ILOG SOLVER

Problemas de dominios finitos

Cambio de mentalidad

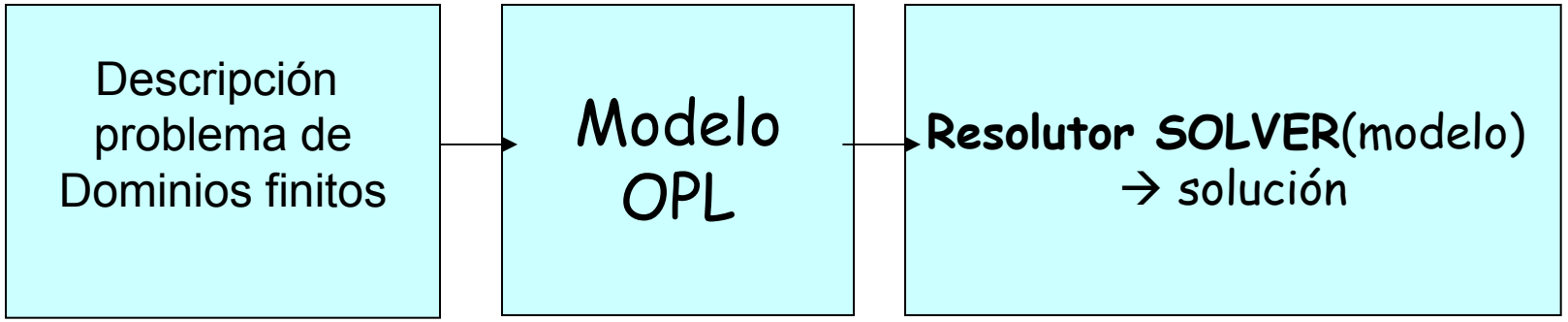
- Dominio de las Variables de decisión.
- Las ecuaciones a las que podemos hacer frente.
- Problemas de satisfacción de restricciones.
- Procedimientos de búsqueda:

  Especial relevancia.
- No parchearemos métodos de resolución CPLEX:

  ¡Hace falta nueva mentalidad!

  Búsqueda + Propagación de restricciones

Ciclo de vida



Estructura de un modelo.

1. Datos
2. Variables
   - Toman valores de un dominio.
3. Restricciones
   - Condiciones en la combinación de valores.
4. Procedimiento búsqueda

Estructura del resolutor

1. Objetivo:
   - Encontrar combinación de valores que satisfaga todas las restricciones.

2. Fases:
   - Propagación inicial de restricciones.
   - Búsqueda.

Índice

- Ejemplo sencillo:
  - Etapa resolución
- Ejemplos más complicados:
  - Etapa modelado

Ejemplo sencillo. Descripción

- En una mesa hay:

  - Número de lápices. Pueden ir desde cinco hasta doce.

  - Número de gomas. Pueden ir desde dos hasta diecisiete.

- La suma de lápices es cinco unidades mayor que el de gomas. Entre ambos hay diecisiete unidades.

- ¿Cuántos lápices y gomas hay?

Ejemplo sencillo. Modelo

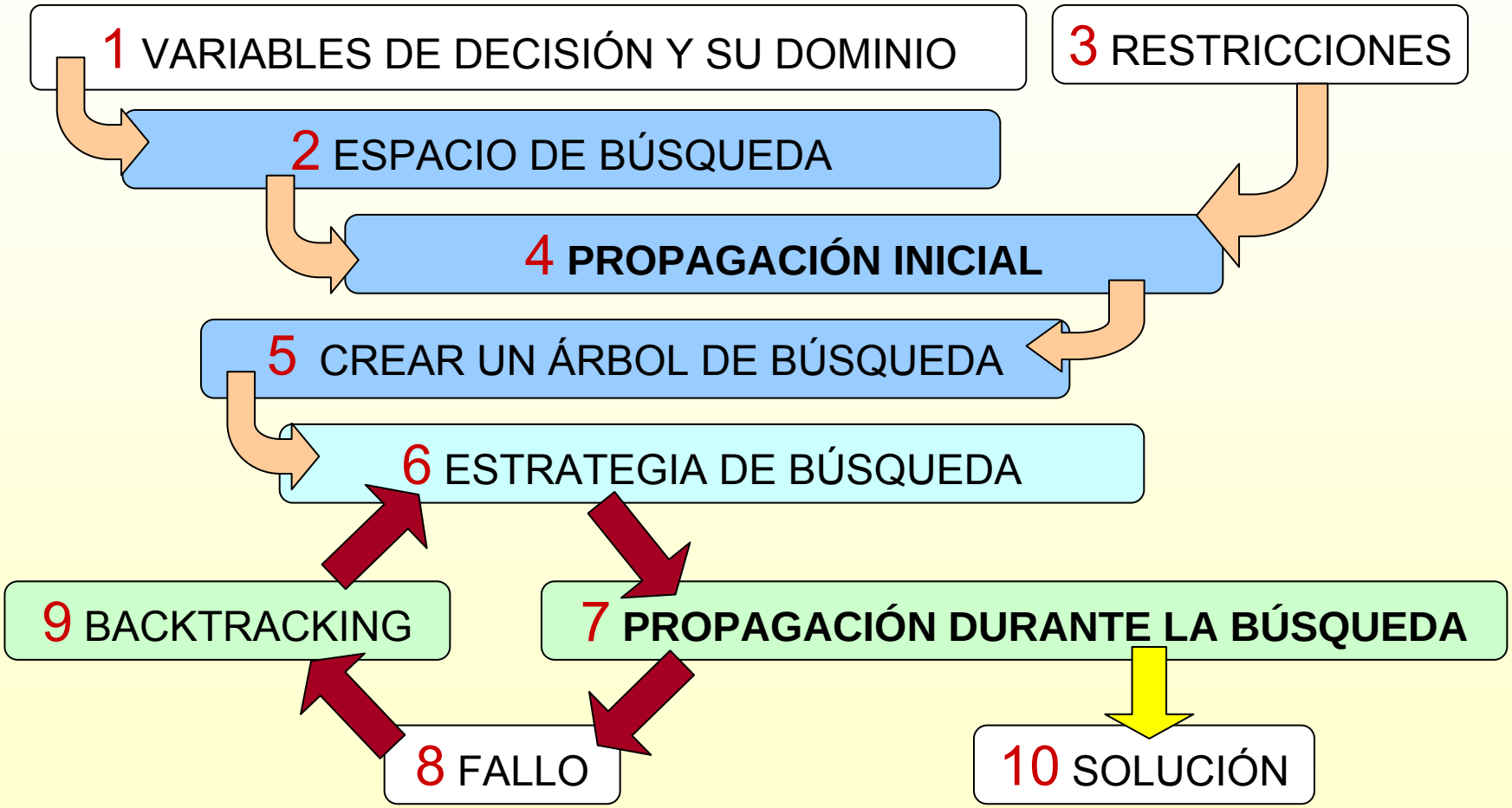
VARIABLES DE DECISIÓN

```
var int X in 5..12;
var int Y in 2..17;
```

RESTRICCIONES DEL PROBLEMA

```
solve{
    X + Y = 17;
    X - Y =5;
};
```

Ejemplo sencillo. Resolución usando SOLVER



Resolución usando SOLVER

1 Variables de decisión y su dominio

```
var int X in 5..12;
var int Y in 2..17;
```

Resolución usando SOLVER

2. Espacio de búsqueda

Todas las posibles combinaciones de valores de las variables de decisión.

X puede tomar 11 valores.

Y puede tomar 16 valores.

176 combinaciones (candidatos a solución)

Resolución usando SOLVER

2. Espacio de búsqueda

X = 5 Y = 2	X = 5 Y = 3	X = 5 Y = 4	X = 5 Y = 6	…	X = 5 Y = 17
X = 6 Y = 2	X = 6 Y = 3	X = 6 Y = 4	X = 6 Y = 5	…	X = 6 Y = 17
…	…	…	…	…	…
X = 12 Y = 2	X = 12 Y = 3	X = 12 Y = 4	X = 12 Y = 5	...	X = 12 Y = 17

