

Problemas de Satisfacción de Restricciones (CSP)

“Constraint Satisfaction, a simple but powerful idea” R. Dechter

1) Modelado de Problemas: Conceptos y Modelos.

- Problemas de Satisfacción de Restricciones (CSP)
- Modelado de CSP. Tipología de Restricciones.
- Ejemplos. Entornos CSP.
- Optimización. Problemas de optimización sujetos a restricciones.

2) Resolución de CSP:

- Conceptos básicos. Operativa.
- Técnicas Inferenciales: Nueva información. Niveles de consistencia.
- Técnicas de Búsqueda. Heurísticas. Técnicas Híbridas: Métodos Looking Ahead.

3) CSP Flexibles (valuados).

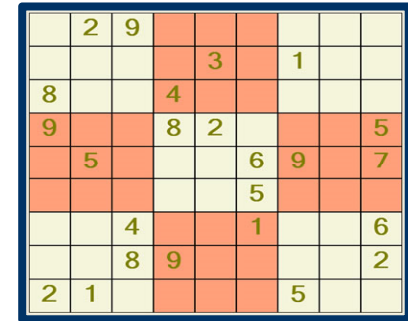
Bibliografía

- *Inteligencia Artificial. Un enfoque moderno* (Cap. 5). S Russell, P. Norvig. Prentice Hall (2010).
- *Inteligencia Artificial. Técnicas, métodos y aplicaciones* (cap. 10). Mc Graw Hill (2008).
- 'On-Line Guide To Constraint Programming' R. Barták. <http://kti.mff.cuni.cz/~bartak/constraints/index.html>
- 'Monografía: Problemas de Satisfacción de Restricciones'. Inteligencia Artificial, vol.7, No. 20
<http://journal.iberamia.org/>
- *Demos en Web:* <http://aispace.org/constraint/> , <http://www.constraintsolving.com/>

1.1 Problemas de Satisfacción de Restricciones

Muchos problemas pueden ser expresados mediante:

- ✓ Un conjunto de variables,
- ✓ Un dominio de interpretación (valores) para las variables.
- ✓ Un conjunto de restricciones entre las variables.



	2	9						
				3		1		
8			4					
9			8	2				5
	5				6	9		7
					5			
		4			1			6
		8	9					2
2	1					5		

tal que la solución al problema es una asignación válida de valores a las variables.

- Problemas de Empaquetamiento,
- Problemas de Rutas, transporte, logística,
- Problemas de Scheduling, Compartición y Asignación de Recursos,
- Problemas de Razonamiento Temporal, Sistemas de Documentación,
- Gestión de Redes de Comunicación,
- Diseño, Planificación, Control, etc.

Ejemplos 1

Criptografía

- Variables: s,e,n,d,m,o,r,y
- Dominios: s,e,n,d,m,o,r,y ∈ {0,...,9}
- Restricciones

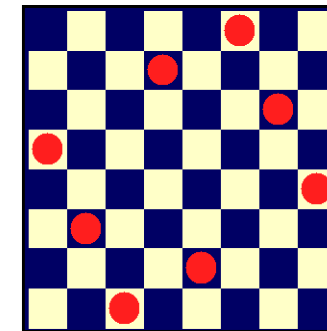
$$\begin{array}{r} \text{ s e n d} \\ + \text{ m o r e} \\ \hline \text{ m o n e y} \end{array}$$

Restricción n-aria

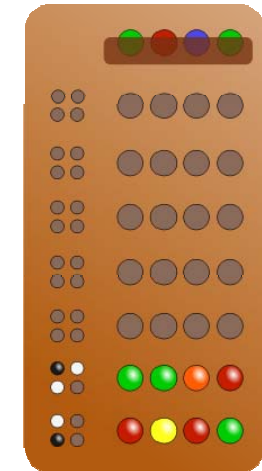
$$10^3(s+m) + 10^2(e+o) + 10(n+r) + d + e = 10^4m + 10^3o + 10^2n + 10e + y$$

Objetivo:

Solución (asignación consistente)



8 Reinas



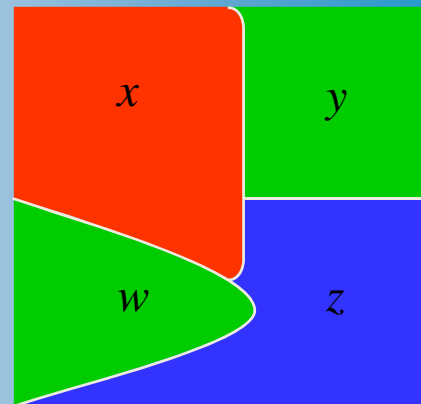
Mastermind

Coloreado de Mapas (Mapa de frecuencias)

- Variables: x,y,z,w
- Dominios: x,y,z,w : {r,v,a}
- Restricciones:

Restricciones Binarias

$$x \neq y, y \neq z, z \neq x, \dots$$



Sudoku



Ejemplo 2

Lectores y Periódicos

Los lectores quieren leer los periódicos en su orden y durante un tiempo determinado.

Todos deben acabar antes de sus dead-lines.



	Ready-Time	P1	P2	P3	Due-Time
L1	0	5'	10'	2'	30'
L2	0	2'	6'	5'	20'
L3	0	10'	15'	15'	60'
L4	0	3'	5'	5'	15'

Objetivo: Obtener la **asignación óptima** (scheduling) de lectura.

Problemas de Horarios,
Asignación de Recursos, Scheduling, etc.

Ejemplo 3

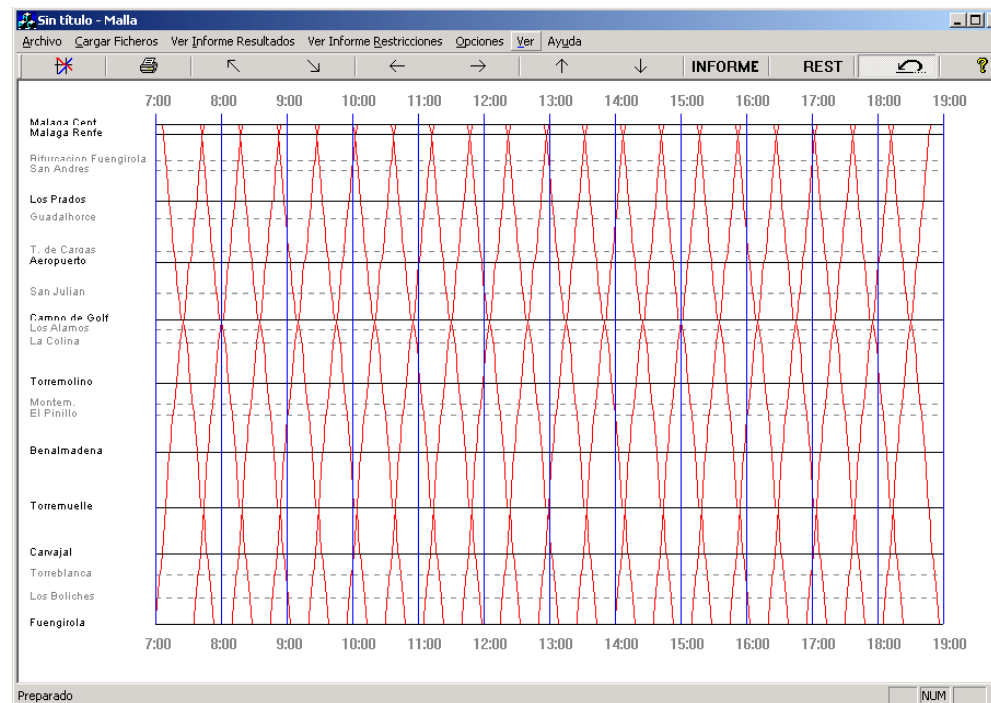
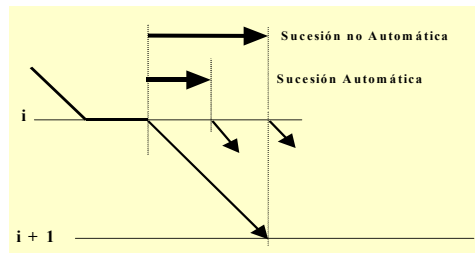
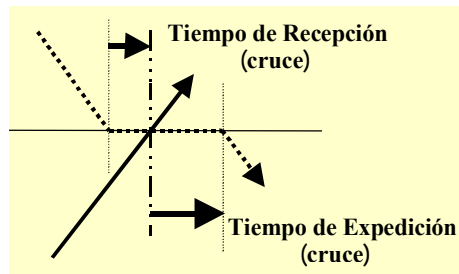
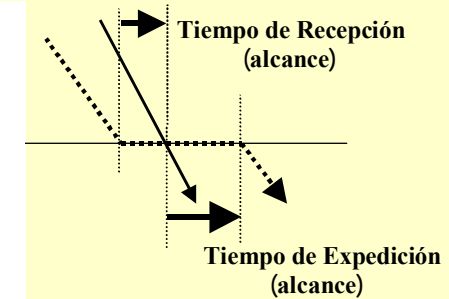
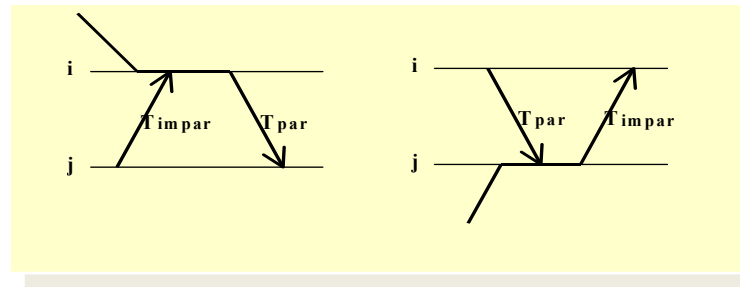
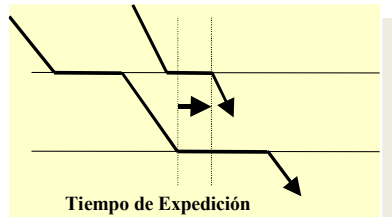
"Juan va de su casa al trabajo en coche (30-40 minutos) o en tren (al menos una hora). Luis va en coche (20-30 minutos) o en metro (40-50 minutos)."

"Hoy Juan parte de casa entre las 8:10 y las 8:20, y Luis llega al trabajo entre las 9:00 y las 9:10. Además, sabemos que Juan llegó al trabajo entre 10 y 20 minutos después de que Luis saliera de casa"

Cuestiones:

- ¿Esta información es consistente?
- ¿Es posible que Juan haya usado el tren y Luis haya usado el Metro?
- ¿Cuáles son los posibles tiempos en los que Luis pudo haber salido de casa?
- ¿Cuánto dura el viaje de Juan en tren?
- etc.

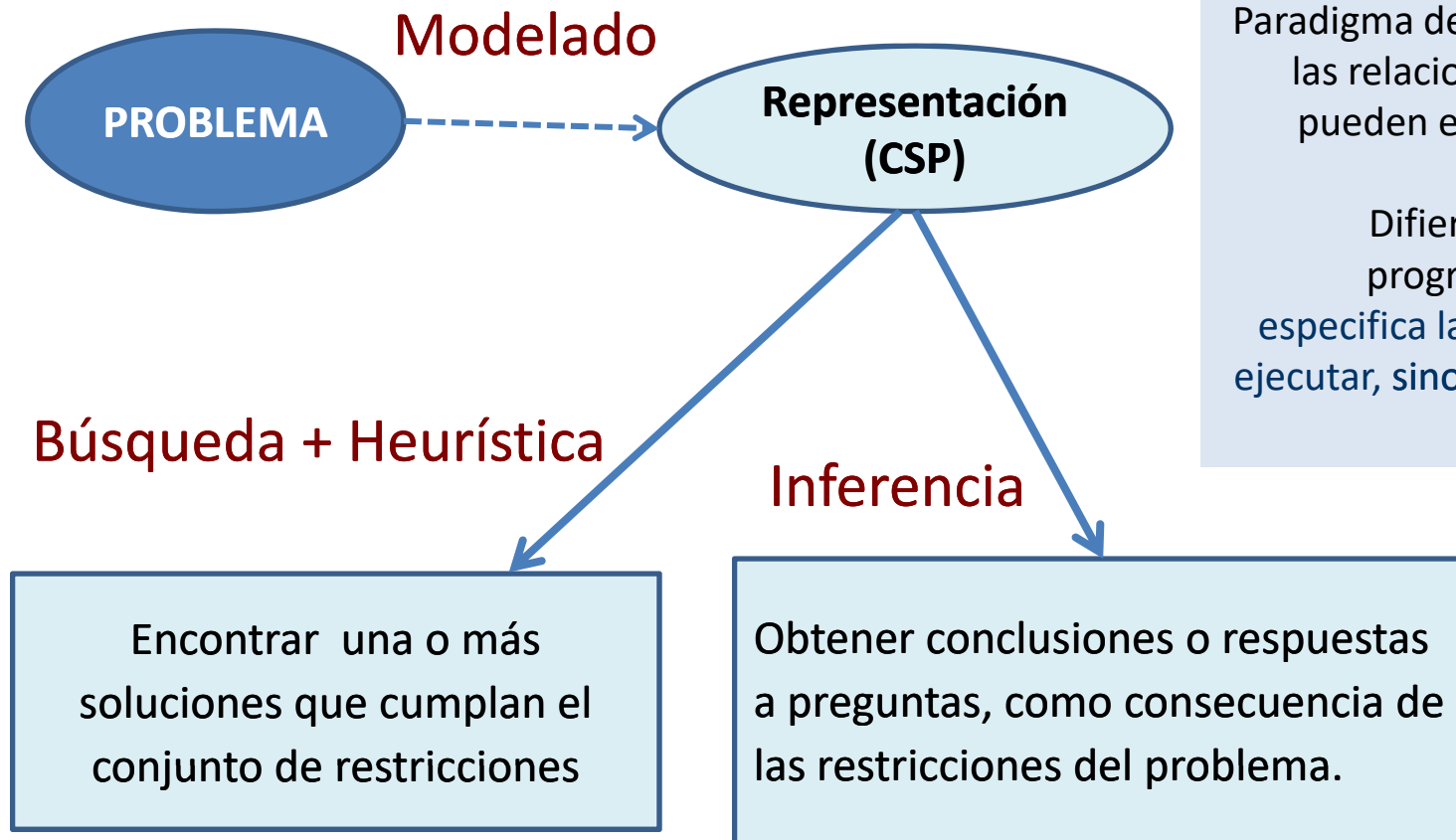
Ejemplo 4. Resolución de problemas reales (logística)



Más demos, tutorials and solvers en:

<http://www.constraintsolving.com/>

¿Qué es lo que queremos?



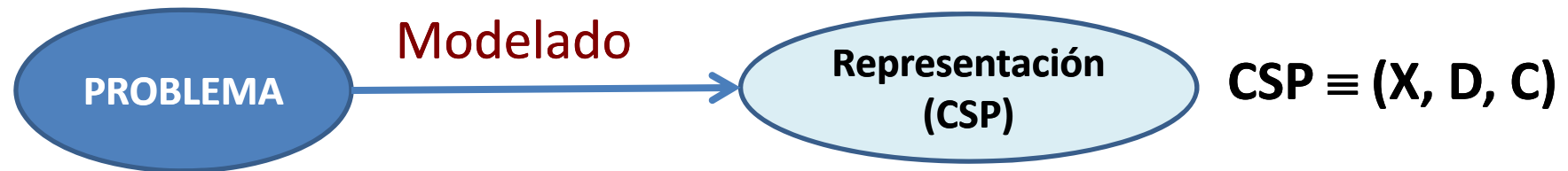
Constraint Solving

Paradigma de programación donde las relaciones entre variables se pueden establecer en forma de restricciones.

Difiere de otras técnicas de programación en que **no se especifica la secuencia de pasos a ejecutar, sino las propiedades de la solución a obtener.**



1.2.- Definición de un CSP.



Un Problema de Satisfacción de Restricciones (CSP) se puede representar como:

- 1) Conjunto de **Variables**: $X = \{x_1, x_2, \dots, x_n\}$
- 2) **Dominios** de Interpretación para las variables: $D = \{D_1, \dots, D_n\} / x_i \in D_i$
- 3) **Conjunto de Restricciones** entre las variables: $C = \{c_1, c_2, \dots, c_m\}$

Las restricciones expresan las combinaciones válidas de valores simultáneos entre las variables.

Una **solución** del CSP es una **instanciación consistente** de todas las variables en sus dominios, tal que todas las restricciones del CSP se cumplan.

Tipología de los CSP's

$$\text{CSP} \equiv (\mathbf{X} = \{x_1, x_2, \dots, x_n\}, \mathbf{D} = \{D_1, \dots, D_n\}, \mathbf{C} = \{c_1, c_2, \dots, c_m\})$$

a) Tipología de las **Dominios** $\langle D_1, \dots, D_n \rangle$:

Dominios Discretos: Enteros (1..10) o Simbólicos (a,b,c)

Dominios Continuos

Los dominios continuos se suelen discretizar con una granularidad dada para evitar un número infinito de valores

b) Tipología de las **Restricciones** (afecta a la expresividad)

1. Intensionales / Extensionales. →
2. Aridad: nº variables involucradas.
3. Cualitativas / Métricas.
4. Disyuntivas o no disyuntivas.
5. Otros tipos.

Extensionales: conjunto de todas las **tuplas permitidas**.

$$\text{CSP} \equiv [\{x_1, x_2\}, \{ [1,3], [1, 2, 5] \}, \{ \mathbf{(3,1)}, \mathbf{(3,2)} \}]$$

Intensionales: expresión lógico-matemática.

$$\text{CSP} \equiv [\{x_1, x_2\}, \{ [1,3], [1, 2, 5] \}, \{ \mathbf{(x_1 > x_2)} \}]$$

Son equivalentes en dominios discretos y finitos

Aridad de las Restricciones (asumiendo restricciones matemáticas, de tipo \leq)

- **Restricción Unaria:** Restricción para una variable. Ejemplo: $a \leq x_i \leq b$, $a, b \in \mathbb{Z}$

- **Restricción Binaria:** Restricción entre dos variables x_i, x_j .

Ejemplo: $a \leq p_i x_i + p_j x_j \leq b$, $a, b, p_i, p_j \in \mathbb{Z}$

- **Restricción no-binaria (n-aria):** Expresa una restricción entre n variables

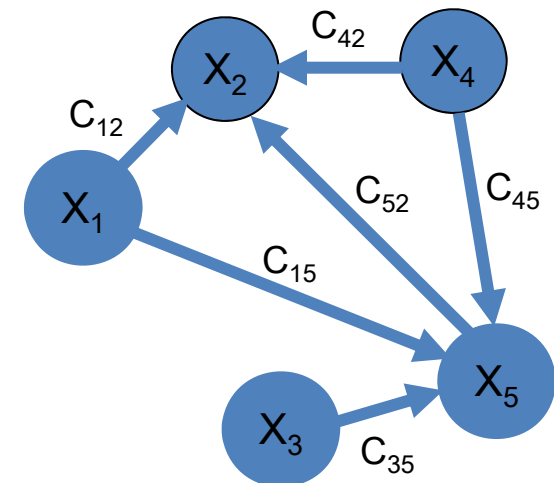
Ejemplo: $a \leq p_1 x_1 + p_2 x_2 + p_3 x_3 + \dots + p_n x_n \leq b$, $a, b, p_k \in \mathbb{Z}$, $k:1,n$

En general: Relación lineal sobre $X=\{x_1, \dots, x_k\}$: $\sum_{i=1}^n p_i x_i \{<, \leq, =, \neq, >, \geq\} b$

CSP binarios y discretos $\{(x_i, c_{ij}, x_j)\}$:

- Problemas bien conocidos (grafo de restricciones binarias).
 - Nodos representan las Variables,
 - Arcos las relaciones binarias entre las variables.
- Requerido para operativa en grafos de restricciones (\oplus , \otimes).
- Todo CSP no-binario puede ser convertido en un CSP binario, *típicamente* mediante la introducción de nuevas variables.

Ver: <http://ktiml.mff.cuni.cz/~bartak/constraints/binary.html>



1. *Intensionales / extensionales*
2. *Aridad: nº variables involucradas.*
3. *Cualitativas / Métricas.*
4. *Disyuntivas o no disyuntivas*
5. *Otros tipos.*

Restricciones Cualitativas:

Establecen la posición relativa (orden) entre las variables.

Típico entre *dominios no escalares (colores, razas, tamaños, lugares, tallas, etc.)*

Ejemplos: $x_i \{<, =, >\} x_j$, $x_i \{<, >\} x_j$

Restricciones Métricas:

Establecen una métrica en las restricciones cualitativas.

Implica una métrica en el dominio (*dominios enteros, reales, etc.*)

Ejemplos: $x_i < x_j + 7$, $(x_i < x_j + 20) \wedge (x_i > x_j + 10)$

Restricciones Disyuntivas / No-disyuntivas

(dominio de soluciones convexo o no convexo)

1. Intensionales / extensionales
2. Aridad: nº variables involucradas.
3. Cualitativas / Métricas.
4. **Disyuntivas o no disyuntivas**
5. Otros tipos.

Restricciones Disyuntivas:

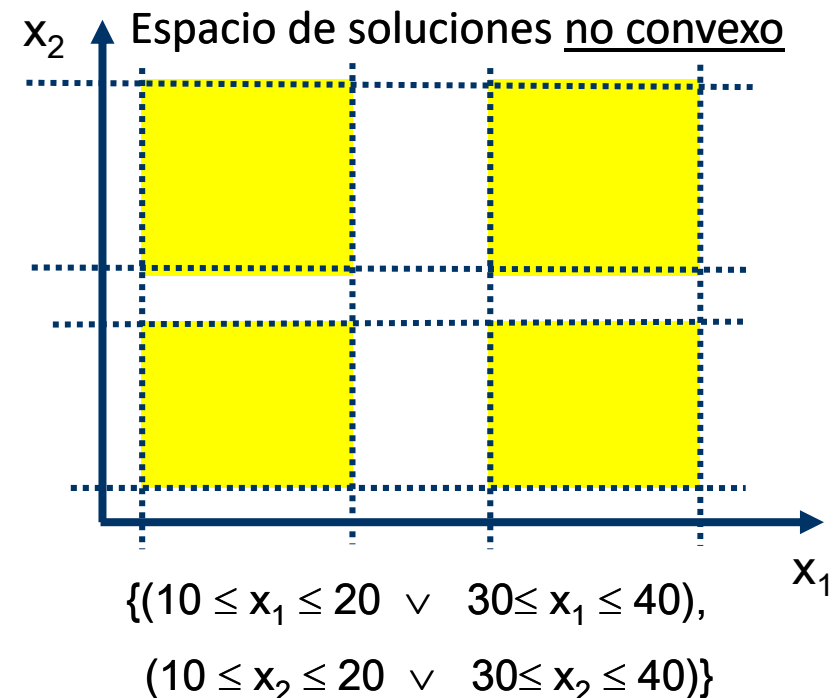
- Disyunción de restricciones entre las variables.
- El espacio de soluciones no es convexo.
- Problema Completo (*Exponencial*)

Ejemplo: $\{ (a_1 \leq x_i - x_j \leq b_1) \vee (a_2 \leq x_i - x_j \leq b_2) \vee \dots \vee (a_p \leq x_i - x_j \leq b_p) \}$

Restricciones no-disyuntivas:

- Problema Simple (*Polinomial*)

Ejemplo: $(a \leq x_i - x_j \leq b)$



Otros tipos de restricciones

1. *Intensionales / extensionales*
2. *Aridad: nº variables involucradas.*
3. *Cualitativas / Métricas.*
4. *Disyuntivas o no disyuntivas*
5. *Otros tipos.*

- **Condicionales:** If <restricción-1> then <restricción-2>. ¡Equivalente a expresión lógica!
- **Fuertes (hard):** son restricciones cuya satisfactibilidad es imprescindible (son obligatorias).
- **Débiles (soft):** son restricciones cuya satisfactibilidad no es imprescindible (son preferencias recomendables).
- **Difusas (fuzzy):** son restricciones definidas con relaciones difusas o sobre valores difusos.
- **Ponderadas:** las restricciones tienen asociadas un peso o ponderación (a maximizar o minimizar).
- **Temporales:** las variables están asociadas a puntos de tiempo, intervalos o duraciones. Restricciones entre primitivas temporales.

