Tema 6.- Problemas de Satisfacción de Restricciones (CSP)

"Constraint Satisfaction, a simple but powerful idea" R. Dechter

Modelizacion: Conceptos y Modelos.

- ¿Qué es un Problema de Satisfacción de Restricciones?
- ¿Cómo se modela? Tipología de Restricciones

Resolución: Técnicas

- ¿Cómo se soluciona? Operativa.
- Técnicas Inferenciales: Nueva información. Niveles.
- Técnicas de Búsqueda. Heurísticas. Técnicas Híbridas: Métodos Looking Ahead.

CSP Flexibles (valuados).

Entornos y Aplicaciones.

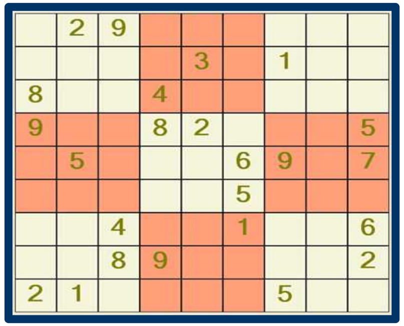
Bibliografía

- *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.- 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.



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

- Problemas de Empaquetamiento, cadenas montaje,
- Problemas de Rutas, transporte, logística,
- Problemas de Scheduling, compartición y asignación de recursos,
- Problemas de Razonamiento Temporal,
- Sistemas de Documentación, Gestión de Redes,
- Diseño, Planificación, Control, etc.

Criptografía

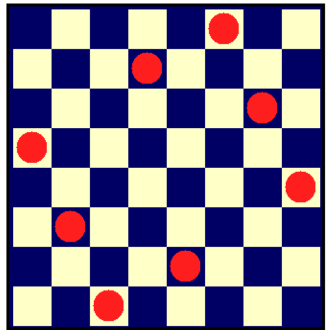
- Variables: s,e,n,d,m,o,r,y
- Dominios: s,e,n,d,m,o,r,y$\in${0,…,9}
- Restricciones

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

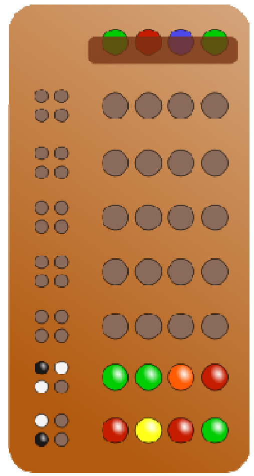
$$10^3(s+m)+10^2(e+o)+10(n+r)+d+e=10^4m+10^3o+10^2n+10e+y$$

Objetivo:

Solución (asignación consistente)



8 Reinas

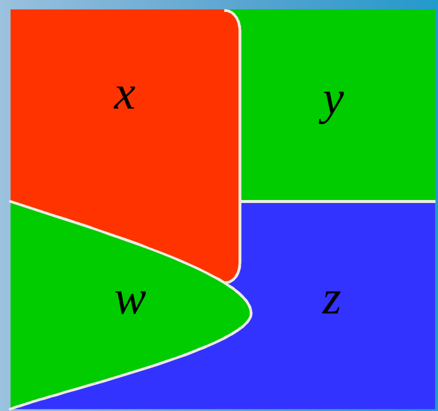


Mastermind

Coloreado de Mapas (Mapa de frecuencias)

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

  $x \neq y,\ y \neq z,\ z \neq x,\ \ldots$



Sudoku

Ejemplo 2

Lectores y Periódicos

	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

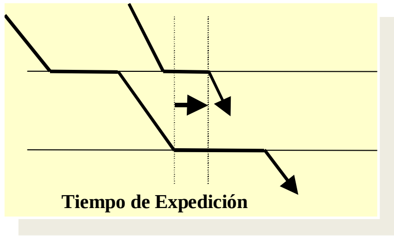
"Juan va de su casa al trabajo en coche (30-40 minutos) o en tren (al menos una hora). Luis va en coche (20-30 minutos) o en metro (40-50 minutos)."

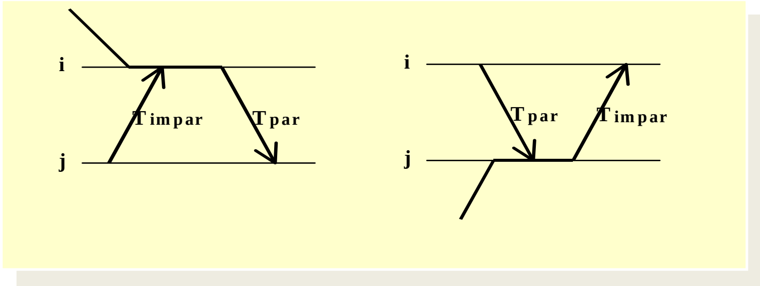
"Hoy Juan parte de casa entre las 8:10 y las 8:20, y Luis llega al trabajo entre las 9:00 y las 9:10. Además, sabemos que Juan llegó al trabajo entre 10 y 20 minutos después de que Luis saliera de casa"

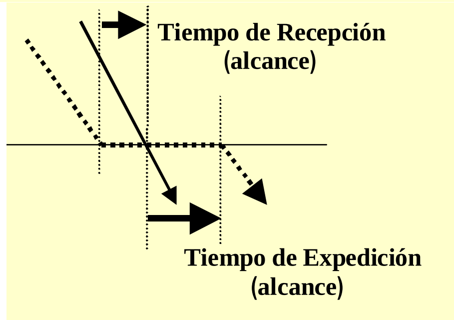
Cuestiones:

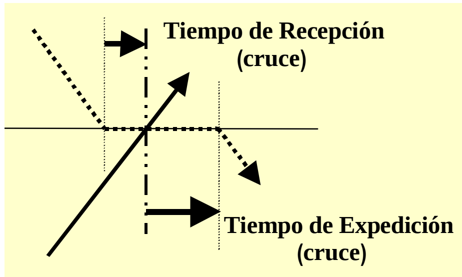
- ¿Esta información es consistente?
- ¿Es posible que Juan haya usado el tren y Luis haya usado el Metro?
- ¿Cuáles son los posibles tiempos en los que Luis pudo haber salido de casa?
- ¿Cuánto dura el viaje de Juan en tren?
- etc.

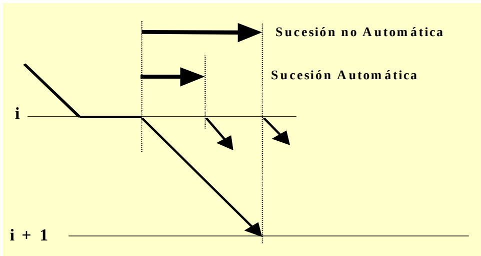
Ejemplo 4. Resolución de problemas reales (logística)

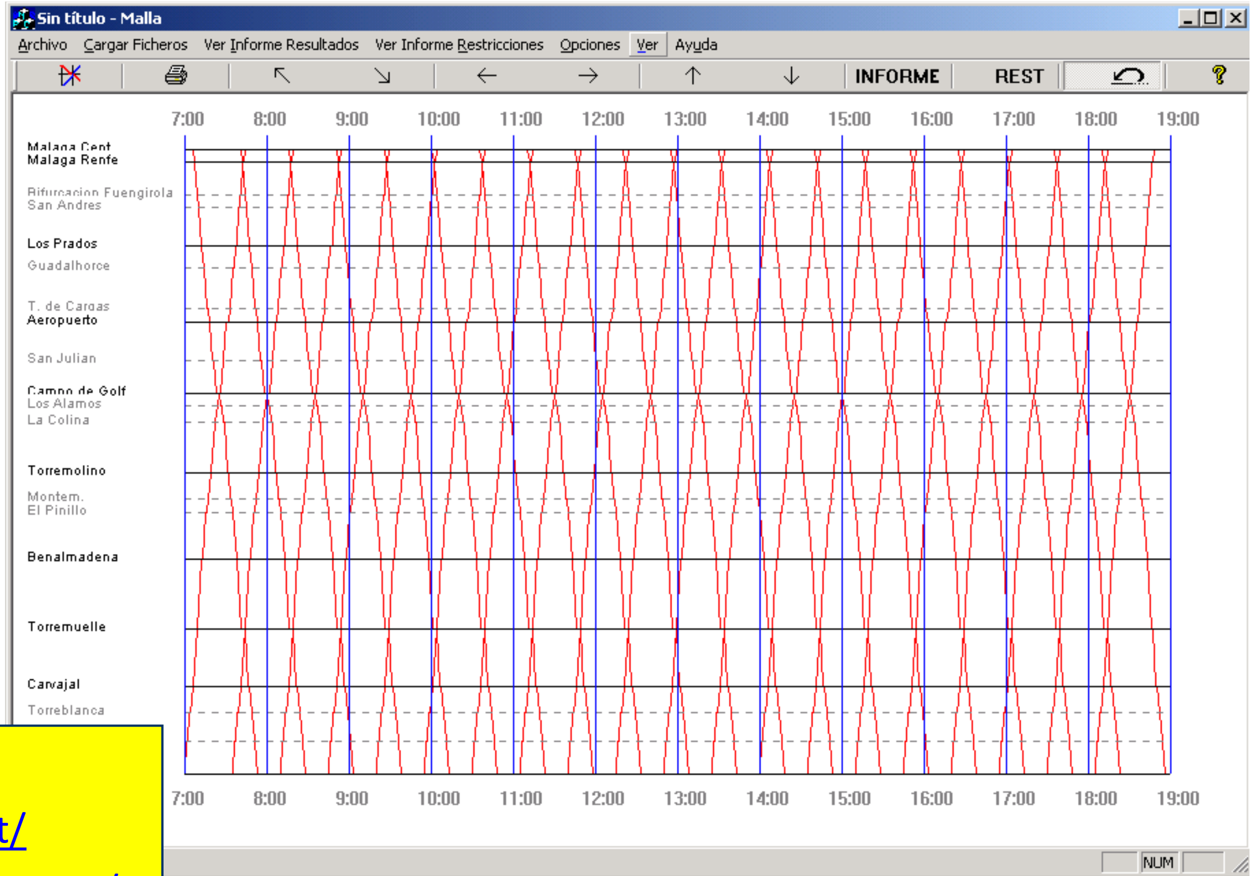












Otras demos en web:

- http://aispace.org/constraint/
- http://www.constraintsolving.com/

¿Qué es lo que queremos?

PROBLEMA

Modelado

Representación (CSP)

Búsqueda + Heurística

Inferencia

Encontrar una o más soluciones que cumplan el conjunto de restricciones

Obtener conclusiones o respuestas a preguntas, como consecuencia de las restricciones del problema.

Constraint Solving

Constraint programming is a programming paradigm where relations between variables can be stated in the form of constraints.

Constraints differ from the common primitives of other programming languages in that they do not specify a step or sequence of steps to execute but rather the properties of a solution to be found.

Problemas de Satisfacción de Restricciones CSP

Metodología de Resolución de problemas
INTELIGENCIA ARTIFICIAL

2.- Definición de un CSP.

$$CSP \equiv (X, D, C)$$

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

1) Conjunto de **Variables**: $\mathbf{X} = \{x_1, x_2, ..., x_n\}$

2) **Dominios** de Interpretación para las variables: $\mathbf{D} = \{D_1,...,D_n\} \,/\, x_i \in D_i$

3) **Conjunto de Restricciones** entre las variables: $\mathbf{C} = \{c_1, c_2, ..., c_m\}$

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

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

Tipología de los CSP's

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

a) Tipología de las Dominios $<D_1,\ldots,D_n>$:

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.

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

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 (asumimos restricciones matemáticas, de tipo $\leq$)

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

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

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

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

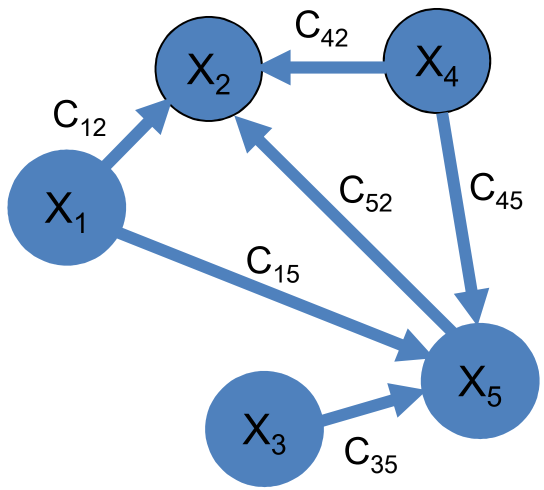
  Ejemplo: $a \leq p_1 x_1 + p_2 x_2 + p_3 x_3 + \ldots + p_n x_n \leq b$, $a, b, p_k \in Z$, $k:1,n$

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

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

- 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



Restricciones Cualitativas / Métricas

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

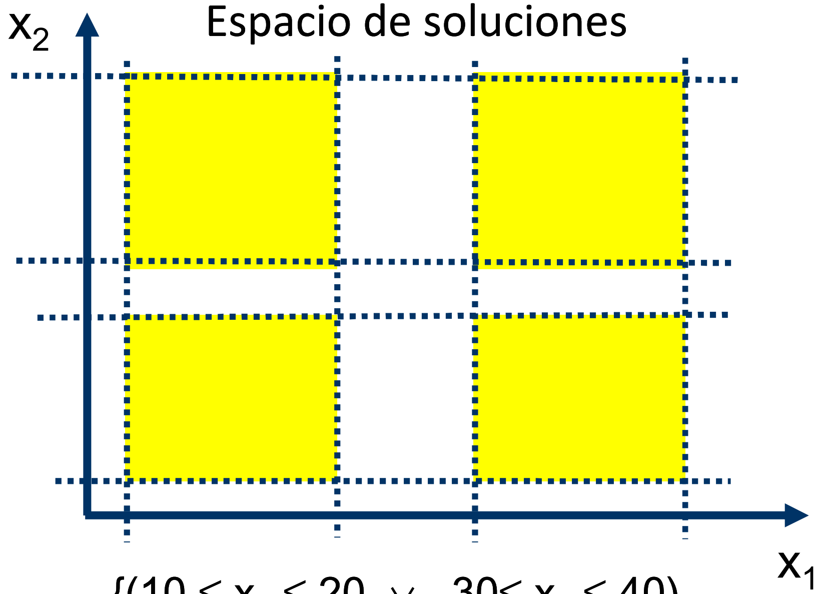
Restricciones Disyuntivas / No-disyuntivas

(dominio de soluciones convexo o no convexo)

Restricciones Disyuntivas:

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

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



$\{(10 \leq x_1 \leq 20 \vee 30 \leq x_1 \leq 40),$
$(10 \leq x_2 \leq 20 \vee 30 \leq x_2 \leq 40)\}$

Restricciones no-disyuntivas:

- Problema Simple *(Polinomial)*

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

Algunos Ejemplos especificación CSP

CSP n-ario, discreto, disyuntivo

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

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),

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

Como alternativa (más eficiente):

y' = d + e;	y = mod (y', 10);
e' = n + r + int (y'/10);	e = mod (e', 10);
n'= e + o + int (e'/10);	n= mod (n', 10);
o' = s + m + int (n'/10);	o = mod (o', 10);
	m = int (o'/10) ;

O incluso:

d + e = y + 10*C1;
C1 + n + r = e + 10*C2;
C2 + o + e = n + 10*C3;
C3 + s + m = o + 10*m;

"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:

$8{:}10 \le T1 \le 8{:}20$

$30 \le T2\text{-}T1 \le 40 \quad \vee \quad 60 \le T2\text{-}T1$

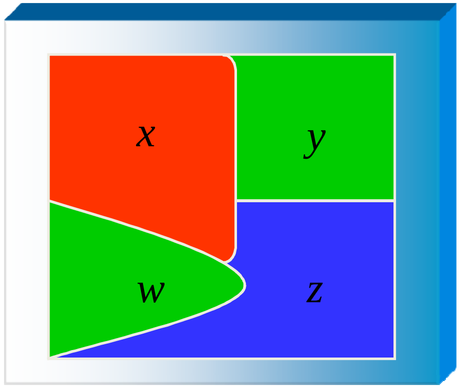
$10 \le T2\text{-}T3 \le 20$

$9{:}00 \le T4 \le 9{:}10$

$20 \le T4\text{-}T3 \le 30 \quad \vee \quad 40 \le T4\text{-}T3 \le 50$

CSP binario, continuo/discreto, disyuntivo

Coloreado de Mapas



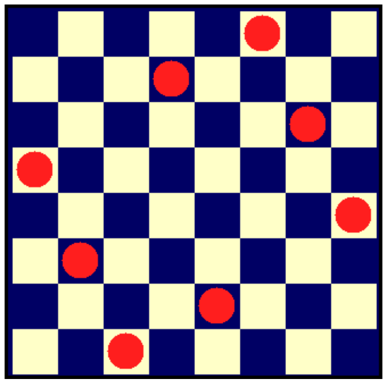
- Variables: x,y,z,w
- Dominios: x,y,z,w : {r,v,a}
- Restricciones (binarias):

$x \neq y$, $x \neq z$,
$x \neq w$, $y \neq z$,
$w \neq z$

CSP binario, cualitativo, disyuntivo y discreto

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

Las 8 Reinas



Con n = 8…

Variables {xi}, indicando la posición en la fila i.

Dominio = {1, 2, 3 … , n}

Restricciones:

$\forall$ xi, xj, i≠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 viv 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?

?

Problema a resolver (How to Solve It: Modern Heuristics. Z. Michalewicz, D. Fogel. 2ed. 2004 (Springer)

Dos matemáticos se encuentran y se ponen a hablar. Uno le dice a otro:

> *"Hoy es un día especial. ¡Mi tres hijos celebran su cumpleaños este mismo día!*
>
> *El producto de sus edades es 36 y la suma es igual al número de ventanas del edificio de enfrente. ¿Sabrías decirme sus edades?"*

El amigo, tras ver las ventanas (13) y pensar un poco, le contesta: *"Me falta un dato"*

\- *"Cierto"*, le contesta. *"Mi hijo mayor tiene los ojos azules, como su madre."*

¿Puedes obtener las edades de sus hijos?

Esto indica que hay más de una combinación de edades que cumplen las restricciones.

Y es suficiente conocer que hay un hijo 'mayor' que el resto, lo que no se cumple en las otras combinaciones.

Otros tipos de restricciones

- *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.

