	1	█																	{1}	
X_3	1	2	1	2	█															{1, 2}	
X_4	1	█																		{1}	
X_5	1	4	2	█																{1, 2, 4}	
X_6	1	3	2	4	3	1	2	3												{1, 2, 3, 4}	
X_7																					
X_8																					

Proceso:

- backjumping (X_j) = max conjunto conflicto(X_j)



- tras backjump de X_j a X_i ,
conjunto conflicto (X_i) = conjunto conflicto (X_i) \cup
{conjunto conflicto (X_j) - X_j }

- se transpasan a X_i los conflictos de X_j con
variables anteriores a X_i

Búsqueda local

Esquema:

- Optimización función objetivo: $\min F(s)$
- Proceso iterativo:
 $s \rightarrow s' \rightarrow s'' \rightarrow s''' \rightarrow \dots$
- Estrategia greedy:
 $F(s) \geq F(s') \geq F(s'') \geq F(s''') \dots$
hasta solución o agotar recursos
- Problema: *mínimos locales*
 - permitir que $F(s) < F(s')$
 - movimientos aleatorios, reinicios, etc.

Elementos:

- Función objetivo $F(s)$: asocia a cada estado s un coste $F(s)$
- Vecindad $N(s)$: estados a los que puede ir desde s , en la siguiente iteración
- Criterio selección: dado $F(s)$ y $N(s)$, elegir el siguiente estado s'

Complejidad: acotada por los recursos

Búsqueda local y CSP

Estado s : asignación con todas las variables

Vecindad $N(s)$:

- s' que difieren de s en valores de i variables
- normalmente $1 \leq i \leq 2$

Función objetivo:

- $F(s) = 0$, si s es solución
- $F(s) > 0$, en otro caso

Algoritmo Breakout: [Morris 92]

- cada restricción tiene un peso
- $F(s)$ = suma pesos restricciones no satisfechas
- si una restricción no se satisface, su peso se incrementa

Algoritmo GSAT: [Selman et al, 92]

- busca un modelo en una fórmula proposicional
- cambia la variable que
 - mejora más o empeora menos
 - aleatoriedad, memoria, etc.

Esquema Global

1. Introducción

- Definiciones
- Ejemplos

2. Métodos de Resolución

- Búsqueda
- Inferencia ←
- Métodos híbridos

3. Modelización

- Primal / dual
- Restricciones globales
- Programación con restricciones

4. Restricciones Blandas

- Modelos
- Algoritmos

Métodos de resolución: Inferencia

Inferencia:

- Deduce nuevas restricciones *implícitas*
- P genera un $P' = P +$ restricciones implícitas
- P' es equivalente a P , $SOL(P) = SOL(P')$

Completa:

- *consistencia global*
- sintetiza una única restricción global
tuplas permitidas = *soluciones*
- resuelve completamente el problema
- coste exponencial

Incompleta:

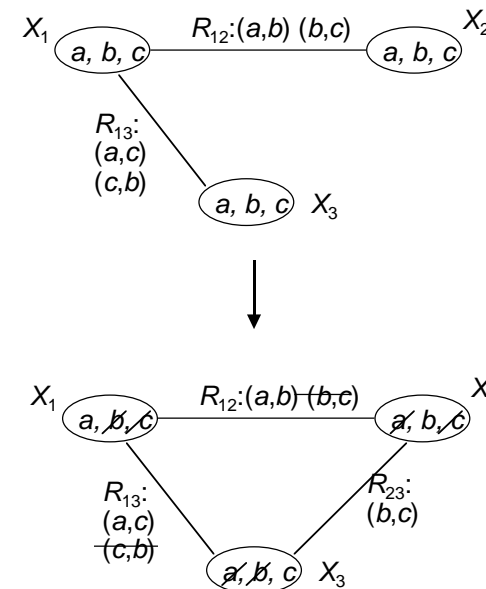
- *consistencia local*
- P' es más fácil de resolver que P
| espacio estados P | > | espacio de estados P' |
- no resuelve completamente el problema,
 - necesita búsqueda
 - puede detectar si no hay solución
- coste polinómico
- uso: antes o durante la búsqueda

Propagación de restricciones

Restricciones:

- explícitas: definición del problema
- implícitas: inducidas por la acción de las explícitas

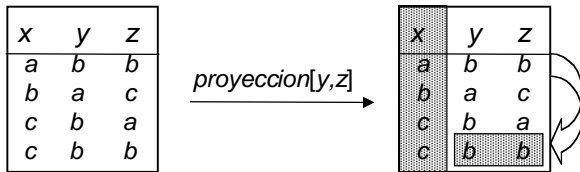
Propagación: descubre ciertas restricciones implícitas



Operaciones con restricciones: Proyección y Join

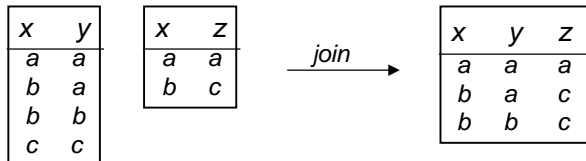
Proyección:

- c restricción, proyección c sobre $var(c) - \{x\}$: c'
 - $var(c') = var(c) - \{x\}$
 - $rel(c')$: formado por las tuplas de $rel(c)$, eliminando la componente de x



Join:

- c, c' restricciones, $join(c, c') = c''$ tal que
 - $var(c'') = var(c) \cup var(c')$
 - t tupla de valores sobre $var(c'')$,
 $t \in rel(c'')$ ssi $t[var(c)] \in rel(c)$ y
 $t[var(c')] \in rel(c')$



Inferencia Completa

Sintetizar una restricción n-aria, global:

- que sustituya a las restricciones iniciales
- sus tuplas son las soluciones del CSP

Es fácil: *join* de todas las restricciones iniciales

$$join(C_1, C_2, \dots, C_n)$$

Complejidad:

- espacial $O(d^n)$, temporal $O(d^n)$
- es MUY costoso!!
- es MAS de lo necesario para encontrar todas las soluciones del CSP !!