Ejemplos de especificación de CSP

s e n d
+ m o r e
—
m o n e y

CSP n-ario, discreto, disyuntivo

Especificación
CSP

- Variables: s, e, n, d, m, o, r, y
- Dominios: s, e, n, d, m, o, r, y: {0,...,9}
- Restricciones: $\neq (s, e, n, d, m, o, r, y),$

$$10^3(s+m) + 10^2(e+o) + 10(n+r) + d + e = 10^4m + 10^3o + 10^2n + 10e + y$$

Como alternativas (más eficientes):

$y' = d + e;$	$y = \text{mod}(y', 10);$
$e' = n + r + \text{int}(y'/10);$	$e = \text{mod}(e', 10);$
$n' = e + o + \text{int}(e'/10);$	$n = \text{mod}(n', 10);$
$o' = s + m + \text{int}(n'/10);$	$o = \text{mod}(o', 10);$
	$m = \text{int}(o'/10);$

O incluso:

$$\begin{aligned}d + e &= y + 10 * C1; \\C1 + n + r &= e + 10 * C2; \\C2 + o + e &= n + 10 * C3; \\C3 + s + m &= o + 10 * m;\end{aligned}$$

"Juan va de su casa al trabajo en coche (30-40 minutos) o en tren (al menos una hora). Luis va en coche (20-30 minutos) o en metro (40-50 minutos).

Hoy Juan parte de casa entre las 8:10 y las 8:20 y Luis llega al trabajo entre las 9:00 y las 9:10.

Además, sabemos que Juan llegó al trabajo entre 10 y 20 minutos después de que Luis saliera de casa"

Especificación CSP

- **Variables:** T1 / T2: Tiempo en que Juan sale de casa / Llega al trabajo
T3 / T4: Tiempo en que Luis sale de casa / Llega al trabajo.

- **Dominio:** {8:00,...,10:00} (* granuralidad en minutos *)

- **Restricciones:**

$$8:10 \leq T1 \leq 8:20$$

$$30 \leq T2-T1 \leq 40 \quad \vee \quad 60 \leq T2-T1$$

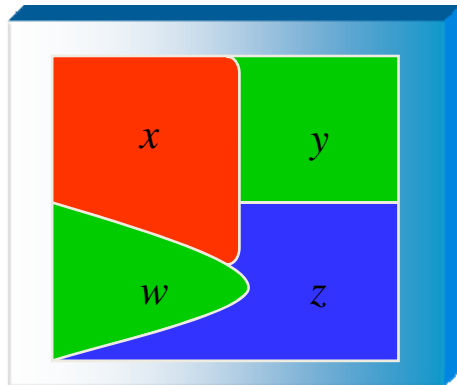
$$10 \leq T2-T3 \leq 20$$

$$9:00 \leq T4 \leq 9:10$$

$$20 \leq T4-T3 \leq 30 \quad \vee \quad 40 \leq T4-T3 \leq 50$$

CSP binario,
continuo/discreto,
disyuntivo

Coloreado de Mapas



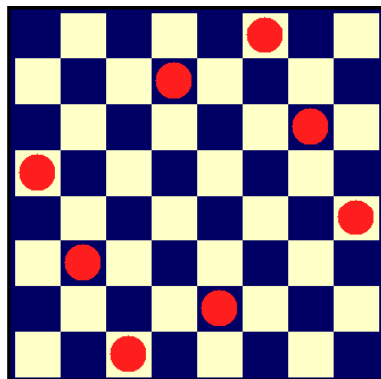
- Variables: x, y, z, w
- Dominios: $x, y, z, w : \{r, v, a\}$
- Restricciones (binarias):

$$\begin{aligned} x &\neq y, & x &\neq z, \\ x &\neq w, & y &\neq z, \\ w &\neq z \end{aligned}$$

CSP binario, cualitativo, disyuntivo y discreto

La restricción \neq implica una relación disyuntiva ($<$, $>$).

Las 8 Reinas



Con $n = 8...$

Variables $\{x_i\}$, indicando la posición en la fila i .

Dominio = $\{1, 2, 3 \dots, n\}$

Restricciones:

$\forall x_i, x_j, i \neq j:$

$x_i \neq x_j$	No en la misma columna
$x_i - x_j \neq i - j$	No en la misma diagonal SE
$x_j - x_i \neq i - j$	No en la misma diagonal SO

CSP binario, disyuntivo, métrico y discreto

Sudoku:

rellenar las celdas con números del 1..9,
tal que no se repitan en columnas, filas o submatrices.

	2	9						
				3		1		
8			4					
9			8	2				5
	5				6	9		7
					5			
		4			1			6
		8	9					2
2	1					5		

Formulación:

- variables: 9x9 celdas
- dominios: {1..9}
- restricciones: propiedades que se deben satisfacer:
 - Todas las variables de una submatriz: distintas
 - Todas las variables de una fila: distintas
 - Todas las variables de una columna: distintas

CSP no binario, con restricciones disyuntivas

Ejercicio

Juan, Pepe y Paco nacieron y viven en ciudades diferentes (Málaga, Madrid y Valencia).

Además, ninguno vive en la ciudad donde nació.

Juan es más alto que el que vive en Madrid. Paco es cuñado del que vive en Valencia.

El que vive en Madrid y el que nació en Málaga tienen nombres que comienzan por distinta letra.

El que nació en Valencia y el que vive ahora en Málaga tienen nombres que comienzan por la misma letra.

¿Donde nació y vive cada uno?

?

1.3. Entornos CSP: Entornos para modelar y resolver CSP.

Proporcionan:

- ✓ Edición de modelos CSP, CSOP.
- ✓ Resolución: Satisfacción & Optimización

- **MiniZinc** (<http://www.minizinc.org/>) **libre, editor 'FlatZinc' genérico para múltiples resolvers**
- Google OR-Tools (<https://developers.google.com/optimization/>) **libre**
CP-SAT Solver (https://developers.google.com/optimization/cp/cp_solver)
- IBM-ILOG: CP-optimizer (<https://www.ibm.com/analytics/cplex-cp-optimizer>). **comercial**
- Choco (en Java) (<http://choco-solver.org/>) **libre**
- GAMS (<http://www.gams.com/>) **versión libre limitada**
- ECLiPSe (<http://eclipseclp.org/>), **libre**
- Gusek interfaz para el resolutor GLPK (<http://gusek.sourceforge.net/gusek.html>)
- Legin (<http://www.stern.nyu.edu/om/software/legin/>), LINGO, LINDO (<http://www.lindo.com/>)
- Repositorios: <http://www.constraintsolving.com>

Entorno CPLEX / CP-Optimizer

The screenshot shows the OPL Studio interface with a file named 'queens.mod'. The code defines a domain for queens and a solve block with constraints for the N-queens problem. The solution window shows a valid solution for 8 queens.

```

var Domain queens[Domain];
solve {
  forall(ordered i,j in Domain) {
    queens[i] <> queens[j] ;
    queens[i] + i <> queens[j] + j ;
    queens[i] - i <> queens[j] - j ;
  };
};

```

N-queens problem

Solution [1]

```

queens[1] = 1
queens[2] = 5
queens[3] = 8
queens[4] = 6
queens[5] = 3
queens[6] = 7
queens[7] = 2
queens[8] = 4

```

Console Solutions Optimization Log Solver CPLEX

Next solution? Ln 11, Col 1 Waiting

Entorno: MiniZinc

- Entorno libre
- Editor > FlatZinc
- Múltiples resolvers
- Windows, Linux, OS Mac

The screenshot shows the MiniZinc IDE with a file named 'nreinas.mzn'. The code defines the N-Reinas problem using MiniZinc syntax. The output window shows the compilation and execution of the model, resulting in a solution for 8 queens.

```

1 % N-Reinas
2 int: n;
3 array [1..n] of var 1..n: q; % queen is column i is in row q[i]
4 include "alldifferent.mzn";
5 % Restricciones
6
7 % No en misma columna
8 constraint forall (i,j in 1..n where i!=j) (q[i] != q[j]);
9 % No en misma diagonal SE
10 constraint forall (i,j in 1..n where i!=j) ((q[i] - q[j]) != (i - j));
11
12 % No en misma diagonal SO
13 constraint forall (i,j in 1..n where i!=j) ((q[j] - q[i]) != (i - j));
14
15 % search
16 solve satisfy;
17
18 output [(show(q[i]) ++ " ") | i in 1..n];
19

```

Output

```

Compiling nreinas.mzn, additional arguments n=8;
Running nreinas.mzn
4 2 7 3 6 8 5 1
-----
Finished in 349msec
Compiling nreinas.mzn, additional arguments n=8;
Running nreinas.mzn
4 2 7 3 6 8 5 1
-----
5 2 4 7 3 8 6 1
-----
3 5 2 8 6 4 7 1
-----
3 6 4 2 8 5 7 1
-----
1 7 4 6 8 2 5 3

```

Line: 2, Col: 1 387msec

Especificación Modelo CSP en MiniZinc

% Esquema de un Modelo CSP en MiniZinc

```
include "alldifferent.mzn";      % Inclusión código restricciones especiales
include "datos1.dzn";           % Inclusión fichero datos
```

```
int a;                          % Parámetros. Valor por asignación, fichero externo o interfaz.
var int: b;                     % Variables tipadas float|int|bool|string/enum
var 0..100: c;
```

```
constraint 250*b + 200*c <= 10*a; % Restricciones (aritmético-lógicas)
constraint .....
```

```
solve maximize 400*b + 450*c;    % Objetivo resolutor (solve satisfy por defecto)
```

% Formato de salida (asignaciones a las variables)

```
Output ["Resultado b= ", show(b), "\n",
        "Resultado c = ", show(c)];
```

Interfaz MiniZinc

The screenshot displays the MiniZinc IDE interface. At the top, the window title is "send-more-money.mzn — Untitled Project". The menu bar includes "File", "Edit", "MiniZinc", "View", and "Help". The toolbar contains icons for "New model", "Open", "Save", "Copy", "Cut", "Paste", "Undo", "Redo", "Shift left", "Shift right", "Run" (highlighted with a red box), and "Solver configuration: Geode 6.1.0 [built-in]". Below the toolbar, the editor shows the MiniZinc model code for "send-more-money.mzn".

Ejecución

Edición del Modelo

```
1 include "alldifferent.mzn";
2 var 1..9: S;
3 var 0..9: E;
4 var 0..9: N;
5 var 0..9: D;
6 var 1..9: M;
7 var 0..9: O;
8 var 0..9: R;
9 var 0..9: Y;
10 constraint 1000 * S + 100 * E + 10 * N + D
11 + 1000 * M + 100 * O + 10 * R + E
12 = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;
13 constraint alldifferent([S,E,N,D,M,O,R,Y]);
14 solve satisfy;
15 output [ " ", show(S), show(E), show(N), show(D), "\n",
16 "+ ", show(M), show(O), show(R), show(E), "\n",
17 "= ", show(M), show(O), show(N), show(E), show(Y), "\n"];
```

Output

```
Compiling send-more-money.mzn
Running send-more-money.mzn
9567
+ 1085
= 10652
-----
Finished in 256msec
```

Salida Ejecución

Line: 2, Col: 1

256msec

Notas:

- Todas las líneas acaban con ;
- %: Comentario (hasta final línea)
- Dependiente de mayúsculas/minúsculas (Identificadores y nombres de variables)
- Identificadores de variables empiezan por letra y pueden contener números, letras y _.
- Debe evitarse en lo posible en uso de variables reales (dominios continuos)
- La configuración de las ventanas es modificable.

Restricciones Especiales en MiniZinc

Diversos ejemplos en boletín y en documentación!

OR:	<code>constraint s1 + d1 <= s2 ∨ s2 + d2 <= s1;</code>
AND:	<code>constraint s1 + d1 <= s2 ∧ s2 + d2 <= s1;</code>
Condicional:	<code>constraint if a > b then c > 10 else c < 10 endif; constraint if b > c then d > 10 endif;</code> <code>constraint if (s1 + d1 <= s2 ∧ s2 + d2 <= s1)</code> <code> then (s1 + d1 >= s3 ∨ s2 + d2 >= s4) else c < 10 endif;</code>
Implicación:	<code>constraint s1 + d1 <= s2 -> s2 + d2 <= s1; % Si</code> <code>constraint s1 + d1 <= s2 <- s2 + d2 <= s1; % Solo si</code> <code>constraint s1 + d1 <= s2 <-> s2 + d2 <= s1; % Si y solo si</code>
Negación:	<code>constraint not (s1 + d1 <= s2 ∧ s2 + d2 <= s1);</code>
forall:	<code>constraint forall (i,j in 1..3 where i < j) (a[i] != a[j]); % a[1] != a[2] ∧ a[1] != a[3] ∧ a[2] != a[3];</code>
exists:	<code>constraint exists (i,j in 1..3 where i < j) (a[i] != a[j]); % a[1] != a[2] ∨ a[1] != a[3] ∨ a[2] != a[3]</code>
alldifferent:	<code>include "alldifferent.mzn"; % requiere incluir restricción global</code> <code>constraint alldifferent ([S,E,N,D,M,O,R,Y]);</code> <code>constraint alldifferent (Q); % Los valores de las celdas del vector Q son todos diferentes</code> <code>constraint alldifferent (j in 1..N) (Q[j]); % Solo los primeros N valores son diferentes</code>

Ejemplo-1:

Un niño entra en un supermercado y compra cuatro elementos. El cajero cobra 8 euros, el niño paga y está a punto de irse cuando el cajero dice al niño: "Espera, que multipliqué los cuatro elementos, en lugar de sumarlos. Voy a intentarlo de nuevo"

Con el cambio, el precio todavía es de 8 euros.

¿Cuáles fueron los precios de los cuatro elementos?

%===== Variables

```
var 1 .. 100: e1;      % Se indica en centimos para evitar reales
var 1 .. 100: e2;      % Mejor limitar dominios (en vez de int)
var 1 .. 100: e3;
var 1 .. 100: e4;
```

%===== Restricciones

```
constraint e1 + e2 + e3 + e4 = 8;
constraint e1 * e2 * e3 * e4 = 8;
```


Ejemplo-2: Juan va de su casa al trabajo en coche (30-40 minutos) o en tren (al menos una hora). Luis va en coche (20-30 minutos) o en metro (40-50 minutos).

Hoy Juan parte de casa entre las 8:10 y las 8:20 y Luis llega al trabajo entre las 9:00 y las 9:10. Además, sabemos que Juan llegó al trabajo entre 10 y 20 minutos después de que Luis saliera de casa.

Formalización Modelo:

$X = \{T0, T1, T2, T3, T4\}$

$D = \{0, \dots, 70\}$; para todas las variables

Constraints =

$\{ (10 \leq T1 - T0 \leq 20),$

$(60 \leq T4 - T0 \leq 70)$

$(30 \leq T2 - T1 \leq 40 \cup 60 \leq T2 - T1),$

$(10 \leq T2 - T3 \leq 20),$

$(20 \leq T4 - T3 \leq 30 \cup 40 \leq T4 - T3 \leq 50)\}$

Running:

T1 = 10; T2 = 40; T3 = 20; T4 = 60;

T1 = 10; T2 = 40; T3 = 30; T4 = 60;

T1 = 10; T2 = 41; T3 = 30; T4 = 60;

T1 = 11; T2 = 41; T3 = 30; T4 = 60;

T1 = 10; T2 = 41; T3 = 31; T4 = 60;

T1 = 11; T2 = 41; T3 = 31; T4 = 60;

T1 = 10; T2 = 42; T3 = 30; T4 = 60;

T1 = 11; T2 = 42; T3 = 30; T4 = 60;

T1 = 12; T2 = 42; T3 = 30; T4 = 60;

T1 — LUIS — T2 T3 — JUAN — T4

%===== Variables

int: T0=0;

var 1 .. 70: T1; %Dominio indica minutos tras 8:00

var 1 .. 70: T2;

var 1 .. 70: T3;

var 1 .. 70: T4;

%===== Restricciones

constraint T1-T0 <= 20;

constraint T1-T0 >= 10;

constraint T4-T0 <= 70;

constraint T4-T0 >= 60;

constraint ((T2-T1 <= 40) /\ (T2-T1 >= 30))
 \/ (T2-T1 >= 60);

constraint T2-T3 <= 20;

constraint T2-T3 >= 10;

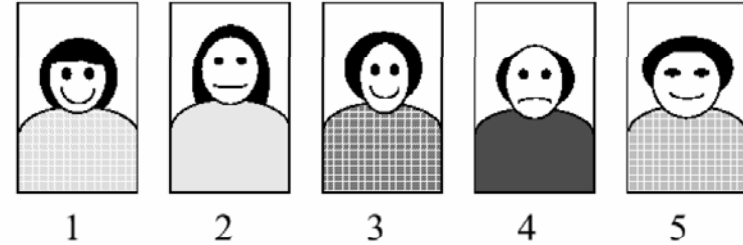
constraint ((T4-T3 <= 30) /\ (T4-T3 >= 20))
 \/ ((T4-T3 <= 50) /\ (T4-T3 >= 40));

% search

solve satisfy;

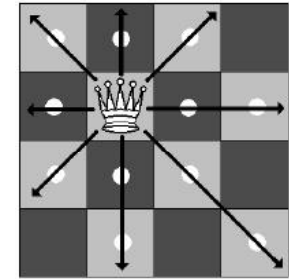
Ejemplo-3: La policía ha detenido a 5 sospechosos. Hay 5 testigos, y cada uno hace dos declaraciones: una es cierta, y la otra es falsa:

Testigo 1:	2 es Adán	3 es Baltasar
Testigo 2:	1 es Carlos	2 es David
Testigo 3:	3 es David	5 es Carlos
Testigo 4:	2 es Adán	4 es Enrique
Testigo 5:	4 es Enrique	1 es Baltasar



Identificar los sospechosos 1,2,3,4 y 5 en base a las declaraciones de los testigos

Ejemplo-4: N-reinas en un tablero $n \times n$, sin que se amenacen.



% Modelo general para N-Reinas.

int: n; % parametro tamaño del tablero

array [1..n] of var 1..n: q; % la reina en columna i esta en la fila q[i]

include "alldifferent.mzn";

% Restricciones

% No en misma columna

constraint forall (i,j in 1..n where i!=j) (q[i] != q[j]);

% No en misma diagonal SE

constraint forall (i,j in 1..n where i!=j) ((q[i] - q[j]) != (i - j));

% No en misma diagonal SO

constraint forall (i,j in 1..n where i!=j) ((q[j] - q[i]) != (i - j));

% search

solve satisfy;

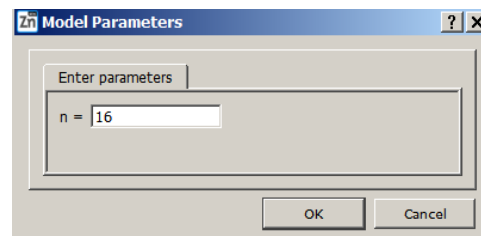
output [(show(q[i]) ++ " ") | i in 1..n];

Resultado de la ejecución

```
1 % N-Reinas
2 int: n;
3 array [1..n] of var 1..n: q; % queen is column
4 include "alldifferent.mzn";
5 % Restricciones
6
7 % No en misma columna
8 constraint forall (i,j in 1..n where i!=j) (q[i]
9 % No en misma diagonal SE
10 constraint forall (i,j in 1..n where i!=j) ((q[i]
11
12 % No en misma diagonal SO
13 constraint forall (i,j in 1..n where i!=j) ((q[j]
14
15 % search
16 solve satisfy;
17
18 output [(show(q[i]) ++ " ") | i in 1..n];|
19
```

Output

```
Compiling nreinas.mzn, additional arguments n=16; };
Running nreinas.mzn
10 12 9 6 8 2 14 16 7 15 4 11 13 5 3 1
-----
12 10 8 6 9 2 16 14 7 11 4 15 13 5 3 1
-----
10 12 9 6 8 14 2 15 7 11 4 16 13 5 3 1
-----
10 2 11 6 8 16 13 15 7 14 4 9 12 5 3 1
-----
10 8 6 9 15 2 14 16 7 12 4 11 13 5 3 1
-----
12 6 9 7 15 8 2 16 14 11 4 10 13 5 3 1
-----
12 8 13 7 9 6 2 16 14 11 4 10 15 5 3 1
-----
12 8 13 7 9 6 2 16 11 15 4 10 14 5 3 1
-----
7 14 8 6 9 16 2 15 12 10 4 13 11 5 3 1
-----
10 7 9 12 2 16 6 15 11 14 4 13 8 5 3 1
-----
13 10 7 9 6 2 14 8 11 16 4 15 12 5 3 1
-----
9 14 10 8 6 2 7 13 11 16 4 15 12 5 3 1
```



Ejemplo-5:

Había una vez una mujer llamada Pandora que tenía un enamorado, que deseaba desposarla. Pero ella solo aceptaría casarse con él si el admirador lograba resolver el acertijo que ella le planteaba. Le presentó tres cajas, una de oro, una de plata y otra de bronce, y en una de ellas estaba escondido un anillo. Cada una de las cajas tenía una inscripción:

Caja de oro: "El anillo está en esta caja"

Caja de plata: "El anillo no está en esta caja"

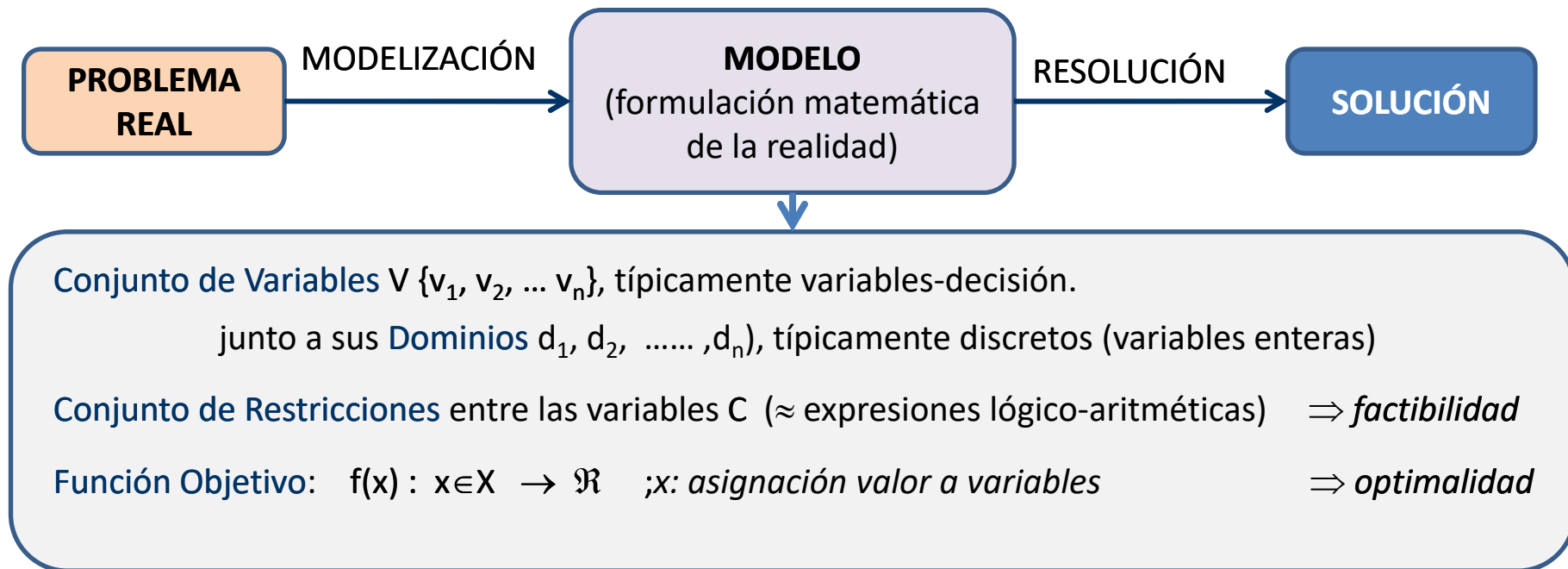
Caja de bronce: "El anillo no está en la caja de oro"

Asumiendo que tan solo una de las inscripciones podía ser cierta,

¿Dónde está el anillo?