Modelando y resolviendo CSPs

CON'FLEX (ms-dos) (https://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/AlgorithmS)

```
##### c4q.csp = four queens problem

############################################
###   Requetes, Parametres et Options  ###
############################################
```

Configuración del resolvedor

```
\alpha = 0.1;
\filtering : f ;
\search : rfla ,
          all_solutions
#         best_solution
#         first_solutions 5
;
\static_labeling_order  :
          smallest_domain
#         greatest_degree
#         smallest_domain_by_degree
;
#\dynamic_labeling_order :
#         smallest_domain
#         smallest_domain_by_degree
#
```

```
\verbose :
          display_solutions ;
#         display_csp
#         display_filtering
```

Dominios, Variables y Restricciones

```
###############################
###      VARIABLES       ###
###############################

\vi : Z1,Z2,Z3,Z4 1..4 ;

###############################
###     CONTRAINTES      ###
###############################

\ci : rd1 , abs (Z1 - Z2) != 1 ;
\ci : rd2 , abs (Z1 - Z3) != 2 ;
\ci : rd3 , abs (Z1 - Z4) != 3 ;
\ci : rd4 , abs (Z2 - Z3) != 1 ;
\ci : rd5 , abs (Z2 - Z4) != 2 ;
\ci : rd6 , abs (Z3 - Z4) != 1 ;

\cim : ct1 , <>(Z1,Z2,Z3,Z4);
```

CON’FLEX. Resultado de la ejecución

Símbolo del sistema

```
C:\>conflex "4queen.csp"

Lecture du fichier... "4queen.csp"
  ... OK

###### Filtering "superficiel" (AC pour les variables entieres) ...

###### Fin filtering du CSP  ######

############  Resolution par Real Full Look Ahead  ############

  - Recherche de toutes les solutions satisfaisantes au moins α 0.090
  - Tri prΘalable des variables, du plus petit domaine au plus grand.
  - Ordre d'examen des valeurs numΘriques : de la plus petite α la plus grande.
----------------------------------------------------


SOLUTION No 1
   Z1 = 2   Z2 = 4   Z3 = 1   Z4 = 3  sat = 1.000 .
  (trouvee apres 5 instanciations et 12 tests de contraintes)

SOLUTION No 2
   Z1 = 3   Z2 = 1   Z3 = 4   Z4 = 2  sat = 1.000 .
  (trouvee apres 9 instanciations et 24 tests de contraintes)

############   Fin du Real Full Look Ahead ############
   Nombre de solution(s) trouvee(s) : 2
   Nb d'instanciations : 10, Nb de tests de contraintes : 24

durΘe de la rΘsolution : 0'00''00

C:\>
```

Ejemplo: **Supongamos un distribuidor automático de bebidas. El cliente inserta monedas por un total de T céntimos de euro y selecciona una bebida cuyo precio es de P céntimos de euro (P y T son múltiplos de 10).**

Calculad las monedas a devolver suponiendo que el distribuidor tiene una reserva de E2 piezas de 2 €, E1 piezas de 1 €, C50 piezas de 50 céntimos, C20 piezas de 20 céntimos y C10 piezas de 10 céntimos.

Formalización:

X = {XE2, XE1, XC50, XC20, XC10}

D = *D(XE2) = {0,1,...,E2}*
D(XE1) = {0,1,...,E1}
D(XC50) = {0,1,...,C50}
D(XC20) = {0,1,...,C20}
D(XC10) = {0,1,...,C10}

*C = { 200*XE2 + 100*XE1 +*
*50*XC50 + 20*XC20 + 10*XC10*
= T - P }

```
##### ejemplo
\filtering : f ;
\search : rfla , first_solutions 5;
\static_labeling_order  : smallest_domain ;
\value_order :  bottom_first;
\verbose :  display_solutions;

###      VARIABLES       ###
\vi : XE2, XE1, XC50, XC20, XC10  0..50 ;
\vi : T 1850;
\vi : P 10;

###      CONTRAINTES      ###
\ci : rd1 ,
 200*XE2 + 100*XE1 + 50*XC50 + 20*XC20 + 10*XC10 = T - P ;
```

Ejemplo: **Un niño entra en un supermercado y compra cuatro elementos. El cajero cobra 7.11, 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 7.11. ¿Cuáles fueron los precios de los cuatro elementos?

Formalización:

$X = \{E1, E2, E3, E4\}$

$D = D(E1,..E4) = \{0,..8\}$

$C = \{ (E1 + E2 + E3 + E4 = 7.11),$
$(E1 * E2 * E3 * E4 = 7.11)\}$

```
##### ejemplo
\filtering : f ;
\search : rfla , first_solutions 5;
\static_labeling_order  : smallest_domain ;
\value_order :  bottom_first;
\verbose :  display_solutions;

###      VARIABLES       ###
\vi : E1, E2, E3, E4  0..800 ;    #para evitar números reales

###      CONTRAINTES       ###
\ci : rd1 , E1 + E2 + E3 + E4 = 711;
\ci : rd2 , E1 * E2 * E3 * E4 = 711;
```

Ejemplo: **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:

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

$D = D(T0,..E4) = \{0,\ldots 70\}$

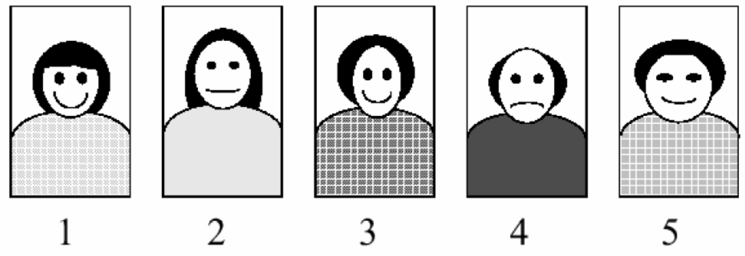
$C = \{ (10 \leq T1\text{-}T0 \leq 20),$
$(60 \leq T4\text{-}T0 \leq 70),$
$\boxed{(30 \leq T2\text{-}T1 \leq 40 \;\cup\; 60 \leq T2\text{-}T1),}$
$(10 \leq T2\text{-}T3 \leq 20),$
$(20 \leq T4\text{-}T3 \leq 30 \;\cup\; 40 \leq T4\text{-}T3 \leq 50)\}$

```
###     VARIABLES      ###
\vi : T0, T1, T2, T3, T4  0..70 ;

###     CONTRAINTES      ###
\ci: ci1 , T0 = 0;
\ci: ci2 , T1-T0 <=  20; \ci: ci3 , T1-T0 >=  10;
\ci: ci4 , T4-T0 <=  70; \ci: ci5 , T4-T0 >=  60;
\doc: doc1
     \coc : C3   \ci :  C13 ,  T2-T1 <=  40;
                    \and   \ci : C12  ,  T2-T1 >= 30;;
     \or     \ci :  C13 ,    T2-T1 >=  60;;
\ci: ci6 , T2-T3 <=  20; \ci: ci7 , T2-T3 >=  10;
\doc: doc2
     \coc :  C4      \ci :  C41  ,  T4-T3 <=  30;
                    \and   \ci : C42  ,  T4-T3 >=  20;;
     \or
      \coc :  C5    \ci :  C51  ,  T4-T3 <=  50;
                        \and    \ci : C52  ,  T4-T3 >=  40;;;
```

```
SOLUTION No 1
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 30  sat = 1.000 .
  (trouvee apres 5 instanciations et 14 tests de contraintes)

SOLUTION No 2
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 31  sat = 1.000 .
  (trouvee apres 6 instanciations et 20 tests de contraintes)

SOLUTION No 3
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 32  sat = 1.000 .
  (trouvee apres 7 instanciations et 26 tests de contraintes)

SOLUTION No 4
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 33  sat = 1.000 .
  (trouvee apres 8 instanciations et 32 tests de contraintes)

SOLUTION No 5
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 34  sat = 1.000 .
  (trouvee apres 9 instanciations et 38 tests de contraintes)

SOLUTION No 6
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 35  sat = 1.000 .
  (trouvee apres 10 instanciations et 44 tests de contraintes)

SOLUTION No 7
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 36  sat = 1.000 .
  (trouvee apres 11 instanciations et 50 tests de contraintes)

SOLUTION No 8
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 37  sat = 1.000 .
  (trouvee apres 12 instanciations et 56 tests de contraintes)

SOLUTION No 9
   T0 = 0   T1 = 10   T4 = 60   T2 = 50   T3 = 38  sat = 1.000 .
  (trouvee apres 13 instanciations et 62 tests de contraintes)

SOLUTION No 10
```

Ejercicio: 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

Testigo1: Adán=2 $\vee$ Baltasar=3

Adán≠2 $\vee$ Baltasar ≠ 3

PROBLEMA

1) ESPECIFICACIÓN

(expresividad)

Modelado CSP
- X: Variables {x1, x2, ..., xn}
- D: Dominios <D1,…,Dn>
- C: Restricciones del problema

2) **RESOLUCION**

(eficiencia)

Inferencia
+
búsqueda

Objetivos de un CSP:

- Obtener respuestas: Nuevas restricciones derivadas, Dominios acotados.
- ¿Tiene solución? Þ 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

Conceptos Básicos CSP

Dado un CSP,

X: Variables $\{x_1, x_2, ..., x_n\}$

D: Dominios $<D_1,..., D_n>$

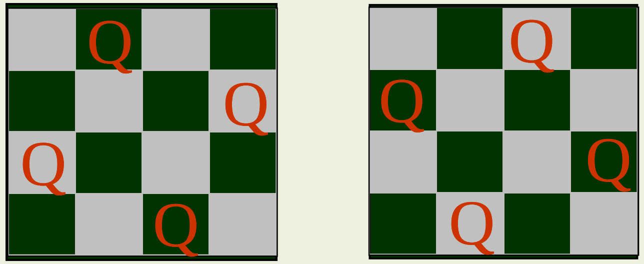
C: Restricciones del problema.

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

$$x_1=v_1,\ x_2=v_2,\ ...,\ x_n=v_n \qquad / \ v_i \in D_i$$

- 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:

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

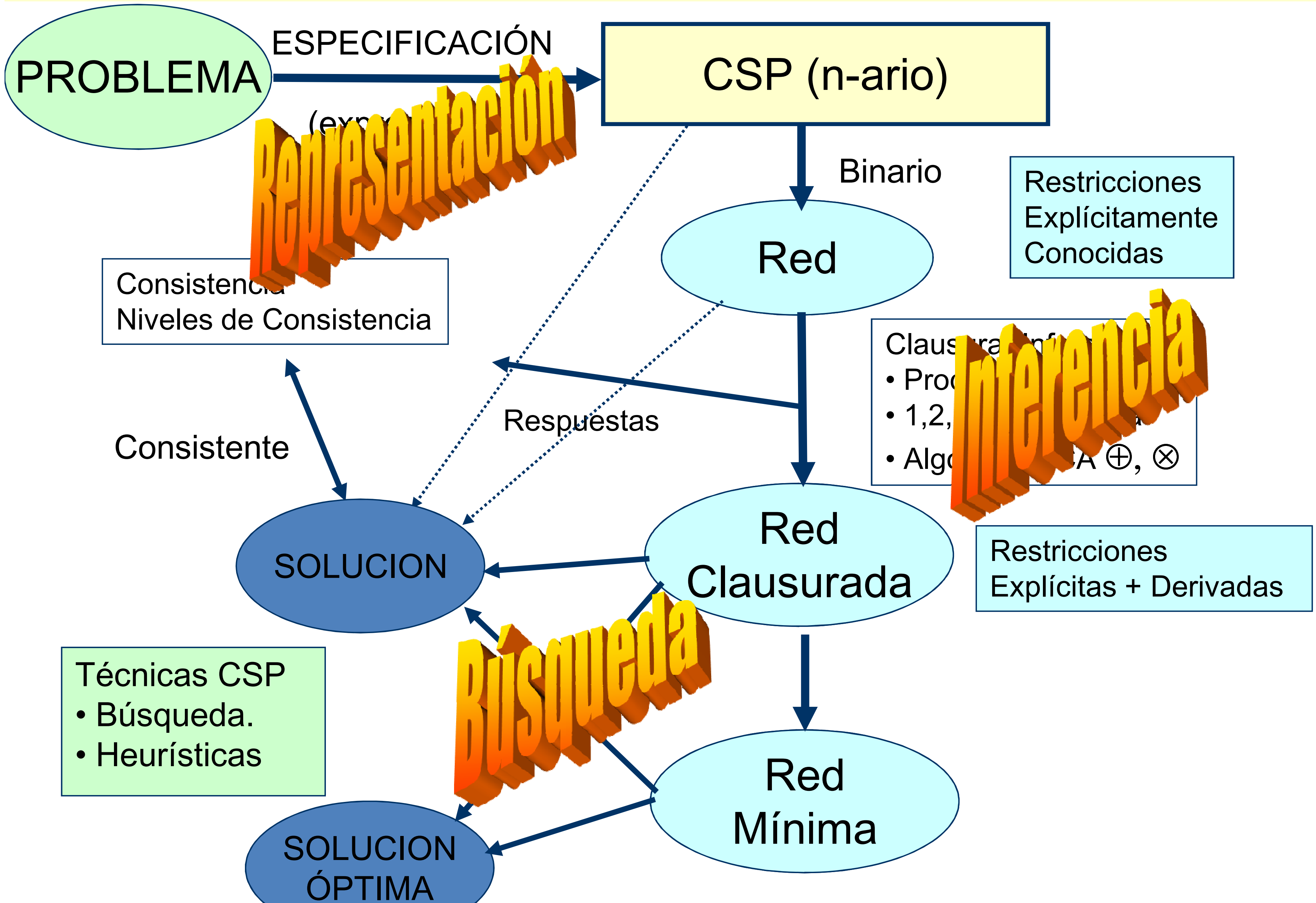
Valor consistente:

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



Técnicas Inferenciales:

- Infieren 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 *(algoritmos de preproceso en la búsqueda)*

Algoritmos de búsqueda de soluciones:

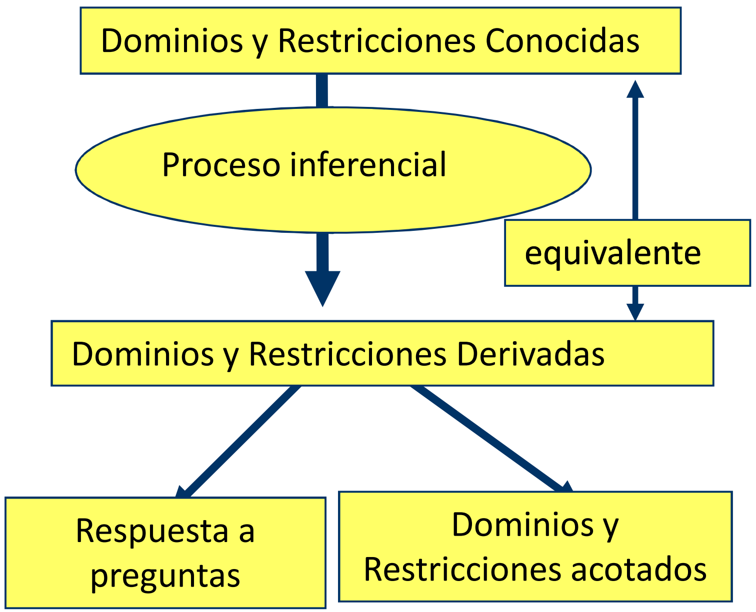
- Buscan una solución que satisfaga todas las restricciones del problema.
- Suelen utilizar información heurística *(ordenación de los valores y/o variables)* para llevar a cabo la búsqueda.
- Pueden utilizar pre-proceso (técnicas inferenciales)
- Técnicas de búsqueda híbridas: combinan la búsqueda e inferencia (durante el propio proceso de búsqueda).
- Heurísticas: Ordenación de variables, Ordenación de valores.

6.4.- Técnicas inferenciales

Los procesos inferenciales en un CSP permiten obtener nueva información de la explícitamente representada.

- *Limitan dominios de las variables.*
- *Limitan restricciones entre las variables*

Esto permite obtener respuestas y limitar el espacio de búsqueda de soluciones (más eficiencia)



El proceso inferencial se puede hacer:

➢ Antes del proceso de búsqueda: acotan dominios y restricciones.

➢ Durante el proceso de búsqueda:
- Acotan dinámicamente la búsqueda.
- Da lugar a los Algoritmos Híbridos (búsqueda + inferencia).

Ejemplo: Sodoku

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…

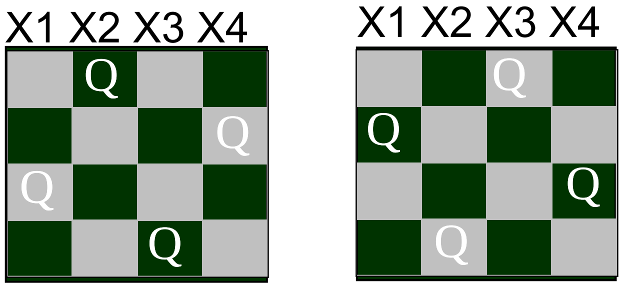
		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

		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		

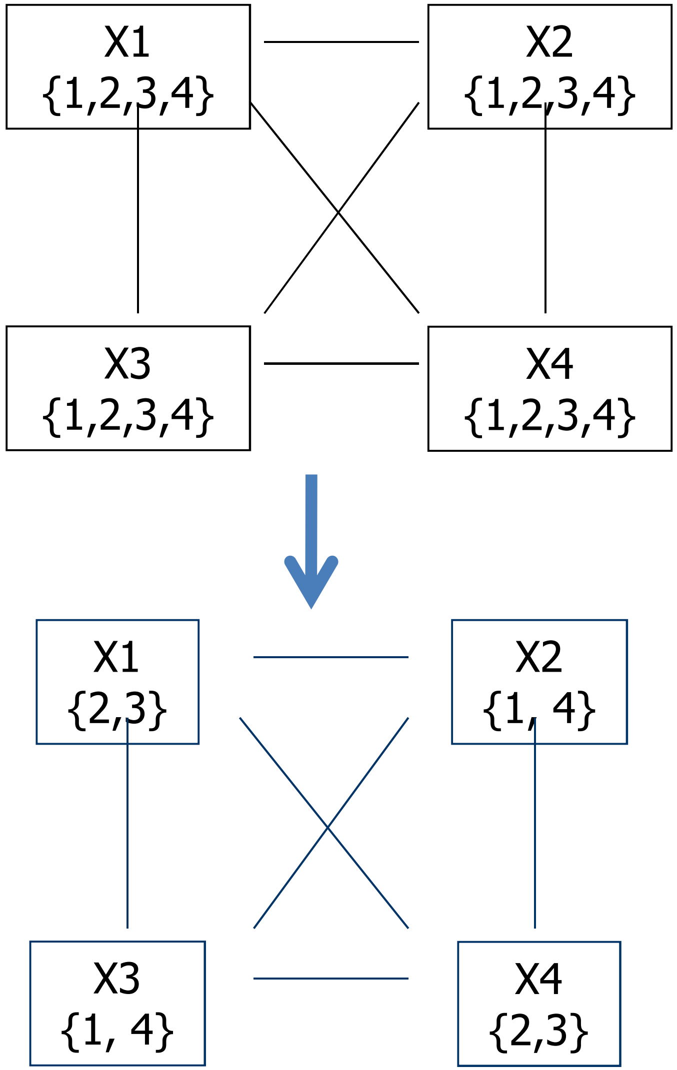
Un simple proceso de 2-consitencia

¿Podríamos resolver todo el sudoku sin búsqueda?