consistencia adaptativa [Dechter, Pearl, 87]

Consistencia Adaptativa

Problema P , variable X , C_x : restricciones sobre X

IDEA:

- Sustituir C_x por \underline{c}
- \underline{c} resume el efecto de C_x sobre P
- \underline{c} no menciona X

} *eliminación de variable*

entonces X se puede eliminar

PROCESO: orden *estático* de variables

problemas	P	\rightarrow	P'	\rightarrow	P''	\rightarrow	\dots	\rightarrow	$P^{(n-1)}$
variables	n		$n-1$		$n-2$				1

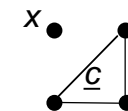
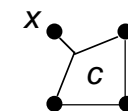
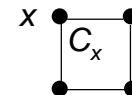
SOLUCION:

- $S^{(n-1)}$ solución de $P^{(n-1)}$
- ↓
- $S^{(n-2)}$ solución de $P^{(n-2)}$ a partir de $S^{(n-1)}$
- ↓
- \dots
- ↓
- S solución de P a partir de S'

Eliminación de Variable

Para eliminar var X :

- *Join* todas las restricciones $C_x \rightarrow c$
- Sustituir C_x por c
- Proyectar la variable x fuera $c \rightarrow \underline{c}$
- Sustituir c por \underline{c}
- Si existe c' $var(c') = var(\underline{c})$, $\underline{c} \leftarrow \underline{c} \cap c'$

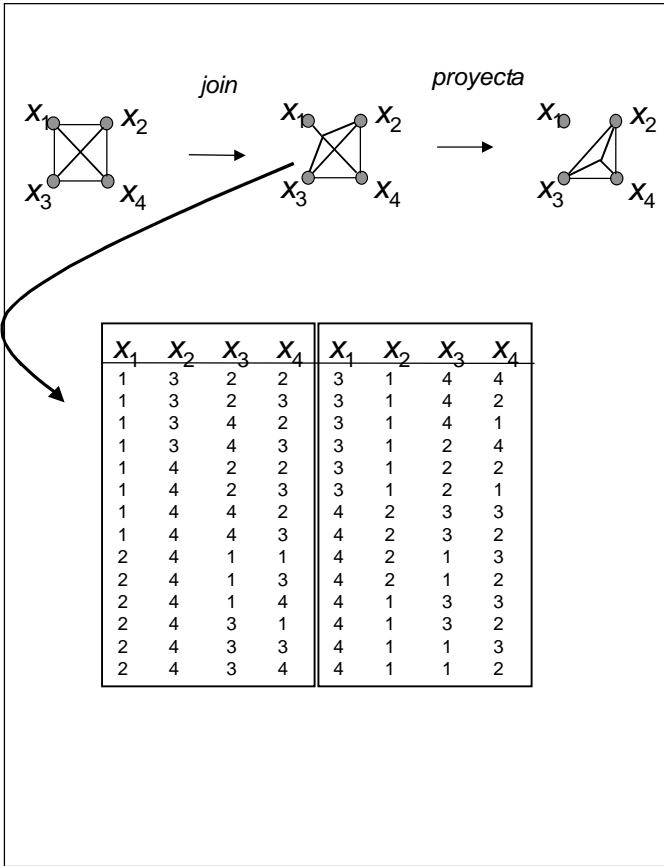


Para obtener la solución:

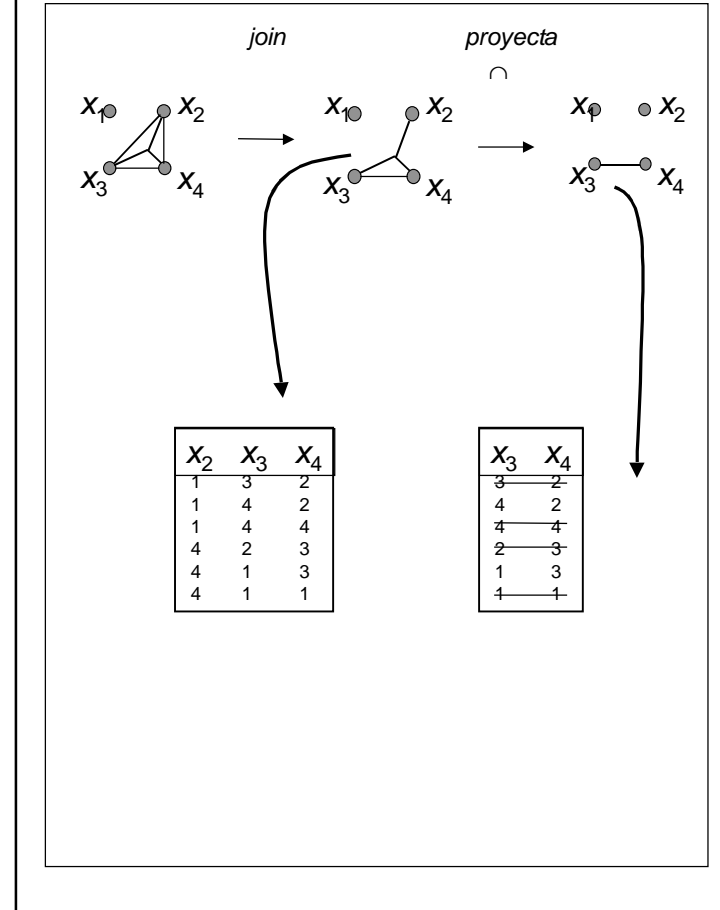
- Variables se procesan en orden inverso
- Se asigna a X un valor consistente con variables anteriores, y con restricciones intermedias totalmente asignadas

Complejidad: espacial $O(nd^{w^*})$, temporal $O(n(2d)^{w^*+1})$
 w^* : anchura inducida del grafo por el orden de vars
 (máxima aridad de las restricciones intermedias)