1.4- Optimización. Problemas de Optimización sujetos a Restricciones (CSOP)



Problema de Optimización sujeta a Restricciones: *Obtener el valor de las variables decisión, tal que se cumplan las restricciones y maximice/minimize una determinada función objetivo.*

Una ***solución factible*** es una asignación consistente a las variables (consistente con C):

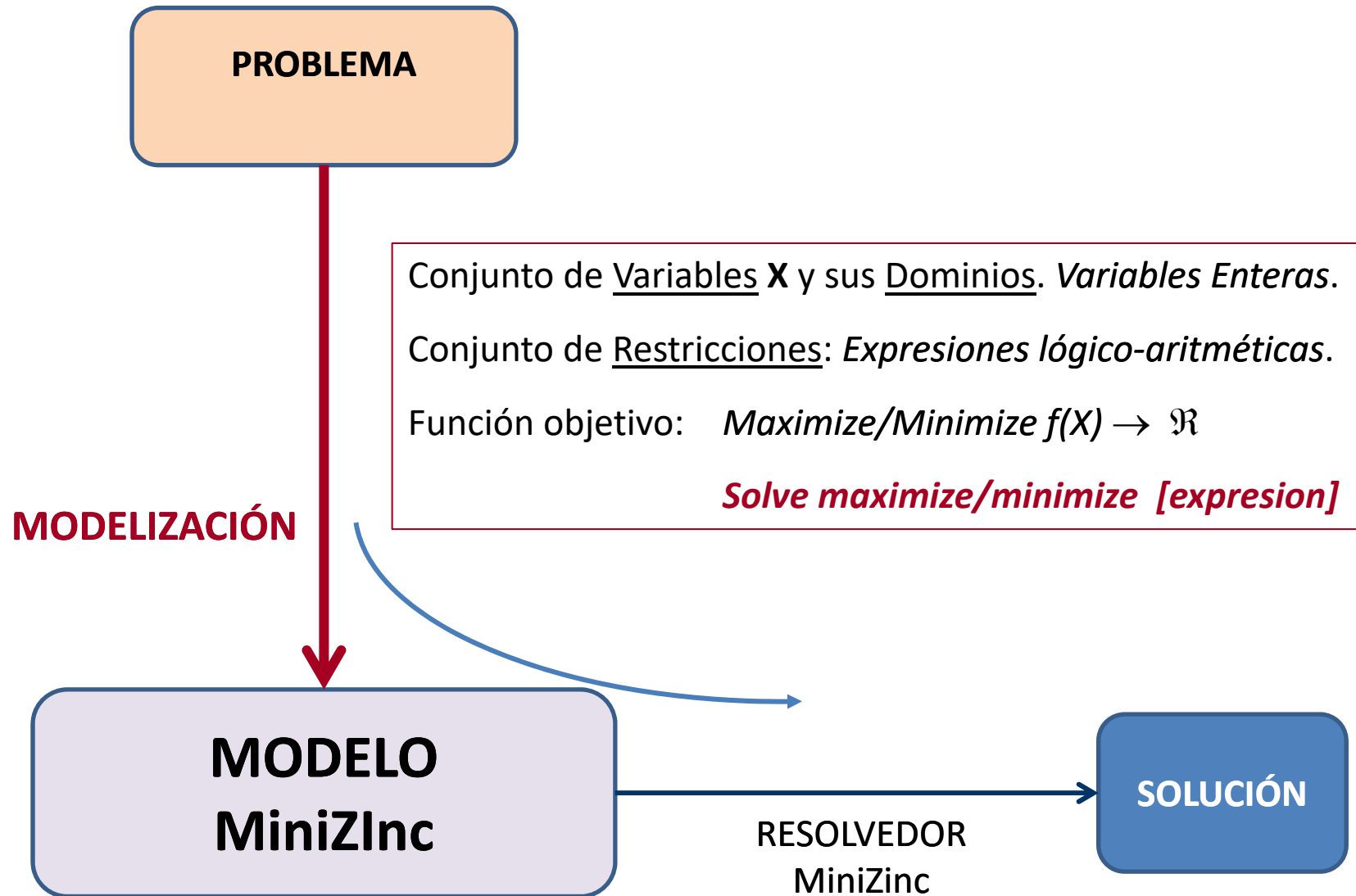
$$v_1=w_1, v_2=w_2, \dots, v_n=w_n \quad / \quad w_i \in d_i, \quad \text{Se satisfacen las restricciones C.}$$

Optimización: Obtención de una solución factible x^* , que cumpla las restricciones C, y:

$$x \in X: \quad f(x^*) \geq f(x), \quad \forall x \in X$$

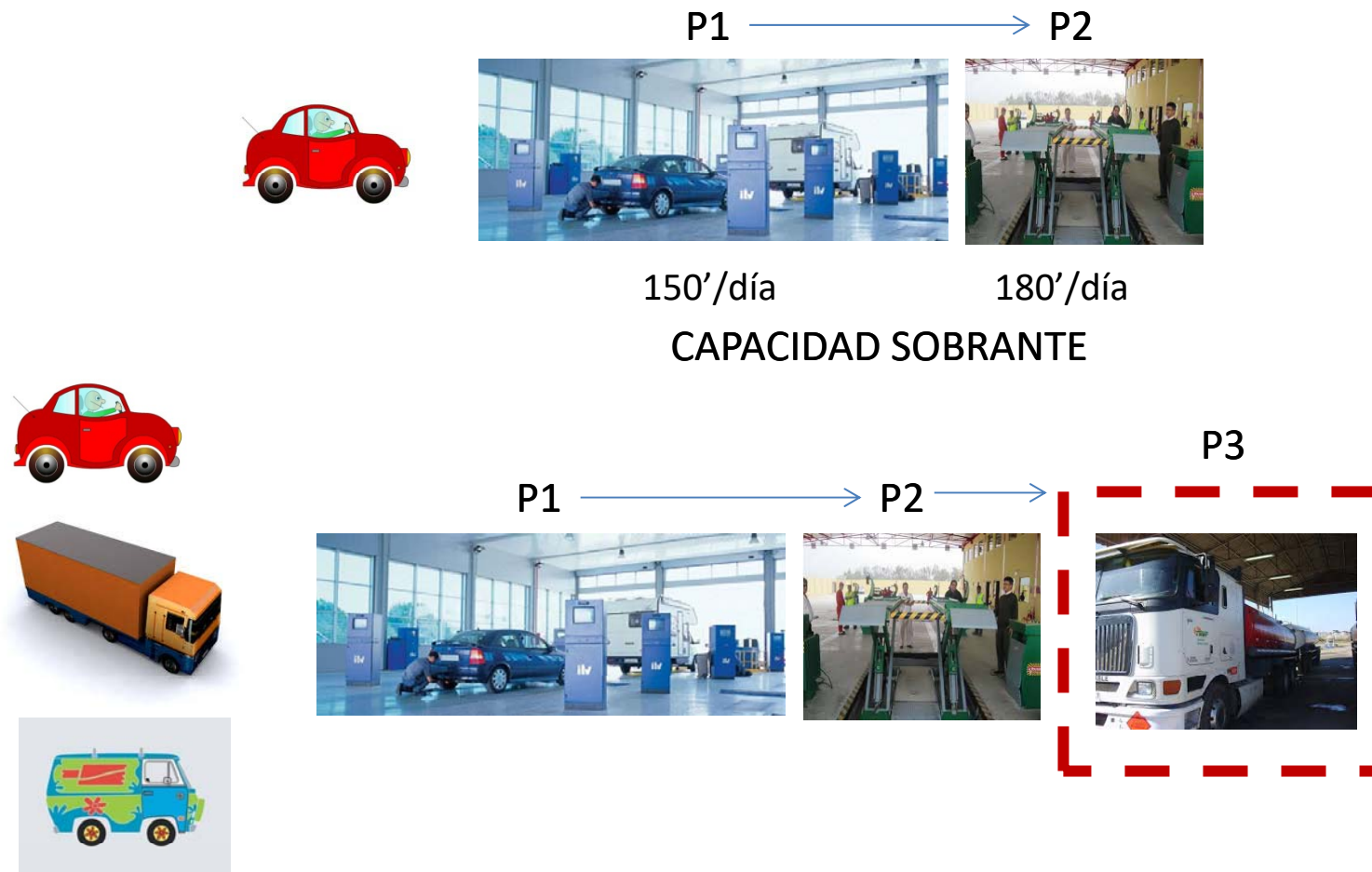
Optimización combinatoria: Dominios discretos (\Rightarrow *problemas de asignación*)

Modelado y Resolución de problemas en MiniZinc



Ejemplo1: Optimización en MiniZinc. Líneas de inspección

Se quiere ampliar una línea de inspección de vehículos de dos etapas {P1, P2} infrautilizadas, con una tercera etapa P3, a fin de poder inspeccionar nuevos tipos de vehículos (furgonetas y camiones).



La capacidad sobrante al día (en minutos) de P1 y P2 se indican en el cuadro,

Hay un tiempo requerido por cada nuevo tipo de vehículo en cada etapa.

La nueva etapa será rentable siempre que se utilice un mínimo de 500 minutos al día.

Por otra parte, la inspección de cada tipo de vehículo supone un determinado coste en material fungible indicado en la tabla, no queriendo sobrepasar un coste diario total de 300 euros.

Variables?

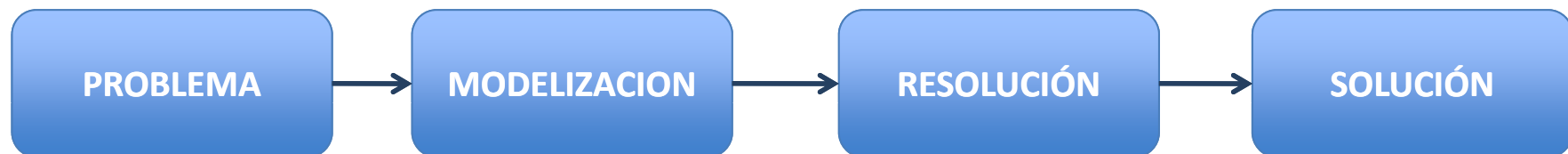
Dominios?

Restricciones?

F optimización?

Suponiendo que se inspeccionarán 3 furgonetas por cada camión, ***se desea estimar el máximo número de cada tipo de vehículo que podría ser inspeccionado en el nuevo servicio.***

	Ocupación P1	Ocupación P2	Ocupación P3 ($\geq 500'$ /día)	Coste Fungible (≤ 300)
Capac. Sobrante	150'	180'		
Furgoneta (3x)	2'	4'	8'	2
Camión (1x)	3'	3'	12'	5



	Ocupación P1	Ocupación P2	Ocupación P3 ($\geq 500'$ /día)	Coste Fungible (≤ 300)
Capac. Sobrante	150'	180'		
Furgoneta (3x)	2'	4'	8'	2
Camión (1x)	3'	3'	12'	5

Modelo del Problema

Variables Enteras {X1, X2}: Nº furgonetas y camiones

Dominio: $X1, X2 \in \{1 \dots 100\}$

Objetivo: Maximizar ($X1 + X2$)

Restricciones?:

Límite ocupación en P1: $2 X1 + 3 X2 \leq 150$

Límite ocupación en P2: $4 X1 + 3 X2 \leq 180$

Mínima ocupación en P3: $8 X1 + 12 X2 \geq 500$

Máximo Coste Fungible: $2 X1 + 5 X2 \leq 300$

Distribución Vehículos: $-3 X1 + 1 X2 = 0$

$X1 \geq 0, X2 \geq 0$

Running vehiculos.mzn

Furgonetas = 13

Camiones = 39

% Modelo MiniZinc

var 0..100: X1; % numero de furgonetas

var 0..100: X2; % numero de camiones

% Límite ocupación en P1

constraint 2*X1 + 3*X2 <= 150;

% Límite ocupación en P2

constraint 4*X1 + 3*X2 <= 180;

% Mínima ocupación en P3

constraint 8*X1 + 12*X2 >= 500;

% Máximo Coste Fungible

constraint 2*X1 + 5*X2 <= 300;

% Distribución Vehículos

constraint -3*X1 + 1*X2 = 0;

% Maximizar cantidad

solve maximize X1 + X2;

output ["Furgonetas = ", show(X1), "\n",

"Camiones = ", show(X2), "\n"];

Ejemplo2: Optimización en MiniZinc. Maximizar ganancia

Podemos hacer dos tipos de tartas:

- De plátano, que necesita 250 g de harina, 2 bananas, 75 g de azúcar y 100 g de mantequilla.
- De chocolate, que necesita 200 g de harina con levadura, 75 g de cacao, 150 g de azúcar y 150 g de mantequilla.

Podemos vender un pastel de chocolate por 4.50 y un pastel de plátano por 4.00.

Tenemos 4 kg de harina con levadura, 6 plátanos, 2 kg de azúcar, 500 g de mantequilla y 500 g de cacao.

¿Cuántos pasteles de cada tipo de pastel debemos hacer para maximizar ganancias?

Ejemplo3: Optimización en MiniZinc. Carga de Camiones

Un conjunto de objetos $\{O_i\}$ de diferentes pesos deben ser cargados en un conjunto de camiones.

Cada camión tiene un máximo peso que puede cargar $\{C_i\}$

Se quiere obtener la distribución de los objetos en los camiones, asumiendo:

Opción a) Todos los objetos caben en los camiones disponibles.

Opción b) No caben todos los objetos y se quiere maximizar el número de objetos cargados.

$$\{O_1, O_2, O_3, \dots, O_n\} \Rightarrow \{C_1, C_2, C_3, \dots, C_m\}$$

Restricciones

R1: La suma de los pesos de los objetos asignados a cada camión no excede su capacidad

Opción a): R2: Todos los objetos están asignados a un (y solo a un) camión.

Opción b): R2: Los objetos están asignados, como mucho, a un camión.



FACTIBILIDAD
OPTIMALIDAD

$$\{O1, O2, O3, \dots, On\} \Rightarrow \{C1, C2, C3, \dots, Cm\}$$

Variables

Peso de Objetos: Peso [o], o in 1..n

Capacidad Camiones: Capacidad [c], c in 1..m

Matriz de asignación: Asigna [1..n, 1..m] $\in 0..1$

A	O1	O2	----	----	On
C1	1	0			0
C2	0	0			1

Cn	0	1			0

Restricciones

R1: La suma del peso de los objetos cargados en cada camión no excede su capacidad

$$\forall c \text{ in } 1..m, \sum_{o \text{ in } 1..n} \text{Asigna}[o, c] * \text{Peso}[o] \leq \text{Capacidad}[c]$$

R2.a) Todos los objetos están asignados a un (y solo a un) camión.

$$\forall o \text{ in } 1..n, \sum_{c \text{ in } 1..m} \text{Asigna}[o, c] = 1$$

R2.b) Todos los objetos están asignados, como mucho, a un camión.

$$\forall o \text{ in } 1..n, \sum_{c \text{ in } 1..m} \text{Asigna}[o, c] \leq 1$$

FACTIBILIDAD

OPTIMALIDAD

Peso Objetos: Peso [o], o in 1..n

Cap. Camiones: Capacidad [c], c in 1..m

Asignación: Asigna [1..n, 1..m] ∈ 0 ..1

Restricciones

$\forall c \text{ in } 1..m, \sum_{o \text{ in } 1..n} \text{Asigna}[o, c] * \text{Peso}[o] \leq \text{Capacidad}[c]$

a) $\forall o \text{ in } 1..n, \sum_{c \text{ in } 1..m} \text{Asigna}[o, c] = 1$

b) $\forall o \text{ in } 1..n, \sum_{c \text{ in } 1..m} \text{Asigna}[o, c] \leq 1$

% Asignacion objetos a camiones

include "datos.dzn"; % fichero de datos

int: ncam; int: nobj;

array [1..nobj] of var 1..100: objeto;

array [1..ncam] of var 1..200: camion;

array [1..nobj, 1..ncam] of var 0..1: asigna;

array [1..ncam] of var 0..200: cargacamion;

var 0..100: objetos;

% DATOS (en fichero aparte)

ncam = 3;

nobj = 10;

objeto = [24, 34, 31, 45, 42, 51, 19, 21, 43, 47];

camion = [117, 133, 111];

% Peso de los objetos

% Capacidad camiones

% Asignacion. Variables Decision

% carga de los camiones. No necesaria.

% número de objetos cargados. No necesaria.

constraint forall (c in 1..ncam) (sum (o in 1..nobj) (asigna [o, c] * objeto[o]) <= camion[c]); **%Restricc. Capacidad**

% constraint forall (o in 1..nobj) (sum (c in 1..ncam) (asigna [o, c]) = 1); **% a) Caben todos**

constraint forall (o in 1..nobj) (sum (c in 1..ncam) (asigna [o, c]) <= 1); **% b) No caben todos**

%===== Expresiones auxiliares

constraint forall (c in 1..ncam) (cargacamion[c] = (sum (o in 1..nobj) (asigna [o, c] * objeto[o]))); **%carga de cam.**

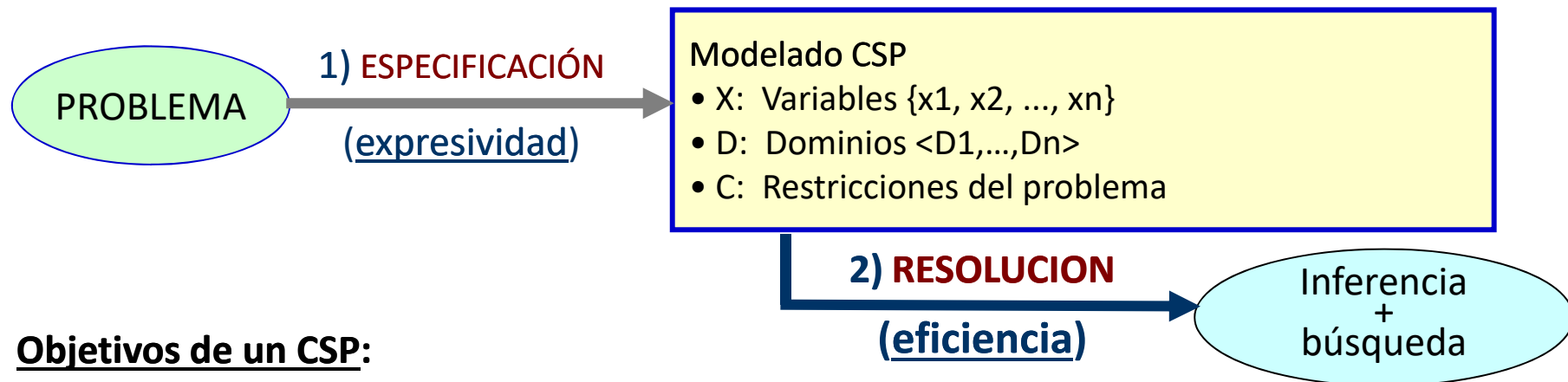
constraint objetos = (sum (o in 1..nobj, c in 1..ncam) (asigna [o, c])); **%numero objetos cargados**

%solve satisfy; **% FACTIBILIDAD: caben todos**

solve maximize objetos; **% OPTIMALIDAD (incluye satisfabilidad)**

output ["bloques cargados= " ++ show(bloques) ++ "\n" ++ "carga de los camiones= "
++ show(cargacamion) ++ "\n" ++ "Asignacion= " ++ "\n" ++ show(asigna)];

2.- Operativa en un CSP.



Objetivos de un CSP:

- Obtener respuestas: Nuevas restricciones derivadas, Dominios acotados.
- ¿Tiene solución? Consistencia.
- Obtener una solución vs. obtener todas las soluciones.
- Obtener una solución óptima, o al menos una buena solución, medida por alguna función objetivo que representa la calidad (CSOP).

CSP
es
NP-completo

CSOP
es
NP-duro

Algoritmos para CSP:

- Técnicas Inferenciales (Procesos de Clausura): Obtienen las consecuencias de las restricciones explícitamente conocidas
⇒ Acotan el espacio de búsqueda
- Técnicas de Búsqueda (Algoritmos CSP): Obtienen una solución, guiados por heurísticas.
⇒ Técnicas Híbridas

2.1.- Conceptos Básicos CSP

Dado un CSP,

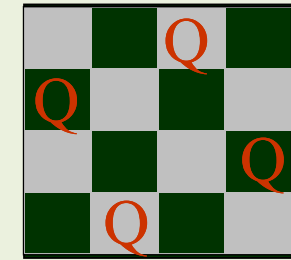
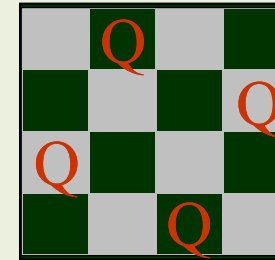
X: Variables $\{x_1, x_2, \dots, x_n\}$
D: Dominios $\langle D_1, \dots, D_n \rangle$
C: Restricciones del problema.