Resolución usando SOLVER

3. Restricciones

$X + Y = 17;$

$X - Y = 5;$

Resolución usando SOLVER

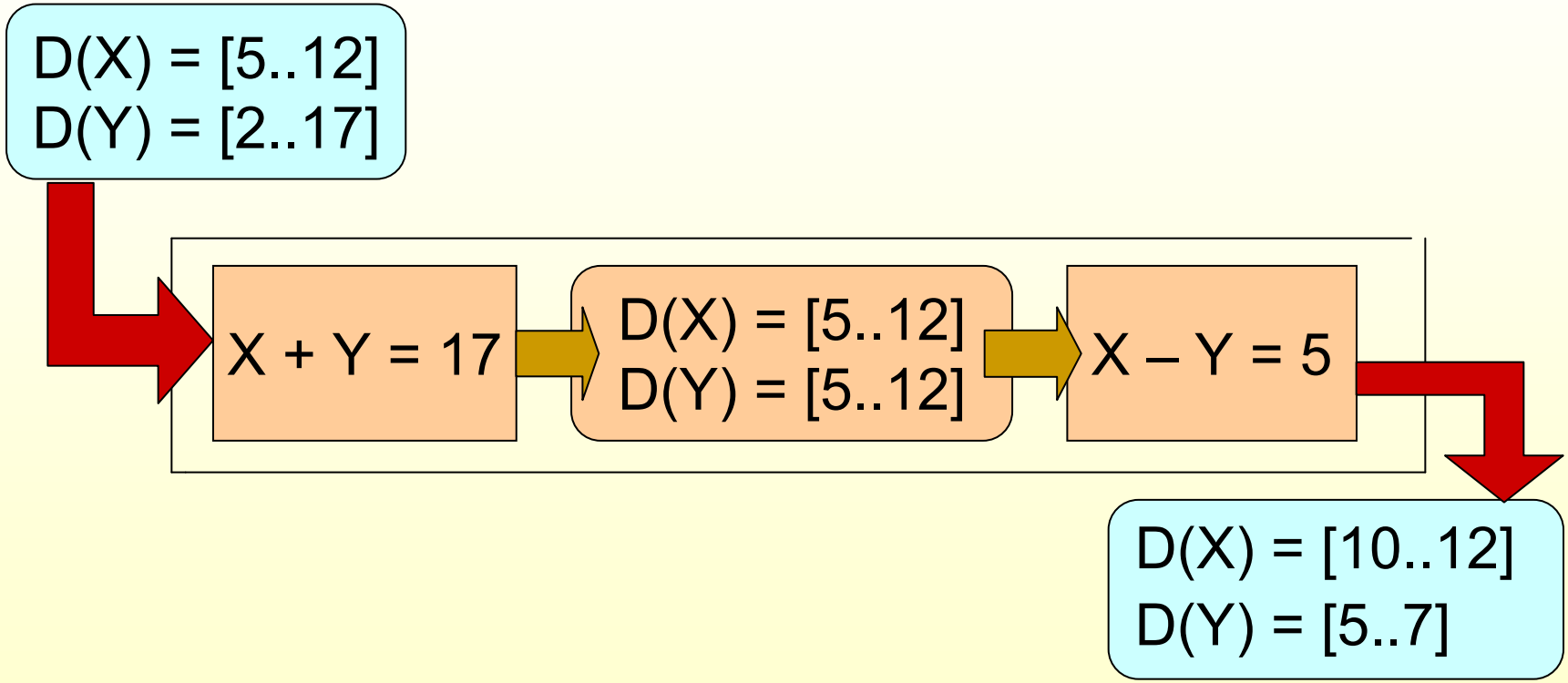
4. Propagación inicial de restricciones

Ideas:

- Variables con dominio
- Candidato a solución
- Restricciones

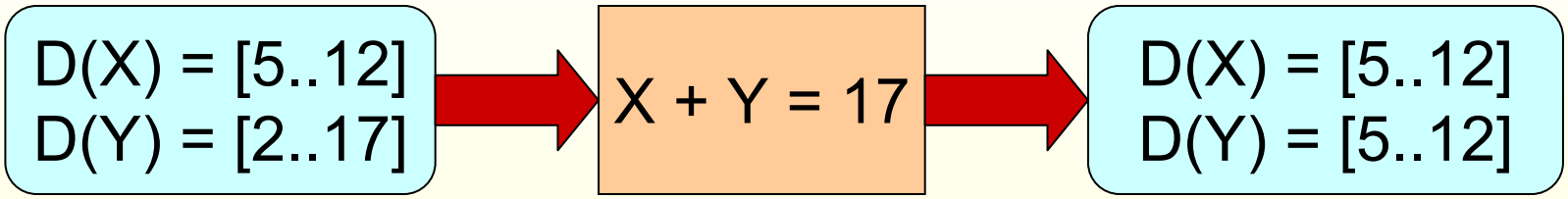
- ✓ Propagación Inicial → Podar valores del dominio de las variables que nunca formarán parte de una solución.

Resolución usando SOLVER



Resolución usando SOLVER

4. Propagación inicial de restricciones



¿Qué se está razonando?

Límite consistencia

Y = 17 - X

Y <= 17 – min(X) Y >= 17 – max(X)

X = 17 - Y

X <= 17 – min(Y) X >= 17 – max(Y)

Resolución usando SOLVER

- Reducción del espacio de búsqueda tras 4

X = 10 Y = 5	X = 10 Y = 6	X = 10 Y = 7
X = 11 Y = 5	X = 11 Y = 6	X = 11 Y = 7
X = 12 Y = 5	X = 12 Y = 6	X = 12 Y = 7

Otros ejemplos de propagadores

4. Propagación inicial de restricciones

Restricción : X = Y + Z

Razonamos

X = Y + Z

X <= max(Y) + max(Z) X >= min(Y) + min(Z)

Y = X – Z

Y <= max(X) - min(Z) Y >= min(X) - max(Z)

Z = X – Y

Z <= max(X) - min(Y) Z >= min(X) - max(Y)

Otros ejemplos de propagadores

4. Propagación inicial de restricciones

Restricción : X <> Y

Sólo hay propagación si una variable toma un valor fijo e igual al mínimo o máximo de la otra variable.

Otros ejemplos de propagadores

4. Propagación inicial de restricciones

Restricción : X = Y * X

Si todas las variables son positivas…

Razonamos

X = Y * Z

X <= max(Y) * max(Z) X >= min(Y) * min(Z)

Y = X / Z

Y <= max(X) / min(Z) Y >= min(X) / max(Z)

Z = X / Y

Z <= max(X) / min(Y) Z >= min(X) / max(Y)