Ejemplo: Colocar 4 reinas en un tablero 4x4



R_{ij}=No misma Fila/Diagonal



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

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 solución parcial de k-1 variables es una solución parcial de k variables.

Dada una asignación consistente a un subconjunto cualesquiera de k-1 variables del CSP, dicha asignación es extensible a k variables.

Consistencia de nodo (1-consistencia)

- 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:

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

$5 \leq Xi \leq 15,$

D(Xi): [0, 10]

Nodo-consistencia

Coste: O(N)

$5 \leq Xi \leq 15,$

D(Xi): [5, 10]

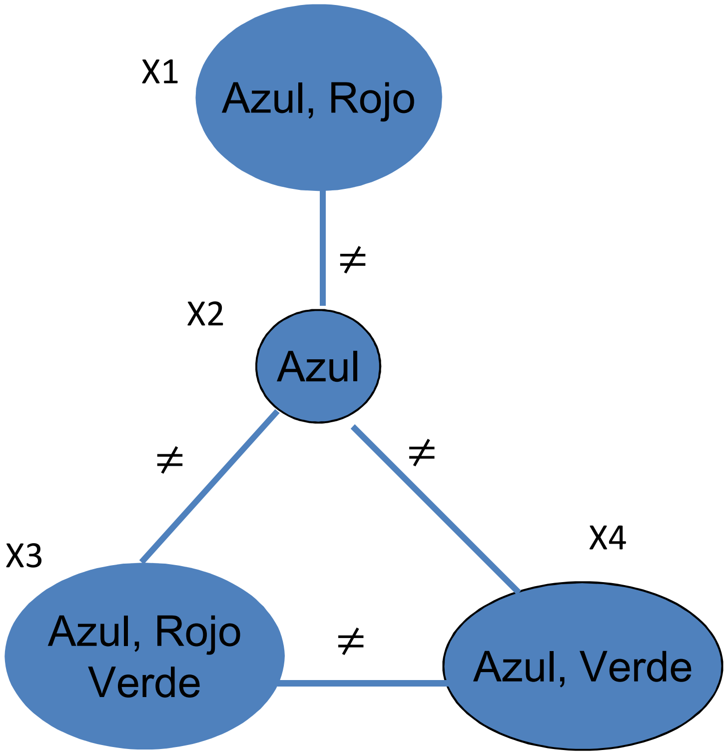
Se han acotado los dominios de las variables

Consistencia de arco (2-consistencia)

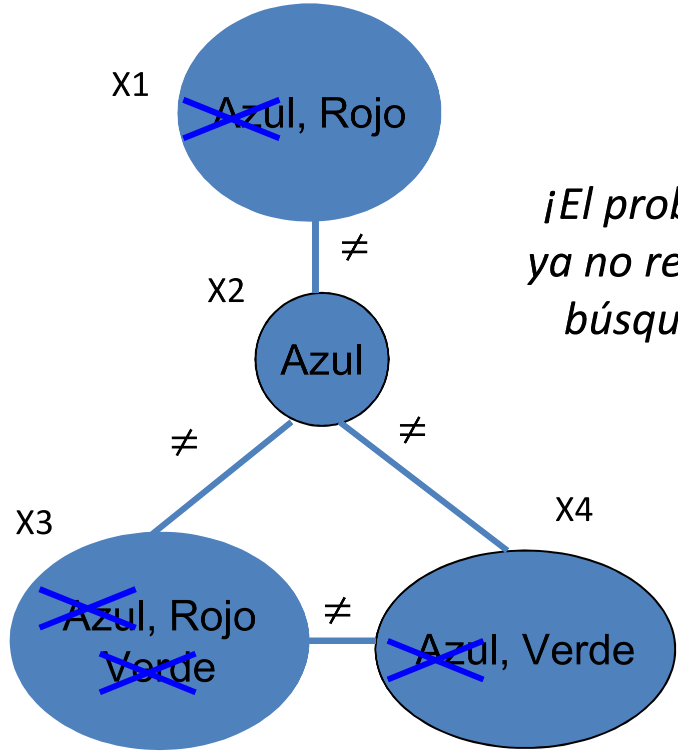
- Para cada restricción entre un par de variables (xi {cij} xj), los dominios de las variables con consistentes con la restricción (arco) entre las variables.
- Una red temporal es *arco-consistente* si todos sus arcos son consistentes.

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

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



Grafo Inicial



¡El problema ya no requiere búsqueda!

Grafo arco-consistente

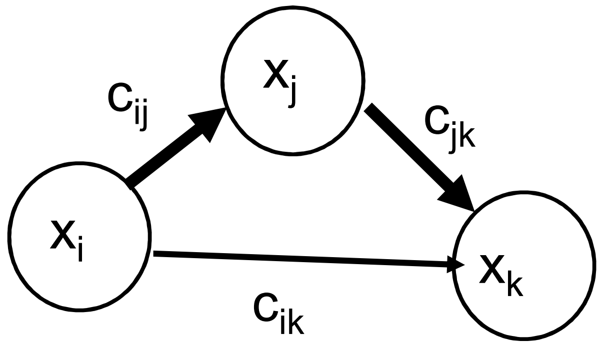
3-consistencia: senda consistencia

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:

$$\forall c_{ij} \subseteq CSP,\ \forall v_i \in d_i,\ \forall v_j \in d_j \ /\ (x_i=v_i\ c_{ij}\ x_j=v_j) \Rightarrow \exists v_k \in d_k \ /\ (x_i=v_i\ c_{ik}\ x_k=v_k) \wedge (x_j=v_j\ c_{jk}\ x_k=v_k)$$

- Cada subred de 3 nodos es consistente.
- Las restricciones entre tres nodos son consistentes si:

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



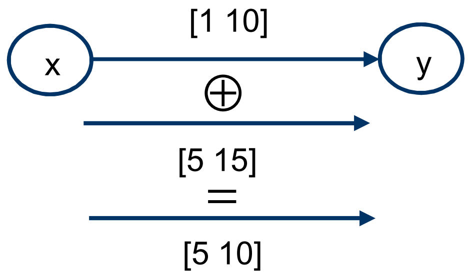
Algoritmo de Clausura Transitiva (TCA)

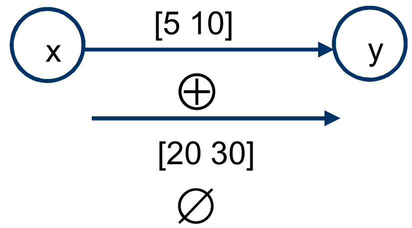
- Operaciones: Adición $\oplus$, Composición $\otimes$
- *Clausura de restricciones en <u>redes binarias</u>*

Operativa de las Restricciones (⊕ ⊗)

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

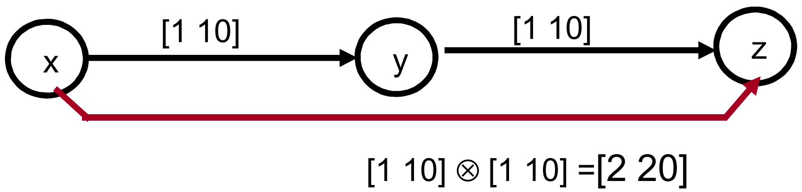
- Actualización, por una nueva restricción, de la restricción existente previa entre los dos variables.
- Cuando no hay intersección se produce una inconsistencia: ¡CSP no consistente!





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

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



Las operaciones ⊕ y ⊗ 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 ⊕

(∅ es la inconsistencia):

⊕	<	≤	>	≥	=	≠	?
<	<	<	∅	∅	∅	<	<
≤	<	≤	∅	=	=	<	≤
>	∅	∅	>	>	∅	>	>
≥	∅	=	>	≥	=	>	≥
=	∅	=	∅	=	=	∅	=
≠	<	<	>	>	∅	≠	≠
?	<	≤	>	≥	=	≠	?

Operación de Combinación (⊗)

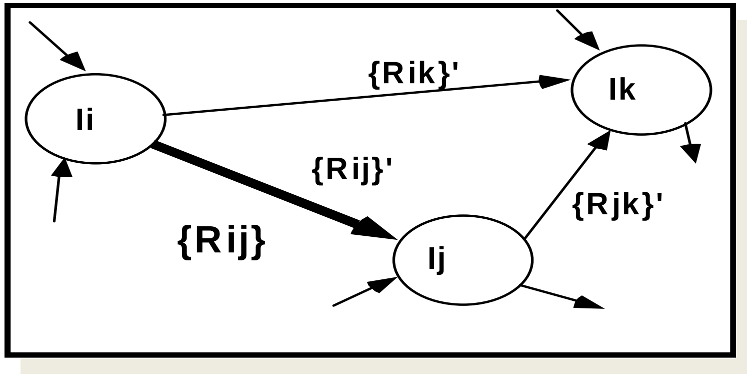
$(ti\ Ri\ tj) \otimes (tj\ Rj\ tk) = ti\ (Ri \otimes Rj)\ tk$

⊗	<	≤	>	≥	=	≠	?
<	<	<	?	?	<	?	?
≤	<	≤	?	?	≤	?	?
>	?	?	>	>	>	?	?
≥	?	?	>	≥	≥	?	?
=	<	≤	>	≥	=	≠	?
≠	?	?	?	?	≠	?	?
?	?	?	?	?	?	?	?

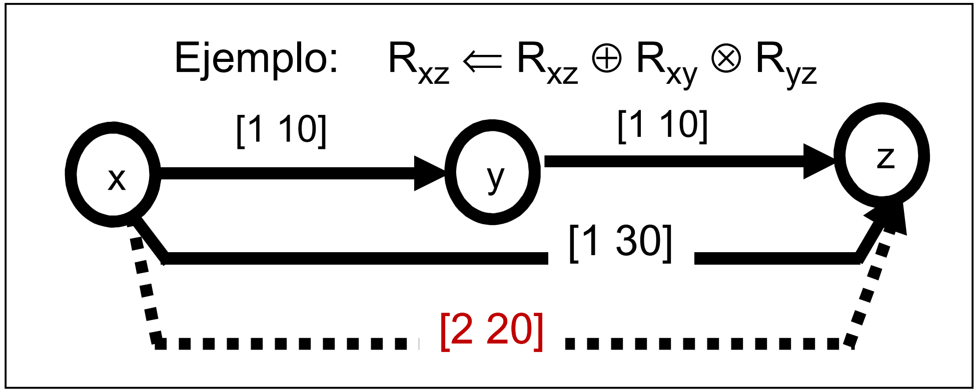
Transitive Closure Algorithm (TCA):

Clausura de una red de restricciones: obtiene una red senda consistente aplicando las operaciones ⊗, ⊕.

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



La 3-consistencia acota dominios



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.

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

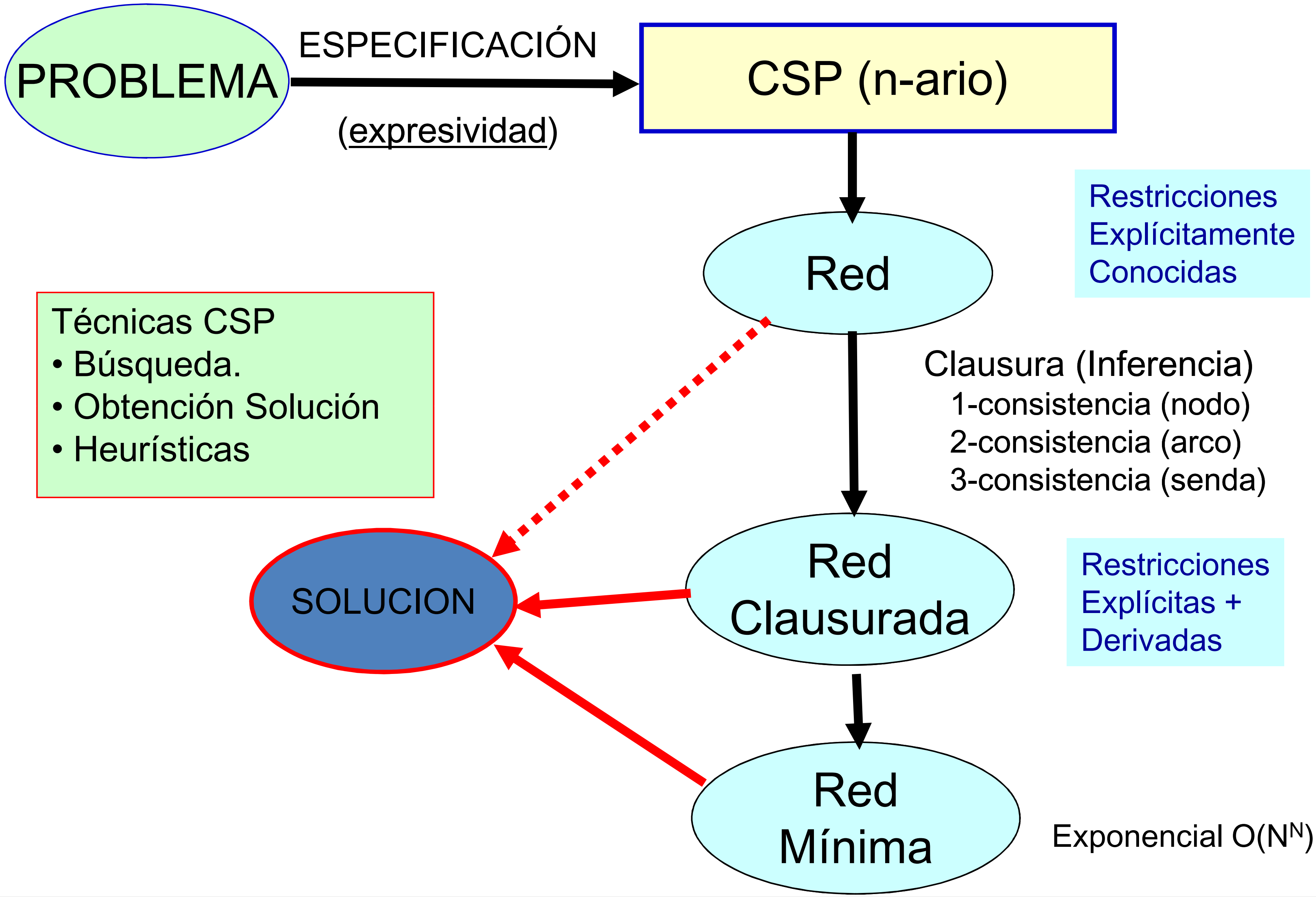
Las técnicas de inferencia nos permiten limitar el dominio de las variables e incluso responder a preguntas sin NECESIDAD de realizar búsqueda

- 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 a*d-hoc*) para la obtención de soluciones.

Se utilizan en multitud de problemas reales

6.5.- Técnicas de Resolución. Búsqueda de soluciones



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

Métodos de Búsqueda

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

Técnicas de Bactracking: 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)*

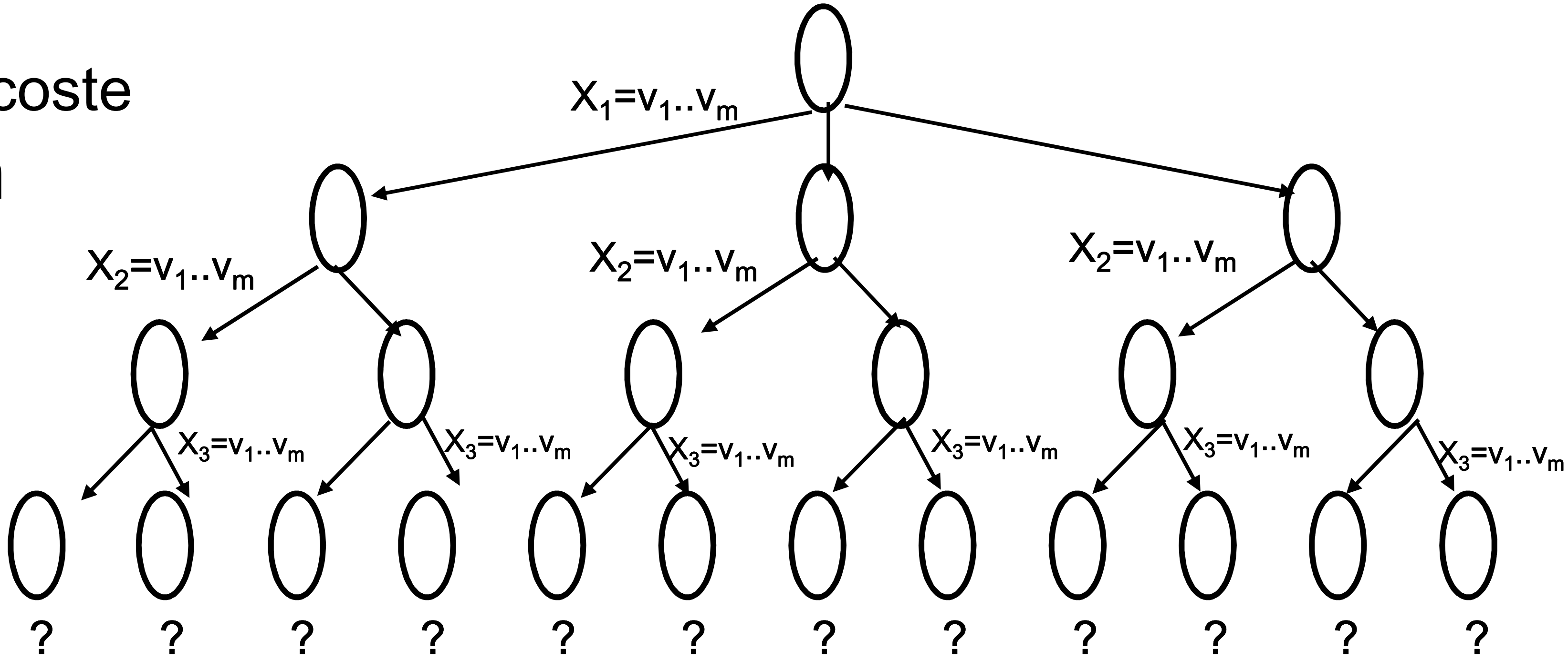
Procedimiento general: Generate and Test

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.

