



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Dpto. Sistemas Informáticos y Computación
Escuela Técnica Superior de Ingeniería Informática
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Técnicas, Entornos y Aplicaciones de Inteligencia Artificial

Práctica CSPs - MiniZinc

Constraint Satisfaction Problems

Contenido

1.- Introducción.....	2
2.- Especificación de modelos en MiniZinc	3
2.1.- Un modelo de ejemplo	4
3.- Parámetros y Ficheros de datos	4
4.- Vectores y Conjuntos.....	5
4.1.- Definición de Conjuntos.....	5
4.2.- Definición u Operativa de Vectores (Arrays)	5
5.- Restricciones Especiales.....	6
6.- Configuración del resolutor	7
7.- Ejemplos Resueltos	9
7.1.- Modelo para el juego Sudoku, con un tamaño indefinido N x N.....	9
7.2.- N-reinas.....	10
7.3.- Cuadrado Mágico.....	10
7.4.- Acabados de carrocería	11
8.- Problema a resolver.....	12

1.- Introducción

El objetivo de esta práctica es modelar y resolver problemas de satisfacción de restricciones. Para ello, se utilizará el entorno **MiniZinc**.

MiniZinc-IDE (<https://www.minizinc.org/>) es un entorno de desarrollo que permite editar modelos basados en restricciones. Incluye un compilador del modelo al lenguaje FlatZinc, que es entendido por una amplia gama de resolutores CSP (Gecode, Chuffed, Gurobi, G12, CBC) para la ejecución y resolución del modelo. Particularmente, usaremos el resolutor GECODE.

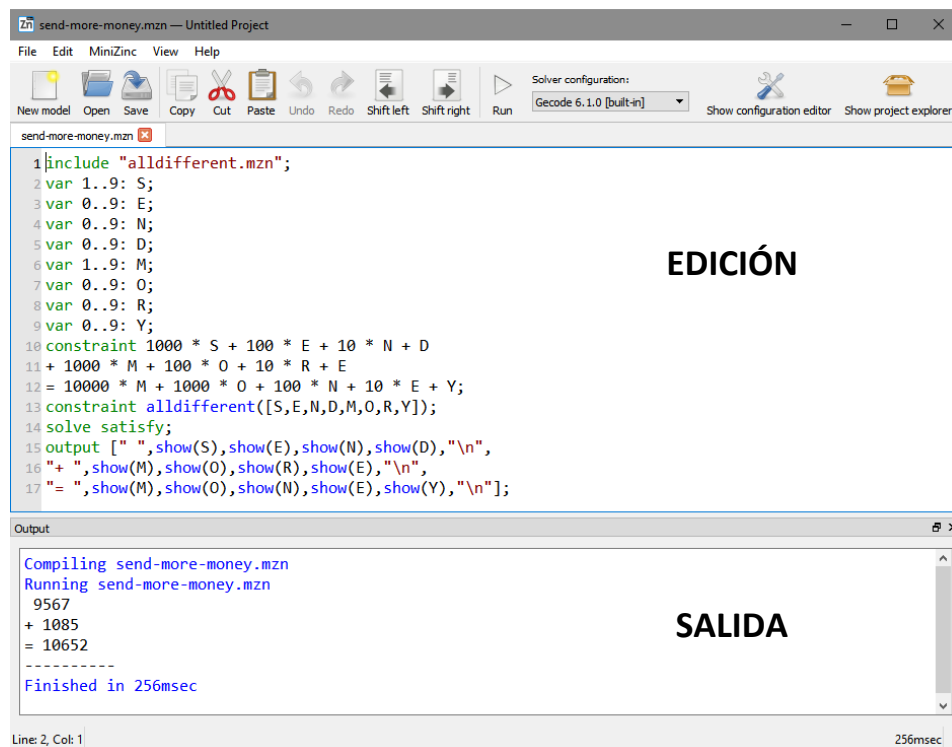


MiniZinc puede descargarse de <https://github.com/MiniZinc/MiniZincIDE/releases/> con versiones para Windows, Mac OS, y Linux.