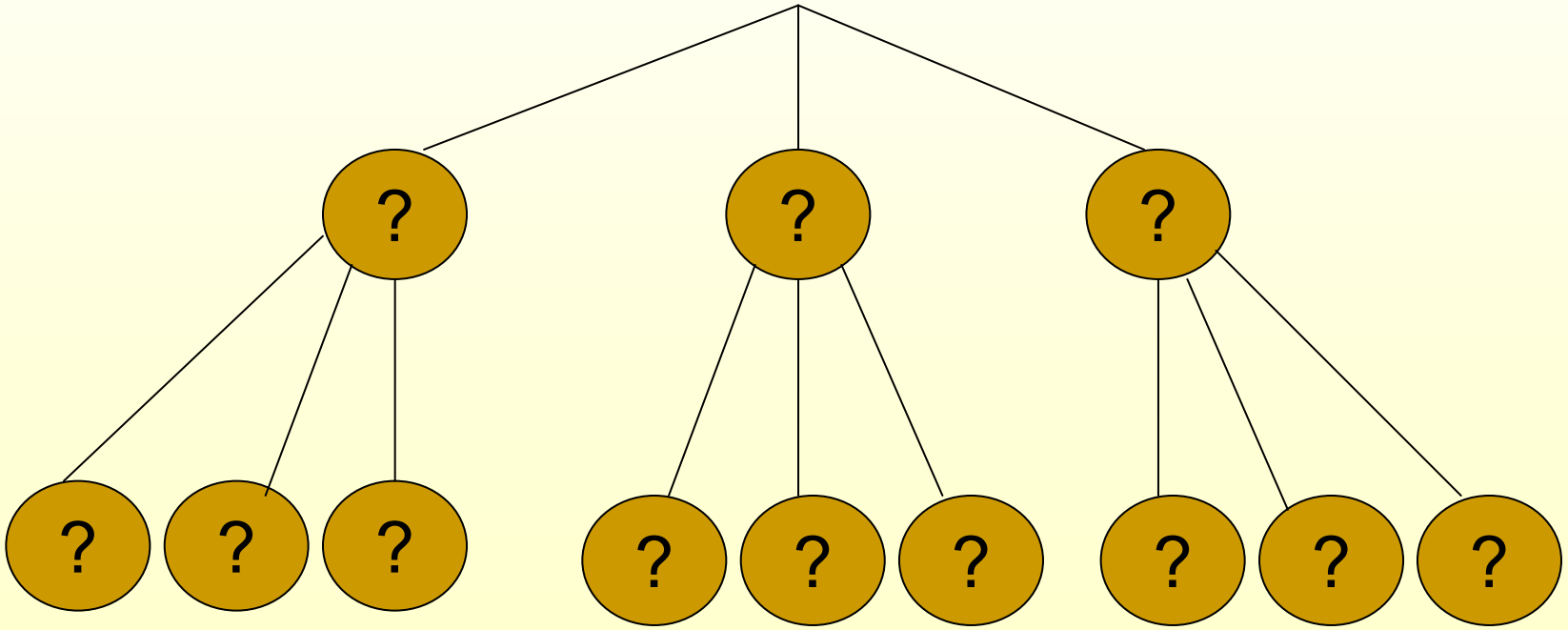
Resolución usando SOLVER

6. Estrategia de búsqueda

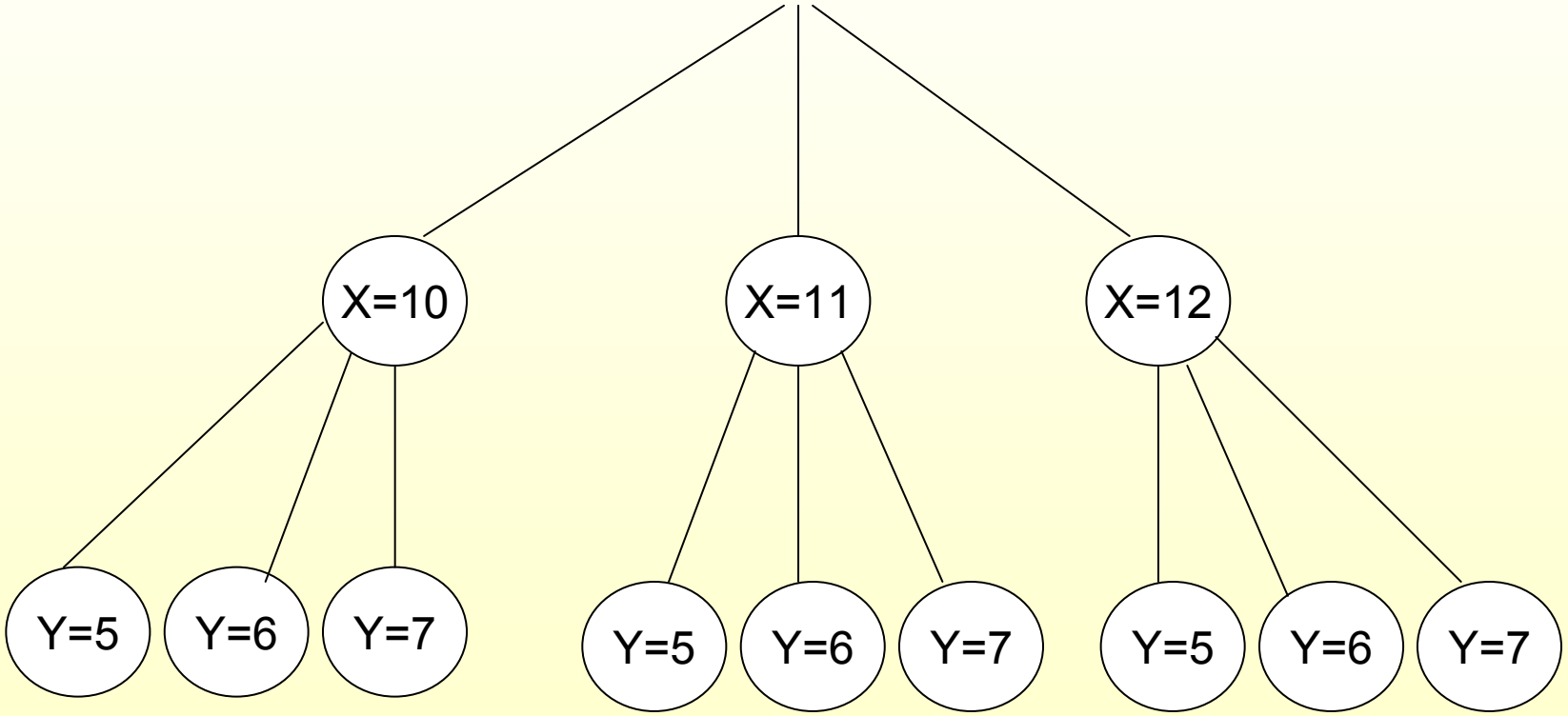
- Idea:
- Si la exploración del árbol depende de su estructura entonces la estrategia es, en sí misma, la propia creación del árbol.

Resolución usando SOLVER

Estrategia de búsqueda en profundidad, explorando de izquierda a derecha.



Resolución usando SOLVER



Resolución usando SOLVER

Iteramos:

- Exploración hasta encontrar un candidato.
- 7 Propagación de restricciones sobre el candidato.
  - 10 Éxito → Solución.

    ¿Quieres más soluciones?
  - 8 Fallo → Inconsistencia del resolutor.

    9 Backtracking.

Resolución usando SOLVER

Exploración hasta encontrar un candidato:

X = 10, Y = 5

7. Propagación de restricciones durante el proceso de búsqueda

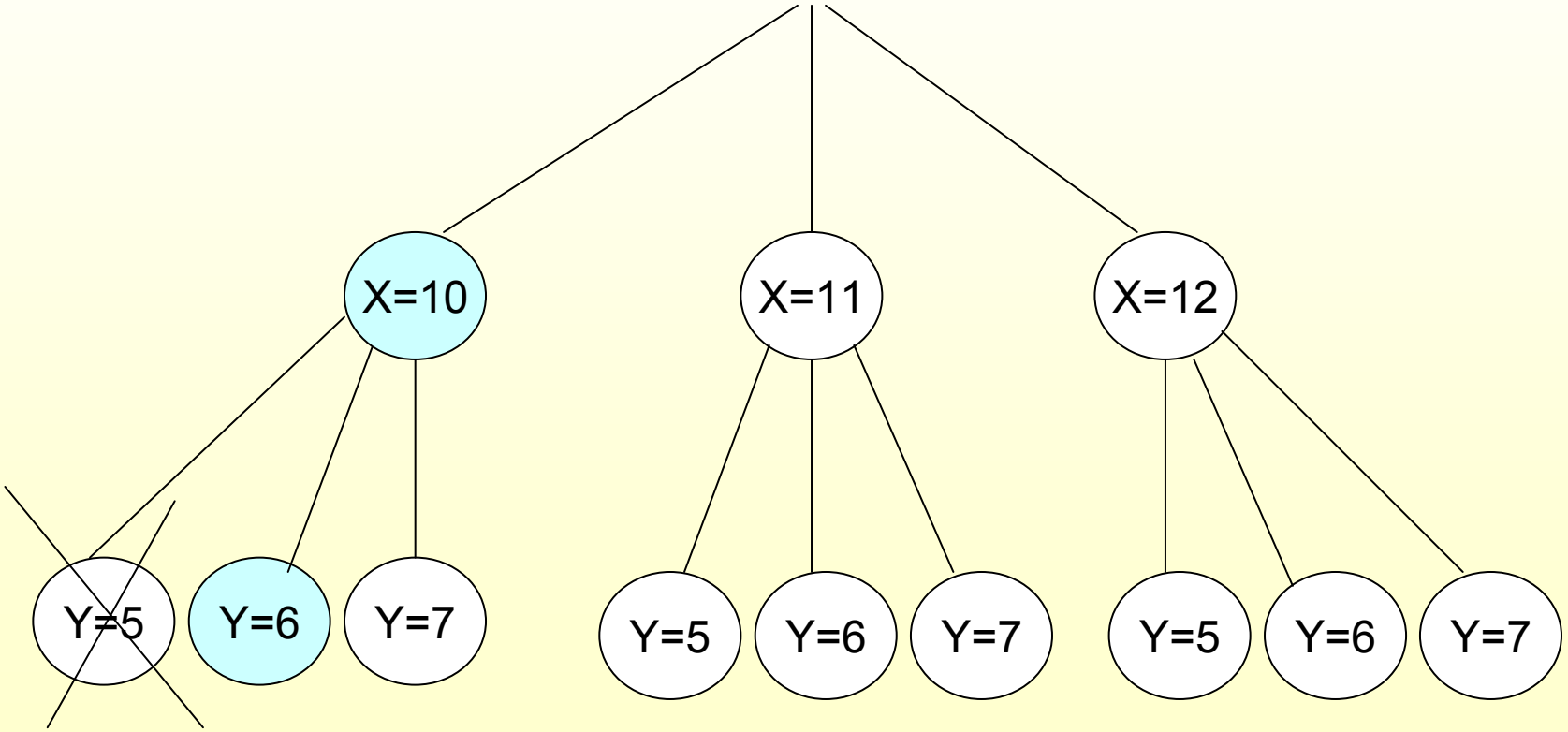
- ☒ X + Y = 17
- ☑ X – Y = 5

8.¡Fallo!

9. Backtracking en el árbol en busca del siguiente candidato.

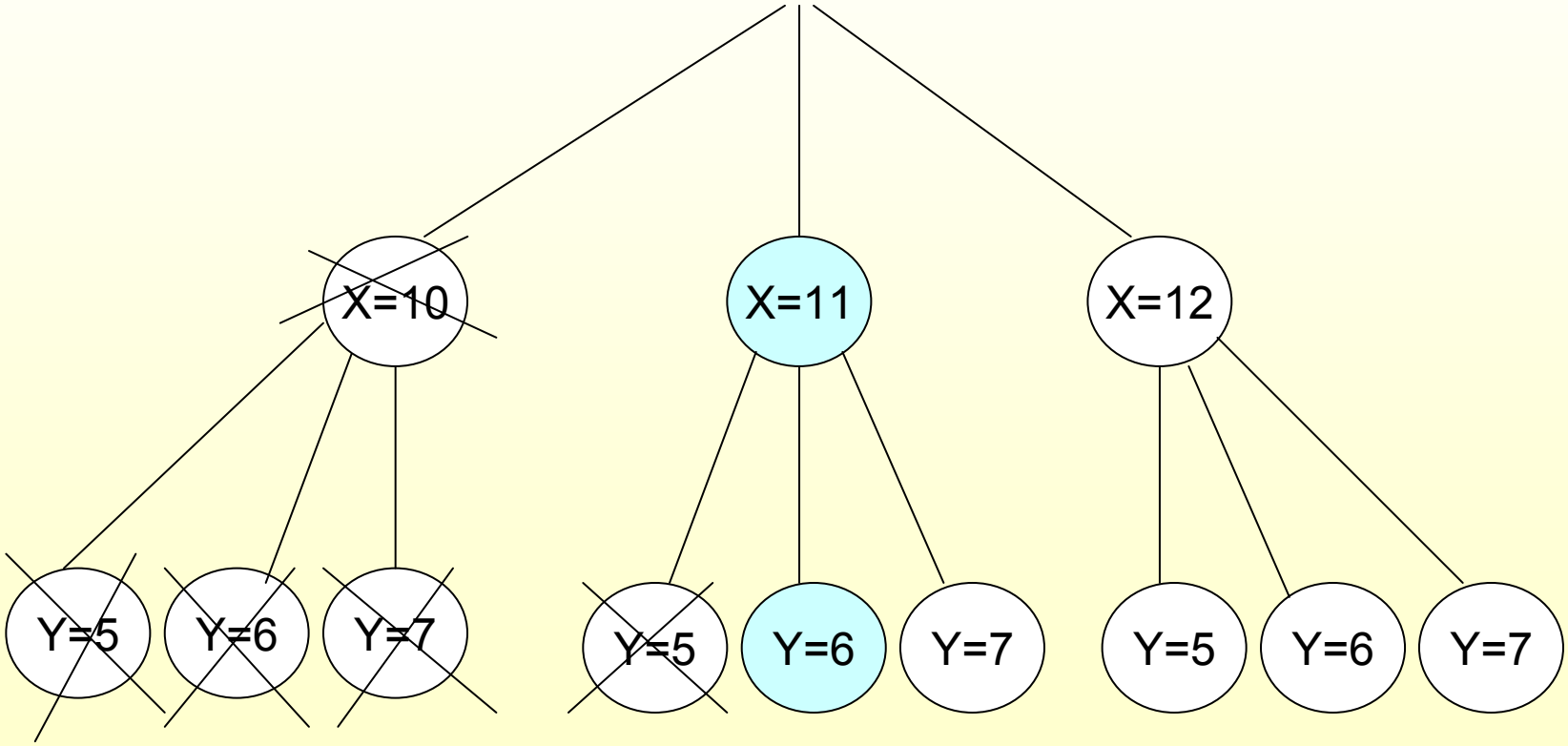
Resolución usando SOLVER

9. Backtracking en busca de otro candidato



Resolución usando SOLVER

… tras varios fallos



Resolución usando SOLVER

Exploración hasta encontrar un candidato:

X = 11, Y = 6

7. Propagación de restricciones durante el proceso de búsqueda

- ☑ X + Y = 17
- ☑ X – Y = 5

10.Éxito!

1ª y única Solución: X = 11, Y = 6.

Ejemplos modelado

- Taller de coches →
  - Ejemplo completo
- N reinas →
  - Introducción a las búsquedas
- Cuadrado relleno
  - Técnicas de Búsquedas
- Series mágicas →
  - Restricciones de alto nivel
- Matrimonios →
  - Fuerte expresividad

Problema del taller de coches

Descripción

- Problema ambientado en un taller de coches.
- El coste de un día de trabajo en el taller es elevado. Luz, consumo eléctrico, sueldos de operarios…
- Hay un coste fijo por hora de funcionamiento del taller.

Problema del taller de coches

- Cada día llegan *n* coches para ser puestos a punto.
- Según la marca de cada coche (clase) necesitará unos últimos retoques u otros.
- Los coches pasan en orden por cinta transportadora de *m* etapas. (1 etapa/hora).

Problema del taller de coches

- Las máquinas que los ponen a punto están situadas a los lados de la cinta.
- Cada máquina tiene un rango de acción y un ritmo de trabajo.

- Objetivo:

  Encontrar, si lo hay, un orden en el cual los coches entran de uno en uno a la cinta sin que haya ningún turno en el que no entra ninguno.

Modelo. Datos del problema

- ¿Qué necesitamos y que NÓ necesitamos?
  - ✓ Número total de coches.
  - ✓ Número de máquinas.
  - ✓ Número de clases diferentes de coches.
  - ✓ Número de coches de cada clase.

  - ✗ Número de etapas.
  - ✗ Posición de trabajo de cada máquina.

Modelo. Datos del problema

- ¿Qué necesitamos y que NÓ necesitamos?
  - ✓ Ritmo de trabajo de cada máquina.
  - ✓ Que máquinas precisa cada clase de coche.

  - ✓ Número total de coches que requerirán cada máquina

Modelo. Datos del problema.

```
int  num_coches   = ...;
enum Clases = ...;
enum Maquinas = ...;
range Coches 1..num_coches;
//Instancia
num_coches = 10;
Clases = {Renault, Fiat, Opel, Seat, Citroen, Ford};
Maquinas = {Aire, Radio, Volante, Ruedas,
                        Maletero};
```

Modelo. Datos del problema

```
int demanda[Clases] = ...;
int maquina_en_clases[Maquinas,Clases] = ...;
//Instancia
demanda = [1, 1, 2, 2, 2, 2];
maquina_en_clases = #[
    Aire: [ 1, 0, 0, 0, 1, 1],
    Radio: [ 0, 0, 1, 1, 0, 1],
    Volante: [ 1, 0, 0, 0, 1, 0],
    Ruedas: [ 1, 1, 0, 1, 0, 0],
    Maletero: [ 0, 0, 1, 0, 0, 0] ]# ;
```

Modelo. Datos del problema

```
struct Ritmo {
   int l;
   int u;  };
Ritmo Ritmo_maquina[Maquinas] = ...;
//Instancia
Ritmo_maquina = #[
      Aire : <1,2>,
      Radio : <2,3>,
      Volante : <1,3>,
      Ruedas : <2,5>,
      Maletero :<1,5>]# ;
```

Modelo. Datos del problema

Saber el número total de coches que requerirá cada máquina nos simplificará después modelar algunas restricciones.

```
int demanda_Maquinas[i in Maquinas] =
    sum(j in Clases) demanda[j] * maquina_en_clases[i,j];
```

Modelo. Variables de decisión

¿Cuál es la información desconocida del problema?

- El orden en que entra cada coche en la cinta.
- Distinguimos por clases de coche, no por total de coches.

var Clases orden[Coches];

Modelo. Variables de decisión

Variables redundantes.

- Nos simplificará la escritura de restricciones.
- Nos ayudarán (sorprendentemente, ¿no?) a aumentar la eficiencia del problema.

var int Maquinas_Coches[Maquinas,Coches] in 0..1;

Modelo. Restricciones

- El número de coches de una clase que hay que poner a punto es x.
- El número de coches de una clase que hay en la variable de decisión es y
- x = y

```
forall(c in Clases )
  sum(s in Coches ) (orden[s] = c) =  demanda[c];
```

Modelo. Restricciones

Toda subcadena de la solución respete el ritmo de trabajo de una máquina.

```
forall(o in Maquinas & s in [1..num_coches - Ritmo_maquina[o].u + 1] )
    sum(j in [s .. s + Ritmo_maquina[o].u - 1]) Maquinas_Coches[o,j]
            <= Ritmo_maquina[o].l;
```

¡Lo logramos expresar gracias a la variable de decisión de apoyo!

Modelo. Restricciones

Restricciones redundantes.

No eliminarán ninguna combinación de variables válida pero sí que ayudarán a reducir el árbol de búsqueda más rápido → Aumenta eficiencia.

Modelo. Restricciones

Relación entre los dos grupos de variables de decisión, para que la solución sea consistente.

Maquinas_coche[i, j] = 1 ⇔ Si el coche que entrará en la posición j requiere la máquina i

```
forall(o in Maquinas & s in Coches )
 Maquinas_Coches[o,s] =   maquina_en_clases[o,orden[s]];
```

Modelo. Restricciones

- Explotar:
  - Numero de coches que van a requerir la máquina *m*.
  - Ritmo de trabajo de la máquina *m*.
- Así nos iremos anticipando a cuando una posible exploración del árbol NUNCA va a conducir a una solución.
- ¡Detectarlo cuanto antes!

Modelo. Restricciones

-Por ejemplo 6 coches van a requerir usar la máquina Aire.

-La máquina Aire tiene un ritmo de trabajo <2,3>.

- Cuando queden 3 coches → 4 que usen Aire han tenido que ser ya escogidos.
- Cuando queden 6 coches → 2 que usen Aire han tenido que ser ya escogidos.

```
forall(o in Maquinas & i in [1..demanda_Maquinas[o]])
    sum(s in [ 1 .. num_coches - i * Ritmo_maquina[o].u])
          Maquinas_Coches[o,s]  >=
                demanda_Maquinas[o] - i * Ritmo_maquina[o].l;
```

Modelo. Procedimiento de búsqueda

- First fail

```
search{
   generateSize(orden);
};
```

Resultados OPL

Solutions

Solution [1]

orden[1] = Renault, orden[2] = Fiat, orden[3] = Ford,
orden[4] = Opel, orden[5] = Citroen, orden[6] = Seat,
orden[7] = Seat, orden[8] = Citroen, orden[9] = Opel,
orden[10] = Ford

Maquinas_Coches[Aire,1] = 1
Maquinas_Coches[Aire,2] = 0
Maquinas_Coches[Aire,3] = 1
Maquinas_Coches[Aire,4] = 0
...

Resultados OPL

Log

Running

Solver

displaying solution ...

SOLVER

Variables: 110 Constraints: 113

Failures: 0 **Choice points: 2**

Solver memory: 300520 Solving time: 0.05

Alternativa sin variables ni restricciones redundantes.

- No nos hacen falta var Maquinas_coches.
- Sólo quedarán la primera restricción y la segunda cambiada.

Variables de decisión.

var Clases orden[Coches];

Alternativa sin variables ni restricciones redundantes.

Restricciones

- forall(c in Clases)

  sum(s in Coches) (orden[s] = c) = demanda[c];

forall(o in Maquinas & s in [1..num_coches - Ritmo_maquina[o].u + 1])

sum(j in [s .. s + Ritmo_maquina[o].u - 1]) Maquinas_Coches[o,j] <= Ritmo_maquina[o].l;

- forall(o in Maquinas & s in [1..num_coches - Ritmo_maquina[o].u + 1])

  sum(j in [s .. s + Ritmo_maquina[o].u - 1]) maquina_en_clases[o,orden[j]]<= Ritmo_maquina[o].l;

Comparativa restricciones redundantes

SÍ NO

Hay 6 posibles soluciones.

