

Técnicas, Entornos y Aplicaciones de Inteligencia Artificial

Práctica 3. Problemas de Satisfacción de Restricciones

Objetivo: Modelar y resolver problemas CSP, utilizando el entorno MiniZinc



MiniZinc:

- Entorno de desarrollo para la edición de modelos basados en Restricciones.
- Compilación del modelo en FlatZinc > Diversos resolutores
- Disponible (libre y código abierto) : <https://www.minizinc.org/>
Windows / MacOS / Linux <https://github.com/MiniZinc/MiniZincIDE/releases/>
- Amplia documentación: (Tutorial, Manual del Usuario, Manual de Referencia) ⇒ Handbook

Interfaz MiniZinc

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

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

Below the editor is the Output window, which shows the following text:

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

Line: 2, Col: 1

Ejecución

Configuración Resolutor

Edición del Modelo

Salida Ejecución

Notas:

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

Especificación Modelo CSP

% Esquema de un Modelo CSP en MiniZinc

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

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

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

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

% Formato de salida (asignaciones a las variables)

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

Especificación Modelo CSP

% Modelo ejemplo (Nº de pasteles de plátano y chocolate)

int: b; % **Parámetro**. Numero de pasteles de plátano

var 0..100: c; % **Variable**. Numero de pasteles de chocolate

% gramos de harina

constraint 250*b + 200*c <= 4000;

% numero de platanos

constraint 2*b <= 6;

% gramos de azucar

constraint 75*b + 150*c <= 2000;

% gramos de mantequilla

constraint 100*b + 150*c <= 500;

% gramos de cacao

constraint 75*c <= 500;

% maximizar cantidad ponderada pasteles

solve maximize 400*b + 450*c;

output

["no. of bananas cakes = ", show(b), "\n",
"no. of chocolate cakes = ", show(c), "\n"];

Tipos: float/int/bool/string/enum

Restricciones (aritmético-lógicas)

Operad. relacionales: = (==), !=, >, <, <=, >=

Operad. aritméticos: +, -, *, /, div, mod, pow

Func. aritméticas: abs, sqrt, pow, ...

solve satisfy; %por defecto

solve maximize {arithmetic expression};

solve minimize {arithmetic expression};

output [{string|expr},];

expression: **show** (var)

"\n": Nueva línea

"\t": Tabulación

Parámetros y fichero de datos

% Modelo con ficheros de datos

```
include "datos1.dzn";
```

%Parametros con valor adquirido por fichero

```
int: flour;    %no. grams of flour available  
int: banana;   %no. of bananas available  
int: sugar;    %no. grams of sugar available  
int: butter;   %no. grams of butter available  
int: cocoa;    %no. grams of cocoa available
```

%Parametros con Valor

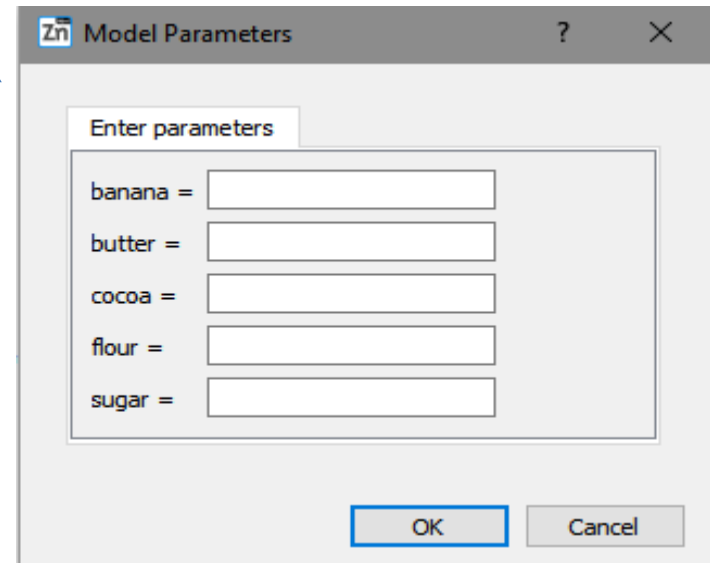
```
int: flour = 4000;  
int: banana = 6;  
int: sugar = 2000;  
int: butter = 500;  
int: cocoa = 500;
```