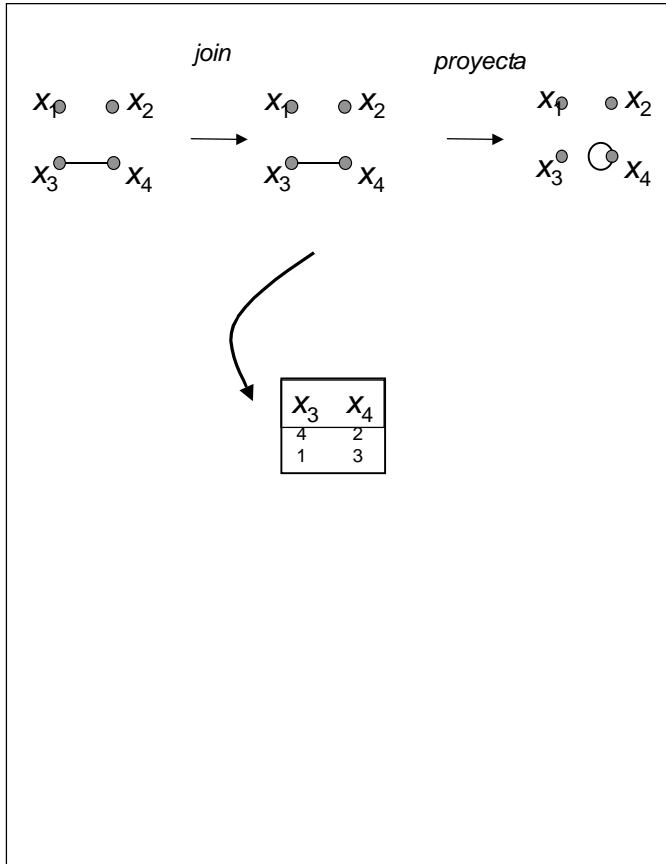
Ejemplo: 4-reinas (x_1)



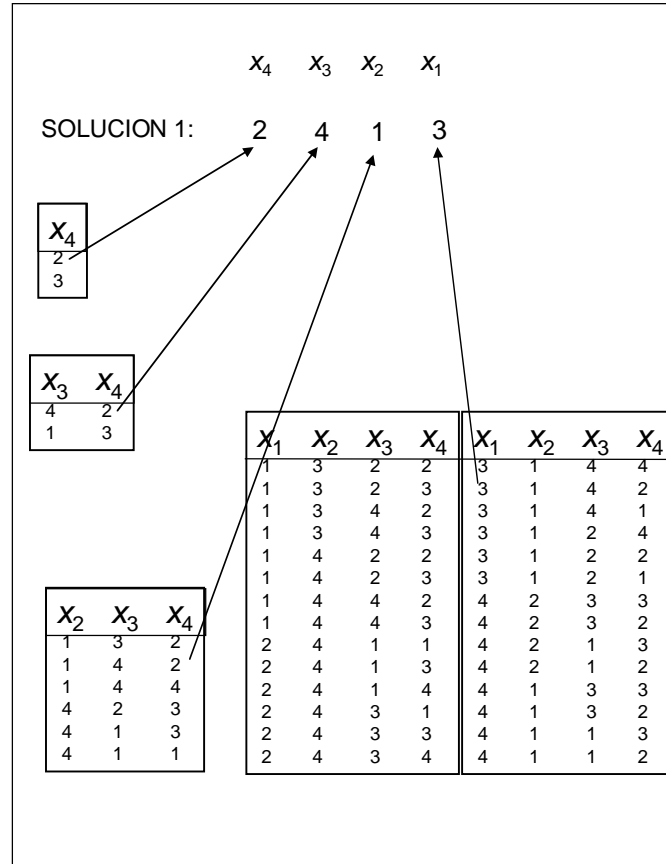
Ejemplo: 4-reinas (x_2)



Ejemplo: 4-reinas (x_3)



Ejemplo: Todas las soluciones 4-reinas



Ejemplo: Todas las soluciones 4-reinas

SOLUCION 2: $x_4 \quad x_3 \quad x_2 \quad x_1$
 3 1 4 2

x_4
2
3

x_3	x_4
4	2
1	3

x_2	x_3	x_4
1	3	2
1	4	2
1	4	4
4	2	3
4	1	3
4	1	1

x_1	x_2	x_3	x_4
1	3	2	2
1	3	2	3
1	3	4	2
1	3	4	3
1	4	2	2
1	4	2	3
1	4	4	2
1	4	4	3
2	4	1	1
2	4	1	3
2	4	1	4
2	4	3	1
2	4	3	3
2	4	3	4

x_1	x_2	x_3	x_4
3	1	4	4
3	1	4	2
3	1	2	4
3	1	2	2
3	1	2	1
4	2	3	3
4	2	3	2
4	2	1	3
4	2	1	2
4	1	3	3
4	1	3	2
4	1	1	3
4	1	1	2

Inferencia incompleta: Consistencia local

Subredes de 1, 2, 3 ... variables de P

Determinar si son consistentes:

- SI, pero hay valores que no aparecen en ninguna solución de la subred, \longrightarrow se eliminan de P
- NO, \longrightarrow P no tiene solución

Niveles de inferencia:

- Subredes de 1 variable: Nodo consistencia
- Subredes de 2 variables: Arco consistencia
- Subredes de 3 variables: Camino consistencia
-
- Subredes de k variables: k -consistencia

- Si un dominio queda vacío: NO hay solución

Consistencia de nodos

- Variable x_i es *nodo consistente* (NC) ssi todo valor de D_i está permitido por R_i
- P es NC ssi todas sus variables son NC

Algoritmo NC:

```
procedimiento NC-1 ( $X, D, C$ )  
  para todo  $x_i \in X$  hacer  
    para todo  $a \in D_i$  hacer  
      si  $a \notin R_i$  entonces  $D_i := D_i - \{a\}$ ;
```