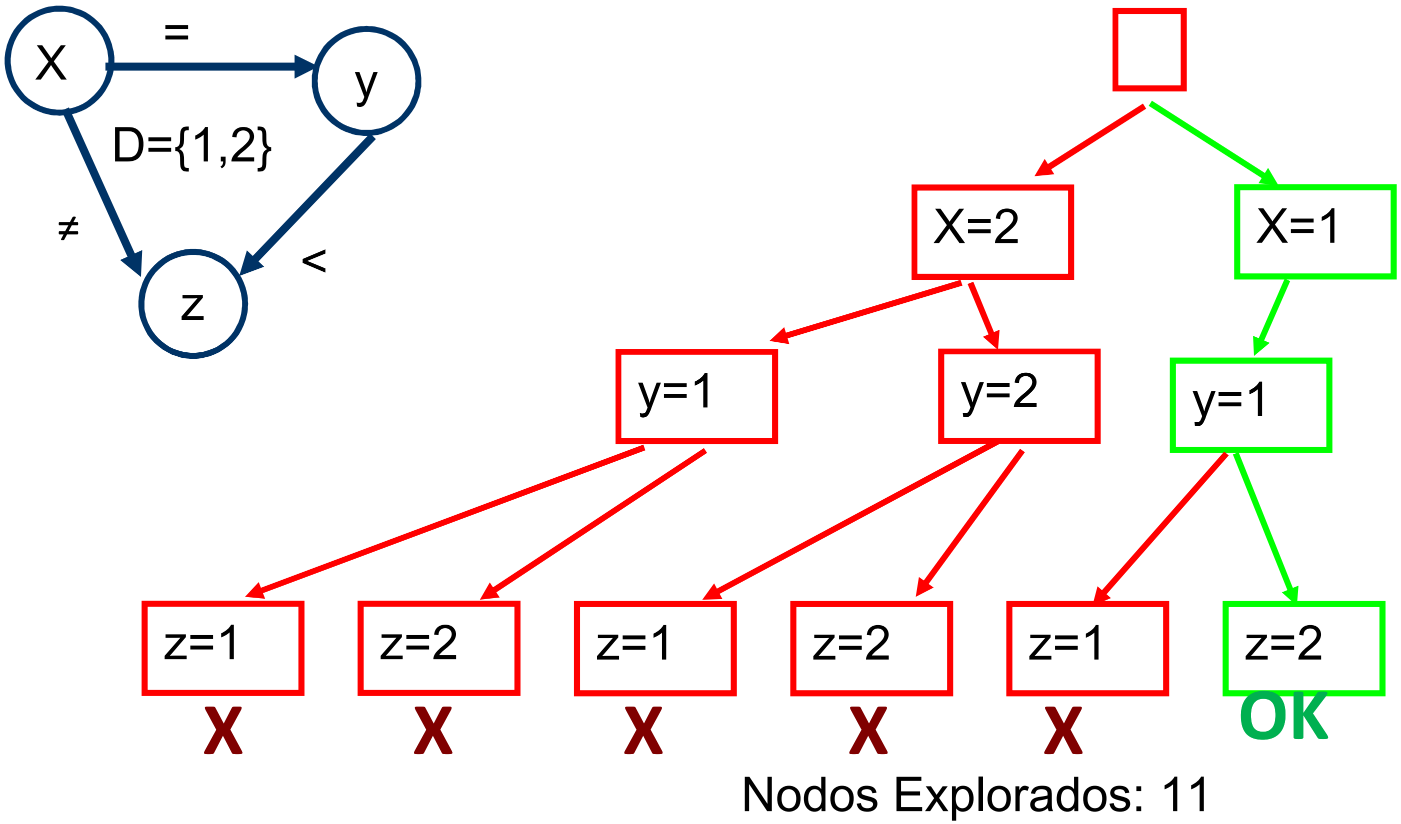
- Para ‘n’ variables, y tamaños de dominio ‘m_i’, se genera un árbol de n niveles, con ramificación de m_i nodos en cada nivel, representando los posibles valores que puede tomar la variable v_i en su dominio d_i.
- Nº de hojas (nodos terminales), sobre los que debe comprobarse las restricciones: $m_1 * m_2 * .... m_n$

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)*

Exploración del árbol mediante backtracking:

<u>Procedure Backtracking (k, V[n])</u> *;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

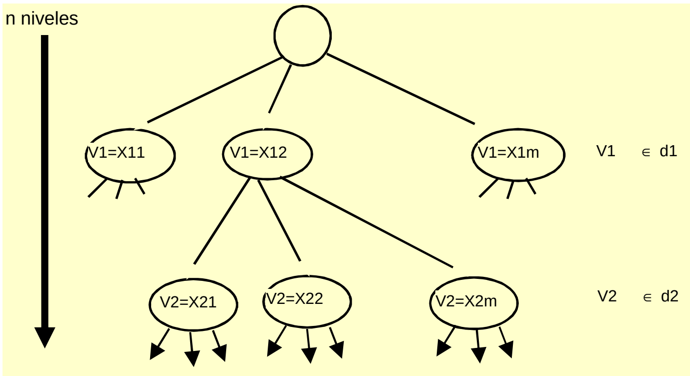
if k=1 then Return (∅) (*fallo*)

Nivel K-1: Vuelvo nivel anterior

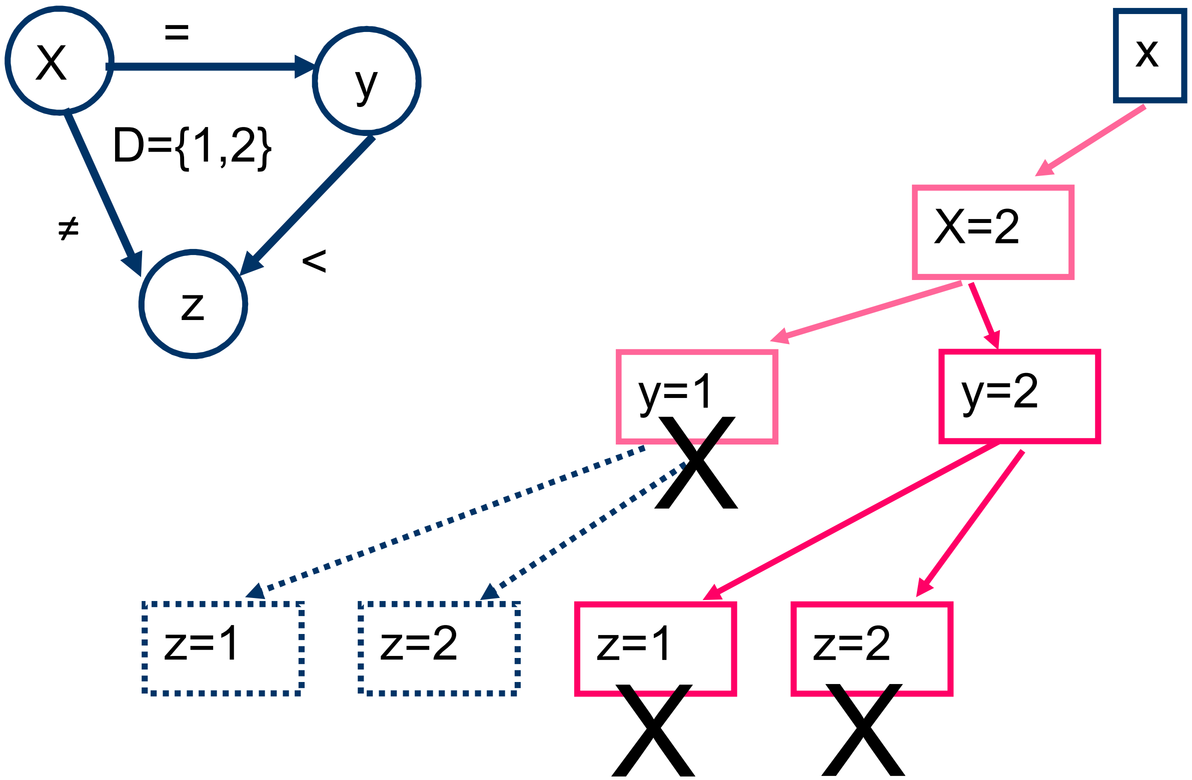
else Backtraking (k-1, V[n])

end

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

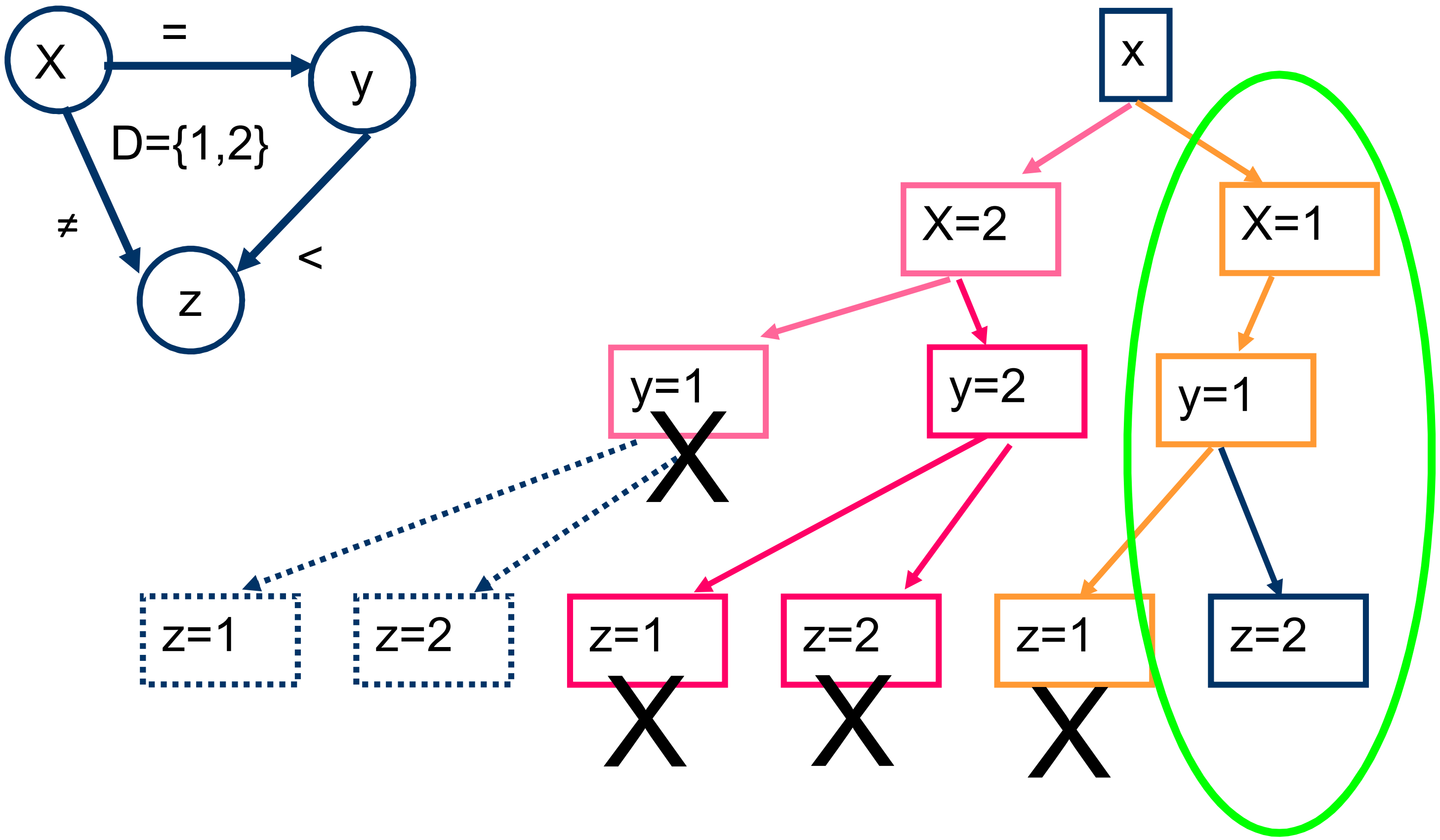


Ejemplo. Standard Backtracking (look backward)



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

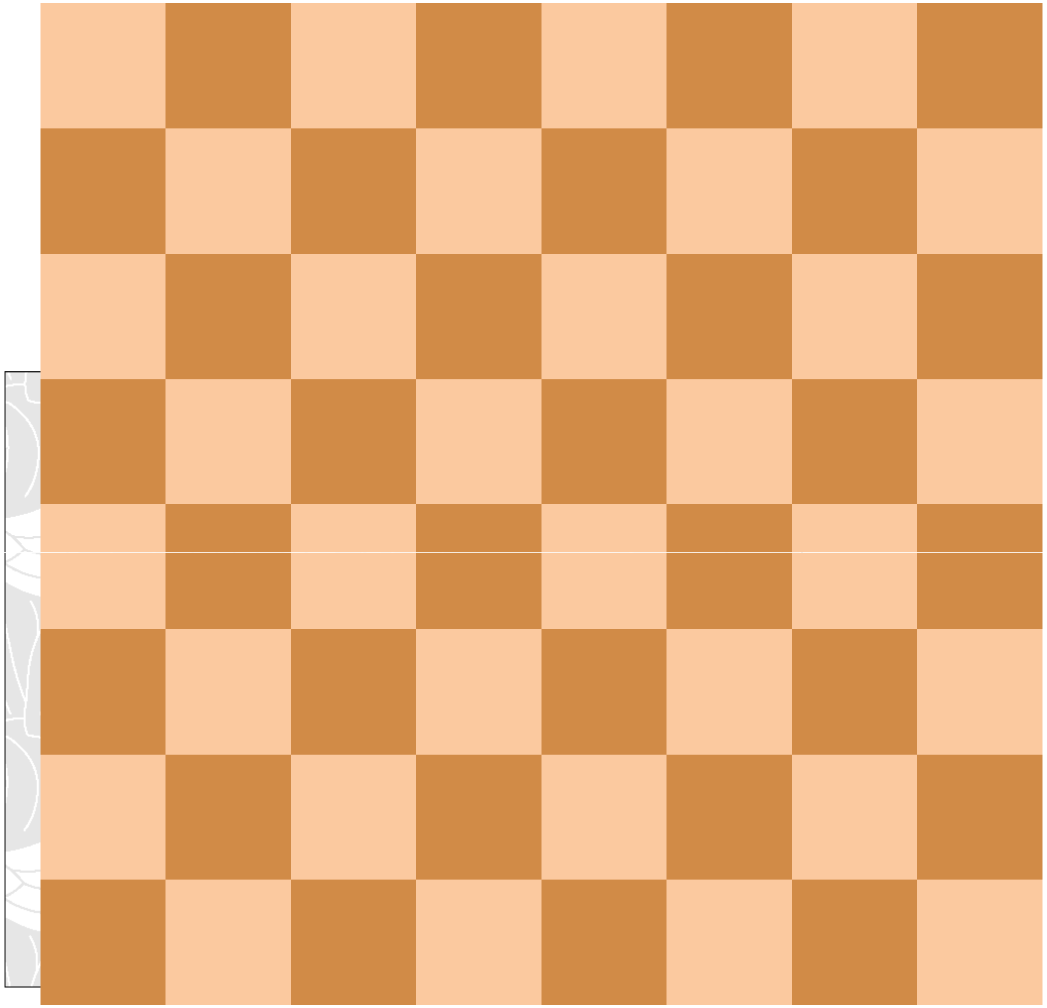
Ejemplo. Standard Backtracking (look backward)

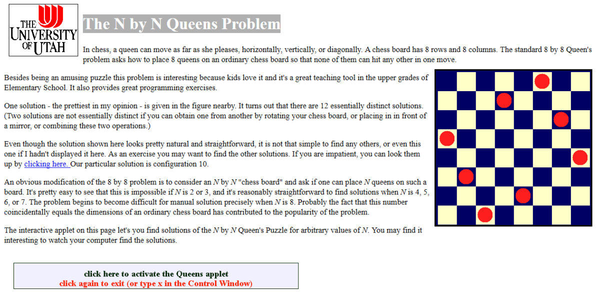


Nodos Explorados: 9

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

Ejemplo Backtracking 8-reinas





The N by N Queens Problem

In chess, a queen can move as far as she pleases, horizontally, vertically, or diagonally. A chess board has 8 rows and 8 columns. The standard 8 by 8 Queen's problem asks how to place 8 queens on an ordinary chess board so that none of them can hit any other in one move.

Besides being an amusing puzzle this problem is interesting because kids love it and it's a great teaching tool in the upper grades of Elementary School. It also provides great programming exercises.

One solution - the prettiest in my opinion - is given in the figure nearby. It turns out that there are 12 essentially distinct solutions. (Two solutions are not essentially distinct if you can obtain one from another by rotating your chess board, or placing in in front of a mirror, or combining these two operations.)

Even though the solution shown here looks pretty natural and straightforward, it is not that simple to find any others, or even this one if I hadn't displayed it here. As an exercise you may want to find the other solutions. If you are impatient, you can look them up by clicking here. Our particular solution is configuration 10.

An obvious modification of the 8 by 8 problem is to consider an *N* by *N* "chess board" and ask if one can place *N* queens on such a board. It's pretty easy to see that this is impossible if *N* is 2 or 3, and it's reasonably straightforward to find solutions when *N* is 4, 5, 6, or 7. The problem begins to become difficult for manual solution precisely when *N* is 8. Probably the fact that this number coincidentally equals the dimensions of an ordinary chess board has contributed to the popularity of the problem.

