Solución de Problemas con CCP Modelos

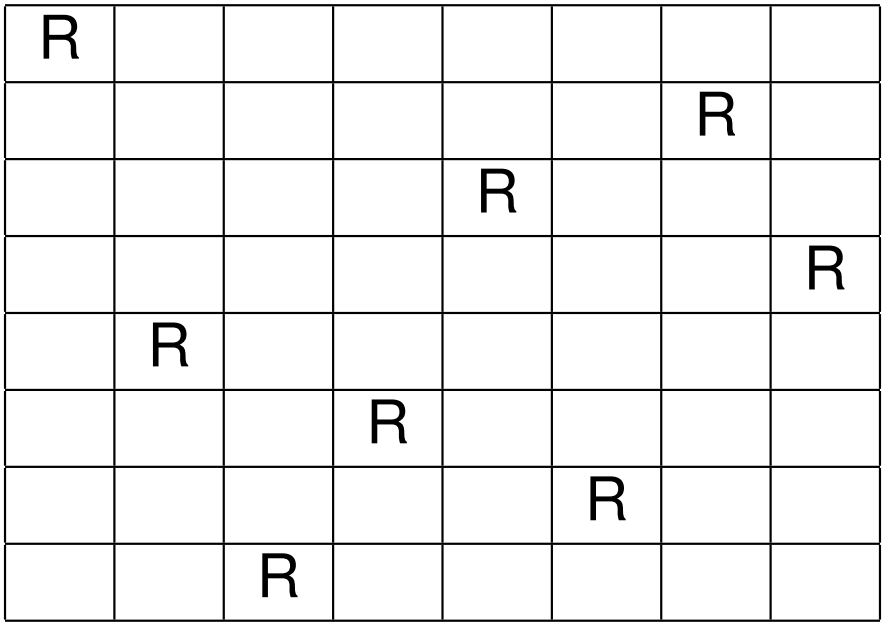
slides basados en el curso “constraint Programming” de Christian Schulte [2]

Profesor: Camilo Rueda [1]

[1] Universidad Javeriana-Cali,
[2] KTH Royal Institute of Technology, Sweden

PUJ 2008

Modelar: ejemplo de 8 reinas



- Encuentre posiciones para 8 reinas en un tablero
- de modo que ninguna se ataque
- Generalizaciones:
  - resolver para *n* reinas
  - colocarlas tan cerca como sea posible

Representación: las variables

- Representar la posición en el tablero

Representación: las variables

- Representar la posición en el tablero
- Dos variables por reina
  - Una para la columna
  - una para la fila

Representación: las variables

- Representar la posición en el tablero
- Dos variables por reina
  - Una para la columna
  - una para la fila
- sobran variables! En cada columna habrá una reina
  - Una variable por columna: su valor dice la fila de la reina en esa columna

    variales: $x_0, ..., x_7$

    dominio: $x_i \in \{0, ..., 7\}$

Representación: restricciones

- No más de una reina en la misma columna
  - se garantia por construcción
- No más de una reina en la misma fila
- No más de una reina en la misma diagonal

Representación: restricciones

- No más de una reina en la misma columna
  - se garantia por construcción
- No más de una reina en la misma fila
  - $x_i \neq x_j$, para $i \neq j$
- No más de una reina en la misma diagonal

Representación: restricciones

- No más de una reina en la misma columna
  - se garantia por construcción
- No más de una reina en la misma fila
  - $x_i \neq x_j$, para $i \neq j$
- No más de una reina en la misma diagonal
  - $x_i - i \neq x_j - j$, para $i \neq j$
  - $x_i + i \neq x_j + j$, para $i \neq j$

Representación: restricciones

- No más de una reina en la misma columna
  - se garantia por construcción
- No más de una reina en la misma fila
  - $x_i \neq x_j$, para $i \neq j$
- No más de una reina en la misma diagonal
  - $x_i - i \neq x_j - j$, para $i \neq j$
  - $x_i + i \neq x_j + j$, para $i \neq j$

$3n(n-1)$ restricciones

Representación: menos restricciones

- Eliminar simetrías
  - $i < j$ en lugar de $i \neq j$
- Restricciones
  - $x_i \neq x_j$, para $i < j$
  - $x_i - i \neq x_j - j$, para $i < j$
  - $x_i + i \neq x_j + j$, para $i < j$

$$3/2n(n-1) \text{ restricciones}$$

Reducir aún más las restricciones

- Restricciones
  - $x_i \neq x_j$ para $i < j$ pueden remplazarse por una sola
- Restricciones
  - $distinct(x_0, ..., x_7)$
  - $x_i - i \neq x_j - j$, para $i < j$
  - $x_i + i \neq x_j + j$, para $i < j$