Fallos	*C.Points*	*Fallos*	*C.Points*
0	2	1	3
2	6	3	7
2	6	3	7
16	21	71	76
24	29	89	95
27	33	97	104

Ejemplos modelado

- Taller de coches →
  - Ejemplo completo
- N reinas →
  - Introducción a las búsquedas
- Cuadrado relleno
  - Técnicas de Búsquedas
- Series mágicas →
  - Restricciones de alto nivel
- Matrimonios →
  - Fuerte expresividad

Problema de las reinas

Descripción

Colocar n reinas en un tablero de *n x n* de tal manera que no se coman entre ellas.

Modelo. Datos del problema

Dimensión del problema.

```
int    n << "Numero de reinas:";
range  Dominio 1..n;
```

Modelo. Variables de decisión

Un array con n posiciones.

El índice de la posición i representa la fila en la que colocar a la reina de la columna i.

var Dominio reinas[Dominio];

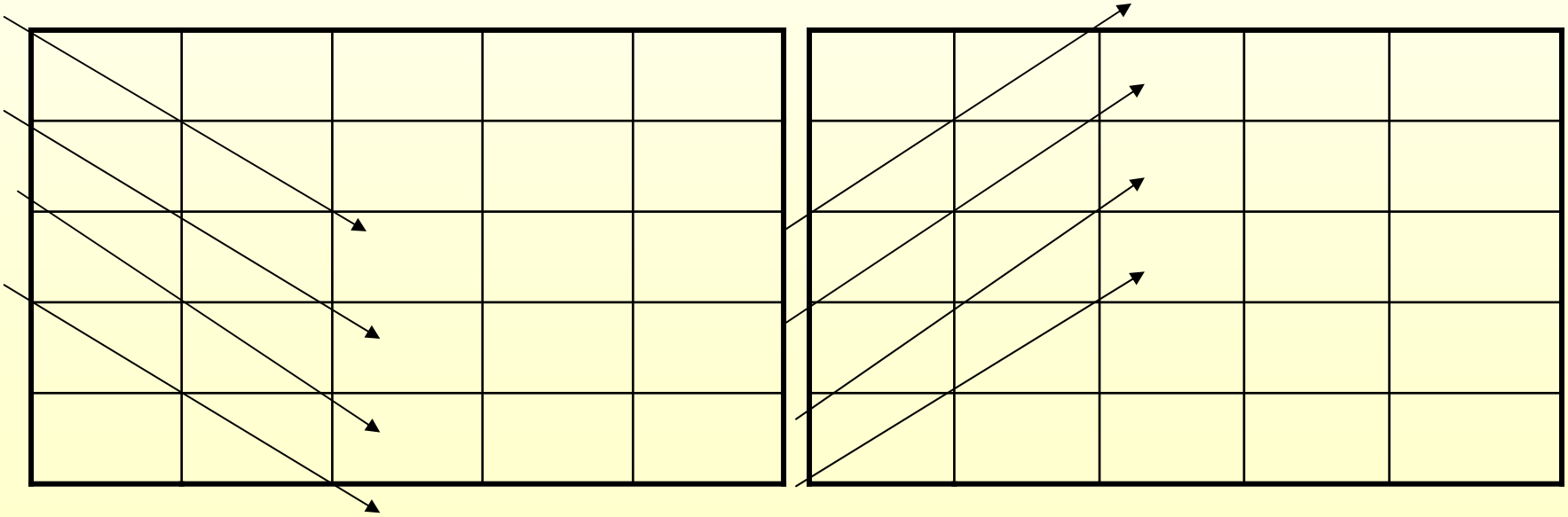
Modelo. Restricciones

- Que todas las reinas estén en diferentes filas.

```
forall(ordered i,j in Dominio)
        reinas[i] <> reinas[j] ;
```

Modelo. Restricciones

- Que no haya nunca dos reinas en la misma diagonal.
- Dividimos esto en dos etapas:
  - Diagonales hacia abajo.
  - Diagonales hacia arriba.



Modelo. Restricciones

Diagonales hacia abajo.

```
forall(ordered i,j in Dominio)
   reinas[i] - i <> reinas[j] – j;
```

Diagonales hacia arriba.

```
forall(ordered i,j in Dominio)
   reinas[i] + i <> reinas[j] + j;
```

Modelo. Procedimiento de búsqueda

- First fail

```
search{
   generateSize(reinas);
};
```

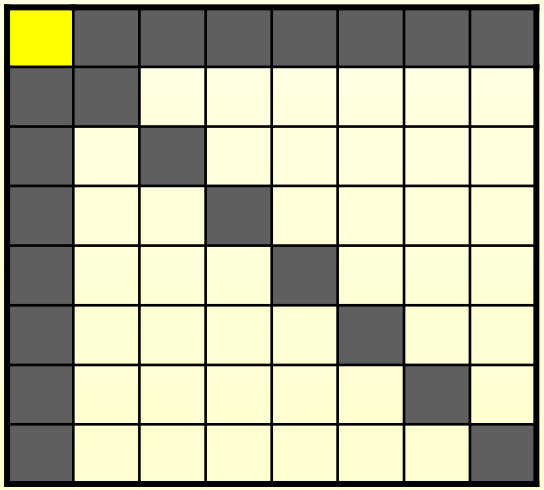
Modelo. Procedimiento de búsqueda

- First fail: En cada toma de decisión…
  1) Asigna valor a la variable de decisión con menor dominio.
  2) El valor asignado es el menor de su dominio.

Veámoslo para 8 reinas…

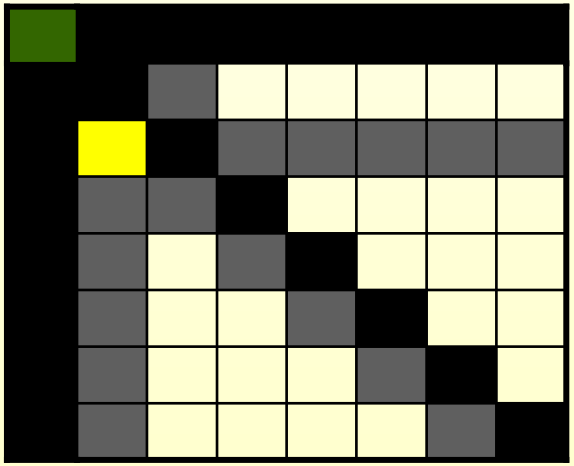
Modelo. Procedimiento de búsqueda

- Paso inicial.
- reinas[1], …, reinas[8] tienen todos los valores en su dominio (8).
- reinas[1] = 1. Propagación de restricciones.



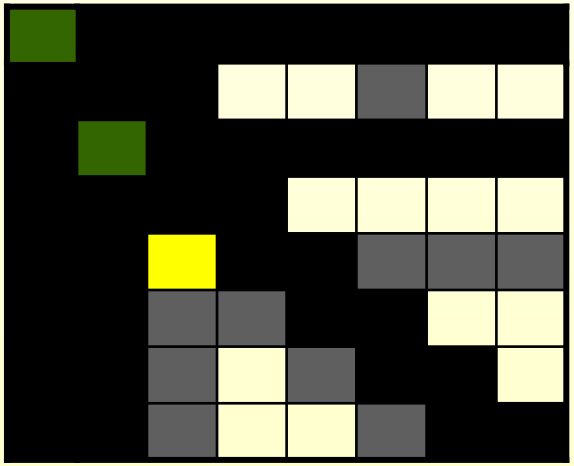
Modelo. Procedimiento de búsqueda

- Paso 2.
- Vemos como se podan ciertas candidatos.
- Ahora elige reinas[2], ya que todas tienen seis valores en su dominio.
- reinas[2] = 3. Propagación de restricciones.



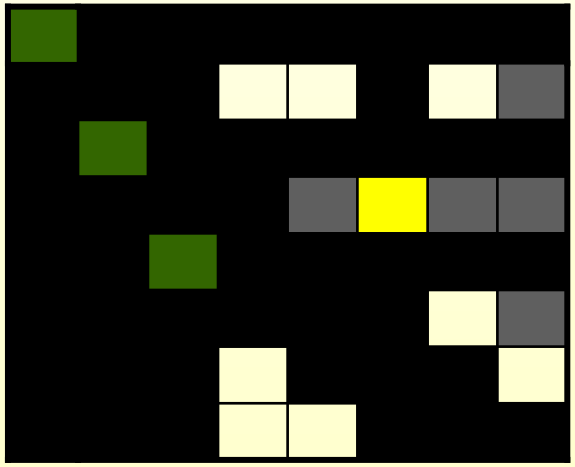
Modelo. Procedimiento de búsqueda

- Paso 3.
- Vemos como se podan ciertas candidatos.
- Ahora elige reinas[3], ya que todas tienen cuatro valores en su dominio.
- reinas[3] = 5. Propagación de restricciones.



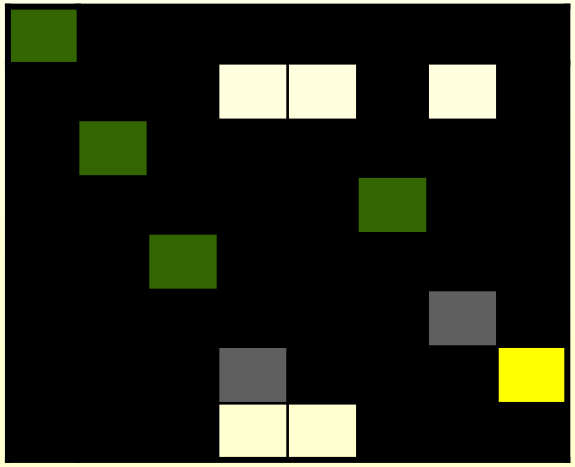
Modelo. Procedimiento de búsqueda

- Paso 3.
- Vemos como se podan ciertas candidatos.
- Coloca reinas[6], ya que solo tiene un valor en su dominio. Fijémonos, esto no cuenta como elección, es obligado. reinas[6] = 4. Propagación de restricciones.