- Una **instanciación** (o interpretación) de las variables X es una asignación de valores a las variables en sus dominios:

$$x_1=v_1, x_2=v_2, \dots, x_n=v_n \quad / \quad v_i \in D_i$$

- Una **solución** del CSP es una **instanciación consistente** de todas las variables, tal que todas las restricciones del CSP se cumplan.
- Un valor $v \in D_i$ es un valor **consistente** (o posible) para x_i si existe una solución del CSP en la cual $x_i=v$.
- El **dominio mínimo** de una variable x_i es el conjunto de todos los valores posibles para la variable. Un CSP es mínimo *sii* todas sus restricciones son mínimas.
- Un CSP es **consistente** *sii* tiene al menos una solución.

Ejemplo: 4-queen (dos soluciones):



Variables: $\{X_1, X_2, X_3, X_4\} \in [1..4]$

Instanciación:

$X_1=3$ es una **instanciación** de X_1

Solución del CSP:

$(X_1=2, X_2=4, X_3=1, X_4=3)$

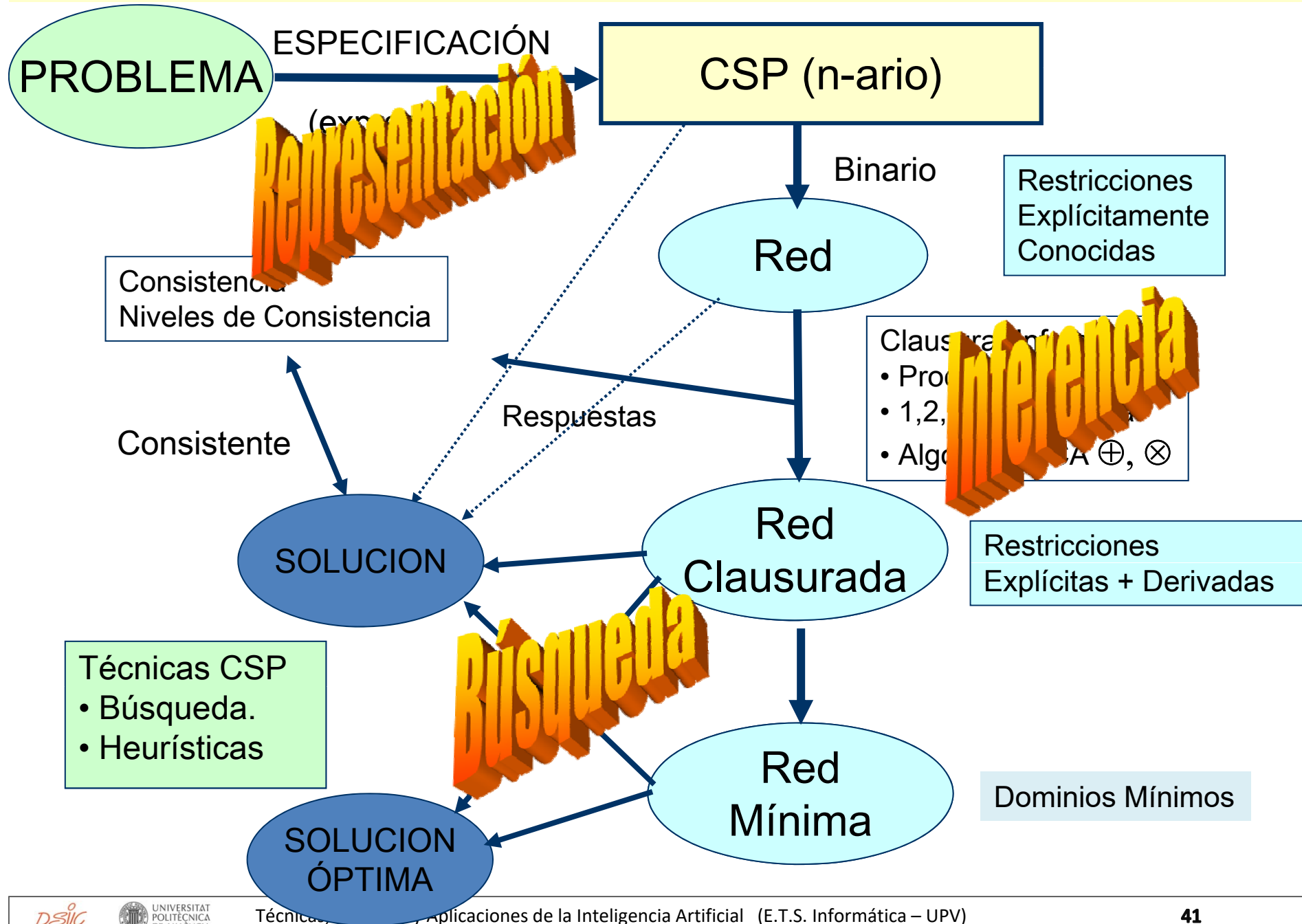
Valor consistente:

3 es un **valor consistente** para X_1 ,
pero 1 no lo es.

Dominio mínimo de $X_2 \Rightarrow \{1, 4\}$

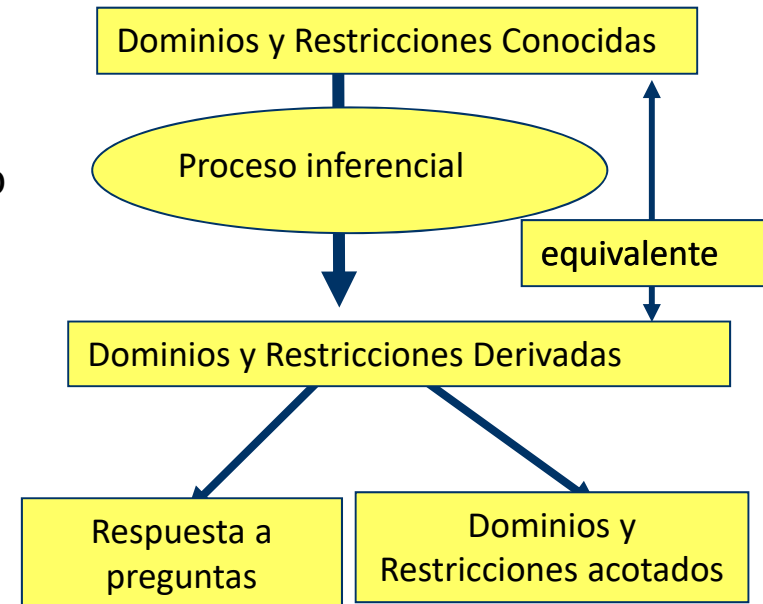
Dominio mínimo de $X_4 \Rightarrow \{2, 3\}$

Operativa en un CSP



2.2.- Técnicas inferenciales

- **Infiere conclusiones** sobre la información explícitamente conocida del problema.
- Los algoritmos de consistencia **filtran y eliminan** de los dominios de las variables aquellos valores que no van a formar parte de ninguna solución.
- Permiten obtener respuestas y limitar el espacio de búsqueda de soluciones (más eficiencia).
- **Diversos niveles de inferencia**



El proceso inferencial se puede hacer:

- Antes del proceso de búsqueda: acotan dominios y restricciones.
- Durante el proceso de búsqueda: Acotan dinámicamente la búsqueda
⇒ *Algoritmos de Búsqueda Híbridos* (búsqueda + inferencia).

Ejemplo: Sudoku

Por donde empezar?

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

¿Fuerza Bruta?

Dominio: {1,2,3,4,5,6,7,8,9}

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7					?			8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

Hagamos alguna inferencia...

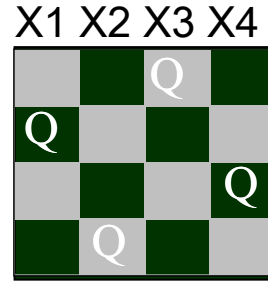
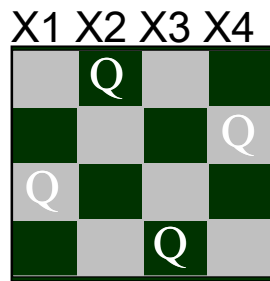
		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7					4			8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

Un simple proceso de 2-consistencia

¿Podríamos resolver todo el sudoku sin búsqueda?

Ejemplo: Colocar 4 reinas en un tablero 4x4



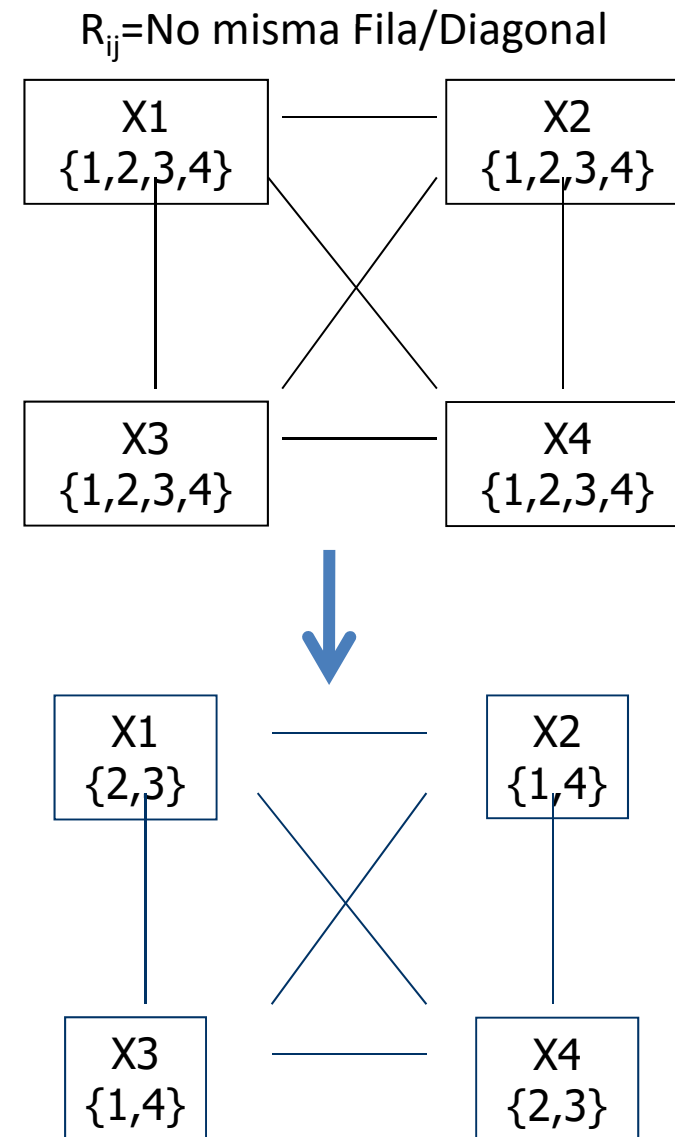
Variables: $\{X1, X2, X3, X4\} \in [1..4]$

Proceso Inferencial:
Se acotan dominios

$X1 \in \{2,3\}, X2 \in \{1,4\}, X3 \in \{1,4\}, X4 \in \{2,3\}$

✓ Acota el espacio de búsqueda.

✓ Respuesta a preguntas:
¿Puede ser $X1=4$? \Rightarrow NO



Procesos Inferenciales. Niveles de Consistencia. K-consistencia.

Los procesos inferenciales en CSP suelen ir ligados al nivel de **k-consistencia** que garantizan.

- Mayor nivel de consistencia implica:
 - Más información deducida: Más poda de dominios/restricciones.
 - Mayor capacidad de dar respuestas.
 - Menos esfuerzo de búsqueda.
 - Más posibilidades de detectar la inconsistencia de un CSP.
 - Más coste.

Niveles de Consistencia:

1-consistencia: Nodo consistencia, $O(n)$

2-consistencia: Arco consistencia, $O(n^2)$

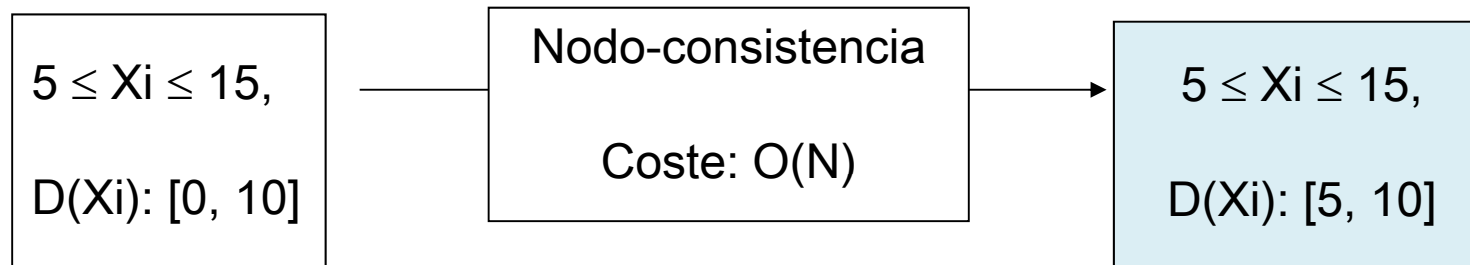
3-consistencia: Senda consistencia, $O(n^3)$

K-consistencia: Cualquier subconjunto de $k-1$ variables es consistente .

Consistencia de nodo (1-consistencia)

- Sub-redes de 1 variable
- Los dominios de las variables son consistentes con las restricciones unarias sobre las variables.
- Una red es *nodo-consistente* sii todos sus nodos son consistentes:

$$\forall x_i \in \text{CSP}, \exists v_i \in d_i / c_i(v_i)$$



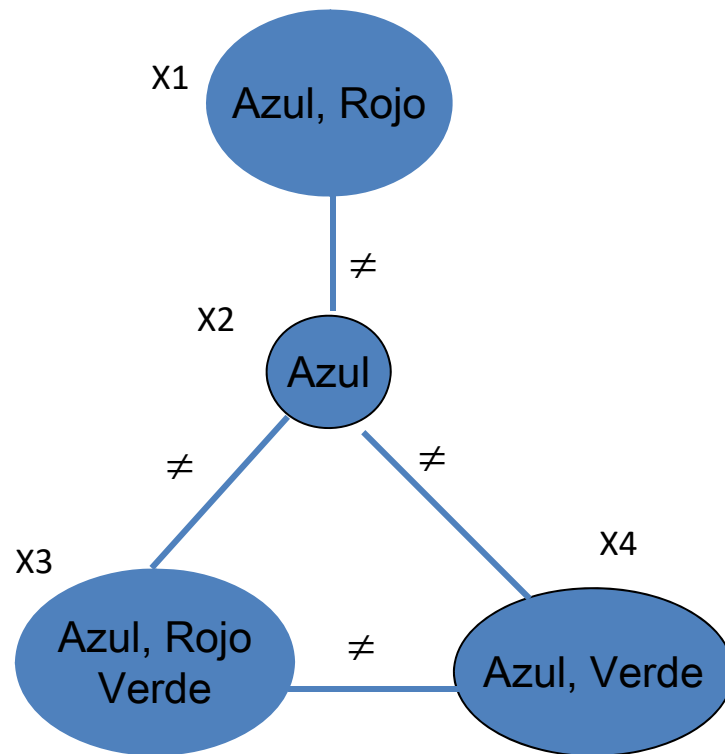
*Se han acotado los
dominios de las variables*

Consistencia de arco (2-consistencia)

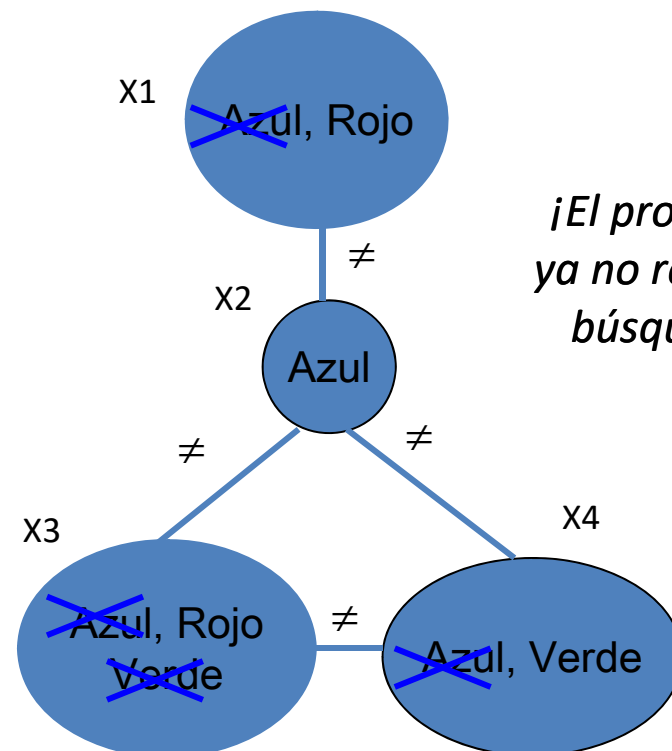
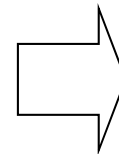
- Sub-redes de 2-variables
- Para cada restricción entre un par de variables ($x_i \{c_{ij}\} x_j$), los dominios de las variables con consistentes con la restricción (arco) entre las variables.
- Una red temporal es *arco-consistente* si todos sus arcos son consistentes.

$$\forall c_{ij} \subseteq \text{CSP}, \forall v_i \in d_i \exists v_j \in d_j / c_{ij}(v_i, v_j)$$

Ejemplo: Cómo elimina valores de dominios la 2-consistencia.



Grafo Inicial



Grafo arco-consistente

¡El problema ya no requiere búsqueda!

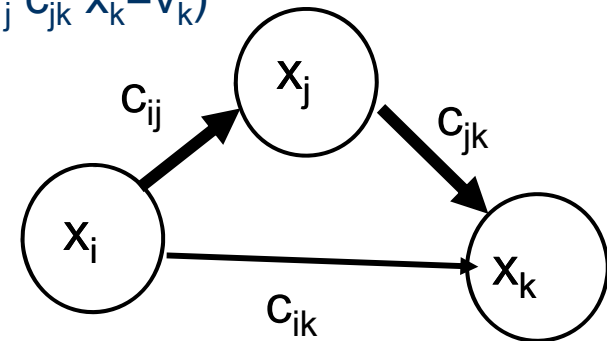
3-consistencia: senda consistencia

Cualquier sub-red de 3 nodos es consistente.

En cada senda de longitud dos, $(x_i \ c_{ij} \ x_j)$, $(x_j \ c_{jk} \ x_k)$, los dominios de las variables con consistentes con las restricciones (binarias) entre las variables:

$$\forall c_{ij} \subseteq \text{CSP}, \forall v_i \in d_i, \forall v_j \in d_j / (x_i=v_i \ c_{ij} \ x_j=v_j) \Rightarrow$$

$$\exists v_k \in d_k / (x_i=v_i \ c_{ik} \ x_k=v_k) \wedge (x_j=v_j \ c_{jk} \ x_k=v_k)$$



Las restricciones entre tres nodos son consistentes si:

$$(c_{ij} \otimes c_{jk}) \oplus c_{ik} \neq \emptyset, \quad \text{Ídem con } c_{ij}, \ c_{jk}$$

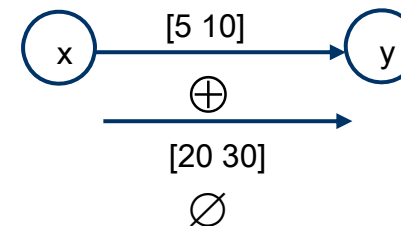
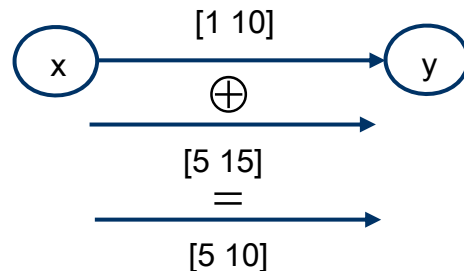
Algoritmo de Clausura Transitiva (TCA)

- Operaciones: Adición \oplus , Composición \otimes
- *Clausura de restricciones en redes binarias*

Operativa de las Restricciones (\oplus \otimes)

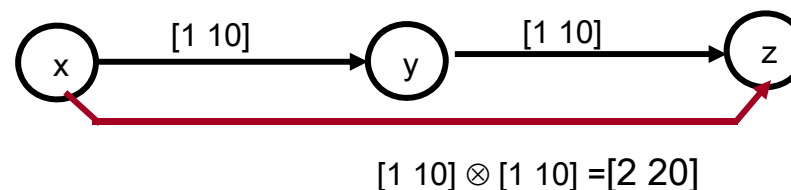
Adición (\oplus): $(X_i \{R_i\} X_j) \oplus (X_i \{R_j\} X_j) = X_i \{R_i \oplus R_j\} X_j$

- Actualización, por una nueva restricción, de la restricción existente previa entre los dos variables.
- Cuando no hay intersección se produce una inconsistencia: ¡CSP no consistente!



Multiplicación o Composición (\otimes): $(X_i \{R_i\} X_j) \otimes (X_j \{R_j\} X_k) = X_i \{R_i \otimes R_j\} X_k$

- Obtiene la restricción temporal entre dos variables a través de una senda entre ellas.



Las operaciones \oplus y \otimes dependen de la tipología de las restricciones binarias:

Álgebra de restricciones de puntos, intervalos, duraciones, etc.

Ejemplo: CSP Temporales cualitativos (puntos)

- Variables: Puntos de Tiempo $\{t_i\}$
- Restricciones Temporales (2^3): $t_i \{<, =, >\} t_j$

Operación de adición \oplus
(\emptyset es la inconsistencia):

\oplus	$<$	\leq	$>$	\geq	$=$	\neq	$?$
$<$	$<$	$<$	\emptyset	\emptyset	\emptyset	$<$	$<$
\leq	$<$	\leq	\emptyset	$=$	$=$	$<$	\leq
$>$	\emptyset	\emptyset	$>$	$>$	\emptyset	$>$	$>$
\geq	\emptyset	$=$	$>$	\geq	$=$	$>$	\geq
$=$	\emptyset	$=$	\emptyset	$=$	$=$	\emptyset	$=$
\neq	$<$	$<$	$>$	$>$	\emptyset	\neq	\neq
$?$	$<$	\leq	$>$	\geq	$=$	\neq	$?$

Operación de Combinación (\otimes)

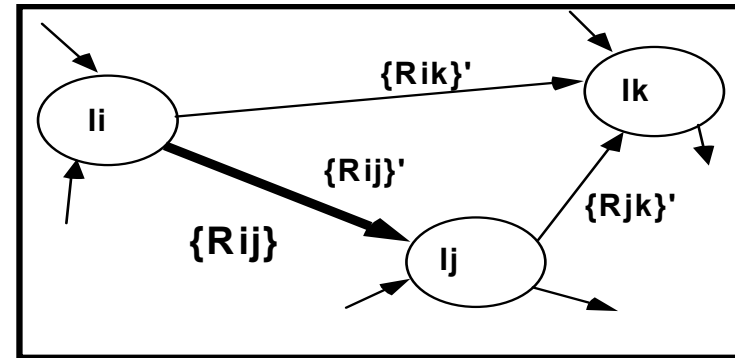
$$(t_i R_i t_j) \otimes (t_j R_j t_k) = t_i (R_i \otimes R_j) t_k$$

\otimes	$<$	\leq	$>$	\geq	$=$	\neq	$?$
$<$	$<$	$<$	$?$	$?$	$<$	$?$	$?$
\leq	$<$	\leq	$?$	$?$	\leq	$?$	$?$
$>$	$?$	$?$	$>$	$>$	$>$	$?$	$?$
\geq	$?$	$?$	$>$	\geq	\geq	$?$	$?$
$=$	$<$	\leq	$>$	\geq	$=$	\neq	$?$
\neq	$?$	$?$	$?$	$?$	\neq	$?$	$?$
$?$	$?$	$?$	$?$	$?$	$?$	$?$	$?$

Transitive Closure Algorithm (TCA):

Clausura de una red de restricciones: obtiene una red senda consistente aplicando las operaciones \otimes , \oplus (Álgebras de restricciones)

- *Recursivamente*, va haciendo consistente todas las 3-subredes de un grafo.
- Aplica las operaciones \oplus y \otimes
- Coste $O(n^3)$

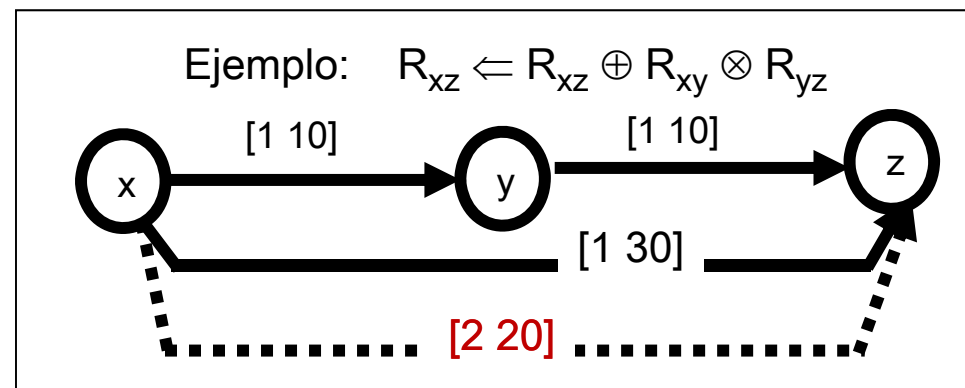


Si cada senda de longitud 2 es consistente, entonces todas las sendas de cualquier longitud son consistentes (Montanari, 1974).