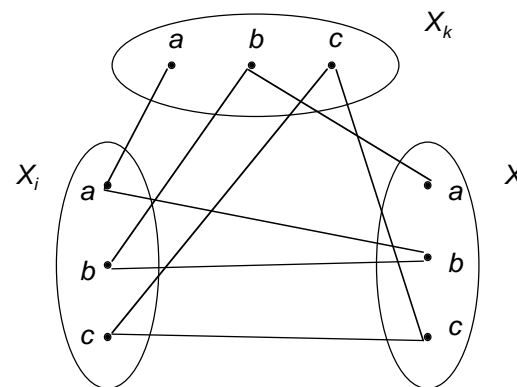
Equivalente a: $D_i := D_i \cap R_i \quad i: 1, \dots, n$

Consistencia de arcos

Una restricción C_{ij} es **arco consistente direccional** (de i a j) ssi para todo valor $a \in D_i$ existe $b \in D_j$ tal que $(a, b) \in R_{ij}$

Una restricción C_{ij} es **arco consistente** si es arco consistente direccional en los dos sentidos

Un problema es **arco consistente** ssi todas sus restricciones lo son.



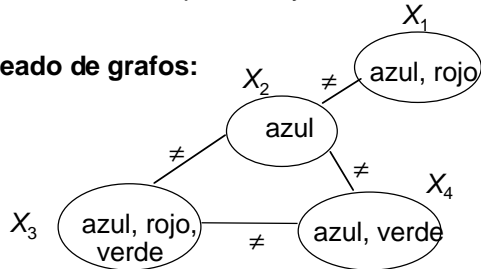
Filtrado por consistencia de arcos

Idea: si para $a \in D_i$ no existe $b \in D_j$ t.q. $(a, b) \in R_{ij}$, se puede eliminar a de D_i porque a no estará en ninguna solución.

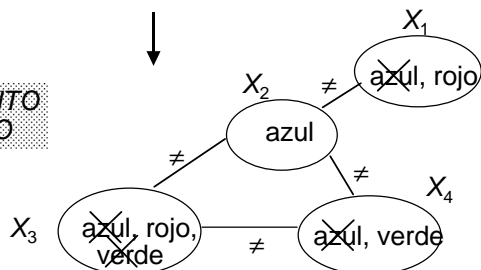
Filtrado de dominios por consistencia de arcos:

- se eliminan valores arco inconsistentes
- se itera hasta que no hay cambios

Coloreado de grafos:



PUNTO FIJO



Función revise(i, j)

Función `revise(i, j)`:

- convierte R_{ij} en arco consistente direccional
- puede eliminar valores del dominio D_i
- se ha de iterar sobre las otras restricciones

funcion `revise (i, j variable): bool;`

`cambio := FALSO;`

para cada $a \in D_i$ **hacer**

si no hay $b \in D_j$ t.q. $(a, b) \in R_{ij}$ **entonces**

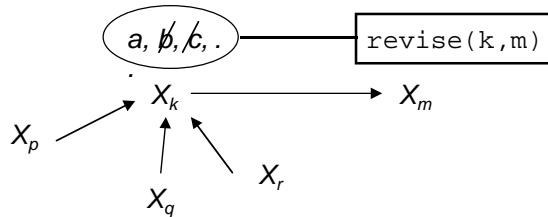
$D_i := D_i - \{a\};$

`cambio := CIERTO;`

retorna `cambio;`

Complejidad: $O(d^2)$

AC-3: Algoritmo consistencia de arcos



`revise(k,m)` borra b y c de D_k

¿Qué arcos hay que visitar? Aquellos que han dejado de ser arco consistentes por el borrado de b y c .

$(k, _)$: no, si era AC, lo sigue siendo tras el borrado
 $(_, k)$: si, puede dejar de ser AC por el borrado

procedimiento AC-3 (G)

```

Q := {(i, j) | (i, j) ∈ arcos(G), i ≠ j}
mientras Q ≠ ∅ hacer
    | selecciona y borra un arco (k,m) de Q
    | si revise(k,m) entonces
    | | Q := Q ∪ {(i, k) | (i, k) ∈ arcos(G), i ≠ k, i ≠ m}
    |

```

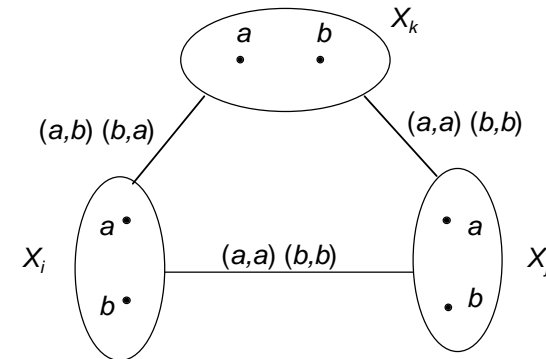
Complejidad: $O(ed^3)$

Consistencia de caminos

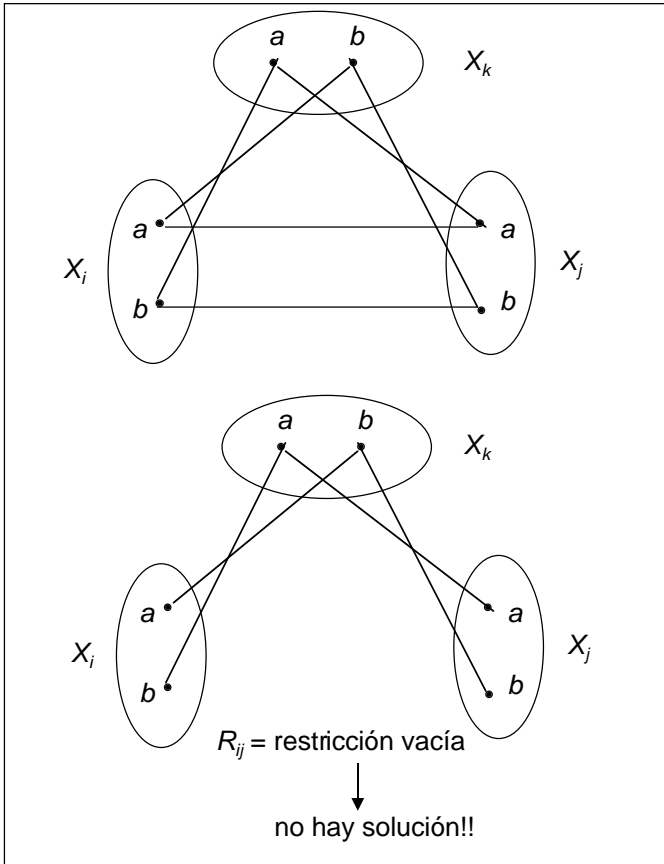
- Un par de valores $((i, a) (j, b))$, tal que $(a, b) ∈ R_{ij}$, es **camino consistente** ssi para todo X_k , $i ≠ k$, $j ≠ k$, existe $c ∈ D_k$ tal que,