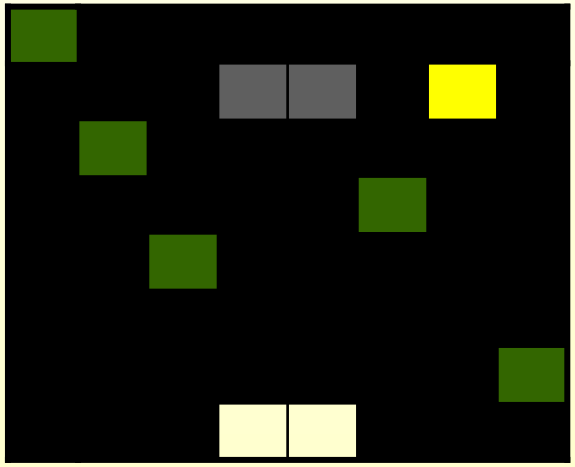
Modelo. Procedimiento de búsqueda

- Paso 3.
- Vemos como se podan ciertas candidatos.
- Coloca reinas[8], ya que solo tiene un valor en su dominio. Fijémonos, esto no cuenta como elección, es obligado. reinas[8] = 7. Propagación de restricciones.



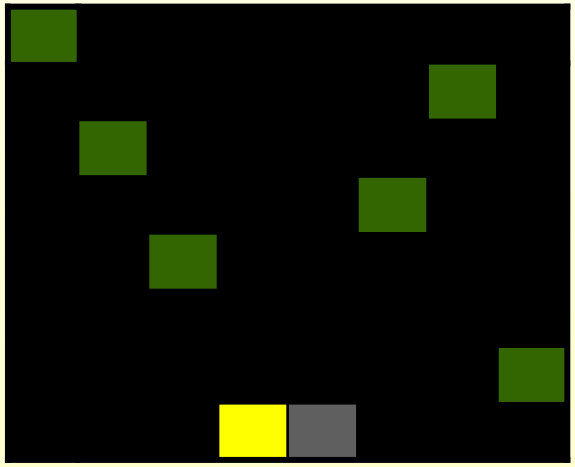
Modelo. Procedimiento de búsqueda

- Paso 3.
- Vemos como se podan ciertas candidatos.
- Coloca reinas[7], ya que solo tiene un valor en su dominio. Fijémonos, esto no cuenta como elección, es obligado. reinas[7] = 2. Propagación de restricciones.



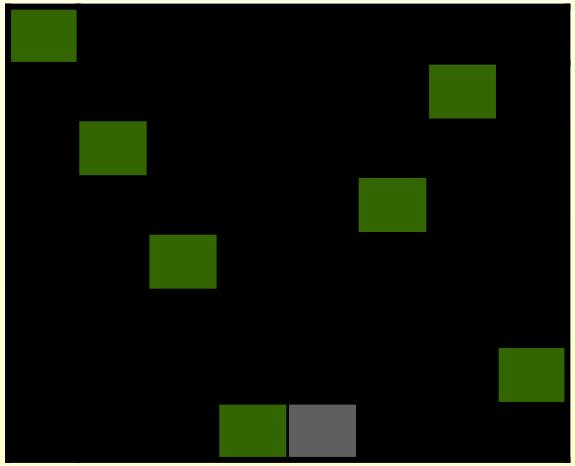
Modelo. Procedimiento de búsqueda

- Paso 3.
- Vemos como se podan ciertas candidatos.
- Coloca reinas[4], ya que solo tiene un valor en su dominio. Fijémonos, esto no cuenta como elección, es obligado. reinas[4] = 8. Propagación de restricciones.



Modelo. Procedimiento de búsqueda

- Paso 3.
- Vemos cómo se poda el único valor que tenía reinas[5]. Como no puede colocar reinas[5] la solución es insatisfactible.
- ¿Dónde hacemos backtracking?
- Backtracking al último checkpoint



Resultados OPL

Solutions

Solution [1]

reinas[1] = 1, reinas[2] = 5,
reinas[3] = 8, reinas[4] = 6,
reinas[5] = 3, reinas[6] = 7,
reinas[7] = 2, reinas[8] = 4

Log

Running

Solver

displaying solution ...

SOLVER

Variables: 8

Restricciones: 84

Failures: 23

Choice points: 25

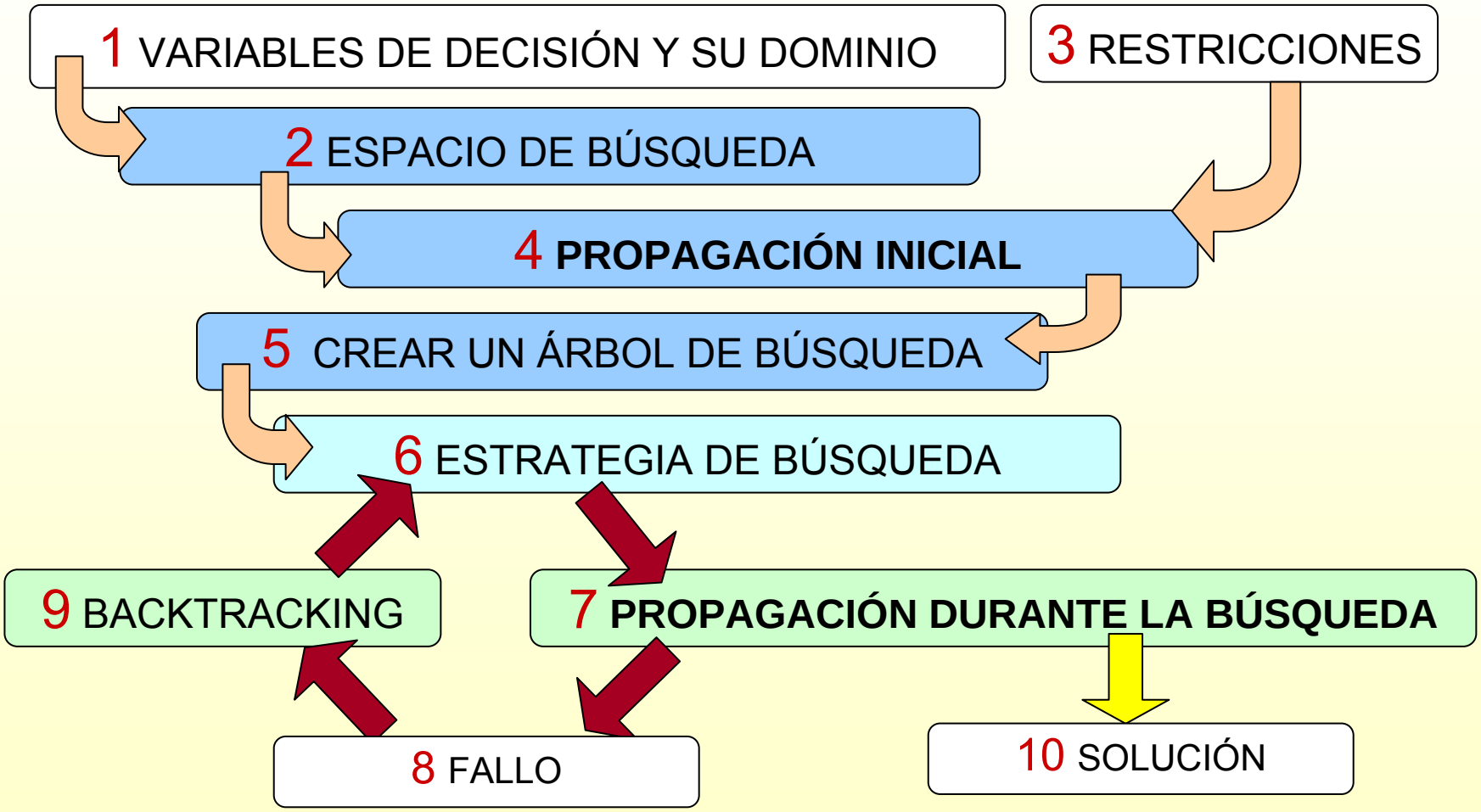
Solver memory: 183940

Solving time: 3.06

Ejemplos modelado

- Taller de coches →
  - Ejemplo completo
- N reinas →
  - Introducción a las búsquedas
- Cuadrado relleno
  - Técnicas de Búsquedas
- Series mágicas →
  - Restricciones de alto nivel
- Matrimonios →
  - Fuerte expresividad

Búsquedas


1 VARIABLES DE DECISIÓN Y SU DOMINIO
3 RESTRICCIONES
2 ESPACIO DE BÚSQUEDA
4 PROPAGACIÓN INICIAL
5 CREAR UN ÁRBOL DE BÚSQUEDA
6 ESTRATEGIA DE BÚSQUEDA
9 BACKTRACKING
7 PROPAGACIÓN DURANTE LA BÚSQUEDA
8 FALLO
10 SOLUCIÓN