Existe diversa documentación y manuales sobre MiniZinc en: <https://www.minizinc.org/doc-latest/index.html>. Particularmente:

- **MiniZinc Tutorial**, una introducción básica del modelado de problemas en MiniZinc (disponible en PoliFormat);
- **MiniZinc User Manual**, que detalla el entorno MiniZinc IDE y herramientas adicionales,
- **MiniZinc Reference Manual**, con las especificaciones oficiales de MiniZinc, FlatZinc, y la librería de funciones y restricciones globales disponibles,
- **MiniZinc Handbook** que contiene toda la documentación anterior (disponible en PoliFormat).

Una vez instalado MiniZinc, su ejecución presenta la interfaz típica del entorno. En la figura puede verse esta interfaz básica, que incluye la **ventana de edición** del modelo (send-more-money.mzn), según el lenguaje de modelado MiniZinc, y la **ventana de resolución** (output). Estas dos ventanas pueden separarse, reconfigurarse y ser independientes.



Los pasos típicos en el diseño y resolución de un modelo CSP son:

1. Diseño del modelo correspondiente al problema a resolver (ver lenguaje MiniZinc). La extensión debe ser .mzn
2. Elección y parametrización (si es necesario) del resolutor. Típicamente utilizaremos el resolutor GECODE (opción por defecto). La elección y parametrización del resolutor se realizaría desde la opción "MiniZinc > Solver configurations > Show configurations editor", o directamente desde el icono del menú superior.
3. Ejecución del modelo (Run), obteniendo el resultado en la ventana Output.

2.- Especificación de modelos en MiniZinc

Un modelo básico en MiniZinc tiene las siguientes partes (ver ejemplo en punto siguiente, y finales):

Componente del Modelo	Sintaxis y Ejemplos
Inclusión de ítems <i>Inclusión de ficheros de restricciones globales, datos, etc.</i>	Include <filename>; <pre>include "datos.dzn"; include "alldifferent.mzn";</pre>
Declaración de parámetros Si no se indica valor, sus valores se adquieren de un fichero externo o mediante interfaz de interrogación que se presentará al inicio de la ejecución (Punto 4). <i>Como criterio, los parámetros suelen escribirse en mayúscula</i>	[par] float int bool string : <var-name> [= <valor>]; <i>[par] es opcional, su propósito es meramente clarificador.</i> <pre>int: Par_1; %Valor obtenido al inicio ejecución float: Par_2 = 3.1416; %Valor establecido</pre>
Declaración de Variables Particularmente, se puede expresar un rango de enteros o reales en lugar del tipo int o float	var float int bool : < var-name> [= <expression>]; <pre>var int: nom_var1; %var entera var float: nom_var2 = (2.5 * 3.1); %Real con valor var 1.5 .. 3.25 : nom_real2; % Var real con rango var 1 .. 10: nom_int1; % Var entera con rango var 0..1 : var_decision; % Var de decisión (T/F)</pre>
Tipos y Variables enumeradas <i>Tipo-enumerado: compuesto por un conjunto de símbolos.</i> <i>Variable enumerada: toma un símbolo como valor.</i> <i>Operaciones: = y !=.</i> Lo usual es definir las variables enumeradas como enteros.	enum <nombre-enum> = {sym ₁ , sym ₂ , ..., sym _n } <pre>enum color = {rojo, azul, verde}; var color : camisa; var color : pantalon;</pre>
Restricciones: En la expresión booleana pueden aparecer: <ul style="list-style-type: none"> • Operadores relacionales: = (==), !=, >, <, <=, >= • Operadores aritméticos: +, -, *, /, div, mod, pow • Funciones aritméticas: abs, sqrt, pow, ... 	constraint <Boolean expression>; donde <Boolean expression> es: < arithmetic expresion> <op_rel> < arithmetic expression> <pre>constraint a <= (2*b) + abs(10*c) - sqrt(d);</pre>
Especificación del resolutor Indica el tipo de solución que se desea (solo un tipo). La expresión aritmética contiene variables del modelo.	<pre>solve satisfy; % por defecto solve maximize <arithmetic expression>; solve minimize <arithmetic expression>;</pre>
Especificación y formato de la salida <ul style="list-style-type: none"> • Cada solución se separa por una línea ----- • En optimización, se inserta línea final ===== cuando se ha encontrado la solución óptima. • En satisfactibilidad, se inserta una línea final ===== cuando se han obtenido todas las soluciones posibles. 	output [<string expression>, ... , <string expression>]; Típicamente, expression: show (var) "\n": Nueva línea "\t": Tabulación <pre>output ["Resultado:", "\t", show(a), "\n", "tambien", "\t", show (b)];</pre>