Luego, 3-consistencia y senda-consistencia (path-consistency) son conceptos equivalentes.

La 3-consistencia acota dominios

Existen muy diversos tipos de técnicas inferenciales, dependientes de la topología de la red de restricciones



Conclusiones

Los CSPs nos permiten definir problemas sin tener en cuenta su proceso de resolución

- Solo nos preocupamos de definir las variables, dominios y las restricciones que deben satisfacerse en el problema

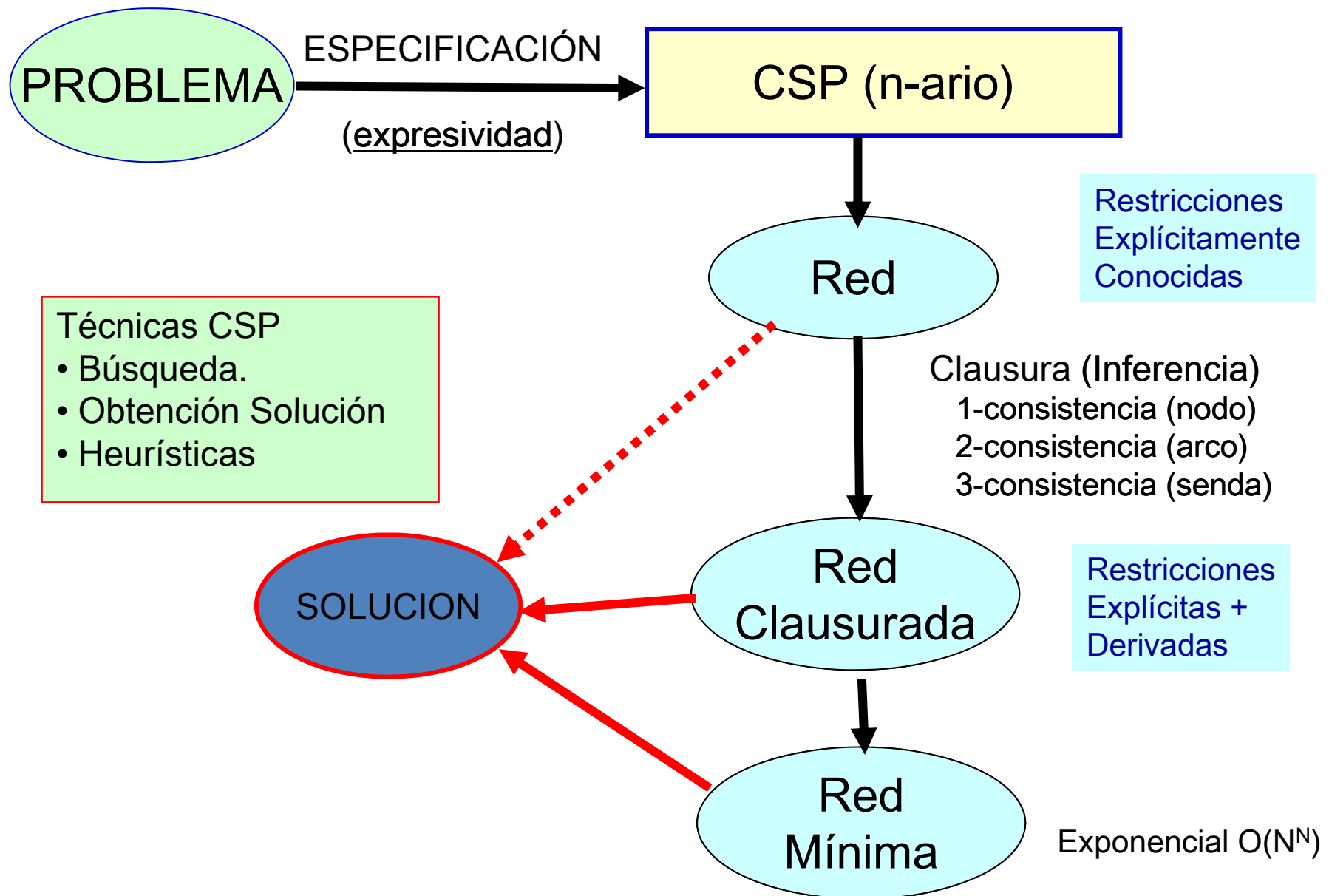
Las técnicas de inferencia nos permiten limitar el dominio de las variables e incluso responder a preguntas sin NECESIDAD de realizar búsqueda

- Podemos obtener distintos grados de consistencia: 1-, 2-, 3-... n- consistencia con algoritmos sencillos, pero a un determinado precio

Permiten definir heurísticas independientes del dominio (así como *ad-hoc*) para la obtención de soluciones.

Se utilizan en multitud de problemas reales

2.3.- Técnicas de Resolución. Búsqueda de soluciones



Búsqueda de soluciones

Especificación CSP $\{V, D, C\} \rightarrow$ Solución: instanciación consistente

Métodos de Búsqueda

Generate & Test: Generar todas las posibles combinaciones de valores y comprobar si se cumplen las restricciones.

Técnicas de Backtracking: Comprobar en cada paso si aparecen inconsistencias y, en ese caso, retroceder en la búsqueda para probar con otros valores.

- *Look-Backward*: En cada instanciación se comprueban las restricciones podando las ramas con asignaciones inconsistentes (*~ branch & bound*).
- *Look-Forward*: En cada instanciación se comprueban también los posibles valores de las restantes variables por instanciar, podando sus dominios y comprobando que quedan posibles valores.
 - *Forward-Checking*
 - *Real-full Looking Ahead (MAC)*

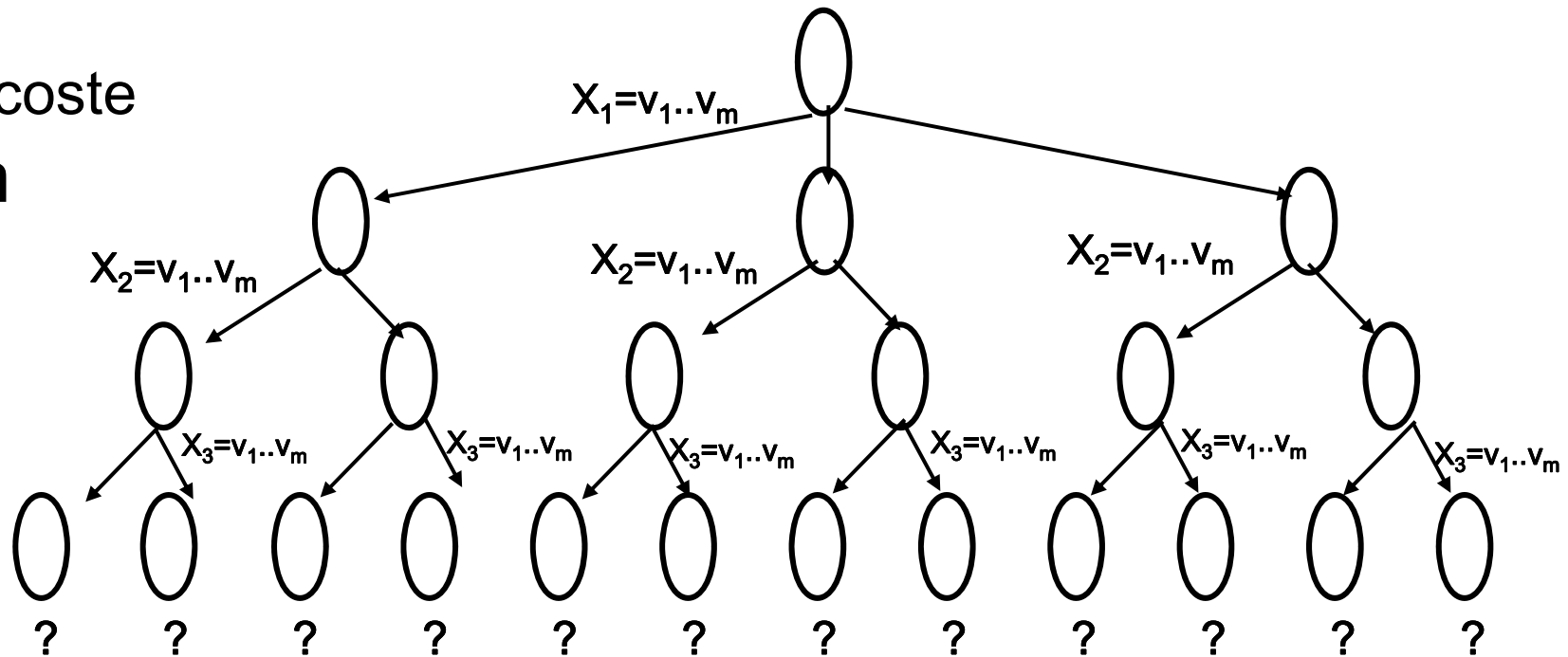
Procedimiento general: Generate and Test

Generar todas las posibles n-tuplas (posibles asignaciones de m valores a las n variables) y comprobar, para cada una de ellas, que se satisfacen todas las restricciones.

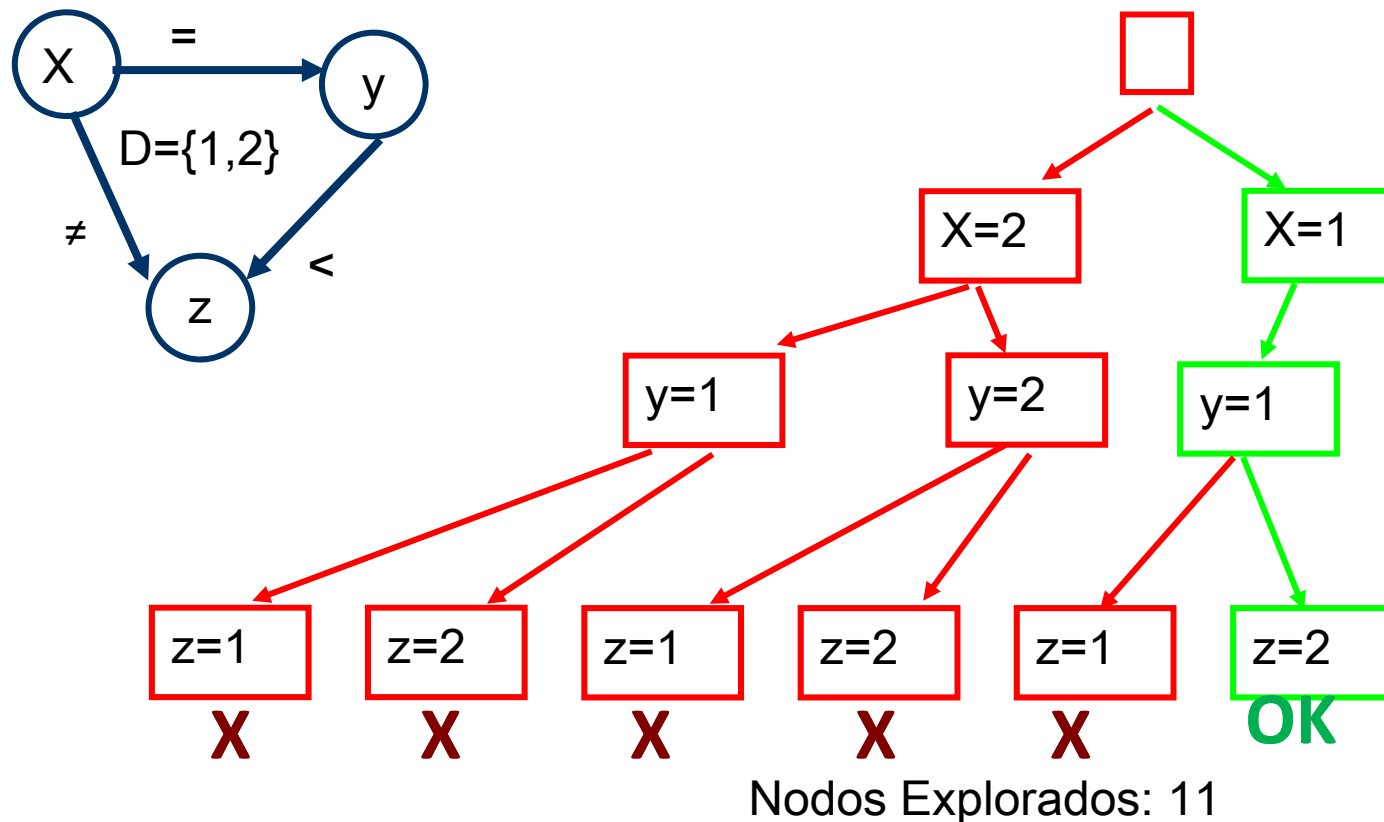
- Para 'n' variables, y tamaños de dominio ' m_i ', se genera un árbol de n niveles, con ramificación de m_i nodos en cada nivel, representando los posibles valores que puede tomar la variable v_i en su dominio d_i .
- N° de hojas (nodos terminales), sobre los que debe comprobarse las restricciones: $m_1 * m_2 * \dots * m_n$

alto coste

m^n



Ejemplo. Generate and Test



⇒ Solución general para reducir el tamaño del árbol:

- Comprobar, en cada instanciación, las asignaciones parciales realizadas hasta ese nodo.
- Si resultan inconsistentes, no se sigue generando por dicho nodo: *Técnicas backtracking (look-backward)*

Procedimientos Backtracking. Técnicas

Exploración del árbol mediante backtracking:

Procedure Backtracking (k, V[n]) ;Llamada: *Backtracking (1, V[n])*. $k=1$.

Begin

Generar $V[k] \in d_k$

If **Comprobar (k, V[n])**

then *(* comprobar=ok *)*

If $k=n$ *(* todas las variables instancias *)*

then **Return (V[n])** *(*solución*)*

← Nivel K+1 else **Backtraking (k+1, V[n])**

else *(* comprobar=NO-ok *)*

If quedan-valores (d_k)

← Nivel K, Otro valor then **Backtraking (k, V[n])**

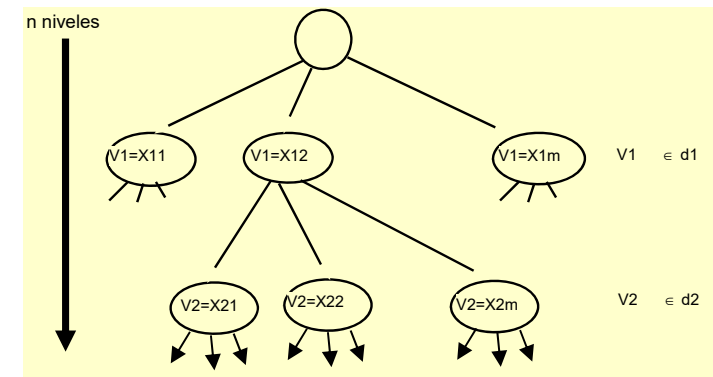
else

if $k=1$ then **Return (\emptyset)** *(*fallo*)*

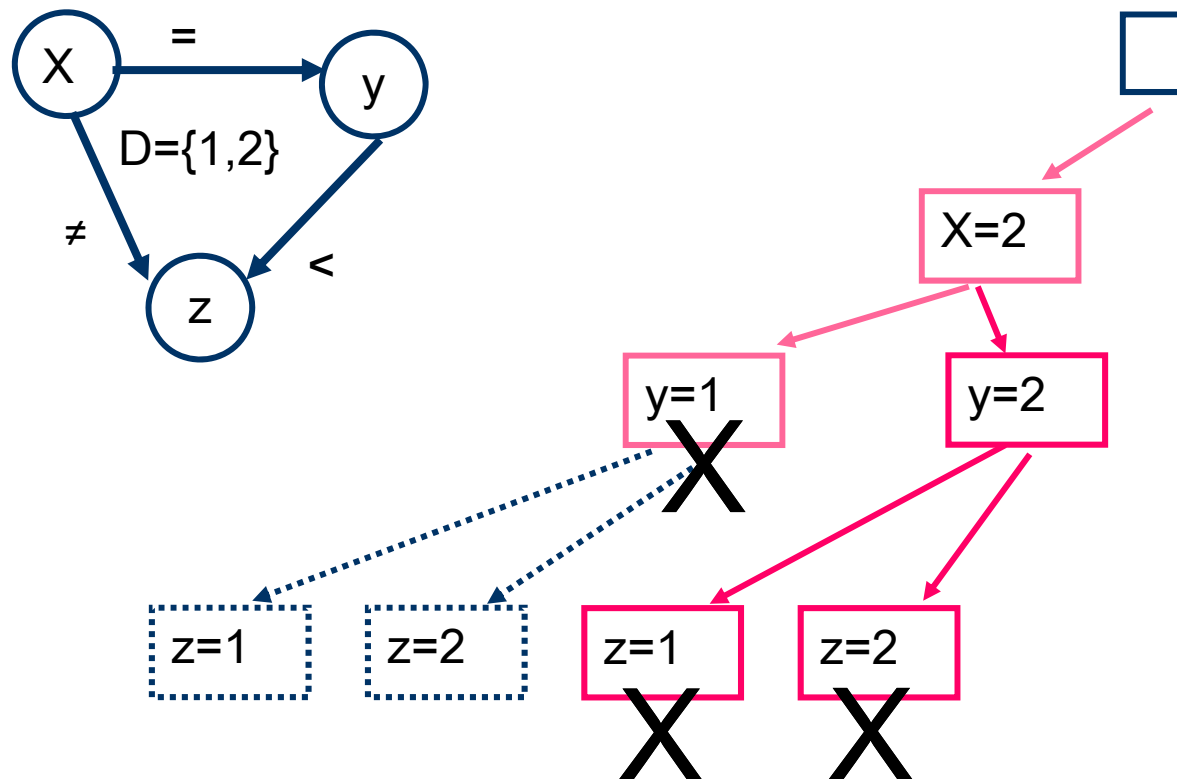
← Nivel K-1: Vuelvo nivel anterior else **Backtraking (k-1, V[n])**

end

La función *Comprobar (K, V[n])* comprueba la validez de las k instanciaciones realizadas: $V[1], V[2], \dots, V[k]$.