$$(a, c) ∈ R_{ik} \quad \text{y} \quad (c, b) ∈ R_{kj}$$

- Un par de variables (X_i, X_j) es **camino consistente** ssi todo par de valores $(a, b) ∈ R_{ij}$ es camino consistente.
- Un problema P es **camino consistente** ssi todo par de variables es camino consistente.



Ejemplo



PC-2

Funcion revise3(i, j, k):

- convierte C_{ij} en camino consistente con x_k
- puede eliminar pares de valores permitidos

funcion revise3 (i, j, k variable): bool;

```

cambio := FALSO;
para cada (a,b) ∈ Rij hacer
  si ∃c ∈ Dk tq. (a,c) ∈ Rik (b,c) ∈ Rjk entonces
    Rij := Rij - {(a,b)};
    cambio := CIERTO;
retorna cambio;
  
```

Complejidad: $O(d^3)$

PC-2: revise3 sobre todos los triangulos posibles

procedimiento PC-2 (X, D, C)

```

Q := {(i, j, k) | 1 ≤ i < j ≤ n, 1 ≤ k ≤ n, k ≠ i, k ≠ j};
mientras Q ≠ ∅ hacer
  selecciona y borra (i, j, k) de Q;
  si revise(i, j, k) entonces
    Q := Q ∪ {(1, i, j), (1, j, i) | 1 ≤ l ≤ n, l ≠ j, l ≠ i}
  
```

Complejidad: $O(n^3 d^5)$

K-Consistencia

K-Consistencia:

- dado un subconjunto de $k-1$ variables asignadas $\{X_1, X_2, \dots, X_{k-1}\}$ consistente;
- para cualquier X_k existe $d \in D_k$ tal que $\{X_1, X_2, \dots, X_k\}$ es consistente.

K-Consistencia: generalización

- 1-consistencia: *consistencia de nodos*
- 2-consistencia: *consistencia de arcos*
- 3-consistencia: *consistencia de caminos*
-

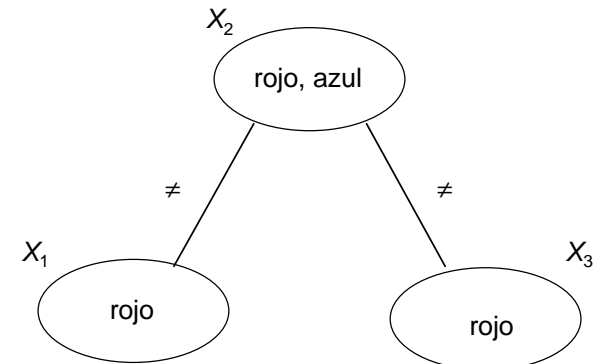
K-Consistencia fuerte:

J-consistente, para $1 \leq J \leq K$

Algoritmos para K-consistencia fuerte:

- Freuder 82, Cooper 89
- Complejidad: $O(\exp K)$

K-consistencia: Ejemplo



- K-consistencia no implica K-consistencia fuerte
- Ejemplo:
 - Es 3-consistente: para cualquier par de dos variables con valores consistentes, existe un valor consistente para la tercera
 - No es 2-consistente: arco $X_2 - X_1$

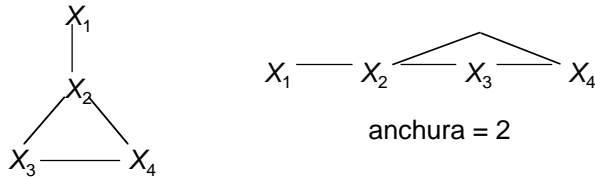
Problemas libres de backtracking

Orden de variables: $\{X_1, X_2, \dots, X_n\}$

Anchura de un nodo: # arcos a nodos anteriores

Anchura de una ordenación: $\max_i \{\text{anchura } X_i\}$

Anchura de un grafo: min anchura ordenaciones



TEOREMA: Dado un orden de variables con anchura K , el problema se puede resolver sin backtracking si el nivel de consistencia fuerte es mayor que K . [Freuder 82]

Algoritmos:

- K -consistencia: $O(\exp k)$.
Añade arcos extras, aumenta la anchura
No añade arcos para anchura 1
Los árboles tienen anchura 1
*Estructura de árbol = libre de backtracking,
tras consistencia de arcos*

Esquema Global

1. Introducción

- Definiciones
- Ejemplos

2. Métodos de Resolución

- Búsqueda
- Inferencia
- Métodos híbridos ←

3. Modelización

- Primal / dual
- Restricciones globales
- Programación con restricciones

4. Restricciones Blandas

- Modelos
- Algoritmos

Métodos de resolución: Algoritmos híbridos

Algoritmo Híbrido = *búsqueda + inferencia*

Búsqueda sistemática + inferencia incompleta:

- Algoritmos de anticipación
- Forward Checking
- Maintaining Arc Consistency

Búsqueda sistemática + inferencia completa:

- Variable Elimination Search

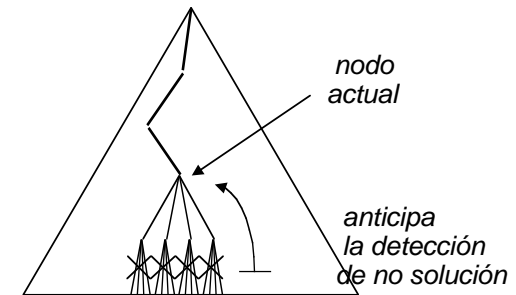
Búsqueda sistemática + inferencia incompleta

Búsqueda: BT (DFS)

ANTICIPACION

Inferencia incompleta:

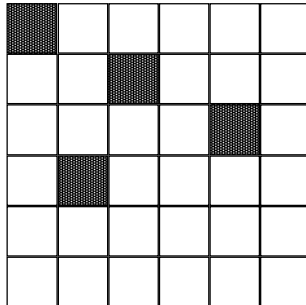
- en cada nodo, consistencia local:
 - se descubren tuplas prohibidas
 - disminuye el tamaño del espacio
 - aumenta la eficiencia de la búsqueda
- si dominio futuro = \emptyset
 - no hay solución en esa rama
 - backtracking
- compromiso:
 - podemos evitar visitar #nodos *exponencial*
 - coste *polinómico* de inferencia en cada nodo



Anticipación: Ejemplo

6-reinas:

¿Hay solución en los descendientes de este nodo?

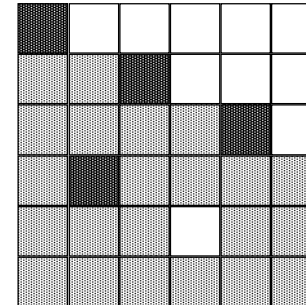


Anticipación: Ejemplo

No hay solución. Última variable sin valores factibles.

Anticipación:

- eliminar valores prohibidos de dominios futuros
- si dominio futuro = \emptyset , no hay solución



Esquema de anticipación

1. Búsqueda:

1. Variable actual X_i $D_i = \{a, b, c\}$
2. Asignación $X_i \leftarrow a \Leftrightarrow D_i = \{a\}$

2. (Consistencia entre pasadas)

3. Tras asignar (\Leftrightarrow reducción de dominio)

- consistencia local
 - parte / todo el problema
 - nivel de consistencia
- si dominio futuro = \emptyset
 - no hay solución en esa rama
 - backtracking
- sino, se continúa (punto 1)

Compromiso coste / beneficio

- coste, beneficio \uparrow con
 - tamaño parte localmente consistente
 - nivel de consistencia
- compromiso óptimo:
 - depende del problema
 - en general, consistencia de arcos

Forward Checking

Forward Checking: [Haralick, Elliot, 80]

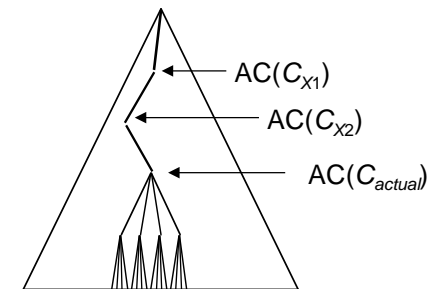
- búsqueda en profundidad
- en cada nodo, arco consistencia sobre las restricciones parcialmente asignadas

Caso binario:

- se eliminan de los dominios futuros los valores incompatibles con el recién asignado
- C_{x_i} : restricciones que involucran a x_i

Proceso:

- en cada nodo $AC(C_{actual})$
- se acumula la poda en la rama actual

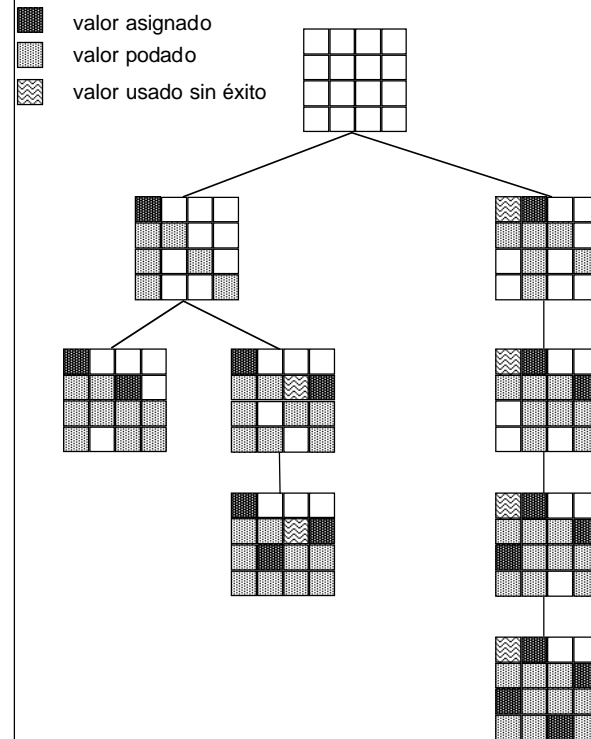


FC: Código

```

funcion FC (i, Past, [Di, ..., Dn]): booleano;
para cada a ∈ Di hacer
  xi := a;
  si i = n entonces retorna CIERTO;
  sino
    C' := {Cij | Cij ∈ C, i < j};
    NewD := AC({xi, ..., xn}, [{a}, Di+1, ..., Dn], C');
    si NewD no contiene ∅ entonces
      si FC(i+1, Past ∪ {xi}, NewD) entonces
        retorna CIERTO;
  retorna FALSO;
  
```

FC: Ejemplo



No hay solución con
 $x_1 = 1$: 4 nodos

9 nodos

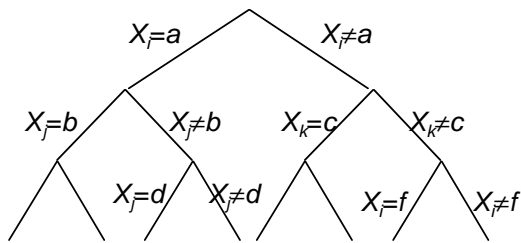
Maintaining arc consistency

MAC: [Sabin, Freuder, 94]