Ejercicio en casa: remplazar también cada conjunto de restricciones de diagonal por distinct

Script

```
public class Reinas extends Space{
  public VarArray < IntVar > q;

  public Reinas(Options opt) {
    super("Reinas");
    q = new VarArray < IntVar > (this ,8, IntVar.class ,0,7);
...
```

Script: restricciones

```
distinct(this , q, opt.icl);
for (int i = 0; i < 8; i++){
   for (int j = i + 1; j < 8; j++){
   int c[] = {1, −1};
   VarArray < IntVar > x =
      new VarArray  < IntVar > (q.get(i), q.get(j));
   linear (this , c, x, IRT_NQ, i − j);
   linear (this , c, x, IRT_NQ, j − i);
     }
 }
```

...

Script: ramificación

$$branch(this\ , q, INT\_VAR\_NONE, INT\_VAL\_MIN);$$

Es buen “branching” ??

Script: ramificación

$$branch(this\ , q, INT\_VAR\_NONE, INT\_VAL\_MIN);$$

Es buen “branching” ??

- Ensayar “first-fail”
  $branch(this\ , q, INT\_VAR\_SIZE\_MIN, INT\_VAL\_MIN);$
- Ensayar escoger el valor de la mitad
  $branch(this\ , q, INT\_VAR\_NONE, INT\_VAL\_MED);$
- Ensayar mínimo dominio con valor de la mitad
  $branch(this\ , q, INT\_VAR\_SIZE\_MIN, INT\_VAL\_MED);$

Lecciones de 8 reinas

- Variables:
  - El modelo debe requerir pocas variables
- Resricciones
  - No imponer la misma restricción dos veces
  - tratar de encontrar restricciones globales que remplazen muchas simples
    - es más eficiente
    - muchas veces propaga más
- Ramificación
  - es fundamental
  - usualmente depende de propiedades específicas del problema

Otro caso: la tienda

- Un turista va a la tienda en Europa y compra cuatro cosas
- Cajero: son 7,11 euros
- Turista: paga y va a salir de la tienda
- Cajero: espere, lo que hice fue multiplicar!
  deje yo sumo etonces!
  huy!, la suma también da 7,11
- Usted: calcular precio de las 4 cosas

La tienda: modelo

- Variables:
  - para cada item A,B,C,D
  - con dominio $\{1,...,711\}$
  - calcular en cientos: permite trabajar con enteros
- Resricciones
  - $A + B + C + D = 711$
  - $A \times B \times C \times D = 711 \times 100 \times 100 \times 100$

La tienda: script

```
public class Tienda extends Space{
  public VarArray < IntVar > x;

  public Tienda(Options opt) {
    super("Tienda");
    int s = 711;
    x = new VarArray < IntVar > (4);
    for (charc : "ABCD".toCharArray())
        x.add(new IntVar (this , "" + c, 0, s));
    IntVar a = x.get(0);
    IntVar b = x.get(1);
    IntVar c = x.get(2);
    IntVar d = x.get(3);
...
```

Script: restricciones

```
int r[] = {1, 1, 1, 1};

// la suma igual a 711
linear (this , r, x, IRT_EQ, s, opt.icl);

// la multiplicación:
IntVar t1 = new IntVar (this , new IntSet(0, s * 100 * 100 * 100));
IntVar t2 = new IntVar (this , new IntSet(0, s * 100 * 100 * 100));
IntVar t3 = new IntVar (this , new IntSet(0, s * 100 * 100 * 100));
rel (this , t3, IRT_EQ, s * 100 * 100 * 100, opt.icl);
mult (this , a, b, t1);
mult (this , c, d, t2);
mult (this , t1, t2, t3);
```

Script: ramificación

$$branch(this\ , x, INT\_VAR\_NONE, INT\_VAL\_MIN);$$

mala idea!

Script: ramificación

$$branch(this, x, INT\_VAR\_NONE, INT\_VAL\_MIN);$$

mala idea!

- Mejor idea: partir las variables
  - para una variable x
  - partir $x < m$ $x \geq m$
- Típicamente apropiado para problemas con restricciones aritméticas

Script: ramificación (2)

Ensayar con la parte baja del intervalo primero

$$branch(this, x, INT\_VAR\_NONE, INT\_VAL\_SPLIT\_MIN);$$

(ver número de nodos explorados)

Script: ramificación (2)

Ensayar con la parte alta del intervalo primero

$$branch(this, x, INT\_VAR\_NONE, INT\_VAL\_SPLIT\_MAX);$$

(ver número de nodos explorados)

Eliminar simetrías

- interesan los valores para A, B, C, D
- el modelo admite soluciones equivalentes
- agregar un orden sobre A, B, C, D:
  $$A \leq B \leq C \leq D$$
- llamada “restricción de rompimiento de simetría”

Script: eliminar simetrías

$$rel\ (this\ , a, IRT\_LQ, b);$$
$$rel\ (this\ , b, IRT\_LQ, c);$$
$$rel\ (this\ , c, IRT\_LQ, d);$$

(ver número de nodos generados)

Más simetrías?