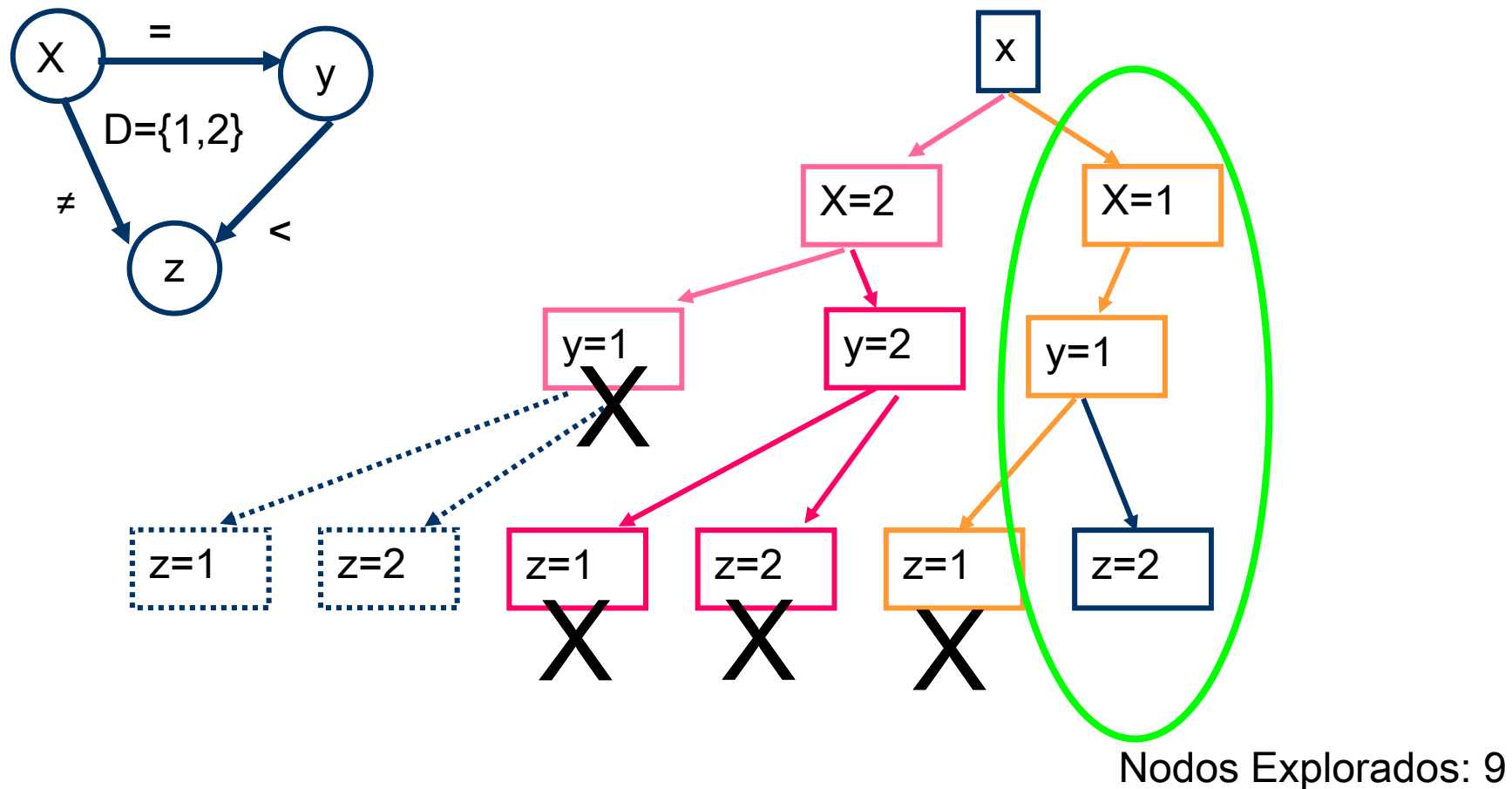


Ejemplo. Standard Backtracking (look backward)



Comprobar, en cada instanciación, las asignaciones parciales realizadas hasta ese nodo.
Si resultan inconsistentes, no se sigue generando por dicho nodo: **Técnicas backtracking**

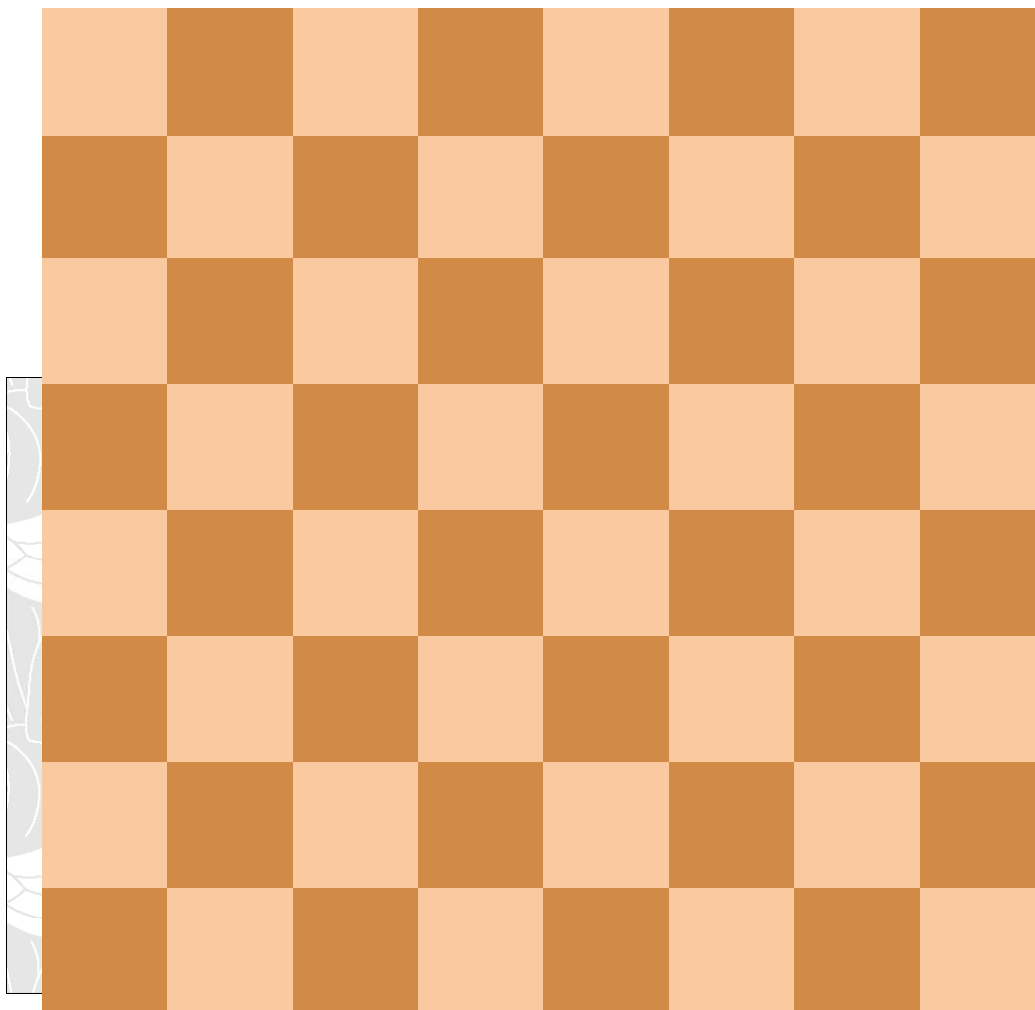
Ejemplo. Standard Backtracking (look backward)



Comprobar, en cada instanciación, las asignaciones parciales realizadas hasta ese nodo. Si resultan inconsistentes, no se sigue generando por dicho nodo: Técnicas backtracking

¿En qué orden es mejor explorar las variables/valores?

Ejemplo Backtracking 8-reinas



Backtracking N by N queens applet

Aunque, para $N > 3$, es posible encontrar soluciones sin búsqueda:

Determinar las posiciones puede hacerse en $O(N)$.

¿En qué orden es mejor explorar las variables/valores?

Backtracking \Rightarrow Métodos Looking Ahead

A costa de mayor esfuerzo en el cálculo de la función 'Comprobar', se intenta efectuar mayor poda en el árbol de búsqueda, comprobando consistencias de variables por instanciar.

```
Begin
  Generar  $V[k] \in dk$ 
  If Comprobar ( $k, V[n]$ ) then....
```

a) Forward-Checking (FC)

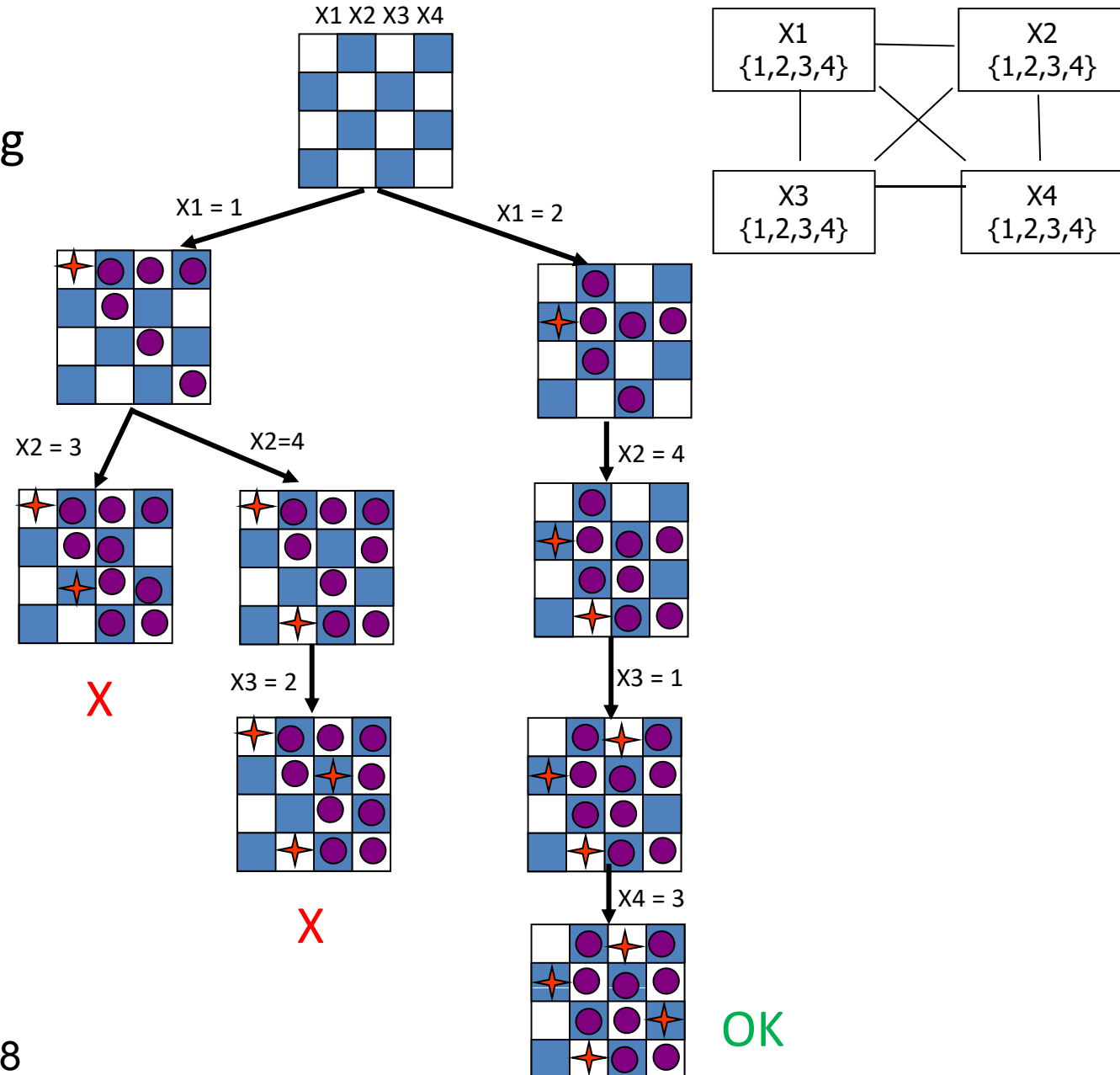
En las variables aún no instanciadas se eliminan de los dominios los valores inconsistentes con respecto a las instanciaciones ya realizadas.

- *Cada variable por instanciar debe quedar al menos con un valor posible en el dominio.*
- *Es como una arco-consistencia, pero solo con las variables ya instanciadas*

b) *Real-full Looking Ahead (MAC)*

Ejemplo. 4-reinas

Forward Checking



Nodos Explorados: 8

Métodos Looking Ahead

a) *Forward-Checking (FC): arco-consistencia, pero solo con las variables ya instanciadas*

b) Maintaining Arc Consistency (Full-Looking Ahead, MAC)

Además de FC,

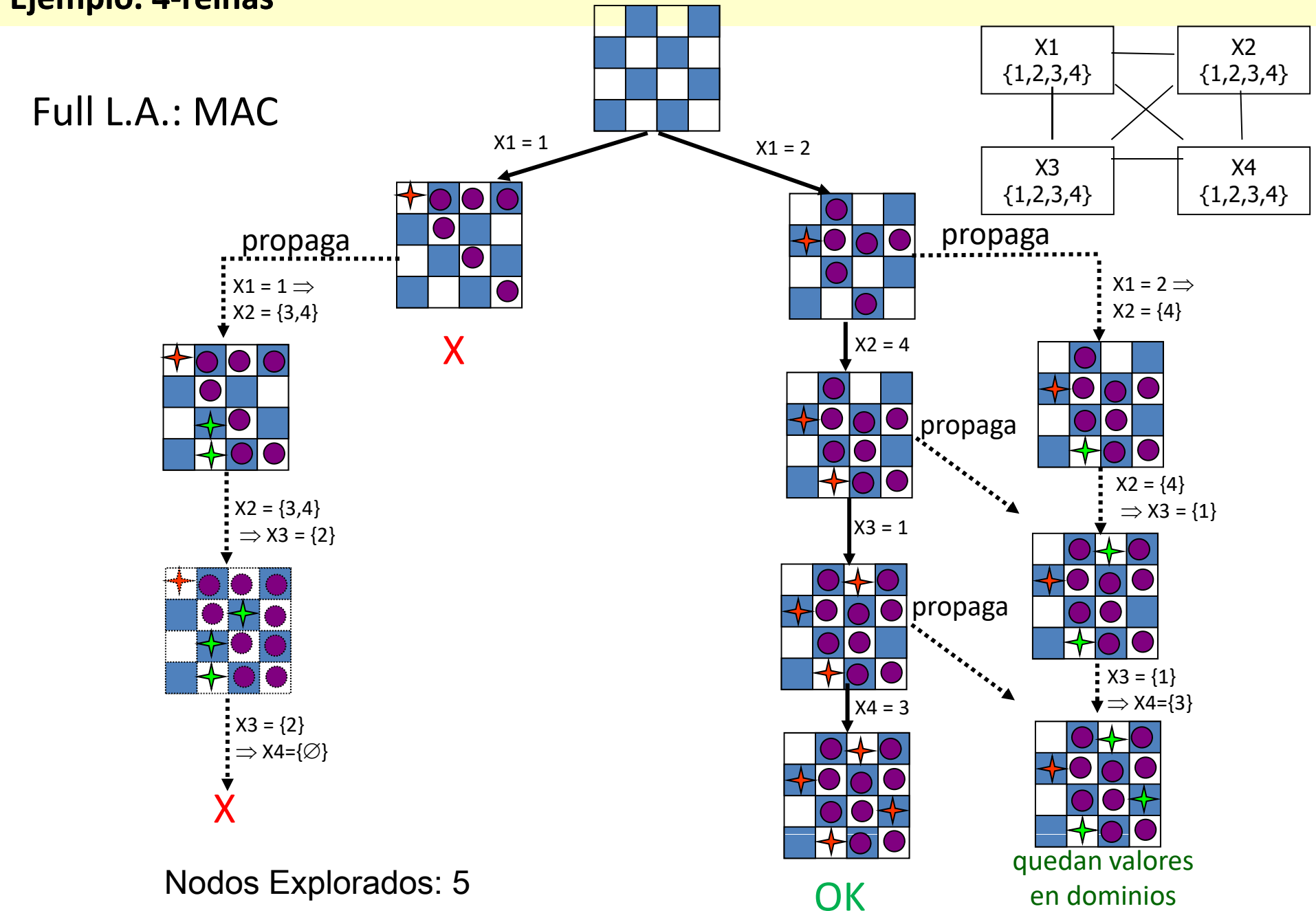
Sobre cada variable por instanciar, se podan aquellos valores inconsistentes con cada posible valor de todas las variables que quedan por instanciar.

Propaga los valores eliminados en el proceso de forward-checking

Similar a mantener una red arco-consistente, con todas las variables, teniendo en cuenta las asignaciones ya efectuadas.

Ejemplo. 4-reinas

Full L.A.: MAC



Nodos Explorados: 5

OK

quedan valores
en dominios

Ejemplo. El puzzle de las 4 casas

D vive en una casa con menor número que B,
B vive junto a A en una casa con mayor número,
Hay al menos una casa entre B y C,
D no vive en una casa cuyo número es 2, C no vive en una casa cuyo número es 4.

Representación:

Variables: A, B, C y D

Dominios: $d_A = d_B = d_C = d_D = \{1, 2, 3, 4\}$

Restricciones:

$$C \neq 4$$

$$D \neq 2$$

$$B = A + 1$$

$$A \neq C$$

$$D < B$$

$$A \neq D$$

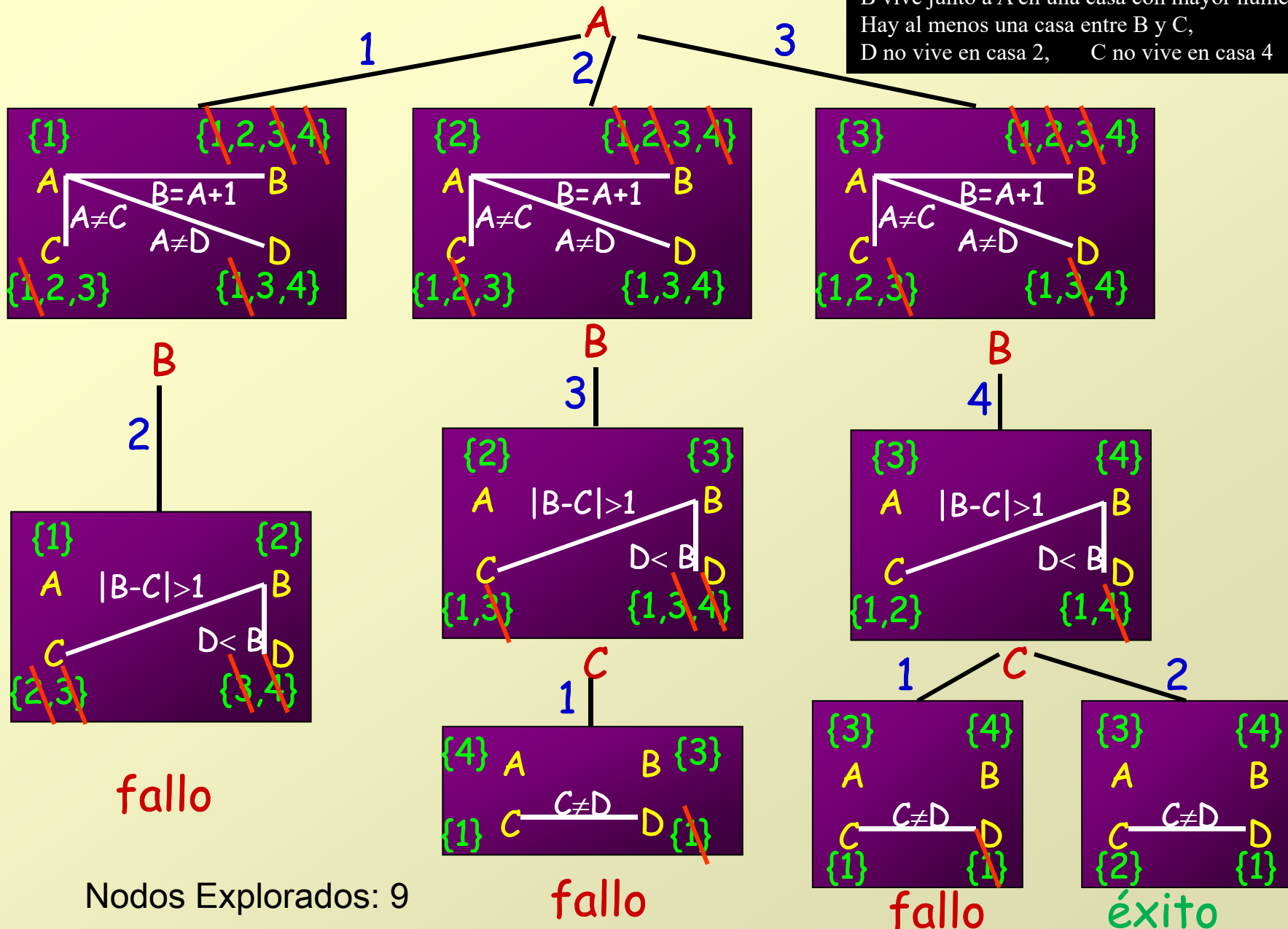
$$|B - C| > 1$$

$$C \neq D$$

¿Qué familia vive en cada casa?

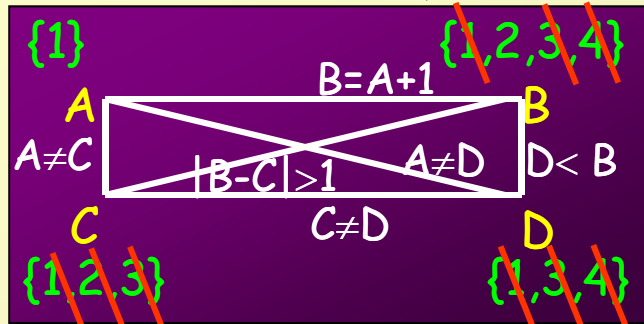
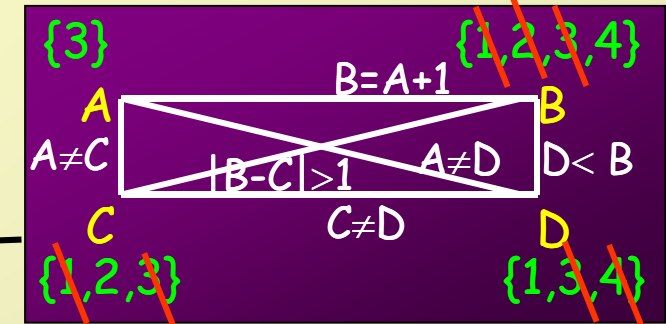
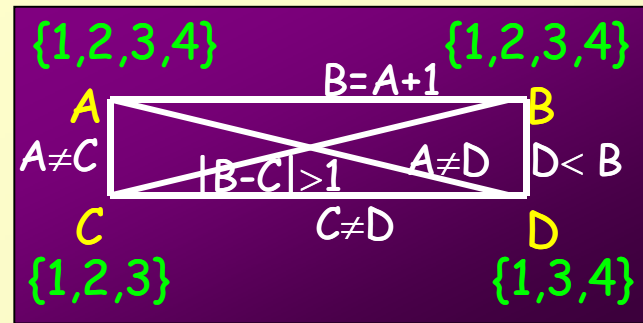
Forward checking

D vive en casa con menor número que B,
B vive junto a A en una casa con mayor número,
Hay al menos una casa entre B y C,
D no vive en casa 2, C no vive en casa 4

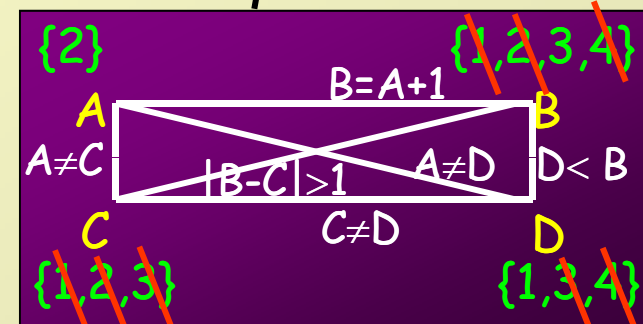


Full Look-Ahead: MAC

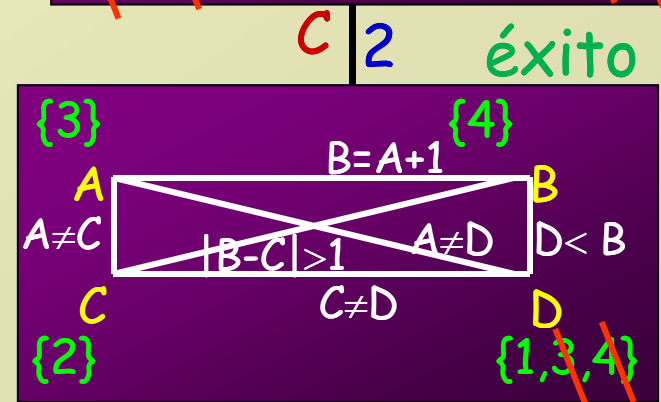
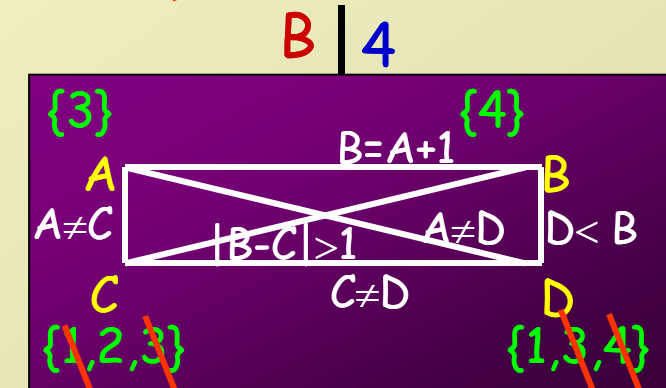
D vive en casa con menor número que B,
B vive junto a A en una casa con mayor número,
Hay al menos una casa entre B y C,
D no vive en casa 2, C no vive en casa 4



fallo



Nodos Explorados: 5 **fallo**



¿Cuál es mejor?