The interactive applet on this page let's you find solutions of the *N* by *N* Queen's Puzzle for arbitrary values of *N*. You may find it interesting to watch your computer find the solutions.

click here to activate the Queens applet
click again to exit (or type x in the Control Window)

Backtracking N by N queens applet

Aunque encontrar una solución para N>3 se puede hacer sin búsqueda, tal que determinar las posiciones puede hacerse en O(N).

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] ∈ dk
   If Comprobar (k, V[n]) then….
```

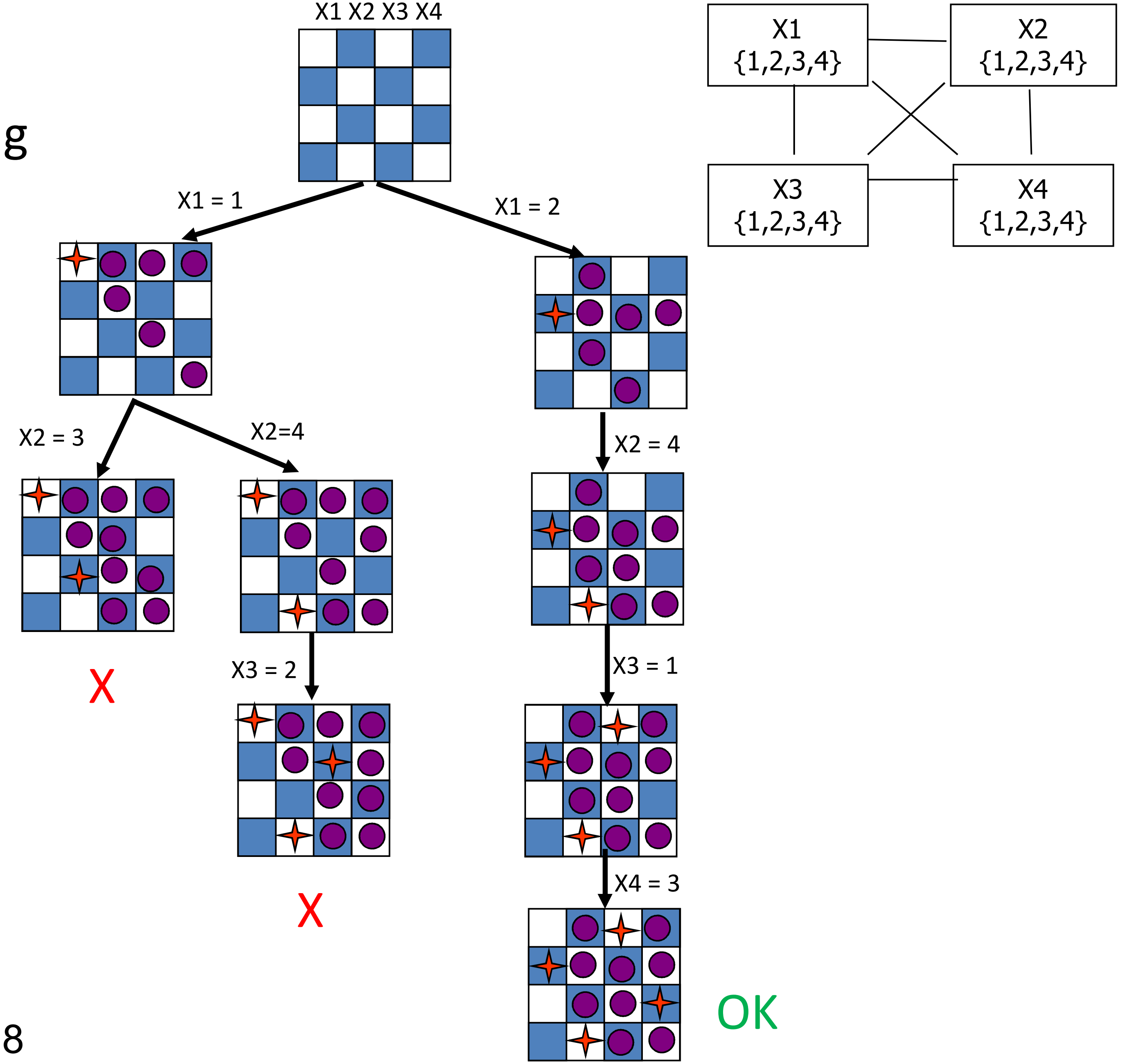
a) Forward-Checking (FC)

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)*

Forward Checking



Nodos Explorados: 8

a) Forward-Checking (FC)

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

Además de FC,

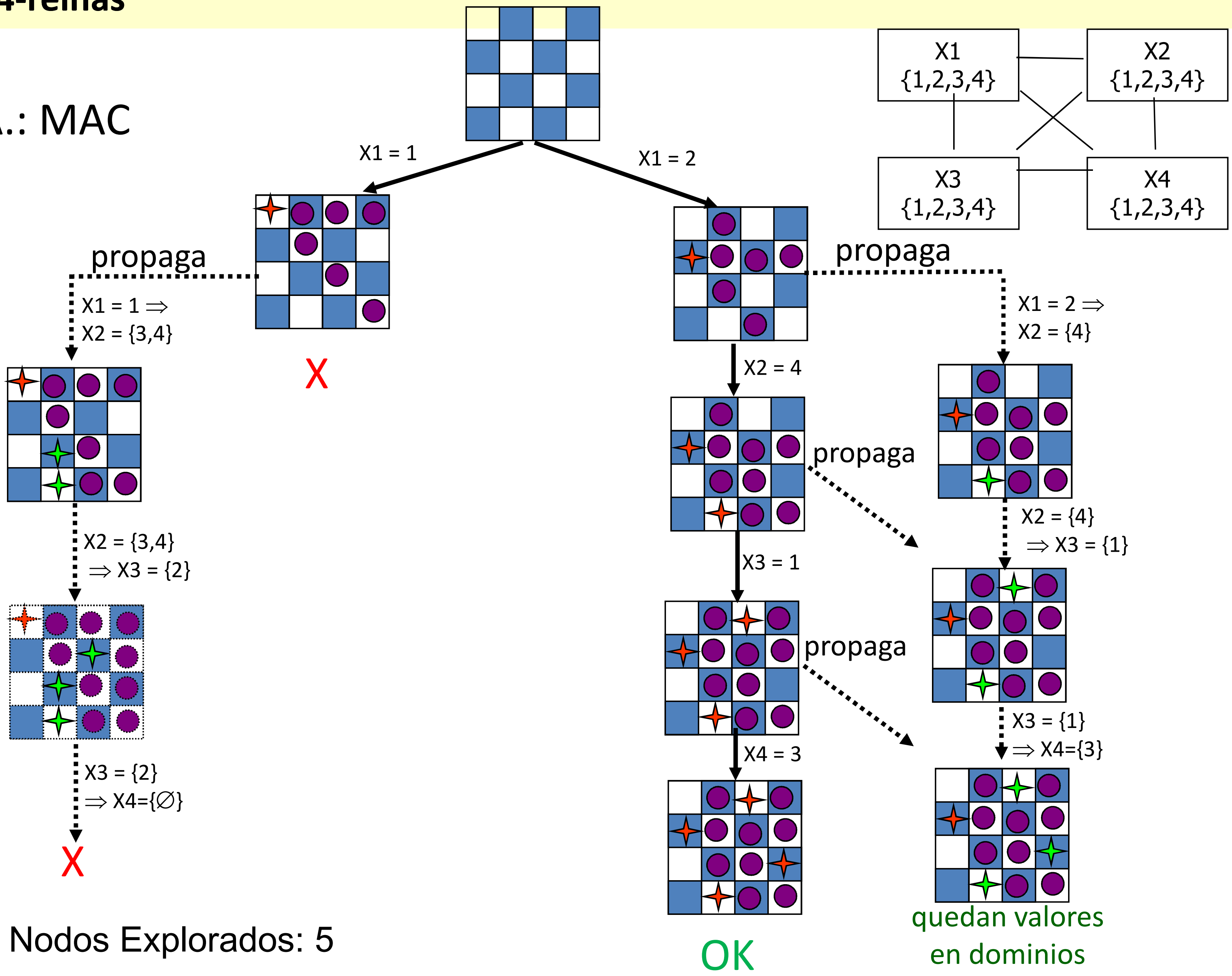
Sobre cada variable por instanciar, se podan aquellos valores inconsistentes con cada posible valor de todas las variables que quedan por instanciar.

Propaga los valores eliminados en el proceso de forward-checking

Es 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

Ejemplo. El puzzle de las 4 casas

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

$C \neq 4$ $\quad$ $D \neq 2$

$B = A + 1$ $\quad$ $A \neq C$

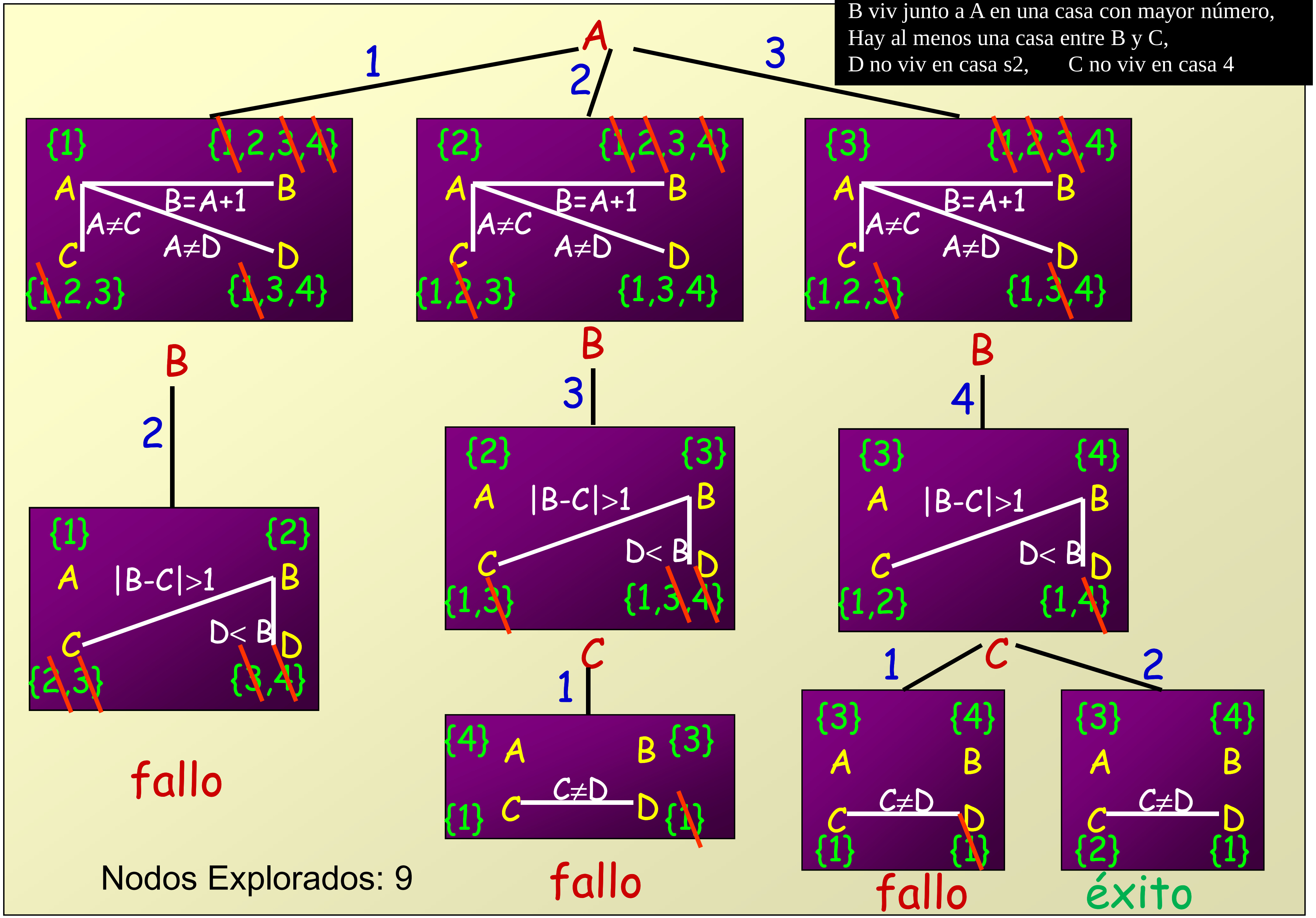
$D < B$ $\quad$ $A \neq D$

$|B - C| > 1$ $\quad$ $C \neq D$

¿Qué familia viv en cada casa?

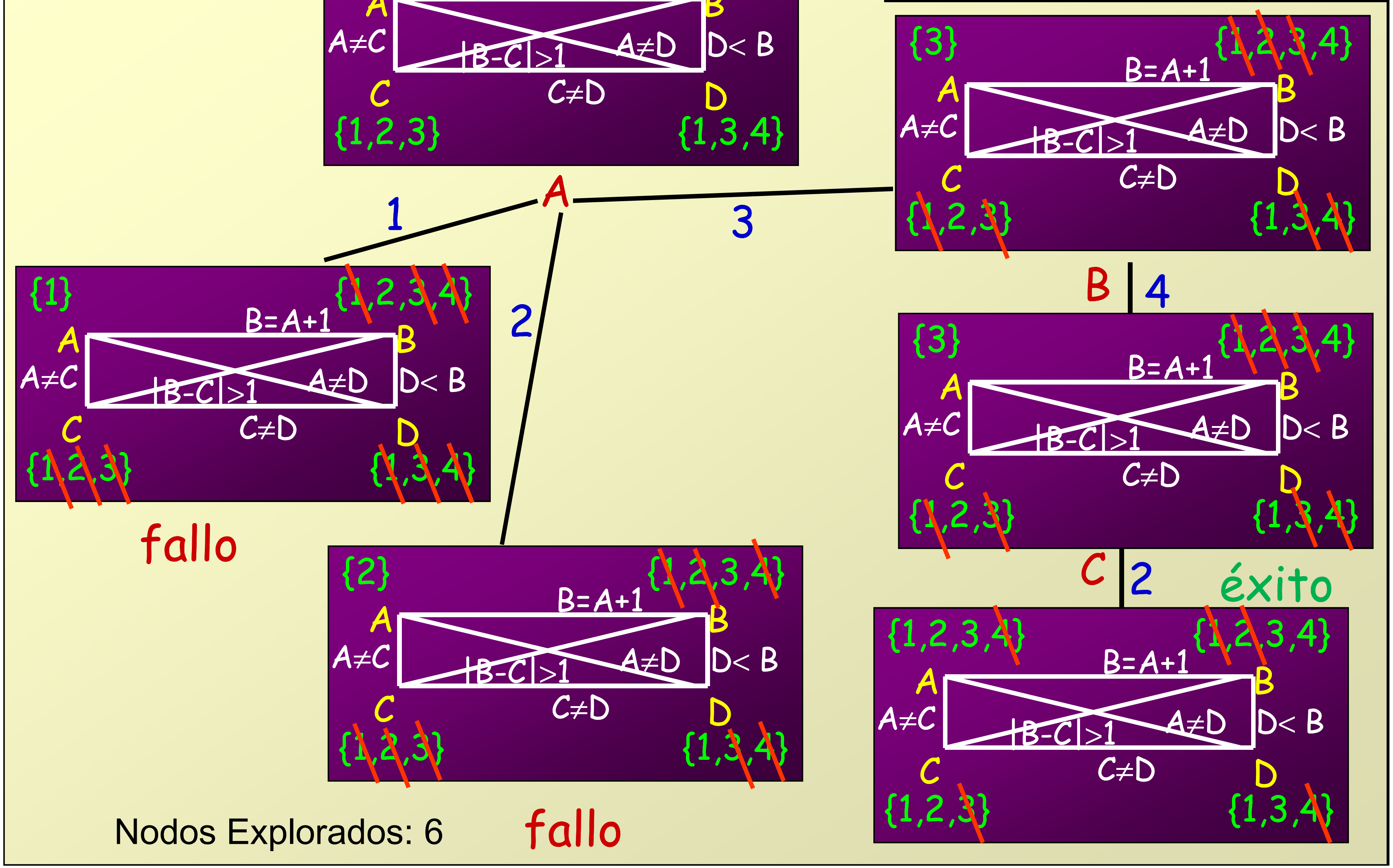
Forward checking

D viv en casa con menor número que B,
B viv junto a A en una casa con mayor número,
Hay al menos una casa entre B y C,
D no viv en casa s2,   C no viv en casa 4



Full Look-Ahead: MAC

D viv en casa con menor número que B,
B viv junto a A en una casa con mayor número,
Hay al menos una casa entre B y C,
D no viv en casa s2,   C no viv en casa 4



¿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:

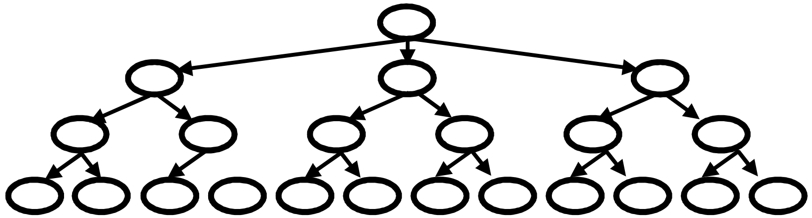
   - Propagación de las instanciaciones parciales para minimizar dominios de las variables por instanciar: *Look-Forward*.

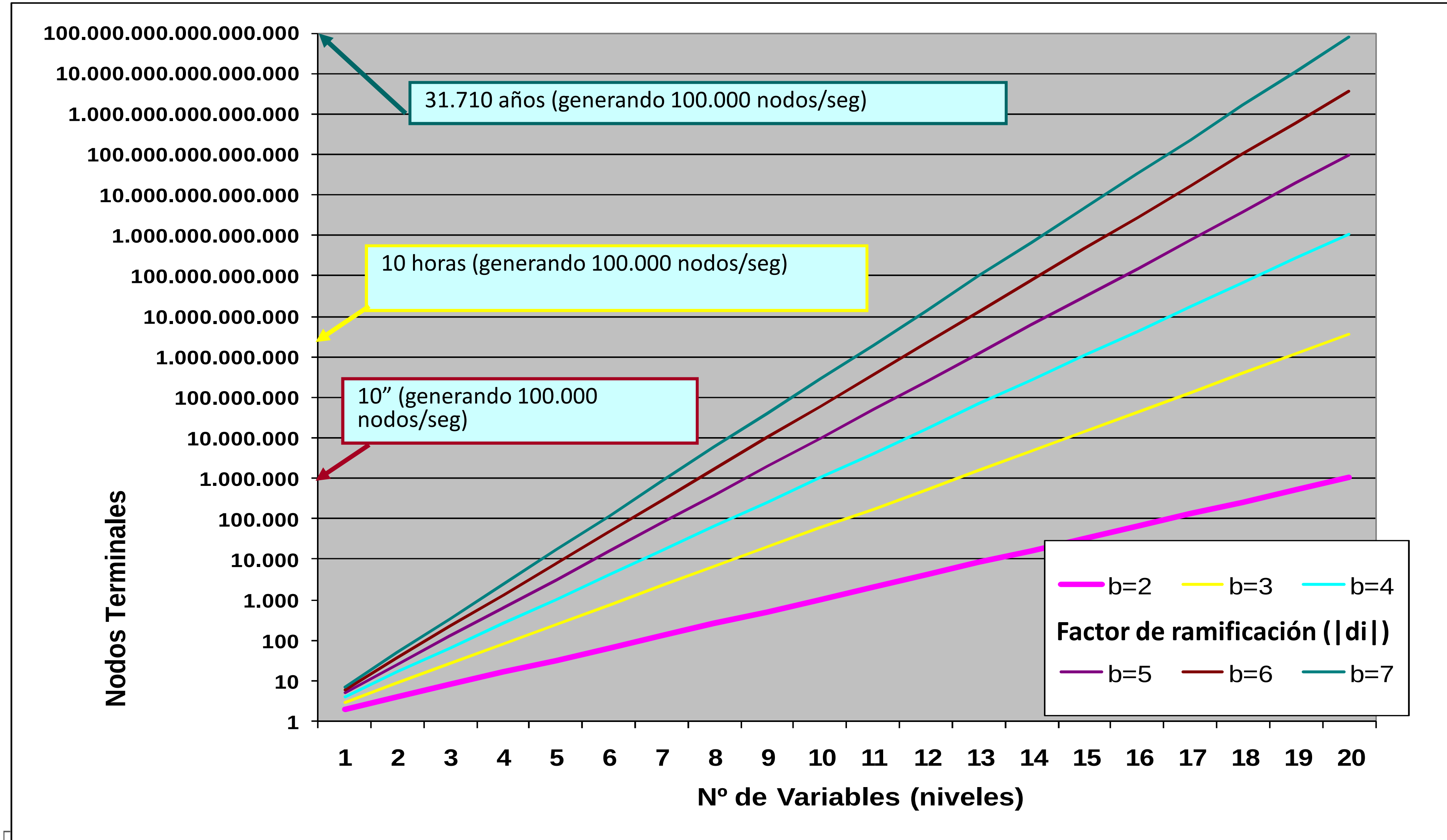
2) Técnicas de **Preproceso** para minimizar dominios iniciales de instanciación y restricciones entre variables: Métodos Inferenciales: 1, 2, 3-consistencia.

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

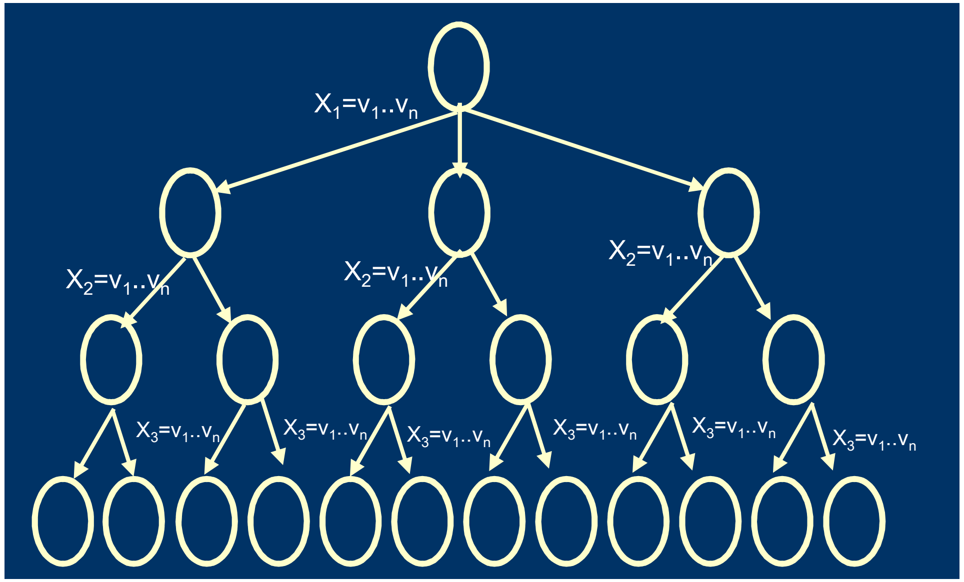
Crecimiento Exponencial de la Búsqueda

Aún con gran preproceso y búsqueda inferencial, se requieren heurísticas





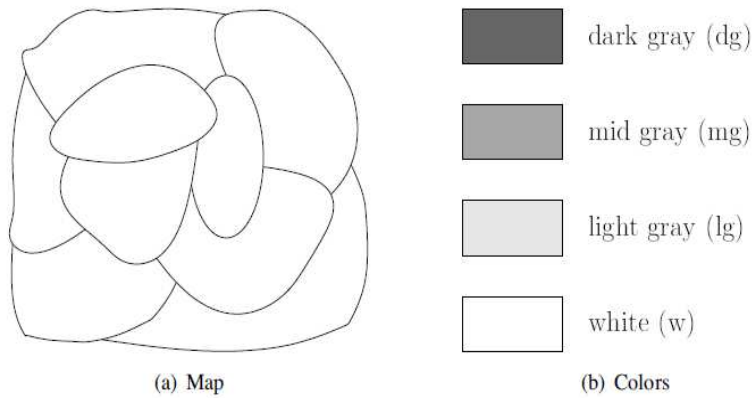
6.6.- Heurísticas de Búsqueda



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.

Heurísticas de Ordenación de Variables

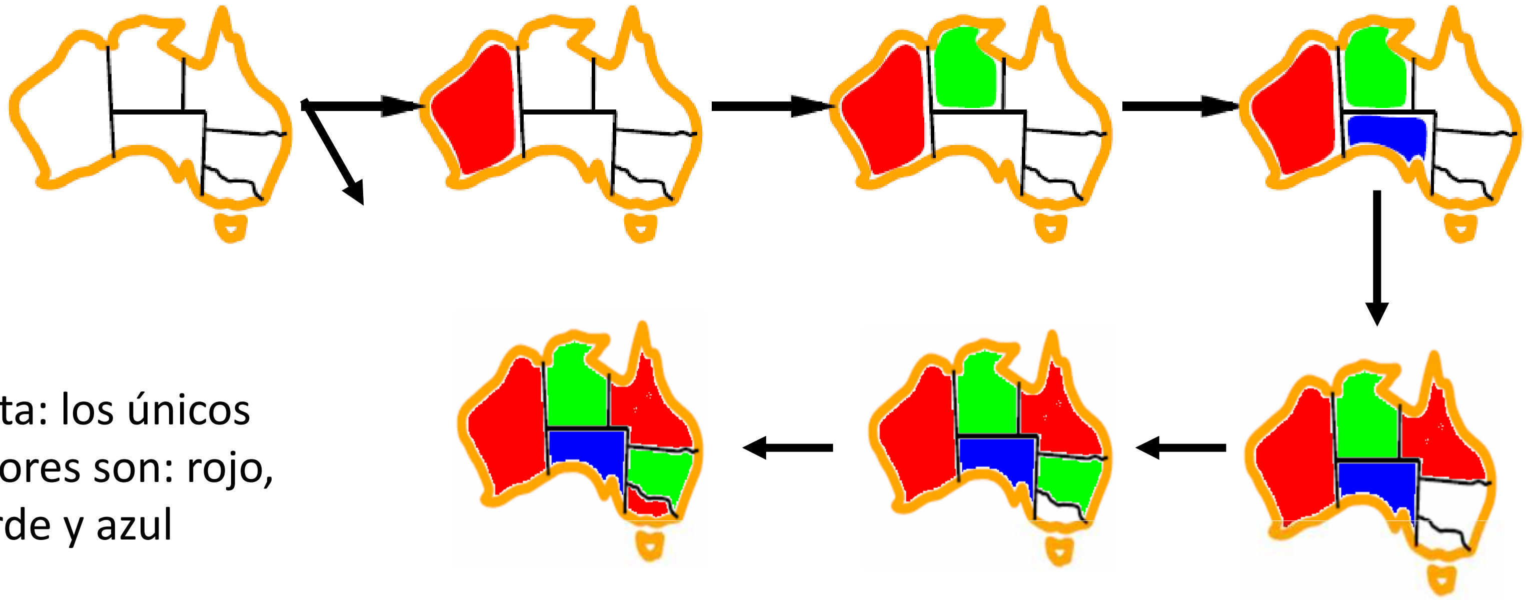
Fija o Estática: se determina un orden fijo para la instanciación de la variables al inicio del proceso.

a) *Dinámica:* el orden de instanciación de las variables se revisa en cada instanciación.

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

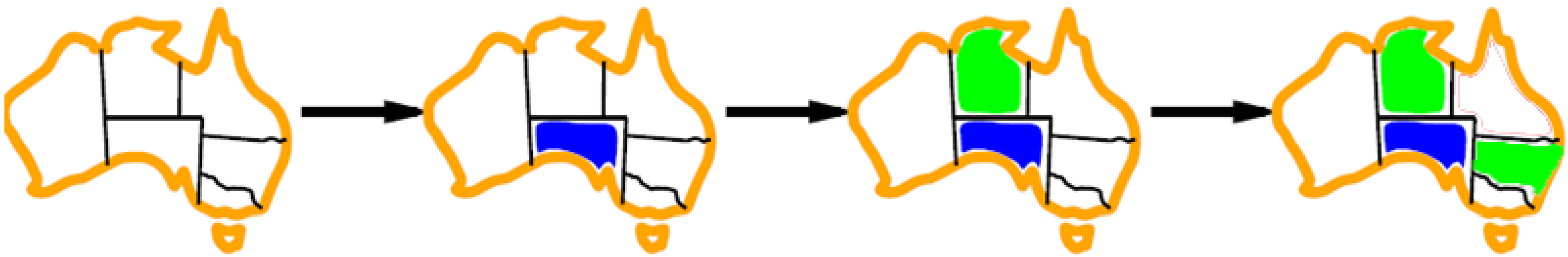
Existen otras muchas heurísticas generales de ordenación