Con fichero datos

Sin fichero datos

%Fichero datos1 ("datos1.dzn")

```
flour = 4000;  
banana = 6;  
sugar = 2000;  
butter = 500;  
cocoa = 500;
```



Variables: Conjuntos

set of int|float|bool : <var-name> = <expresion> | {<exp₁> <expr₂>, ...<expr_n>} ;

Ejemplos:

```
int: P;           % P es un parámetro
```

```
set of int: Conjunto1 = 1 ..P;    % Conjunto1: variable de 1..P enteros
```

```
set of float: Conjunto2 = {2.5, 6.5, 10.5}; % Variable conjunto de reales
```

Operaciones :

- Pertenencia (in, subset, superset),
- Union (union), Interseccion (inter),
- Diferencia (diff)
- Cardinalidad (card).

Variables: Vectores

array [**<index-1>**, **<index-2>**,....., **<index-n>**] **of var** **int|float|string|bool** : **<var-name>** ;

Ejemplos:

int: N; %N es un parámetro (que se leera en ejecución)

int: k=10; %k es un parámetro con valor indicado

array [1..N, 1..N] **of var** **int**: celda1; % celda1:array bi-dimensional de enteros

array [1..k] **of var** 1..100: celda2; % Array uni-dimensional, con valores 1..100

array [1..10, 1..5, 1..15] **of var** **bool**: celda3; % 3-dimensional de booleanos

Los vectores se pueden inicializarse :

celda1 = [| 3, 5, | 6, 7,]; % celda1: array bi-dimensional de N x N elementos

celda2 = [3, 5, 6, 7,..... 76, 66]; % celda2: array unidimensional de 10 elementos

O adquirir sus valores de ficheros de datos externos.

Ver diversos ejemplos en boletín sobre la definición, operativa e impresión de vectores!

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

Hay varios problemas resueltos en el boletín

```
Running sudoku.mzn
sudoku:
2 7 5 1 4 3 8 6 9
1 3 6 7 9 8 2 4 5
8 4 9 5 6 2 7 1 3
7 1 2 8 3 5 4 9 6
4 6 3 2 1 9 5 7 8
5 9 8 4 7 6 1 3 2
6 5 4 3 2 1 9 8 7
3 2 1 9 8 7 6 5 4
9 8 7 6 5 4 3 2 1
```

%Modelo de un Sudoku N x N

par int: S; %Parámetro pedido en la ejecución del modelo.

int: N = S*S; %parametro, para usarlo como índice de la matriz

array [1..N, 1..N] of var 1..N: celda; % Sudoku, celda[i, j]

include "alldifferent.mzn";

% Todas las celdas en una fila son diferentes.

constraint forall (i in 1..N) (alldifferent (j in 1..N) (celda[i,j]));

% Todas las celdas en una columna son diferentes.

constraint forall(j in 1..N) (alldifferent (i in 1..N) (celda[i,j]));

% Todas las celdas en una submatriz son diferentes.

constraint forall (i,j in 1..S)
 (alldifferent (p,q in 1..S) (celda[S*(i-1)+p, S*(j-1)+q]));

solve satisfy; %solo requerimos satisfactibilidad

output ["sudoku:\n"] ++ [show(celda[i,j]) ++ % Blancos separadores de submatrices

if j = N then if i mod S = 0 /\ i < N then "\n\n" else "\n" endif else

if j mod S = 0 then " " else " " endifendif | i,j in 1..N];

Tarea:

- Realizar el ejercicio propuesto (se necesitará para el día de la evaluación, en el que se plantearán ampliaciones o modificaciones)

Calendario:

Sem	<u>LABORATORIO</u>	Evaluación
15-XI	CSP-MiniZinc	
22-XI	CSP-MiniZinc	
10-I		P3: <i>Eval:</i> CSP-MiniZinc

Práctica CSP (15%) P3