Forward checking:

- Arco consistencia parcial: acota dominios de acuerdo a restricciones con nodos ya instanciados.
- Lleva a cabo menos chequeos de consistencia.
- Tiene más ramificación: más próximo a backtracking.

Full Look-ahead (MAC):

- Arco consistencia total: acota dominios de acuerdo a restricciones con nodos ya instanciados y por instanciar pues '*propaga*' los valores eliminados en el proceso FC.
- Consume más tiempo en la consistencia, **pero** limita más el espacio de búsqueda.

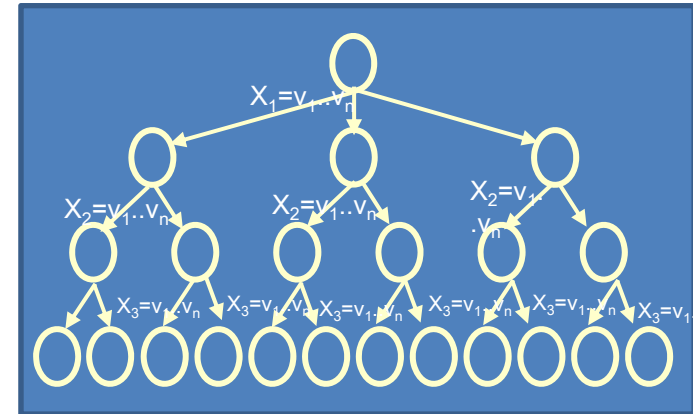
- Generalmente, Forward Checking es más útil: requiere menos tiempo en la propagación, confiando más en la heurística de búsqueda.
- Para problemas altamente restringidos / complejos, Full Look-ahead permite podar más ramas.
- Full Look-ahead es útil cuando se desea obtener más de una solución: la mayor poda realizada acota la búsqueda en la obtención de las nuevas soluciones.

Mejora de las Técnicas Básicas CSP

Las técnicas de búsqueda son los procesos básicos de resolución de un CSP.

La complejidad (exponencial: m^n) de un CSP depende de:

- Número de Variables (n),
- Tamaño de los Dominios ($m_i = |\{d_i\}|$)



Las mejoras de las técnicas básicas CSP se centran en:

1) **Incrementar el cálculo de la función 'Comprobar'** en cada instanciación:

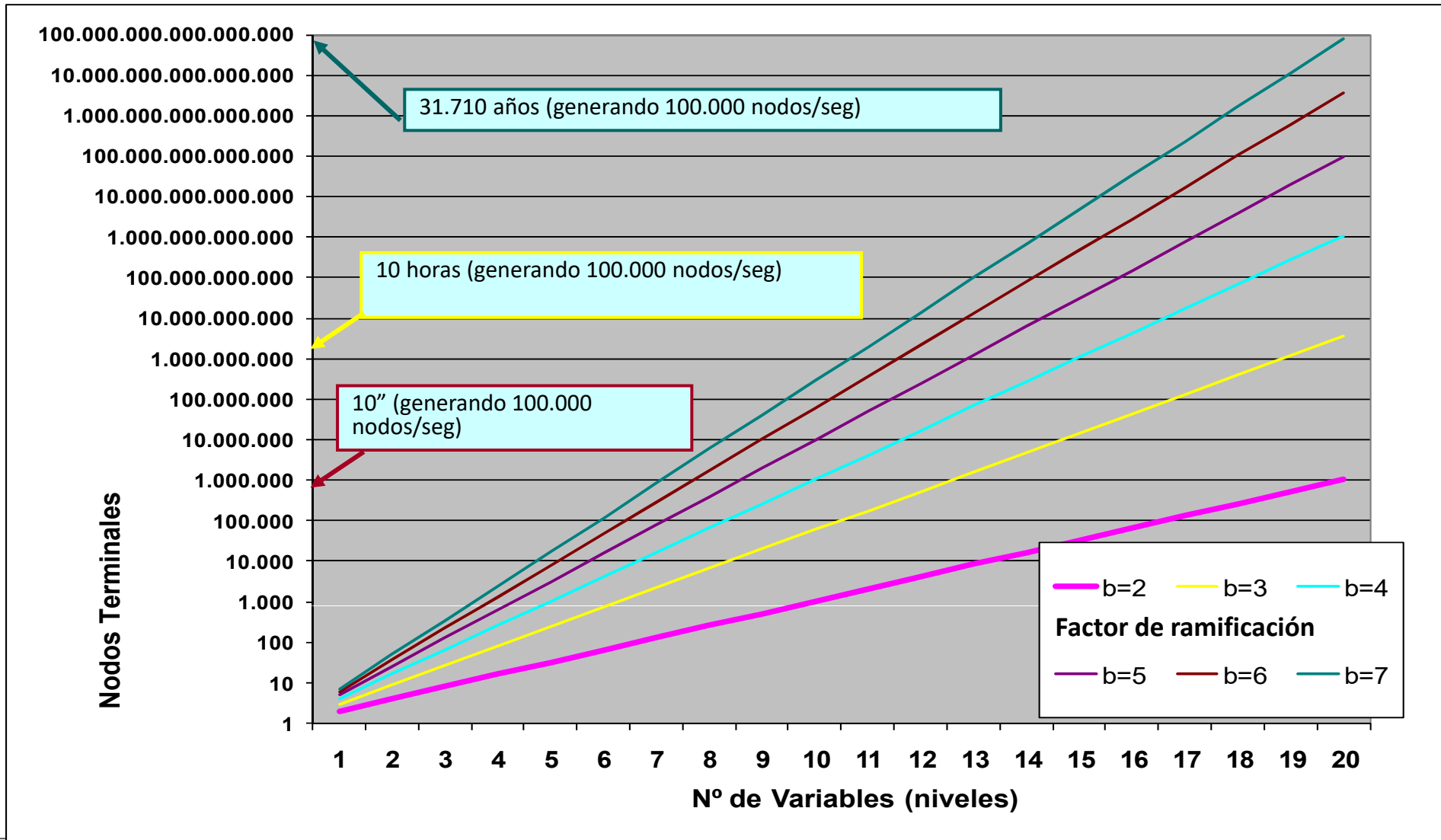
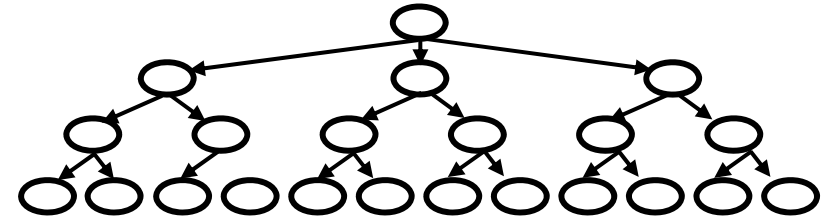
- Propagación de las instanciaciones parciales para minimizar dominios de las variables por instanciar: *Look-Forward*.

2) Técnicas de **Preproceso** para minimizar dominios iniciales de instanciación y restricciones entre variables: Métodos Inferenciales: 1, 2, 3-consistencia.

3) **Heurísticas** para la ordenación de variables/valores de instanciación

Crecimiento Exponencial de la Búsqueda

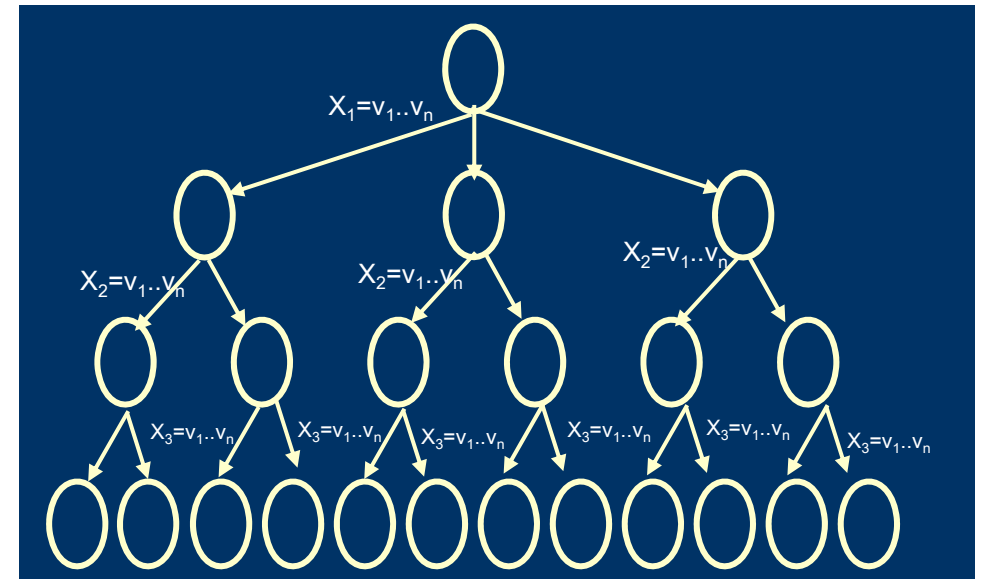
Aún con gran preproceso y búsqueda inferencial, se requieren heurísticas



Heurísticas de Búsqueda

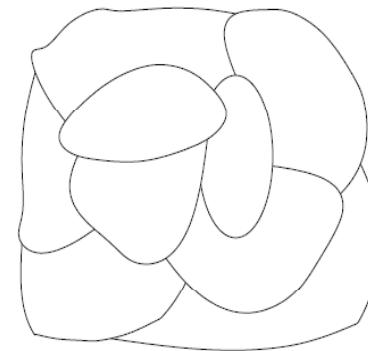
a) Heurísticas Independientes del Dominio

- **Ordenación de Variables**
¿Qué variable será la próxima a instanciar?
- **Ordenación de Valores**
Seleccionada una variable, ¿en qué orden asignar sus valores?

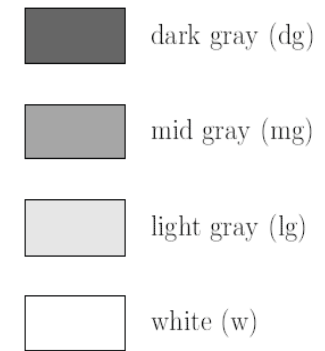


b) Heurísticas dependientes del dominio:

Orientadas a scheduling, empaquetamiento, etc.



(a) Map



(b) Colors

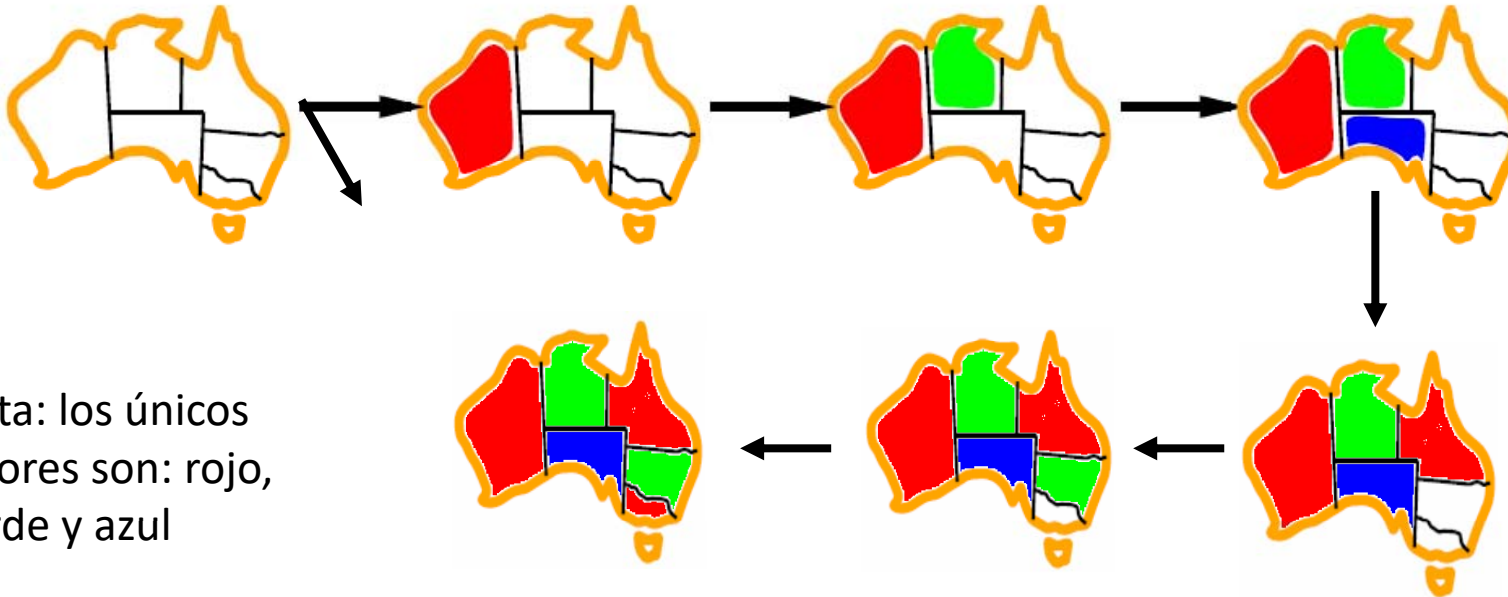
Heurísticas de Ordenación de Variables

- a) *Fija o Estática*: se determina un orden fijo para la instanciación de la variables al inicio del proceso.
 - b) *Dinámica*: el orden de instanciación de las variables se revisa en cada instanciación.
-

- **Máximo Grado (MD)**: prioriza las variables según **nº de restricciones en las que participa** (grado). *En caso de ordenación dinámica, según nº restricciones con variables no asignadas: Most Constraining Variable, MCV)*
- **Mínimo Dominio (MDV)**: se selecciona la variable con el **menor tamaño de dominio** (menor número de valores restantes en su dominio). *En caso de ordenación dinámica: Minimum Remaining Values (MRV).*
- **Máxima Cardinalidad (MC)**: selecciona la variable **conectada con el mayor número de variables ya instanciadas** (aplicable en ordenación dinámica).

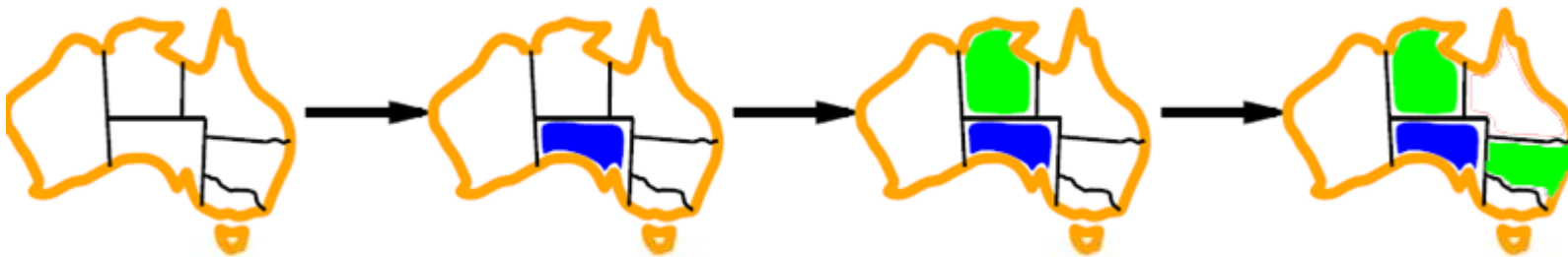
Existen otras muchas heurísticas generales de ordenación

Heurística Mínimo Dominio: (MDV) Elige la variable con **menor número de valores posibles** (\cong Variable más restringida)



Nota: los únicos colores son: rojo, verde y azul

Heurística Máximo Grado (MD): Selecciona la variable incluida en **más restricciones** con otras variables no asignadas

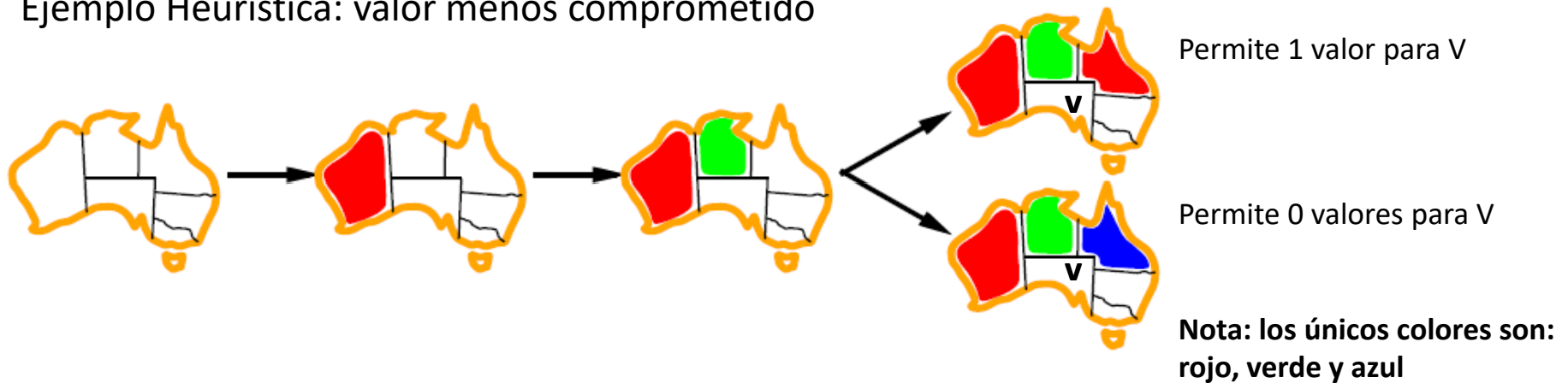


Heurísticas de Selección de Valores

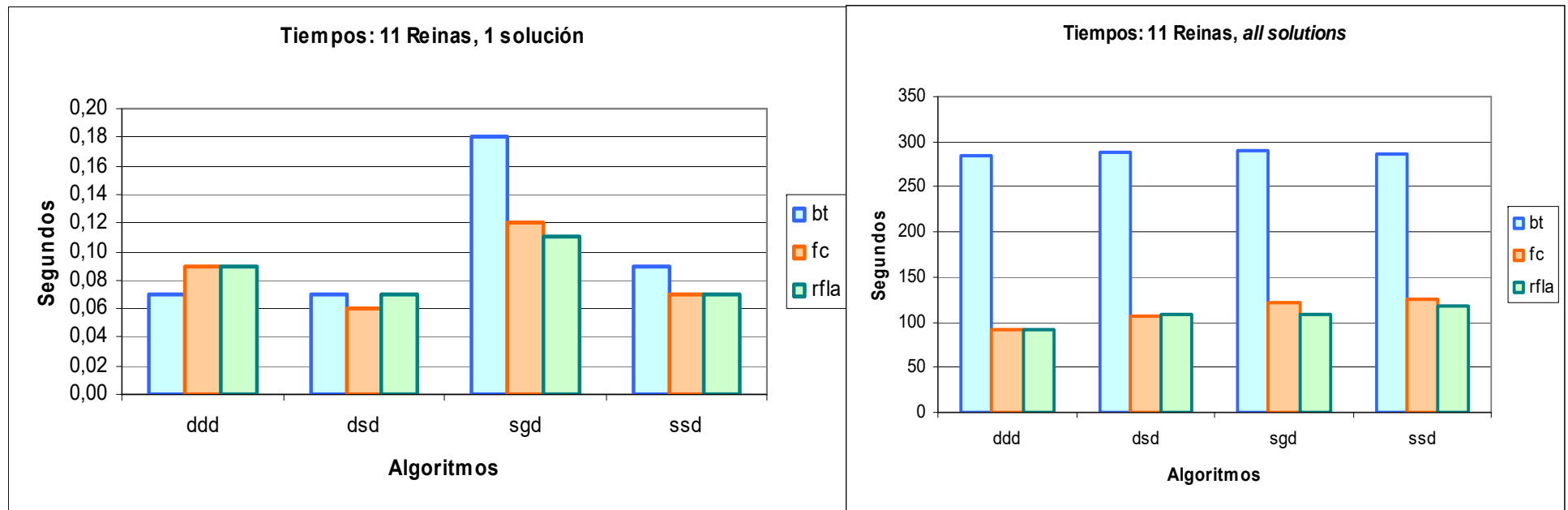
Estrategia para la elección del valor en d_i para instanciar la variable v_i .

- Elegir el **máximo/mínimo/medio** valor del dominio.
- Elegir el **valor menos comprometido** (*least constraining value*): el que menos restringe los dominios de las variables no asignadas relacionadas con la variable a instanciar (maximiza dominios y deja la máxima flexibilidad a variables no asignadas).
- Elegir el valor de **menor inconsistencia** (*mayor supervivencia*): el que es consistente con la mayor parte (%) de los valores de las variables relacionadas con la variable a instanciar.

Ejemplo Heurística: valor menos comprometido



Evaluación: 11 Reinas



Heurísticas Variables

Static Labeling Order:

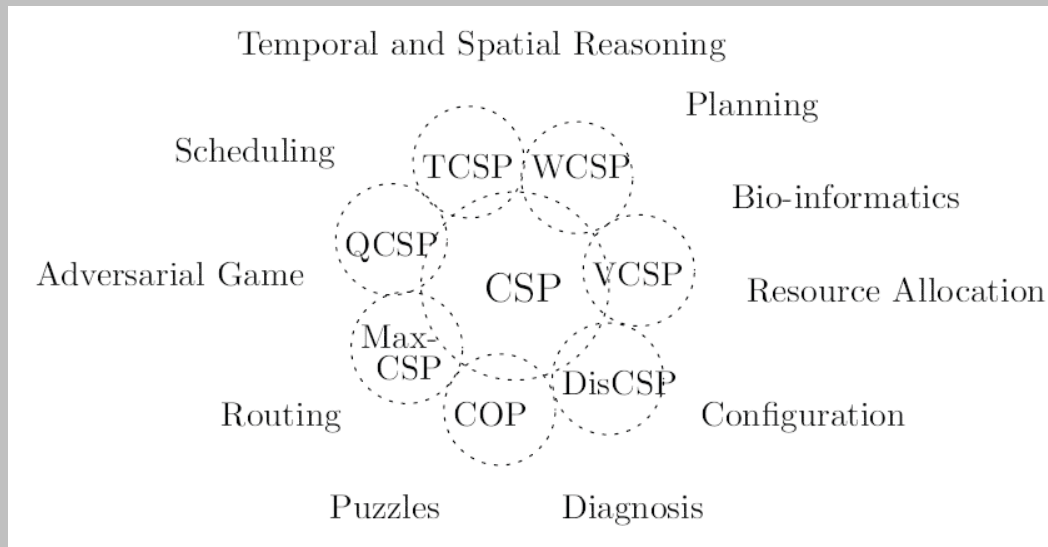
- smallest_domain (ssd): mínimo dominio
- + greatest_degree (sgd): máximo grado

Dinamyc Labeling Order:

- smallest_domain (dsd)
- smallest_domain_by_degree (ddd)

- Con todas las soluciones, las heurísticas de valores son menos relevantes.
- Con todas las soluciones, rfla, fc se vuelven más relevantes.

3.- CSPs flexibles



Temas en desarrollo:

- Expresividad,
- Eficiencia,
- Optimalidad,
- Robustez, Flexibilidad, etc.