Heurística Mínimo Dominio: (MDV) Elige la variable con menor número de valores posibles (≅ 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

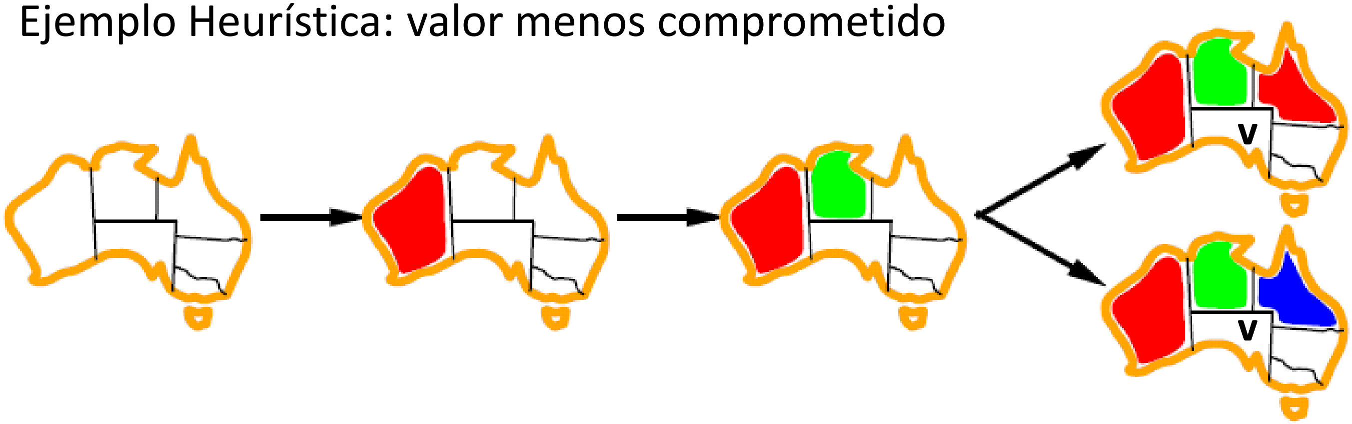


Heurísticas de Selección de Valores

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 supervivncia)*: 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

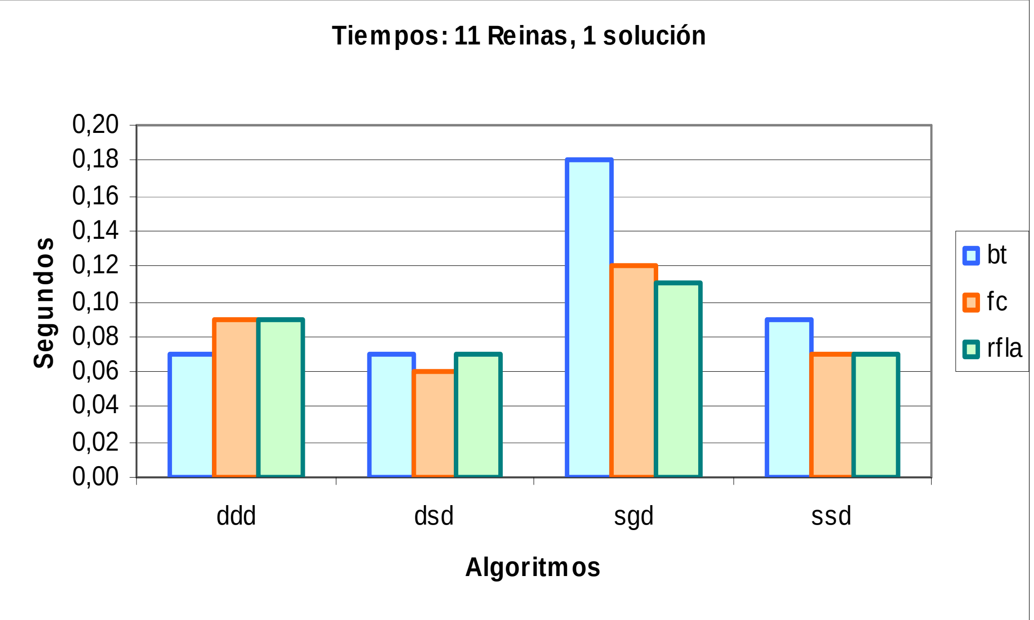


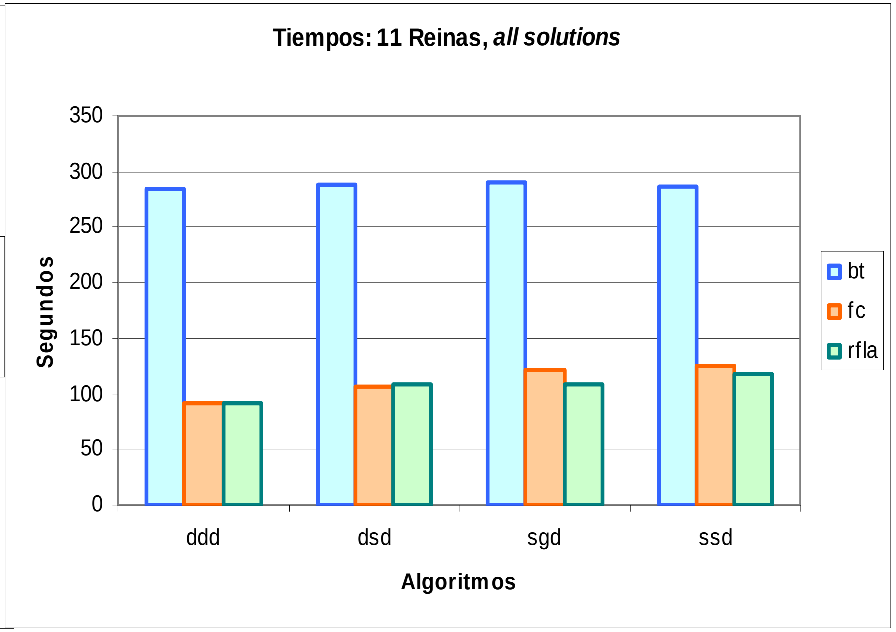
Permite 1 valor para V

Permite 0 valores para V

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

Evaluación: 11 Reinas





Heurísticas Variables

Static Labeling Order:

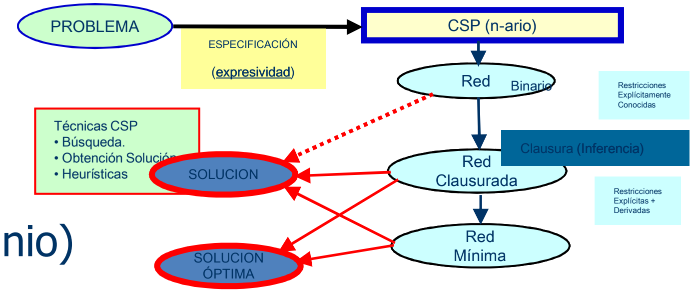
- smallest_domain (ssd): mínimo dominio
- greatest_degree (sgd): máximo grado

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.

6.7.- Aplicaciones y Entornos CSP



Proporcionan:

- ✓ Editores de CSP (X, D, C), CSOP.
- ✓ Métodos Inferenciales, de proceso.
- ✓ Métodos de búsqueda (heurísticas independientes del dominio)
- ✓ Satisfacción & Optimización

- Conflex (http://www.inra.fr/bia/T/conflex/): Libre, sencillo, flexible.
- IBM-ILOG: CP-optimizer (https://www.ibm.com/analytics/cplex-cp-optimizer).
- MiniZinc (medium-level constraint modelling language): http://www.minizinc.org/
- Choco (en java). (http://choco-solver.org/)
- GAMS (http://www.gams.com/)
- CP-SAT Solver (https://developers.google.com/optimization/cp/cp_solver)
- GPLK (GNU Linear Programming Kit). http://www.gnu.org/s/glpk/)
- ECLiPSe (http://eclipseclp.org/), Mozart-Oz (http://www.mozart-oz.org/)
- Cream (http://bach.istc.kobe-u.ac.jp/cream/), MINION (http://constraintmodelling.org/minion/)
- Lekin (http://www.stern.nyu.edu/om/software/lekin/), LINGO, LINDO (http://www.lindo.com/)
- Repositorios: (http://scom.hud.ac.uk/planet/repository/schedulers.html)

Entornos Generalistas

(Satisfacción / Optimización)

Notaciones y Resolutores CSP

Existen diversas herramientas para la (i) modelización y (ii) resolución de CSP. Por ejemplo:

Lenguajes/Entornos para el Modelado de CSP

- **AMPL** (http://www.ampl.com/), integrado con diversos resolvedores
- FlatZinc (lenguaje de alto nivel e independiente para muchos resolvedores),
- xCSP: Notación XML para modelado CSP. Interfaz gráfica: TAILOR
- Essence': Notación que permite representar 'clases' de problemas (en vez de instancias),
- **Gusek**, interfaz para el resolvedor GLPK (http://gusek.sourceforge.net/gusek.html), etc.

Analizadores (parsers)

Resolutores CSP /
Librerías en C, C++, Java, Python, etc.

- GLPK – Librería en C (www.gnu.org/software/glpk/)
- Abscon
- GECODE (librería C++)
- ToulBar2 (para WCSP)
- MINION

etc.

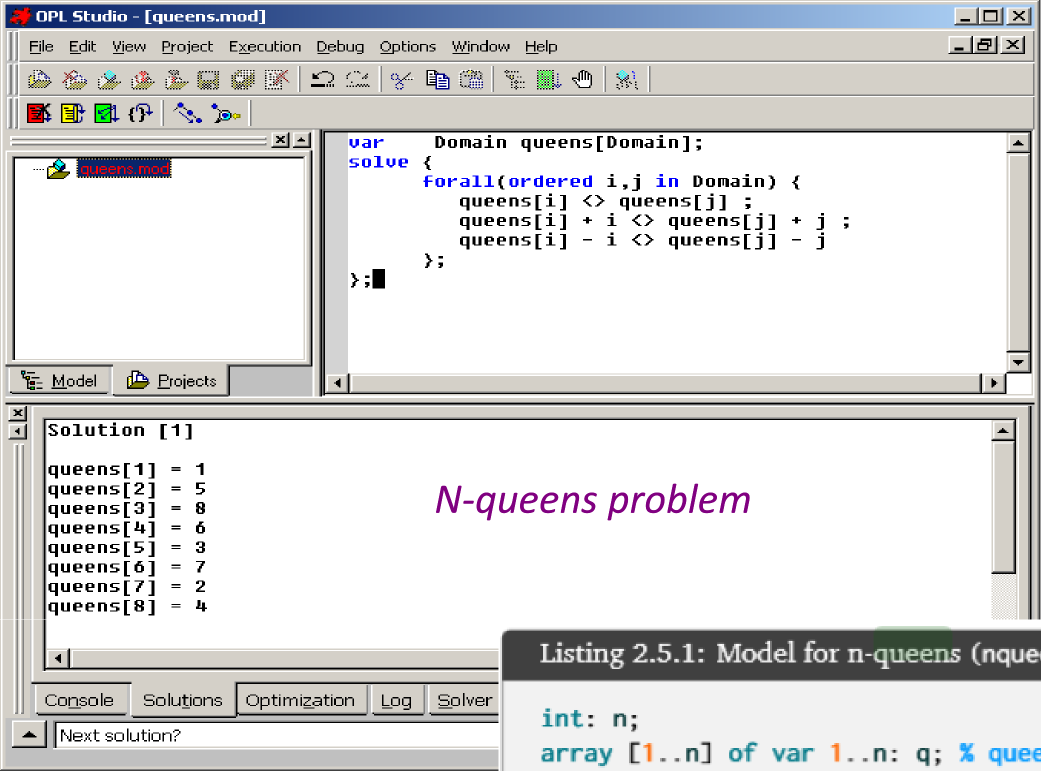
Herramientas Libres:

- Or-Tools (Google): https://developers.google.com/optimization/
- MiniZinc + Gecode: http://www.minizinc.org/
- Choco (http://choco-solver.org/)
- Repositorio:http://www.constraintsolving.com/

Herramientas Comerciales:

- CP-Optimizer (IBM) - CPLEX (ILOG/IBM)

Entorno CPLEX / CP-Optimizer



N-queens problem

Confle'x