Técnicas de búsqueda

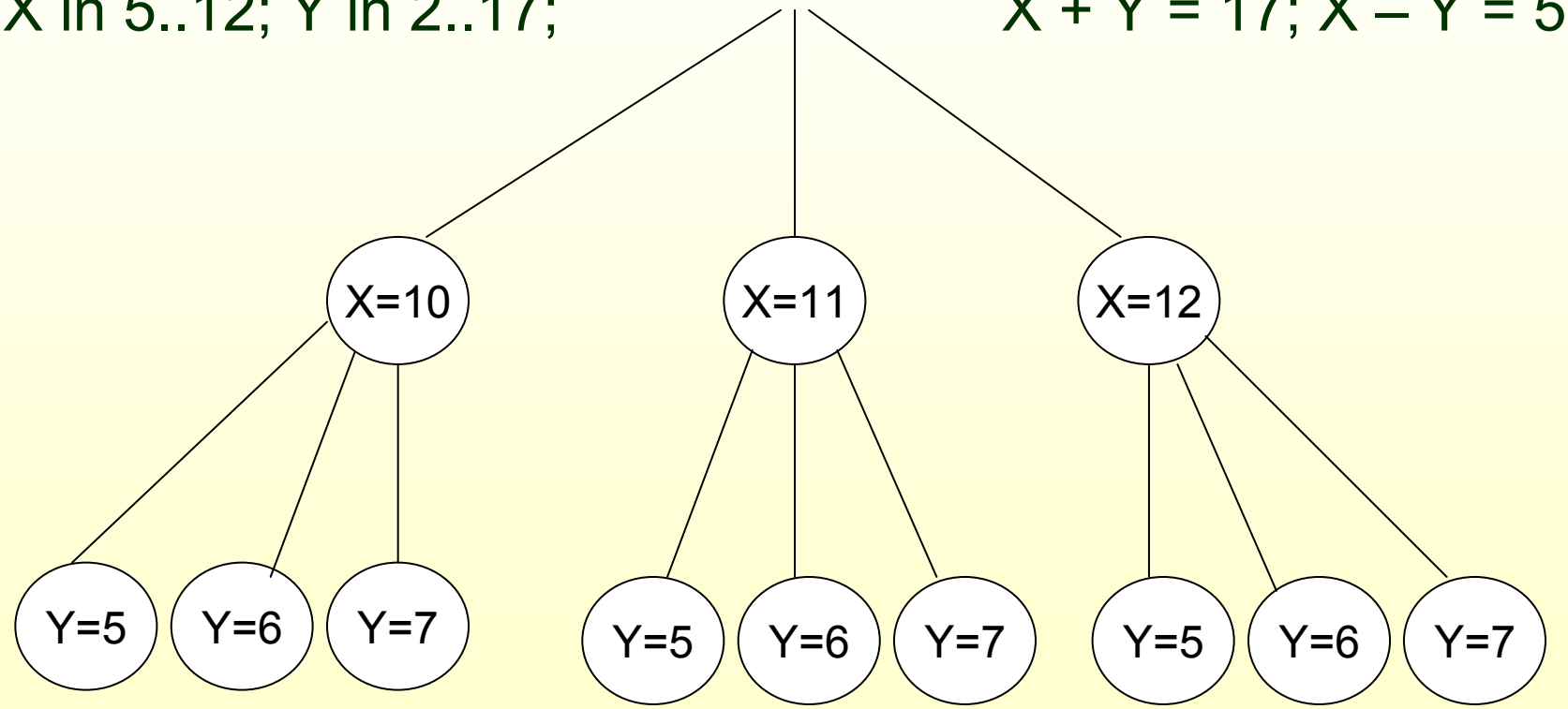
- Búsqueda en profundidad. Estrategia:
  1) Elegir asignación variable/nivel
  2) Elegir orden sobre valores dominio de la variable

- Búsqueda dinámica. Generate

Búsqueda en profundidad

Imaginemos el árbol del ejemplo sencillo.

X in 5..12; Y in 2..17; X + Y = 17; X – Y = 5;



Búsqueda en profundidad.

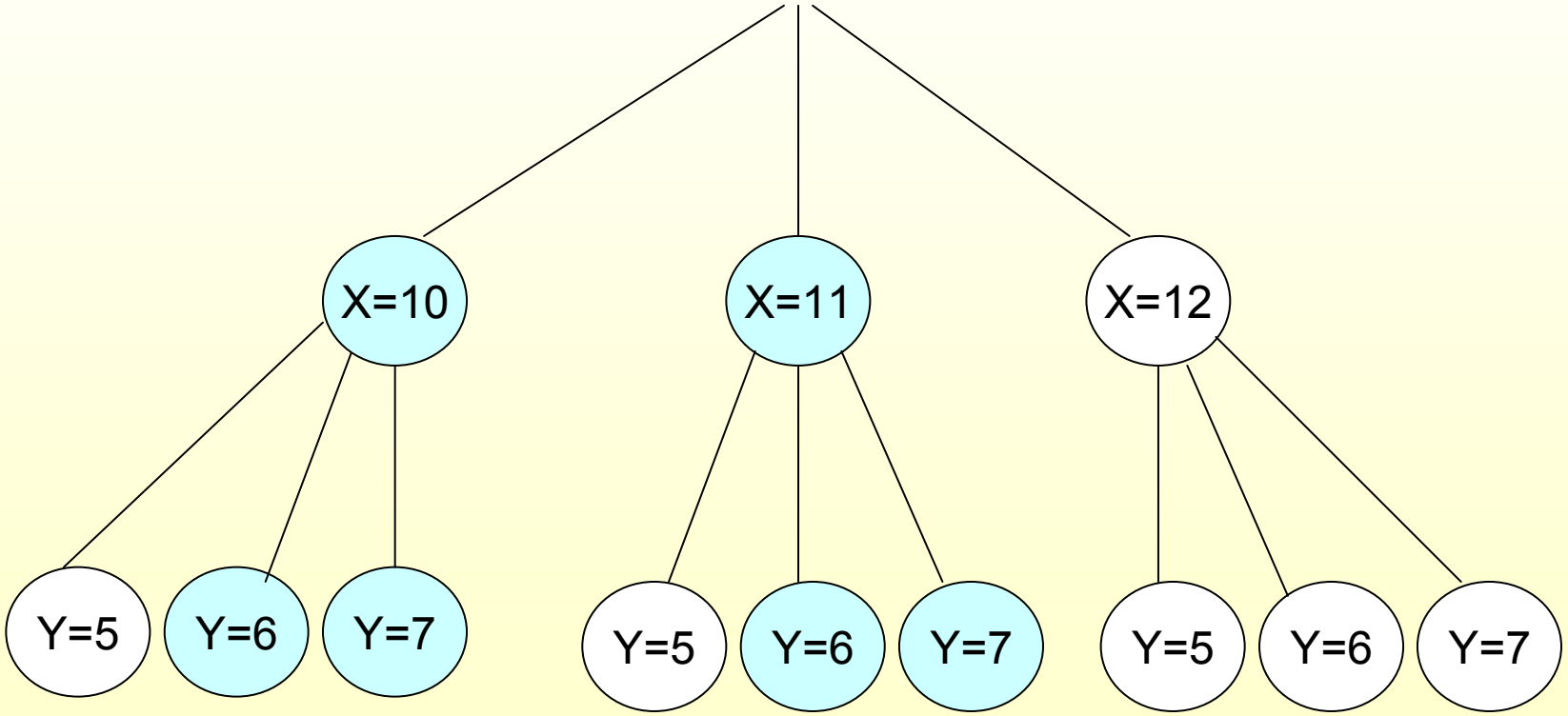
- Elegir asignación variable/nivel
- Elegir orden sobre valores dominio de la variable

1. Instrucción try

```
search {
        try x = 10 | x = 11 endtry;
        try y = 6 | y = 7 endtry;
};
```

Concepto de choice point

Búsqueda en profundidad



Búsqueda en profundidad

- Elegir asignación variable/nivel
- Elegir orden sobre valores dominio de la variable

2. Instrucción tryall

Para explorar todo el espacio de búsqueda.

```
search{
    tryall(i in 10..12)
            x = i;
    tryall(j in 5..7)
            y = j;
};
```

Búsqueda en profundidad

Expresiones con tryall

```
tryall(i in 10..12 ordered by decreasing i)
tryall(i in 10..12: i mod 2 = 0)

//Para arrays
forall(i in 1..8)
    tryall(v in 1..8)
        reinas[i] = v;
```

¡No aprende de los fallos!

Búsqueda dinámica

Idea.

¿Y si la creación del árbol fuera dinámica?

- En el ejemplo de las reinas…
  - El orden o nivel en el árbol de cada variable depende de su dominio.

    Cada vez que haya un backtracking las variables que estén por debajo en el árbol serán generadas dinámicamente.
  - El orden de los valores del dominio es de menor a mayor.

Búsqueda dinámica

Siguiente variable a asignar…

- Variable con menor dominio → generateSize(reinas)
- Variable con menor valor en su dominio → generateMin(reinas)
- Variable con mayor valor en su dominio → generateMax(reinas)
- Variables con índice siguiente en el array → generateSeq(reinas)

Problema del cuadrado.

Descripción

Tengo un cuadrado de $n \times n$ y quiero colocar dentro de su área un conjunto k de cuadrados más pequeños cuya suma total de áreas es n^2.

¿Cómo los coloco dentro del cuadrado grande?

Modelo. Datos del problema

```
int dimension_grande = 112;
int num_cuadrados  = 21;

range      Cuadrados   1..num_cuadrados;
range      Posiciones 1..dimension_grande;

int tamanos[Cuadrados]=  [50,42,37,35,33,29,27,
                          25,24,19,18,17,16,15,
                          11,9,8,7,6,4,2];
```

Modelo. Variables de decisión.

var Posiciones x[Cuadrados];

var Posiciones y[Cuadrados];

Modelo. Restricciones

Que todo cuadrado pequeño se encuentre complétamente dentro del cuadrado grande.

```
forall(s in Cuadrados) {
    x[s] <= dimension_grande + 1 - tamanos[s];
    y[s] <= dimension_grande + 1 - tamanos[s]
  };
```

Modelo. Restricciones

Que la posición asignada a cada cuadrado respete que ningún par de cuadrados pequeños se solapen.

```
forall(ordered i, j in Cuadrados)
    x[i] + tamanos[i] <= x[j] \/
    x[j] + tamanos[j] <= x[i] \/
    y[i] + tamanos[i] <= y[j] \/
    y[j] + tamanos[j] <= y[i];
```

Modelo. Restricciones

Restricción redundante.