- búsqueda en profundidad
- en cada nodo, arco consistencia sobre todas las restricciones

Arbol de búsqueda:

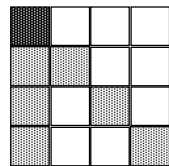
- binario
- en cada nivel
 - una variable x_i
 - dos opciones: $a, \neg a$
- se puede cambiar de variable sin agotar valores
- en cada nodo, AC del subproblema actual



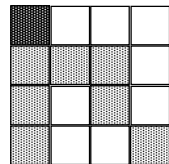
MAC: Código

```
funcion MAC (i, [D1, ..., Dn]): booleano;  
  para j:=i+1, ..., n hacer D'j:=Dj;  
  para cada a∈Di hacer  
    /* xi:=a */ /  
    D'i := {a};  
    si i = n entonces retorna CIERTO  
    sino  
      NewD := AC(X, [D1, ..., Di-1, D'i, ..., D'n], C);  
      si NewD no contiene ∅ entonces  
        si MAC(i+1, NewD) entonces  
          retorna CIERTO;  
        /* xi:=a */ /  
      Di := Di - {a};  
      D'i := Di;  
      NewD := AC(X, [D1, ..., Di-1, D'i, ..., D'n], C);  
      si NewD contiene ∅ exit bucle  
      sino  
        para j:=i+1, ..., n hacer D'j := NewD[j];  
  retorna FALSO;
```

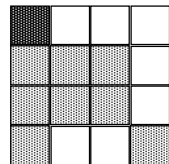
MAC: AC entre Futuras



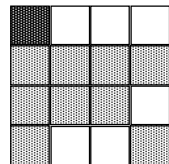
AC (2,3)



AC (3,4)

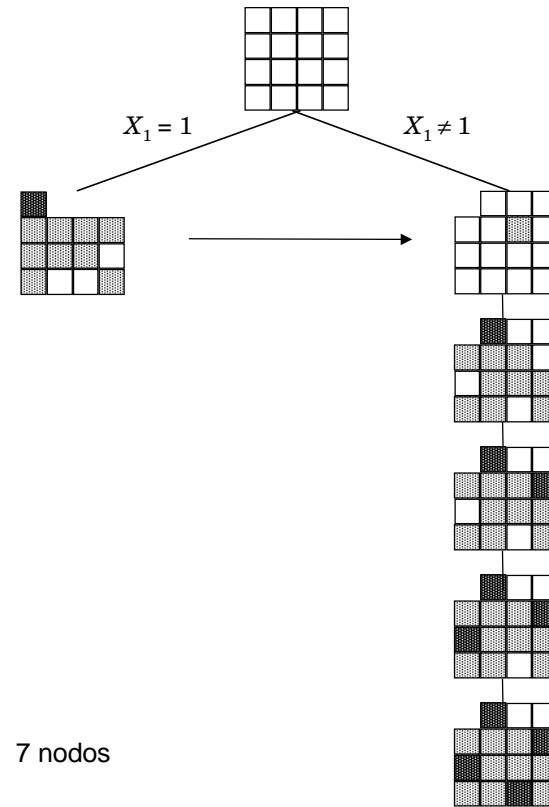


AC (2,3)



No hay solución con $x_1 = 1$: 1 nodo

MAC: Ejemplo



7 nodos

Búsqueda sistemática + inferencia completa

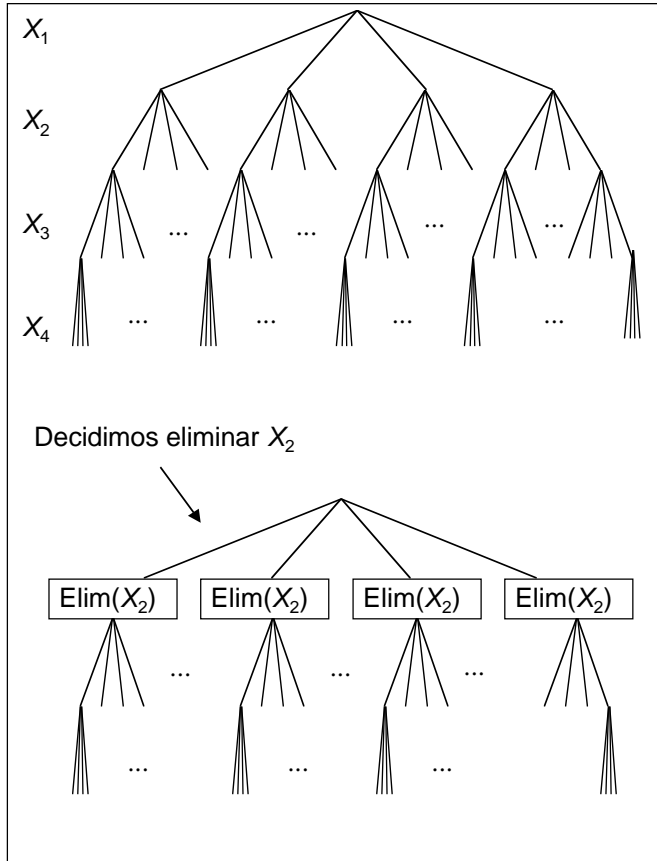
Búsqueda sistemática + inferencia completa:

- DECISIÓN: a cada nueva variable x aplicamos
 - búsqueda o
 - eliminación
- si búsqueda,
 - árbol, backtracking
 - tras asignación x , nuevo grafo
- si eliminación
 - generamos un nuevo problema

Compromiso:

- coste de eliminación: $\exp(w^*)$ anchura grafo
- coste búsqueda: $\exp(\#\text{variables búsqueda})$

Variable Elimination Search



Variable Elimination Search

VES: orden estático de variables

Variable actual: x

Búsqueda:

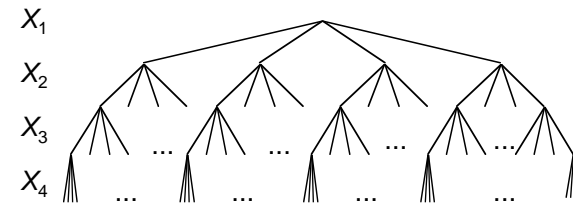
- tras asignar x , anticipación
- x queda fijada en esa rama
- modifica la topología del grafo

Eliminación:

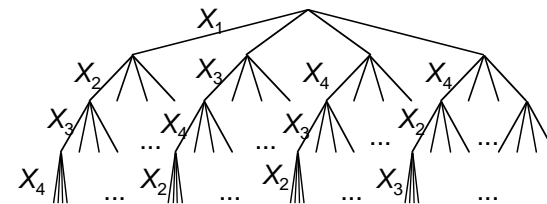
- coste: d^{w^*} w^* anchura de x
- si w^* pequeña, eliminación es competitiva

Ordenación de variables

Ordenación estática de variables: cada nivel del árbol de búsqueda se asocia con una variable



- No es necesaria para que el árbol de búsqueda sea exhaustivo.
- Es necesario que cada nodo se asocie con una variable para la generación de sucesores.



- Diferentes variables en el mismo nivel:
ordenación dinámica de variables

Heurística: selección de variable

En nodo q ¿qué variable asignar a continuación?

1. Hay solución en $\text{sucesores}(q)$:
cualquier variable es adecuada
2. No hay solución en $\text{sucesores}(q)$:
*asignar la variable que antes
descubre que no hay solución*

En general, ¿qué situación es más frecuente?

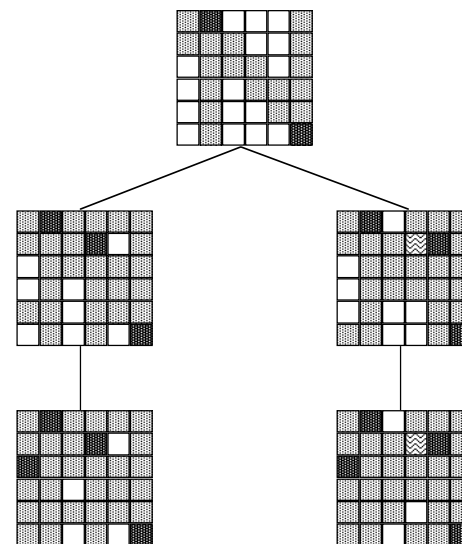
- Salvo problemas triviales, situación 2
- Mayor esfuerzo del algoritmo: salir de subproblemas sin solución

Suponemos situación 2: asignar primero aquella variable que, aparentemente, antes nos conduce a un fallo (principio *fail-first*)



Heurística dominios mínimos: *asignar primero la variable con menor número de valores factibles*

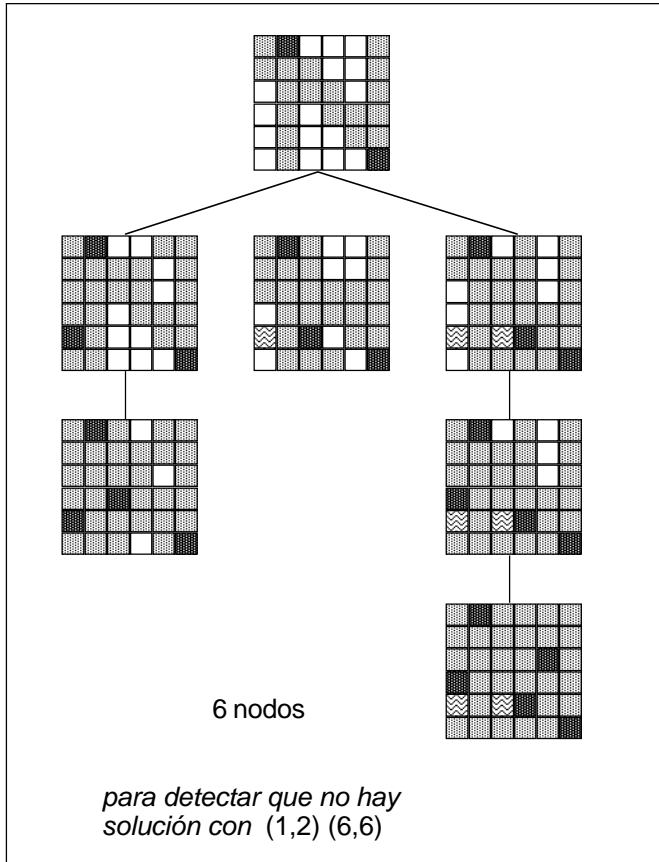
Ejemplo: 6 reinas



4 nodos

para detectar que no hay solución con (1,2) (6,6)

Ejemplo



Heurística: selección de valor

En nodo q ¿qué valor asignar a continuación?

1. Hay solución en $\text{sucesores}(q)$:
un valor que mantenga la resolubilidad del nodo sucesor
2. No hay solución en $\text{sucesores}(q)$:
cualquier valor es adecuado

Suponemos situación 1: asignar primero aquél valor que, aparentemente, antes nos conduce al éxito (principio *success-first*)



Heurística anticipación valores: *asignar primero el valor que es consistente con mayor número de valores factibles del resto de variables*

Coste heurísticas

Compromiso coste / beneficio:

- coste: suma de costes de cálculo en cada nodo
- beneficio en todo el árbol
- sale a cuenta si $\text{coste} < \text{beneficio}$

Dominios mínimos y anticipación:

- Dominios mínimos: calcula #valores factibles
- Alg. anticipación: calculan los valores factibles
- obtiene dominios mínimos *sin coste* adicional

Evaluación de las heurísticas:

- Evaluación empírica
- Esfuerzo computacional:
 - constraint checks
 - tiempo CPU

Resultados empíricos, 20 reinas, 1ª solución

BT	25.428.842 cc
FC	2.398.022 cc
FC+dom min	4.144 cc