```
######## CSP para 8x8 reinas
### VARIABLES ###
\vi : z1,z2,z3,z4,z5,z6,z7,z8
### RESTRICCIONES ###
\ci : rd1 , abs (z1 - z2) != 1 ;
\ci : rd2 , abs (z1 - z3) != 2 ;
\ci : rd3 , abs (z1 - z4) != 3 ;
\ci : rd4 , abs (z1 - z5) != 4 ;
\ci : rd5 , abs (z1 - z6) != 5 ;
\ci : rd6 , abs (z1 - z7) != 6 ;
\ci : rd7 , abs (z1 - z8) != 7 ;
\ci : rd10 , abs (z2 - z3) != 1 ;
\ci : rd11 , abs (z2 - z4) != 2 ;
\ci : rd12 , abs (z2 - z5) != 3 ;
\ci : rd13 , abs (z2 - z6) != 4 ;
\ci : rd14 , abs (z2 - z7) != 5 ;
\ci : rd15 , abs (z2 - z8) != 6 ;
\ci : rd18 , abs (z3 - z4) != 1 ;
\ci : rd19 , abs (z3 - z5) != 2 ;
\ci : rd20 , abs (z3 - z6) != 3 ;
\ci : rd21 , abs (z3 - z7) != 4 ;
\ci : rd22 , abs (z3 - z8) != 5 ;
\ci : rd25 , abs (z4 - z5) != 1 ;
\ci : rd26 , abs (z4 - z6) != 2 ;
\ci : rd27 , abs (z4 - z7) != 3 ;
\ci : rd28 , abs (z4 - z8) != 4 ;
\ci : rd31 , abs (z5 - z6) != 1 ;
\ci : rd32 , abs (z5 - z7) != 2 ;
\ci : rd33 , abs (z5 - z8) != 3 ;
\ci : rd36 , abs (z6 - z7) != 1 ;
\ci : rd37 , abs (z6 - z8) != 2 ;
\ci : rd40 , abs (z7 - z8) != 1 ;
\cim : ct1 , <>(z1,z2,z3,z4,z5,z6,z7,z8);
```

Entorno: MiniZinc

Listing 2.5.1: Model for n-queens (nqueens.mzn

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

include "alldifferent.mzn";

constraint alldifferent(q);                          % distinct rows
constraint alldifferent([ q[i] + i | i in 1..n]); % distinct diagonals

constraint alldifferent([ q[i] - i | i in 1..n]); % upwards+downwards

% search
solve :: int_search(q, first_fail, indomain_min, complete)
      satisfy;
output [ if fix(q[j]) == i then "Q" else "." endif ++
         if j == n then "\n" else "" endif | i,j in 1..n]
```

Herramientas CSP (optimización), MIP: LINGO

LINGO - Solution Report - LINGO1

File Edit LINGO Window Help

LINGO Model - LINGO1

```
MODEL:
!FUNCION OBJETIVO: Minimizar Distancia Reina2-Reina-4;
MIN = x4 - x2;

!RESTRICCIONES SOBRE FILAS;
X1 >= 1;
X1 <= 4;
X2 >= 1;
X2  <= 4;
X3 >= 1;
X3  <= 4;
X4 >= 1;
X4  <= 4;

!RESTRICCIONES SOBRE COLUMNAS xi <> xj;

X1 + 0.01 < X2  + 32000 * a1;
X2 + 0.01 < X1  + 32000 * (1

X1 + 0.01 < X3 + 32000 * a2;
X3 + 0.01 < X1 + 32000 * (1-

X1 + 0.01 < X4 + 32000 * a3;
X4 + 0.01 < X1 + 32000 * (1-

X2 + 0.01 < X3 + 32000 * a4;
X3 + 0.01 < X2 + 32000 * (1-

X2 + 0.01 < X4 + 32000 * a5;
X4 + 0.01 < X2 + 32000 * (1-

X3 + 0.01 < X4 + 32000 * a6;
X4 + 0.01 < X3 + 32000 * (1-
```

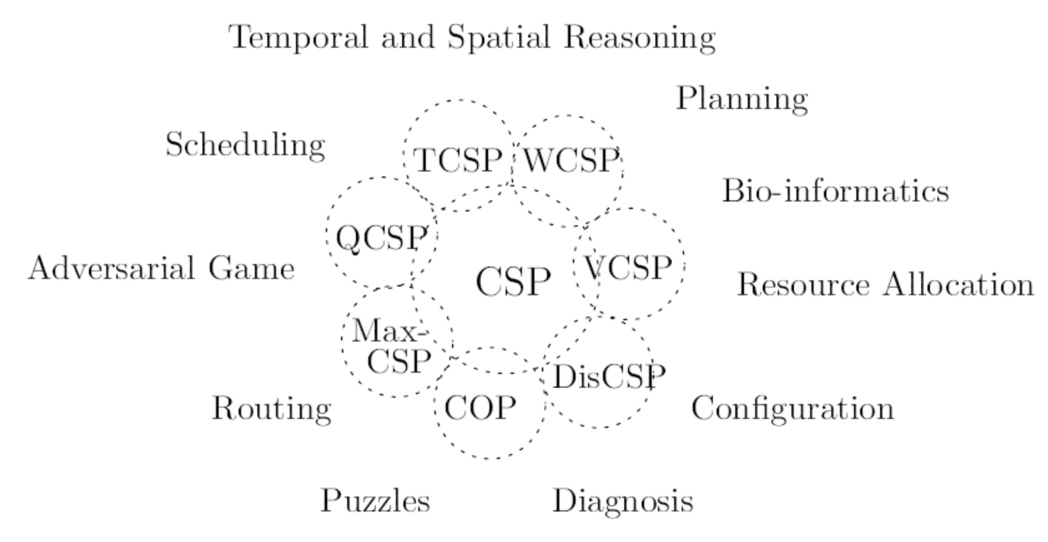
Solution Report - LINGO1

```
Global optimal solution found at step:        111
Objective value:                        -3.000000
Branch count:                                   3
```

Variable	Value	Reduced Cost
X4	1.000000	0.0000000
X2	4.000000	0.0000000
X1	1.010000	0.0000000
X3	3.020000	0.0000000
A1	0.0000000	0.0000000
A2	0.0000000	0.0000000
A3	1.000000	0.0000000
A4	1.000000	0.0000000
A5	1.000000	0.0000000
A6	1.000000	0.0000000

For Help, press F1 | NUM | Ln 7, Col 44 | 6:07 pm

6.8.- CSPs flexibles



Temas en desarrollo:
- Expresividad,
- Eficiencia,
- Optimalidad,
- Robustez, Flexibilidad, etc.

- Temporal CSP (TCSP): variables primitivas temporales (t_i, I_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 ($\exists x_j$).
- 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.

Ejemplo. Cena de clase/trabajo

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 **optimizacion** (NP-completo *vs.* NP-duro)?

CSPs flexibles (valuados)

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

$X = \{x_1, x_2, \ldots, x_n\}$ conjunto de variables, $D = \{D_1, D2, \ldots, D_n\}$ conjunto de dominios.

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

Una restricción dura (hard) *es:*

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

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

Cuestiones:

- Existen muchas soluciones, pero *¿qué soluciones son mejores que el resto?*
- No existen soluciones, pero *¿algunas asignaciones satisfacen más restricciones que otras?*
- No conocemos las restricciones exactas, sino solo *aproximadas* (difusas).
- ¿Estaríamos dispuestos a *violar algunas restricciones s*i a cambio pudiésemos obtener una solución mejor que las demás?

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, sino que incluyen funciones de coste (*$Coste_i$*)

$$\text{Restricción}_i \equiv \text{Relación}_i(x_1, x_2, \ldots, x_n),\quad Coste_i$$

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

La tupla $\{x_1=v_1, x_2=x_2, \ldots, x_k=v_k\}$, donde $\{x_1, x_2, \ldots, x_k\} \subseteq X$

tiene asociado un $Coste_i$ al aplicarse a la $Restricción_i$

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

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

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

Coste-Solución: $(X_t=\{x_1=v_1, x_2=x_2, \ldots, x_n=v_n\}) = Coste_1 \oplus Coste_2 \oplus \ldots \oplus Coste_m$

Formalmente, $\text{Coste}(X_t) = \bigoplus_{Ci \in C,\ var(Ci) \subseteq Xt} [\text{Coste}_i(X_t \downarrow_{var(Ci)})]$.

Esta función de coste da lugar a los diferentes CSP flexibles.

En un CSP clásico puede considerarse que $u \oplus v = u \wedge v$.

Objetivo: Encontrar una asignación con el mejor coste (**optimización**) combinado *(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 $Soft_k \Rightarrow Coste_k$ 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 $\Rightarrow C_1\ (Coste_1) \vee C_2\ (Coste_2) \vee \ldots \vee C_k(Coste_k)$

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)*

CSP posibilista

- Permite que algunas restricciones soft no se cumplan.
- El objetivo es minimizar el coste de insatisfacción de las restricciones soft.

Un *CSP Posibilista* es un CSP {X, D, C}, donde *(informalmente)*:

Cada restricción 'soft' tiene asociado un coste de insatisfabilidad [0,1], que *representa su coste cuando no es satisfecha*. Una restricción es 'hard' si coste=1.

Cada 'tupla' $t=\{x_1=v_1, x_2=v_2, .., x_n=v_n\}$, que *satisface todas las restricciones hard*, tiene un coste combinado de insatisfabilidad, sobre el conjunto de *restricciones soft no satisfechas*: ***u ⊕ v = max (u, v)***.

El objetivo es encontrar la tupla que *minimiza* el coste. Puede definirse un máximo nivel de insatisfabilidad (1).

Ejemplo:
Dado el CSP [X={x, y, z, t}, D={0, 10}, C_{SOFT}={(x > y, *0.5*) , (y > z, *0.3*), (x > t, *0.4*)}]:

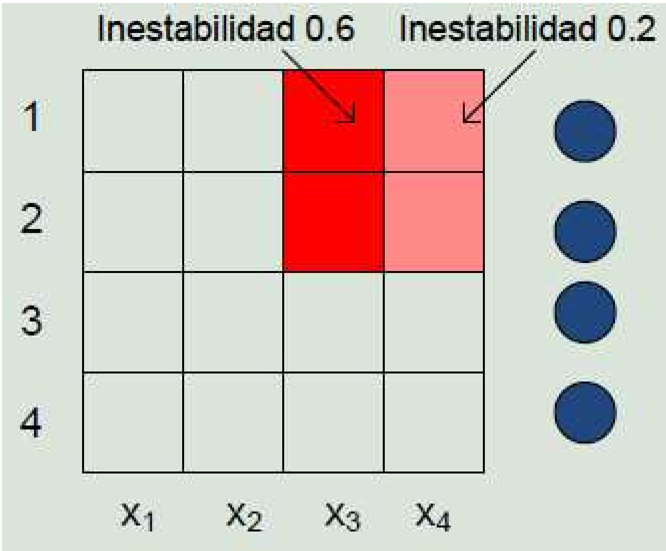
La tupla x=0, y=5, z = 3, t=1 tiene un coste *(insatisfabilidad)*: max (0.5 , 0.4) = 0.5

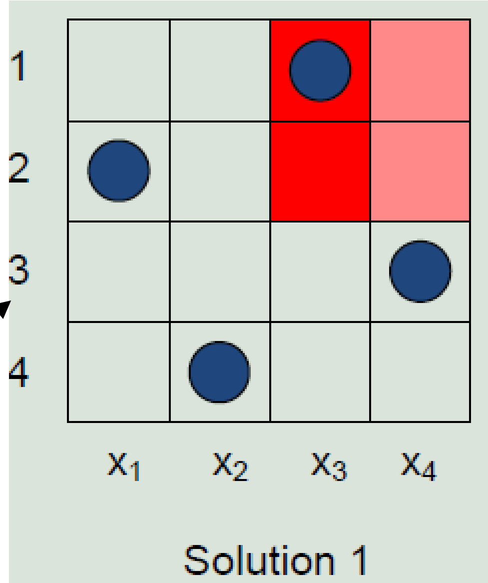
La tupla x=10, y=6, z = 7, t=1 tiene un coste *(insatisfabilidad)*: max (0.3) = 0.3

La segunda tupla es mejor solución (Insatisfabilidad = 0.3) ⇒ Satisfabilidad=1-0.3=0.7

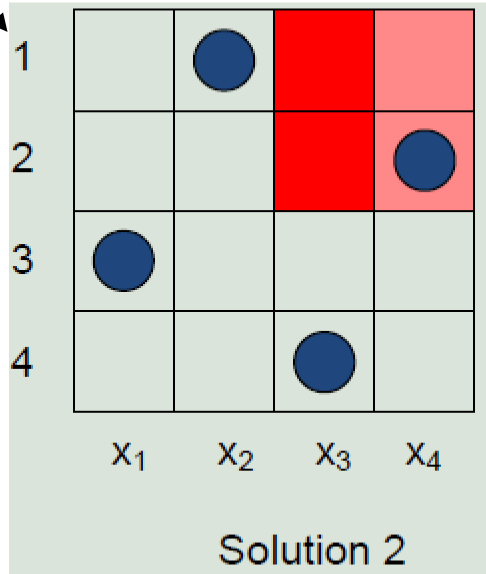
Una tupla con Satisfabilidad = 0 (Insatisfabilidad=1) no se considera solución

Ejemplo 4 reinas





Insatisf=0.6



Insatisf=0.2

mejor

Costes de Insatisfactibilidad

(restricciones soft)

$(x3 \neq 1)$, 0.6

$(x3 \neq 2)$, 0.6

$(x4 \neq 1)$, 0.2

$(x4 \neq 2)$, 0.2

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

CSP ponderado (Weighted CSP)

- Cada disyunción en restricciones disyuntivas (o cada valor alternativo de las variables en su dominio), tienen un peso (coste): *Es preferible la componente disyuntiva de menor valor.*
- El objetivo es obtener una solución con el **mínimo** peso combinado.