Si no hay ningún hueco libre para cada coordenada x (recta vertical) y cada coordenada y (recta horizontal) la dimensión de los cuadrados que ocupen esa fila/columna será = a la dimensión del cuadrado grande.

```
forall(p in Posiciones) {
  sum(s in Cuadrados)
    tamanos[s] * (x[s] <= p & p <= x[s] + tamanos[s] - 1) =
                dimension_grande;
  sum(s in Cuadrados)
    tamanos[s] * (y[s] <= p & p <= y[s] + tamanos[s] - 1) =
                dimension_grande;  };
```

Modelo. Procedimiento de búsqueda

Cada cuadrado pequeño tendrá una posición (x,y).

- Exploro el espacio de búsqueda así:
  - Para cada posición pruebo si puede ser la posición de inicio para cada cuadrado.
  - Si falla, añado al resolutor de ese choice point que ese punto no es el inicio de ese cuadrado.

Modelo. Procedimiento de búsqueda

```
search {
  forall(p in Posiciones)
    forall(s in Cuadrados)
      try x[s] = p | x[s] <> p endtry;
  forall(p in Posiciones)
    forall(s in Cuadrados)
      try y[s] = p | y[s] <> p endtry;
};
```

¡Aprende de los fallos!

Resultados OPL.

Solutions

Solution [1]

x[1] = 1, y[1] = 1, x[2] = 71, y[2] = 71,

x[3] = 76, y[3] = 34, x[4] = 1, y[4] = 51,

x[5] = 80, y[5] = 1

Log

Running

Solver

displaying solution ...

Solver

Variables: 42 Restricciones: 476

Fallos: 10722 C.Points: 15426

S.memory: 16529700 S.time: 1.00

Resultados OPL

Sin aplicar search…

¡No encuentra la primera solución en 8 horas!

Ejemplos modelado

- Taller de coches →
  - Ejemplo completo
- N reinas →
  - Introducción a las búsquedas
- Cuadrado relleno
  - Técnicas de Búsquedas
- Series mágicas →
  - Restricciones de alto nivel
- Matrimonios →
  - Fuerte expresividad

Problema de las series mágicas

Descripción

Dado un n encontrar una secuencia $S=(s_0,s_1,\ldots,s_n)$ tal que s_i represente el número de apariciones de i en S.

Por ejemplo, S=(1,2,1,0) para n=3

Modelo. Datos del problema

```
int n << "Numero de variables:";

range Rango  0..n-1;
range Dominio 0..n;
```

Modelo. Variables de decisión

var Dominio s[Rango];

Modelo. Restricciones

Toda casilla *i* del array S deberá contener un número *k*.

K = número de apariciones de *i* en S.

```
forall(i in Rango)
    s[i] = sum(j in Rango) (s[j] = i);
```

Modelo. Restricciones

¿Cómo explota las restricciones de alto nivel?

s[2] = sum(k in 0..3) (s[k] = 2);

Crea 4 restricciones asociadas a variables booleanas [0,1]

- s[0] = 2 → Variable V02 s[1] = 2 → Variable V12
- s[2] = 2 → Variable V22 s[3] = 2 → Variable V32

En realidad tenemos

s[2] = V02 + V12 + V22 + V32

Modelo. Restricciones

¿Cómo trabaja OPL con estas variables durante la propagación de restricciones?

Imaginemos que tenemos V32 = s[2] >1

DOBLE COLABORACIÓN

a) Si en algún momento el dominio de la variable s[2] pasa a ser {2,3,4} ➔ V32 = 1 (Vinculamos la variable)

b) Si en algún momento la variable V32 se hace cierta ➔ Añadimos s[2] > 1 al resolutor de restricciones.

Modelo. Restricciones

Conclusión:

A medida que vamos vinculando las variables Vij podemos despejar el resultado de la suma de la "gran restricción":

s[i] = sum(j in Rango) (s[j] = i);

Resultados OPL

Solutions

Solution [1]

s[0] = 1

s[1] = 2

s[2] = 1

s[3] = 0

SOLVER

Variables: 4

Failures: 2

Solver memory: 175900

Log

Running

Solver

displaying solution ...

Constraints: 4

Choice points: 4

Solving time: 4.09

Restricciones globales

- alldifferent(reinas) → Si son todas diferentes
- circuit(reinas) → Si forman un circuito hamiltoniano.

int card[1..n]

int value[1..n] = [0,1,2,…,n-1];

int base[1..m]

- distribute(card, value, base) ≡

    forall(i in indice)

        card[i] = sum(j in R) (value[i] = base[j]);

Series mágicas → distribute(s, value, s);

Restricciones globales

- onDomain → garantiza la arco-consistencia para los dominios de las variables implicados en la restricción

alldifferent(reinas) on domain

Importante:

Estas restricciones globales tienen algoritmos especializados de propagación que aumentan su eficacia.

Ejemplos modelado

- Taller de coches →
  - Ejemplo completo
- N reinas →
  - Introducción a las búsquedas
- Cuadrado relleno
  - Técnicas de Búsquedas
- Series mágicas →
  - Restricciones de alto nivel
- Matrimonios →
  - Fuerte expresividad

Problema de las parejas

Descripción

- Tenemos *n* mujeres y *n* hombres.
- Cada persona tiene un ranking de preferencias sobre los miembros del otro grupo.
- Se trata de encontrar *n* parejas estables

Una pareja (a,b) si →

- En el caso de que a prefiera a c antes que a b, c esté más contento con su actual pareja que con a.

- En el caso de que b prefiera a d antes que a a, d esté más contento con su actual pareja que con b.

Modelo. Datos del problema

```
enum Mujeres ...;
enum Hombres ...;
int orden_hombres[Mujeres,Hombres] = ...;
int orden_mujeres[Hombres,Mujeres] = ...;
//Instancia
Hombres = {Richard,James,John,Hugh,Greg};
Mujeres = {Helen,Tracy,Linda,Sally,Wanda};
```

Modelo. Datos del problema

```
orden_hombres = #[
    Helen: #[Richard:1, James:2, John:4, Hugh:3, Greg:5 ]#,
    Tracy: #[Richard:3, James:5, John:1, Hugh:2, Greg:4 ]#,
    Linda: #[Richard:5, James:4, John:2, Hugh:1, Greg:3 ]#,
    Sally: #[Richard:1, James:3, John:5, Hugh:4, Greg:2 ]#,
    Wanda: #[Richard:4, James:2, John:3, Hugh:5, Greg:1 ]#  ]#;

orden_mujeres = #[
    Richard: #[Helen:5, Tracy:1, Linda:2, Sally:4, Wanda:3 ]#,
    James  : #[Helen:4, Tracy:1, Linda:3, Sally:2, Wanda:5 ]#,
    John   : #[Helen:5, Tracy:3, Linda:2, Sally:4, Wanda:1 ]#,
    Hugh   : #[Helen:1, Tracy:5, Linda:4, Sally:3, Wanda:2 ]#,
    Greg   : #[Helen:4, Tracy:3, Linda:2, Sally:1, Wanda:5 ]#  ]#;
```

Modelo. Variables de decisión

La solución al problema será un array con:

- La mujer de cada hombre
- El marido de cada mujer

var Mujeres mujer_de[Hombres];

var Hombres marido_de[Mujeres];

Modelo. Restricciones

Para ser estables:

- Hombre h1 tiene su mujer, m1.
  Pues el marido de m1 tiene que ser h1.
- Mujer m2 tiene su marido, h2.
  Pues la mujer de h2 tiene que ser m2

```
forall(m in Hombres)
    marido_de[mujer_de[m]] = m;

forall(w in Mujeres)
    mujer_de[marido_de[w]] = w;
```

Modelo. Restricciones

Aspectos a destacar

- Expresividad.
  - ✓ Indexar array con variable de decisión.

- Expresión marido_de[mujer_de[m]] = m; produce doble vinculación de variables.
  - ✓ Cuando conoces una variable, automáticamente puedes vincular su pareja.

Modelo. Restricciones

Una pareja (a,b) si →

- En el caso de que a prefiera a c antes que a b, c esté más contento con su actual pareja que con a.

- En el caso de que b prefiera a d antes que a a, d esté más contento con su actual pareja que con b.

```
forall(m in Hombres & o in Mujeres)
    orden_mujeres[m,o] < orden_mujeres[m,mujer_de[m]] =>
        orden_hombres[o,marido_de[o]] < orden_hombres[o,m];

forall(w in Mujeres & o in Hombres)
    orden_hombres[w,o] < orden_hombres[w,marido_de[w]] =>
        orden_mujeres[o,mujer_de[o]] < orden_mujeres[o,w];
```

Resultados OPL

Solutions

Solution [1]

mujer_de[Richard] = Tracy,
mujer_de[James] = Helen,
mujer_de[John] = Wanda,
mujer_de[Hugh] = Linda,
mujer_de[Greg] = Sally

marido_de[Tracy] = Richard,
marido_de[Helen] = James,
marido_de[Wanda] = John,
marido_de[Linda] = Hugh,
marido_de[Sally] = Greg

Log

Running
Solver
displaying solution ...

SOLVER

Variables: 120 Restric: 60
Fallos: 0 Choice Points: 2
S.memory: 268360 Solving time: 0.05

Fin