- **Temporal CSP (TCSP)**: variables primitivas temporales (t_i, l_i, d_i), restricciones temporales.
- **Quantified CSP (QCSP)**: Obtención de soluciones para todo valor posible de (algunas) variables ($\forall x_i$), o si hay solución para un valor de (algunas) variables ($\exists x_i$).
- **Constraint Optimization Problem (COP)**: Max/minimizar una función de optimización sobre las variables
- **Valued CSP (VCSP)**: Restricciones con valor/peso asociado (utilidad en soft constraints, ponderados, fuzzy, max-CSP, preferencias, etc.): CSP Flexibles
- **Distributed CSP (DisCSP)**: Subconjuntos de variables distribuidas entre agentes.
- **Dynamic CSP (DynCSP)**: Sucesión de CSP's donde aparecen nuevas variables de forma incremental y las restricciones se van modificando (restringiendo).
- **Geometric CSP (G-CSP)**: Restricciones geométricas: distancias, volúmenes, etc.
- Etc.



Cada persona tiene sus **propias restricciones**

¿**Siempre será posible** encontrar una fecha+hora que venga bien a todo el mundo?

Posiblemente existirán **muchas soluciones parciales**, a priori todas válidas. Pero, ¿algunas podrían ser **más interesantes** que otras?

¿Estamos ante un problema de **satisfactibilidad** o de **optimización** (NP-completo vs. NP-duro)?

CSPs flexibles (valuados)

Un Problema de Satisfacción de Restricciones (CSP) es una terna:

$X = \{x_1, x_2, \dots, x_n\}$ conjunto de variables, $D = \{D_1, D_2, \dots, D_n\}$ conjunto de dominios.

$C = \{C_1, C_2, \dots, C_m\}$ conjunto de restricciones duras entre las variables

Una restricción dura (hard) es:

Imperativa: Una solución válida debe satisfacerla **siempre**.

Inflexible: La restricción es completamente satisfecha o insatisfecha: {T, F}.

Cuestiones:

- Existen muchas soluciones (*disyuntivas*), pero *¿hay soluciones mejores que otras?*
- No existen soluciones, pero... *¿algunas asignaciones satisfacen más restricciones que otras?*
- No conocemos las restricciones exactas, sino solo *aproximadas* (difusas).
- ¿Podemos *violar algunas restricciones* si a cambio pudiéramos obtener alguna solución?

Una **Restricción Blanda** (soft o preferencia) es:

✓ No imperativa: Una solución válida puede no satisfacerla.

✓ Flexible: La restricción es satisfecha en cierto grado.

¿Para que se utilizan las restricciones blandas?

Problemas sobre-restringidos (over-constrained)

Optimización: hay preferencias por unas restricciones frente a otras.

Falta de información: Las restricciones solo se conocen de forma difusa.

CSPs Flexibles: Permiten encontrar soluciones que satisfagan, en mayor o menor medida, las restricciones del problema.

CSPs flexibles => Problema de Satisfacción de Restricciones *Valuado*

Un **CSP Flexible** se diferencia del modelo clásico de CSP en que las restricciones no son SOLO relaciones lógico-aritméticas, sino que incluyen **funciones de coste** ($Coste_i$)

$$\text{Restricción}_i \equiv [\text{Relación}_i(x_1, x_2, \dots, x_n), \text{Coste}_i]$$

que expresan el **grado de satisfacción** para cada posible tupla:

La tupla $X_t = \{x_1=v_1, x_2=v_2, \dots, x_k=v_k\}$, donde $\{x_1, x_2, \dots, x_k\} \subseteq X$, tiene asociado un $Coste_j$ en la Restricción $_j$

En un CSP clásico, puede considerarse que las funciones de coste se interpretan sobre $\{True, False\}$.

El **coste de una tupla** $X_t = \{x_1=v_1, x_2=v_2, \dots, x_k=v_k\}$, sobre el **conjunto de restricciones del CSP**, resulta como la **combinación de los costes** de cada restricción $_i$ ($i=1..m$) en la que participan sus variables.

La combinación aplica un operador \oplus (asociativo y conmutativo):

$$\text{Coste}(X_t = \{x_1=v_1, x_2=v_2, \dots, x_k=v_k\}) = \text{Coste}_1 \oplus \text{Coste}_2 \oplus \dots \oplus \text{Coste}_m$$

Las diferentes funciones de coste \oplus da lugar a los diferentes CSP flexibles.

- *En un CSP clásico puede considerarse que $u \oplus v = u \wedge v$.*
- **En un CSP flexible, el objetivo es encontrar** la asignación X_t con el mejor coste combinado (maximización o minimización).
Puede definirse un mínimo/máximo valor de coste admisible para que sea solución. Da lugar a un problema de optimización (**Complejidad NP-hard**)

Principales tipos de CSPs flexibles

CSPs Posibilista: Permite que **algunas restricciones no se cumplan**, lo que conlleva un coste de insatisfabilidad.

Restricción $\text{Soft}_k \Rightarrow \text{Coste}_k$ asociado a su Insatisfabilidad

El objetivo es obtener una solución que **minimice el coste de las restricciones no satisfechas**.

CSPs Ponderados (Weighted CSPs): Existen restricciones disyuntivas ponderadas (o valores posibles de las variables). Cada disyunción tiene un coste asociado a su satisfactibilidad.

Restricción Soft $\Rightarrow C_1 (\text{Coste}_1) \vee C_2 (\text{Coste}_2) \vee \dots \vee C_k (\text{Coste}_k)$

El objetivo es obtener una solución que **minimice el coste de las restricciones satisfechas**.

CSPs Probabilistas: Caso similar a CSP posibilista, permitiendo que algunas restricciones no se satisfagan, pero los costes representan la **probabilidad** de la restricción (*difiere en la función de combinación*).

El objetivo es obtener una solución que **minimice el coste de las restricciones no satisfechas**.

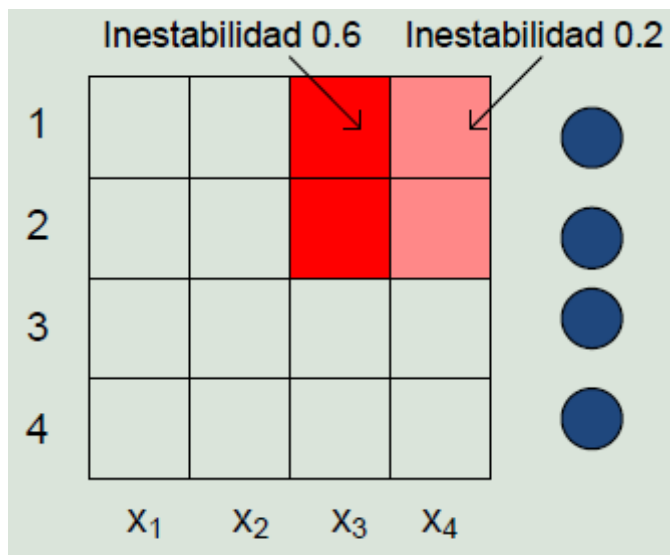
CSPs Difusos: Las restricciones son **difusas**, tal que no se interpretan a $\{T, F\}$, sino que son definidas como una función continua $[0, 1]$ de sus variables.

El objetivo es obtener una solución que **maximice la satisfacción de las restricciones** (*valores próximos al núcleo*)

Tipo	Función de coste Ci	Coste (Xt= {x1=v1, ...,xk=vk})	OBJETIVO						
CSP Posibilista Restricciones soft [Ri, Ci] que pueden no cumplirse.	Ci ∈[0,1] Asociada a la insatisfabilidad de Ri	Coste (Xt) = max (C1, C2, ...Cm) <i>Sobre el conjunto de restricciones {R1, R2, ...Rm} no satisfechas</i>	Minimizar coste (insatisfacción de las restricciones soft)						
CSP ponderado Restricciones con disyunciones, con un coste asociado de satisfabilidad	Ci ∈[0, P] Asociada al coste de satisfabilidad de cada disyunción	Coste (Xt) = (C1 + C2 ++ Cm) <i>Sobre el conjunto de disyunciones satisfechas (de cada restricción disyuntiva)</i>	Minimizar coste (conjunto de disyunciones satisfechas)						
CSP probabilístico Restricciones soft [Ri, Ci] con una probabilidad de ocurrencia	Ci ∈[0,1] Asociada a la probabilidad de Ri	Coste (Xt) = 1 – ((1-C1) * (1-C2) * * (1-Cm)) <i>Sobre el conjunto de restricciones {R1, R2, ...Rm} no satisfechas</i>	Minimizar coste (probabilidad de que Xt no sea solución)						
CSP difuso Restricciones difusas, definidas como una función continua [0, 1] sobre sus variables, que representa su grado de satisfabilidad	Ci: μ(x1, ...,xk) ∈[0,1] Ejemplo: a >> b / a,b∈{0,100}, <table> <tr> <td>0,</td> <td>si a ≤ b</td> </tr> <tr> <td>(a-b)/10</td> <td>si b<a<b+10</td> </tr> <tr> <td>1</td> <td>si a ≥ b+10</td> </tr> </table>	0,	si a ≤ b	(a-b)/10	si b<a<b+10	1	si a ≥ b+10	Coste (Xt) = min (C1 + C2 +....+ Cm) <i>Mínimo sobre el conjunto de restricciones (≈AND difuso).</i>	Maximizar coste (grado de satisfactibilidad).
0,	si a ≤ b								
(a-b)/10	si b<a<b+10								
1	si a ≥ b+10								

Ejemplo CSP posibilista

Los pesos indican costes de insatisfacción



Costes de Insatisfacción (restricciones soft)

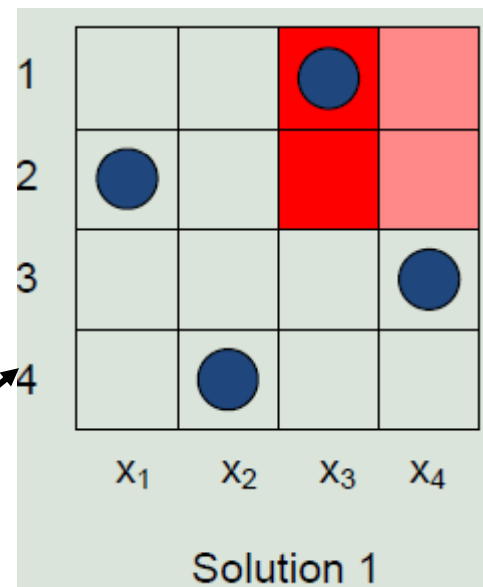
($x_3 \neq 1$), 0.6

($x_3 \neq 2$), 0.6

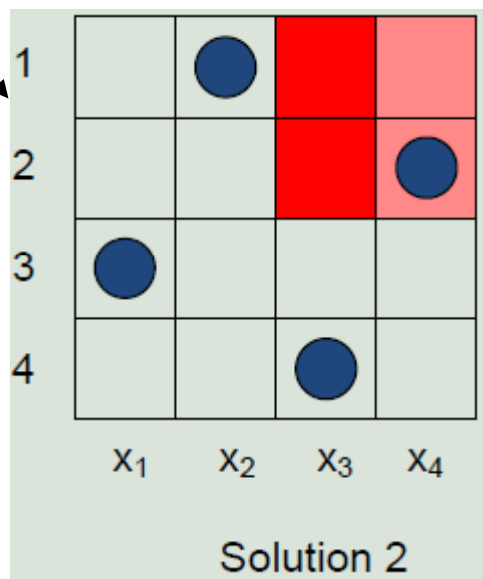
($x_4 \neq 1$), 0.2

($x_4 \neq 2$), 0.2

Adicionalmente, existen las restricciones hard relativas a la no amenaza mutua de las reinas.



Insatisf=0.6



Insatisf=0.2

mejor

Tipo	Función de coste Ci	Coste (X _t = {x ₁ =v ₁ , ...,x _k =v _k })	OBJETIVO
CSP Posibilista Restricciones soft [R _i , C _i] que pueden no cumplirse.	Ci ∈[0,1] Asociada a la insatisfabilidad de R _i	Coste (X_t) = max (C₁, C₂, ...C_m) <i>Sobre el conjunto de restricciones {R₁, R₂, ...R_m} no satisfechas</i>	Minimizar coste (insatisfacción de las restricciones soft)
CSP ponderado Restricciones con disyunciones, con un coste asociado de satisfabilidad	Ci ∈[0, P] Asociada al coste de satisfabilidad de cada disyunción	Coste (X_t) = (C₁ + C₂ ++ C_m) <i>Sobre el conjunto de disyunciones satisfechas (de cada restricción disyuntiva)</i>	Minimizar coste (conjunto de disyunciones satisfechas)
CSP probabilístico Restricciones soft [R _i , C _i] con una probabilidad de ocurrencia	Ci ∈[0,1] Asociada a la probabilidad de R _i	Coste (X_t) = 1 – ((1-C₁) * (1-C₂) * * (1-C_m)) <i>Sobre el conjunto de restricciones {R₁, R₂, ...R_m} no satisfechas</i>	Minimizar coste (probabilidad de que X _t no sea solución)
CSP difuso Restricciones difusas, definidas como una función continua [0, 1] sobre sus variables, que representa su grado de satisfabilidad	Ci: μ(x₁, ...,x_k) ∈[0,1] Ejemplo: a >> b / a,b∈{0,100}, 0, si a ≤ b (a-b)/10 si b<a<b+10 1 si a ≥ b+10	Coste (X_t) = min (C₁ + C₂ +....+ C_m) <i>Mínimo sobre el conjunto de restricciones (≈AND difuso).</i>	Maximizar coste (grado de satisfactibilidad).

Ejemplo CSP ponderado

Los pesos indican costes de las celdas

1		20	13		
2					
3	11			8	
4					
	x_1	x_2	x_3	x_4	

Costes:

$[x_1 = 3, 11] \vee [x_1 = 1, 0] \vee [x_1 = 2, 0] \vee [x_1 = 4, 0]$

$[x_2 = 1, 20] \vee [x_2 = 2, 0] \vee [x_2 = 3, 0] \vee [x_2 = 4, 0]$

$[x_3 = 1, 13] \vee [x_3 = 2, 0] \vee [x_3 = 3, 0] \vee [x_3 = 4, 0]$

$[x_4 = 3, 8] \vee [x_4 = 1, 0] \vee [x_4 = 2, 0] \vee [x_4 = 4, 0]$

1		20	13	
2				
3	11			8
4				
	x_1	x_2	x_3	x_4

Solution 1

Coste=21

mejor

1		20	13	
2				
3	11			8
4				
	x_1	x_2	x_3	x_4

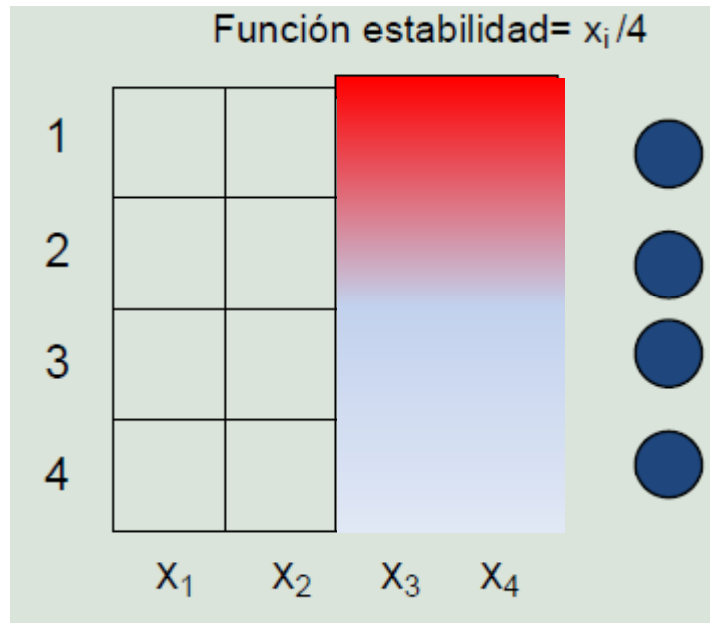
Solution 2

Coste=31

Tipo	Función de coste Ci	Coste (Xt= {x1=v1, ...,xk=vk})	OBJETIVO
CSP Posibilista Restricciones soft [Ri, Ci] que pueden no cumplirse.	Ci ∈[0,1] Asociada a la insatisfabilidad de Ri	Coste (Xt) = max (C1, C2, ...Cm) <i>Sobre el conjunto de restricciones {R1, R2, ...Rm} no satisfechas</i>	Minimizar coste (insatisfacción de las restricciones soft)
CSP ponderado Restricciones con disyunciones, con un coste asociado de satisfabilidad	Ci ∈[0, P] Asociada al coste de satisfabilidad de cada disyunción	Coste (Xt) = (C1 + C2 ++ Cm) <i>Sobre el conjunto de disyunciones satisfechas (de cada restricción disyuntiva)</i>	Minimizar coste (conjunto de disyunciones satisfechas)
CSP probabilístico Restricciones soft [Ri, Ci] con una probabilidad de ocurrencia	Ci ∈[0,1] Asociada a la probabilidad de Ri	Coste (Xt) = 1 – ((1-C1) * (1-C2) * * (1-Cm)) <i>Sobre el conjunto de restricciones {R1, R2, ...Rm} no satisfechas</i>	Minimizar coste (probabilidad de que Xt no sea solución)
CSP difuso Restricciones difusas, definidas como una función continua [0, 1] sobre sus variables, que representa su grado de satisfabilidad	Ci: μ(x1, ...,xk) ∈[0,1] Ejemplo: a >> b / a,b∈{0,100}, 0, si a ≤ b (a-b)/10 si b<a<b+10 1 si a ≥ b+10	Coste (Xt) = min (C1 + C2 +....+ Cm) <i>Mínimo sobre el conjunto de restricciones (≈AND difuso).</i>	Maximizar coste (grado de satisfactibilidad).

Ejemplo CSP-difuso

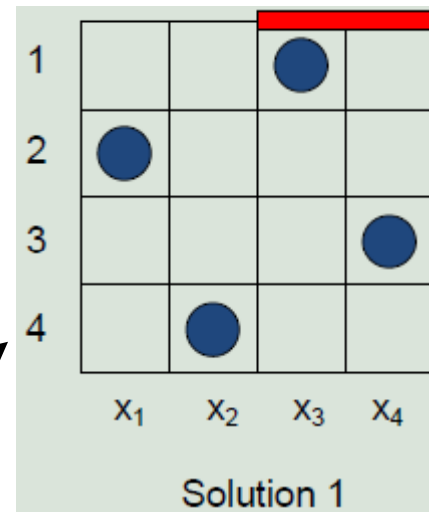
Preferencia a alejar x_3 y x_4 de la primera fila



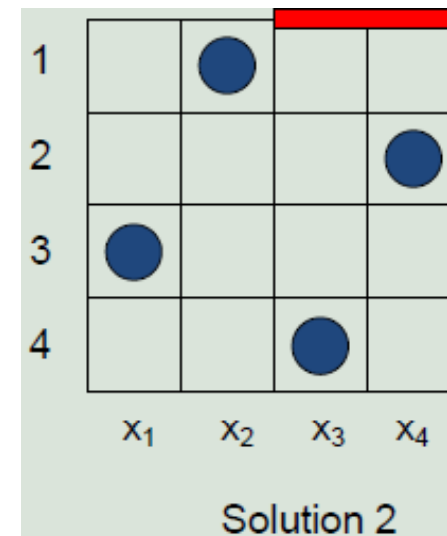
La función de estabilidad se asocia a los valores de x_3 y x_4 , y penaliza los valores cercanos a la primera fila.

$$x_3 \gg 0 \text{ / } x_3 \in \{1, 4\}, \quad \mu = x_3/4$$

$$x_4 \gg 0 \text{ / } x_4 \in \{1, 4\}, \quad \mu = x_4/4$$



$$\text{Coste} = \min(1/4, 3/4) = 1/4$$



$$\text{Coste} = \min(4/4, 2/4) = 2/4$$

mejor

Métodos de Resolución de los CSP Valuados

Objetivo:

Buscar una solución que optimice su evaluación/coste (*minimice la violación de restricciones, minimice su coste, o maximice su satisfacción*).

Mejor solución para el CSP Flexible:

- **CSP Posibilístico / CSP Probabilístico:** Solución que minimiza el coste de insatisfacción de restricciones.
- **CSP Ponderado:** Solución que minimiza el coste acumulado de las disyunciones que se cumplen.
- **CSP Difuso:** Solución con la máxima satisfacción de restricciones.

Se suelen emplear algoritmos CSP combinados con Ramificación y Poda:

- En cada paso, las asignaciones parciales son evaluadas por la función de poda.
- El mejor valor de la función para la cada solución se almacena como el mejor coste hasta el momento.
- Si la función de evaluación, aplicada sobre una asignación parcial, es peor que el mejor coste almacenado hasta el momento, se poda la rama.
- RETOS: Eficiencia, Robustez, Expresividad

CSP Valuados \Rightarrow Problema de Optimización

Ejemplo en MiniZinc. Priorizar soluciones donde X7, X8 estén alejadas de la primera columna

% N-Reinas, priorizando x7, x8 alejadas primera columna

int: n;

array [1..n] of var 1..n: q; % queen is column i is in row q[i]

include "alldifferent.mzn";

% Restricciones

% No en misma columna

constraint forall (i,j in 1..n where i!=j) (q[i] != q[j]);

% No en misma diagonal SE

constraint forall (i,j in 1..n where i!=j) ((q[i] - q[j]) != (i - j));

% No en misma diagonal SO

constraint forall (i,j in 1..n where i!=j) ((q[j] - q[i]) != (i - j));

% search

solve maximize (q[7]+q[8]);

output [(show(q[i]) ++ " ") | i in 1..n];

Running nreinas-opt.mzn

4 2 7 3 6 8 5 1

5 2 4 7 3 8 6 1

3 5 2 8 6 4 7 1

4 6 1 5 2 8 3 7

4 8 1 3 6 2 7 5

7 2 6 3 1 4 8 5

3 5 7 1 4 2 8 6

=====

X1			●					
X2					●			
X3							●	
X4	●							
X5				●				
X6		●						
X7								●
X8					●			

+ inestabilidad

Conclusiones

Los CSPs nos permiten abordar los problemas de forma distinta a la tradicional

- En lugar de pensar en la forma de **resolver** el problema pensamos en la forma de **modelarlo** (vía variables, dominios y restricciones) y la forma de las soluciones
Ej: Sudoku, en el que no sé cómo lo resolveré pero sí sé la forma (restricciones) que debe tener una solución válida

Existen distintas técnicas para resolver un CSP

- Aplicar técnicas de **inferencia**, garantizando distintos niveles de consistencia → pueden detectar si no existe solución, e incluso encontrarla, sin necesidad de aplicar búsqueda
- **Búsqueda**
 - Existen distintas aproximaciones: bt, fc, rfla...
 - Con heurísticas sobre variables y valores para incrementar la eficiencia
- Nos puede interesar encontrar una solución arbitraria (**satisfactibilidad**) o la mejor con respecto a una métrica (**optimalidad**, obviamente de forma más costosa)

Se pueden relajar ciertas restricciones (soft) y trabajar con CSPs flexibles bajo la idea de optimización

- CSPs posibilistas, ponderados, probabilísticos, difusos, etc.

Se aplican en multitud de entornos problemas

- Además existen multitud de **herramientas, librerías (libres y gratuitas) y entornos de resolución**