Un CSP Ponderado es un CSP {X, D, C}, donde *(informalmente)*:

Cada restricción (o disyuntiva en una restricción) tiene asociado un peso [0, P] asociado.

La combinación de costes de una tupla, sobre el conjunto de restricciones en las que satisface es:

$$u \oplus v = (u + v).$$

El objetivo es obtener la tupla con *mínimo valor de la función de coste*. Problema de **optimización.**

Una tupla con un coste final $\geq$ P no es válida.

Ejemplo:

Dado el CSP [X={x, y, z, t}, D={0, 10}, P=2.0,

C_{SOFT}={ $(x > y, 0.6) \vee (x \leq y, 0.4)$, $(y > z, 0.0) \vee (y \leq z, 1)$, $(x > t, 0.3) \vee (x \leq t, 0.7)$ }]:

La tupla x=0, y=5, z = 3, t=1 tiene un coste *(peso)*: (0.4 + 0.0 + 0.7) = 1.1

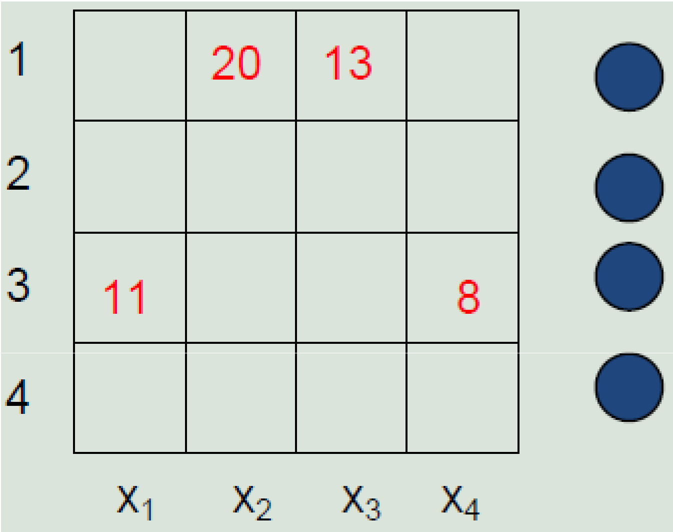
La tupla x=10, y=6, z = 7, t=1 tiene un coste *(peso)*: (0.6 + 1 + 0.3) = 1.9

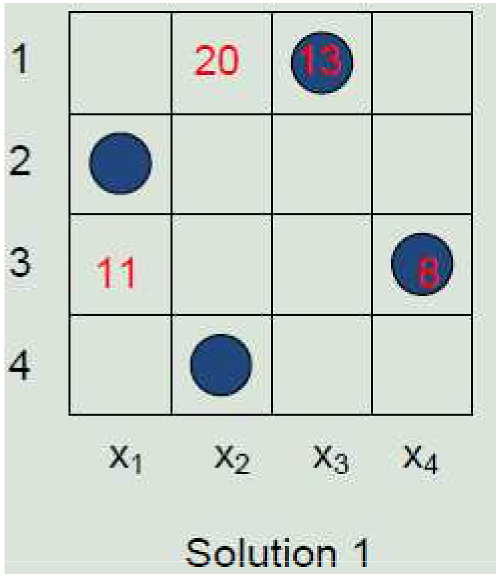
La tupla x=9, y=5, z = 7, t=10 tiene un coste *(peso)*: (0.6 + 1 + 0.7) = 2.3 *(*no válida; excede de P=2.0*)*

La primera tupla es mejor solución (coste= 1.1).

Aplicación a CSP de optimización

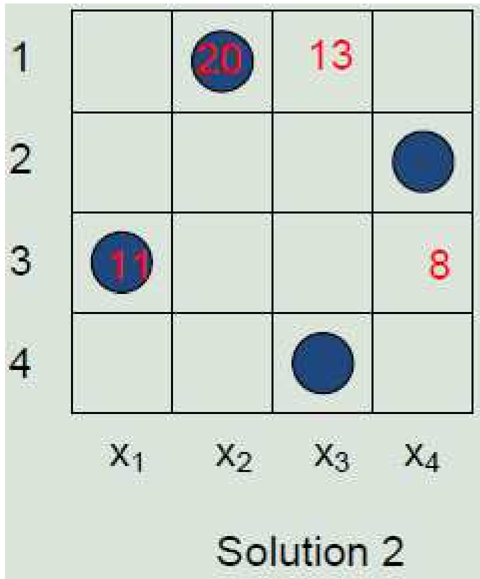
Ejemplo 4 reinas
(los pesos indican costes de las celdas)





Coste=21

mejor



Coste=31

Costes:

(x1 = 3), 11. Resto de valores es posible, con coste 0.
(x2 = 1), 20 . Resto de valores es posible, con coste 0.
(x3 = 1), 13. Resto de valores es posible, con coste 0.
(x4 = 3), 8. Resto de valores es posible, con coste 0.

CSP probabilístico

➤ Las restricciones tienen una probabilidad (de que 'existan' en el problema).

Un CSP Probabilístico es un CSP {X, D, C}, donde *(informalmente)*:

- Cada restricción 'soft' tiene asociado un peso [0, 1] asociado a su probabilidad.
- La combinación de costes de una tupla, sobre el conjunto de restricciones que *no satisface*, es:

$$u \oplus v = 1 - (1-u)\,(1-v).$$

- *El objetivo es obtener el menor coste (grado de satisfactibilidad)*

  Coste=0 indica satisfactibilidad total

  Coste=1 indica insatisfactibilidad total (solución no válida)

- Permite expresar algunos tipos de inferencia probabilística.

El coste de cada restricción representa la probabilidad de que exista dicha restricción.

En la solución, representa la probabilidad de que no sea solución al problema.

Ejemplo:

Dado el CSP [X={x, y}, D={0, 10}, C_{SOFT}={(x>y, *1*), (y>3, *0.6*), (x>10, *0.4*)}

La tupla (x=8, y=4), tiene un coste: 1 - (1 - 0.4) = 1 - 0.6 = 0.4 *(* mejor *)*

La tupla (x=8, y=1), tiene un coste: 1 - ((1 - 0.6) * (1 - 0.4)) = 1 - 0.24 = 0.76 *(* peor *)*

La tupla (x=1, y=4), tiene un coste: 1 - ((1 - 1) * (1 - 0.4)) = 1 - 0 = 1 *(* no válida, insatisfactible *)*

CSP difuso

- Restricciones difusas, definidas como una función continua [0, 1] de sus variables

Un CSP Difuso es un CSP {X, D, C}, donde *(informalmente)*:

Las restricciones son difusas, tal que no se interpretan a {T, F}, sino que son definidas como una función continua [0, 1] de las variables implicadas en la restricción.

La satisfactibilidad total es 1, la insatisfactibilidad es 0.

Ejemplo: $a \gg b$ / $a, b \in \{0, 100\}$, $\begin{cases} 0, & \text{si } a \leq b \\ (a-b)/10 & \text{si } b < a < b+10, \\ 1 & \text{si } a \geq b+10 \end{cases}$

Esta función representa el coste de la restricción: Satisfactibilidad

La combinación de costes de una tupla, sobre el conjunto de restricciones que satisface, es:

$$u \oplus v = \min(u, v).$$

La satisfactibilidad de una solución es la mínima sobre el conjunto de restricciones.

El objetivo es obtener la tupla con *máximo valor* de la función de coste (grado de satisfactibilidad). Puede definirse un mínimo nivel de satisfactibilidad.

Ejemplo:

Dado el CSP [X={x, y, z}, D={0, 10}, C_{SOFT}={(x >> y) , (y >> z)}]:

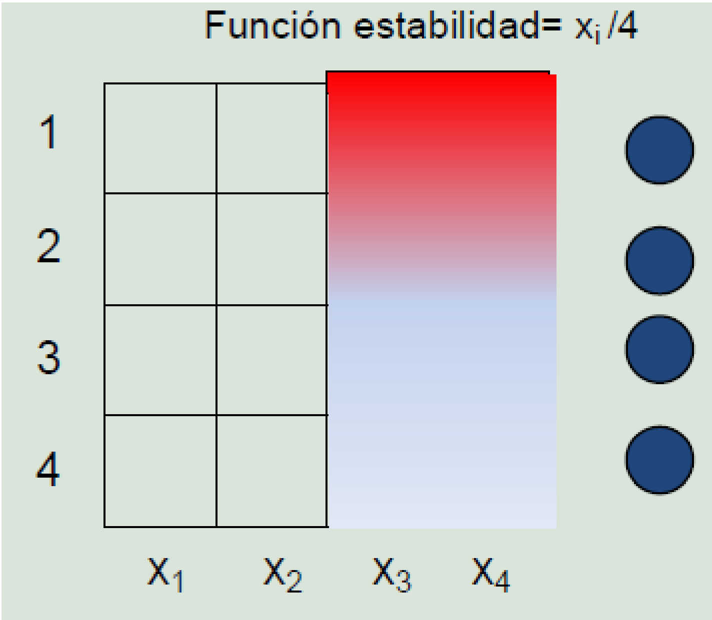
La tupla x=6, y=5, z=3 tiene un coste *(satisfactibilidad)*: min((6-5)/10, (5-3)/10) = min(0.1, 0.2)= 0.1

La tupla x=10, y=6, z=0 tiene un coste *(satisfactibilidad)*: min((10-6)/10, (6-0)/10)= min(0.4, 0.6) = 0.4

La segunda tupla es mejor solución (satisfactibilidad = 0.4).

Ejemplo CSP-difuso

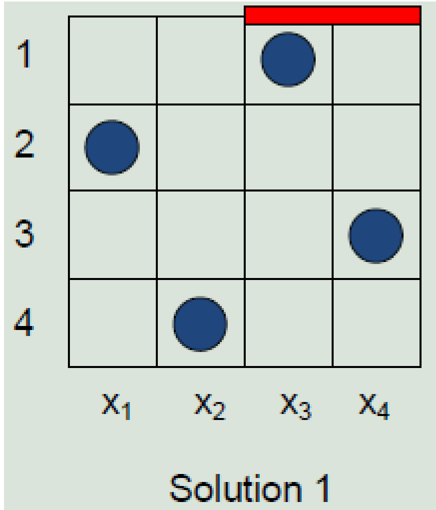
Ejemplo 4 reinas



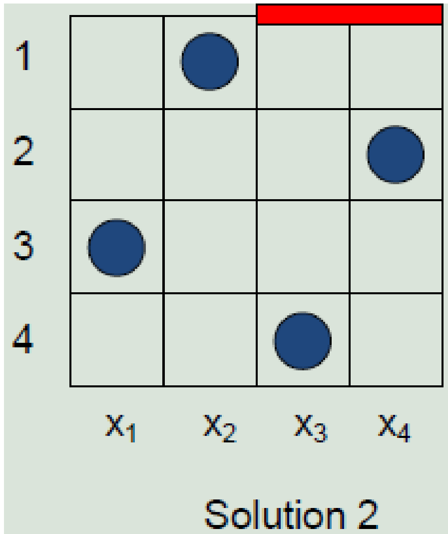
La función de estabilidad se asocia a los valores de x_3 y x_4, y penaliza los valores cercanos a la primera fila.

$x_3 >> 0 \;/\; x_3 \in \{1, 4\}, \quad x_3/4$

$x_4 >> 0 \;/\; x_4 \in \{1, 4\}, \quad x_4/4$



Coste=
min (1/4, 3/4)=
1/4



Coste=
min (4/4, 2/4) =
2/4

mejor

Métodos de Resolución de los CSP Valuados

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

Mejor solución para el CSP Flexible:

- CSP Posibilístico / CSP Probabilístico: Solución que minimiza el coste de la insatisfacción de restricciones.
- CSP Ponderado: Solución que minimiza el coste acumulado de las restricciones 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.

Con'flex implementa un tipo de CSP Difuso,
pero únicamente admite restricciones extensionales

Conflex implementa un tipo de CSP Difuso, únicamente con restricciones extensionales

Se puede especificar, opcionalmente, el **grado de satisfactibilidad (c)** de cada combinación de valores que simultáneamente pueden tomar las variables

\ce: <nombre-restriccion>

var_1	var_2	var_3		... ,
$valor_{11}$	$valor_{21}$	$valor_{31}$	...	**$[(c_1)]$**
$valor_{12}$	$valor_{22}$	$valor_{32}$	...	**$[(c_2)]$**
....				**$[(c_n)]$** ;

Combinaciones de valores posibles entre las variables

Ejemplo:

\ce: restriccion

Peso	Tamanyo	Coste ,	
50	grande	30.5	**(0.8)**
20	mediano	(10.5 [40.5, 50.1])	**(0.2)**
5	pequenyo	[90.1 90.5]	**(0.4);**

\ce: unaria-real X1, [0.0, 1.0] **(0.6)**;

Las combinaciones de valores no especificadas en la restricción son consideradas como NO VÁLIDAS (salvo /allbut)

\allbut:

Especifica que las combinaciones de valores posibles, **no especificadas** en la restricción, satisfacen totalmente la restricción (satisfactibilidad =1)

Ejemplo:

\alpha = 0.5; **#Corte de Satisfactibilidad (valor por defecto 0.5)**
\fuzzy_cut_step : 0.1; **#Precisión en la representación de la satisfactibilidad (valor por defecto 0.1)**

```
\vi = X,Y   1..2 ;
\ce:   C1            X          Y,
                     1          1    (0.2)
                     1          2    (1)
                     2          1    (1)
                     2          2    (0.6);
```

Es equivalente a:

```
\vi = X,Y   1..2 ;
\ce: C1              X          Y,
        \allbut
                     1          1    (0.2)
                     2          2    (0.6);
```

Todas las combinaciones posibles son válidas, con satisfactibilidad 1, excepto…

Ejemplo-1

\vi = X 1..2 ;
\vi = Y 1..2 ;

\ce: C1 X Y,
\allbut
1 1 (0.2)
2 2 (0.6);

\ce: C2 X Y,
1 1 (0.8)
2 2 (0.9)
1 2 (0.3);

\alpha = 0.5;
\fuzzy_cut_step : 0.1;

SOLUTION No 1
X = 2 Y = 2, sat = 0.6

Las restricciones son aditivas. Por tanto, el grado de satisfactibilidad de 2 2 es min(0.6, 0.9) = 0.6, que es mayor que \alpha = 0.5

Soluciones con sat<0.5 no son válidas

\alpha = 0.1;
\fuzzy_cut_step : 0.1;

SOLUTION No 1
X = 1 Y = 1 sat = 0.2

SOLUTION No 2
X = 1 Y = 2, sat = 0.3

SOLUTION No 3
X = 2 Y = 2, sat = 0.6

Ejemplo-2

Dadas las variables enteras (A, B), con dominios {0,..,5} y la variable real X con dominio [2.1,.., 4.2]:

el par de valores para las variables A, B: (0, 0) está excluido (pues tiene grado 0),

el par (0,1) es válido en el intervalo de X [2.1, 2.9] con grado 0.7,

en el intervalo de X [2.901, 3.4] con grado 0.8,

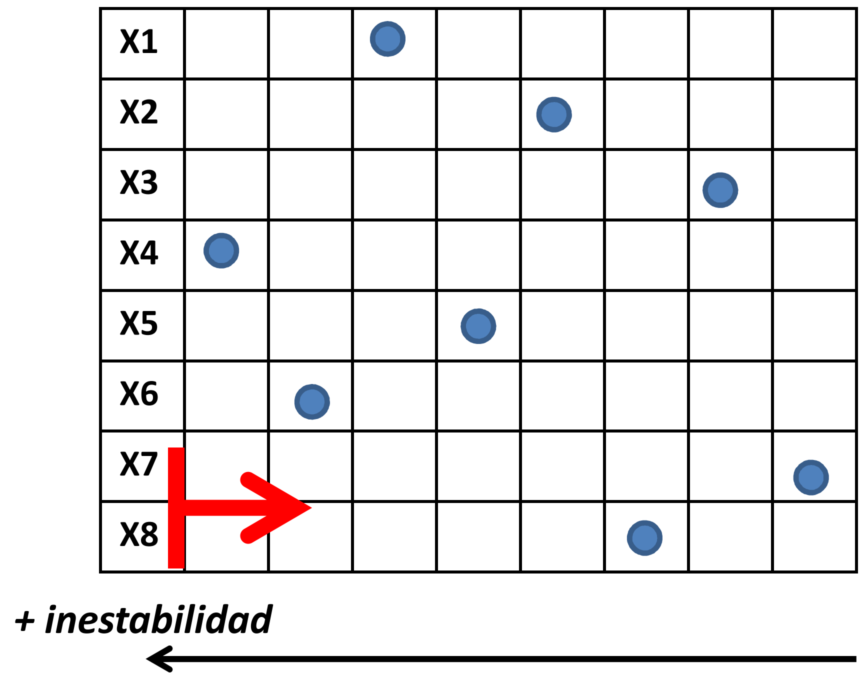
y con grado 1 en el restante dominio de X.

todos los demás valores son completamente posibles (tienen grado 1).

```
\vr = X  0.1 [2.1, 4.2];
\vi = A  0 .. 5 ;
\vi = B  0 .. 5 ;

\ce:  C1     A    B         X ,
   \allbut
             0    0     ( [2.1, 4.2])    (0)
             0    1     ( [2.1, 2.9,          (0.7)]
                          [2.901, 3.400,     (0.8)]
                          [3.5, 4.2,            (1)]  )     (1);
```

Ejemplo en Con'flex. Priorizamos soluciones donde X7, x8 están alejadas de la primera columna

```
\alpha = 0.75;   #corte de las soluciones

\search : rfla , all_solutions;

\vi : Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8 1..8 ;

###     CONTRAINTES     ###
# resto restricciones 8-reinas

\ce:  C1    Z7,   \allbut
                8        (1)
                7        (0.9)
                6        (0.8)
                5        (0.7)
                4        (0.6)
                3        (0.5)
                2        (0.4)
                1        (0.3) ;

\ce:  C2    Z8,   \allbut
                8        (1)
                7        (0.9)
                6        (0.8)
                5        (0.7)
                4        (0.6)
                3        (0.5)
                2        (0.4)
                1        (0.3) ;
```




```
SOLUTION No 1
 Z1 = 3
 Z2 = 5
 Z3 = 7
 Z4 = 1
 Z5 = 4
 Z6 = 2
 Z7 = 8
 Z8 = 6
 sat = 0.790 .
 (trouvee apres 32 instanciations et 179 tests de contraintes)
```

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**