- observar que 711 tiene un factor primo $= 79$
  - $711 = 79 \times 9$
- alguna de las variables (digamos A) debe etonces ser múltiplo de 79
  - agregar $A = 79 \times z$
  - para alguna variable z de dominio finito
  - borrar el orden para A (dejando el de las otras)

Script: eliminar más simetrías

$$IntVar\ z = new\ IntVar\ (this\ , new\ IntSet(1, s * 100 * 100 * 100));$$
$$IntVar\ sn = new\ IntVar\ (this\ , new\ IntSet(79, 79));$$
$$mult\ (this\ , z, sn, a);$$

(ver número de nodos generados)

Otras consideraciones

- La multiplicación se descompuso en
  $a \times b = t_1 \quad c \times d = t_2 \quad t_1 \times t_2 = t_3$
- podría haber sido diferente:
  $a \times b = t_1 \quad c \times t_1 = t_2 \quad d \times t_2 = s \times 100 \times 100 \times 1000$
  (ensayar)

Resumen

- Principios de modelamiento
  - cuále son las variables
  - cuáles son las restricciones
  - encontrar los propagadores
  - restricciones redundantes
  - cuál debe ser la ramificación
  - romper simetrías

Estrategia de modelamiento

- Entender bien el problema
  - cuále son las variables
  - cuáles son las restricciones
  - criterio de optimización
- ensayar un modelo inicial
  - chequearlo en ejemplos para evaluar su corrección
- mejorar el modelo: mucho más difícil
  - escalar al tamaño del problema real

Importancia de restricciones redundantes

El cuadrado mágico

2	9	4
7	5	3
6	1	8

El problema

- Encontrar matriz de $n \times n$ tal que
  - cada entrada es u número entre 1 y n^2
  - no hay entradas guales
  - cada suma de fila, columna y diagonales principales es igual
- problema muy difícil para n grande

Modelo

Para n=3:

- Variables:
- Restricciones:

Modelo

Para n=3:

- Variables:
  - para cada entrada de la matriz, variable x_{ij}
  - $x_{ij} \in \{1, ..., 3^2\}$
  - una variable adicional para la suma: $s \in \{1, ..., 9 \times 9\}$
- Restricciones:

Modelo

Para n=3:

- Variables:
  - para cada entrada de la matriz, variable x_{ij}
  - $x_{ij} \in \{1, ..., 3^2\}$
  - una variable adicional para la suma: $s \in \{1, ..., 9 \times 9\}$
- Restricciones:
  - todos los campos diferentes: $distinct(xij)$
  - para cada fila i:

    $x_{i0} + x_{i1} + x_{i2} = s$
  - igual para cada columna y diagonal

Cuadrado: script

```
public class Cuadrado extends Space{
  private int n;
  private VarMatrix < IntVar > m;

  public Cuadrado(int size) {
    super("Cuadrado");
    n = size;
    final int nn = n * n
    m = new VarMatrix < IntVar > (this , n, n, IntVar.class , 1, nn);
    IntVar s = new IntVar (this , new IntSet(1, n * nn));
...
```

Script: restricciones

```
// filas y columnas
  for (int i = n; i--!= 0; ){
      linear (this , m.row(i), IRT_EQ, s);
      linear (this , m.col(i), IRT_EQ, s);
  }
// Diagonales
  VarArray < IntVar > d1 = new VarArray < IntVar > (n);
  VarArray < IntVar > d2 = new VarArray < IntVar > (n);
  for (int i = 0; i < n; ++i){
     d1.add(i, m.get(i, i));
     d2.add(i, m.get(n - i - 1, i));
  }
  linear (this , d1, IRT_EQ, s);
  linear (this , d2, IRT_EQ, s);
  distinct(this , m, ICL_DEF);
```

Script: ramificación

$$branch(this\ , x, INT\_VAR\_SIZE\_MIN, INT\_VAL\_SPLIT\_MIN);$$

(correr el script)

Modelo: mejoras

- Se sabe la suma de toda la matriz:
  $1 + 2 + ... + 9 = 9(9 + 1)/2 = 45$
- Se sabe que la suma de una fila es igual a s
- Luego,
  $3 \times s = 45$

En el script:

final $s = nn * (nn + 1)/(2 * n)$; *//la suma de cada fila*

(correr el programa)