Notas Importantes:

- %: Símbolo de comentario (hasta final de línea)
- Todas las líneas acaban con ";" excepto un comentario que acaba con la línea.
- Las variables y cualquier símbolo (función, predicado, tipo, etc.) son dependientes de mayúsculas/minúsculas.
- Los identificadores de variables empiezan por una letra y pueden contener números, letras y el carácter "_".
- Debe evitarse en lo posible en uso de variables reales. Al ser dominios continuos el número de soluciones puede ser muy grande. Formatos posibles de reales: 1.05, 1.3e-5, 1.3+E5.
- En operaciones sobre variables enteras, no debe sobrepasarse el máximo entero posible, dados sus dominios.

2.1.- Un modelo de ejemplo

"Podemos hacer dos tipos de tortas: una torta de plátano que contiene 250 g de harina, 2 plátanos triturados, 75 g de azúcar y 100 g de mantequilla, y una torta de chocolate que contiene 200 g de harina, 75 g de cacao, 150 g de azúcar y 150 g de mantequilla. Podemos vender un pastel de chocolate por \$4.50 y un pastel de banana por \$4.00. Y tenemos 4 kg de harina, 6 plátanos, 2 kg de azúcar, 500 g de mantequilla y 500 g de cacao. La cuestión es cuántos de cada tipo de pastel podemos hacer para maximizar el beneficio."

```
% Modelo para determinar numero de pasteles de plátano
y chocolate

var 0..100: b; % numero de pasteles de plátano
var 0..100: c; % 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 de pasteles
solve maximize 400*b + 450*c;

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

```
Output
Compiling cakes.mzn
Running cakes.mzn
no. of banana cakes = 0
no. of chocolate cakes = 0
-----
no. of banana cakes = 1
no. of chocolate cakes = 0
-----
no. of banana cakes = 2
no. of chocolate cakes = 0
-----
no. of banana cakes = 3
no. of chocolate cakes = 0
-----
no. of banana cakes = 2
no. of chocolate cakes = 1
-----
no. of banana cakes = 3
no. of chocolate cakes = 1
-----
no. of banana cakes = 2
no. of chocolate cakes = 2
=====
Finished in 297msec
```

3.- Parámetros y Ficheros de datos

MiniZinc permite la aplicación de un mismo modelo a diferentes casos (diferentes conjuntos de datos). Para ello:

- Se definen parámetros en el modelo, sin asignarles valor: **[Par] float/int/bool/string : (var-name);**
- Se definen ficheros de datos (extensión **.dzn**), tal que cada uno contiene un conjunto de valores para los parámetros del modelo.
- En cada ejecución del modelo se indica el fichero de datos a usar: **Include (filename)**

Por ejemplo, consideremos el ejemplo el punto anterior, pero aplicado a diferentes cantidades posibles de harina, plátanos, azúcar y cacao. Este modelo puede ahora ser ejecutado con diferentes ficheros de datos.

Si no se especifica un fichero de datos, al inicio de la ejecución el, sistema presentaría una pantalla para la introducción de los valores de estos parámetros (ver figura).

% Modelo con ficheros de datos

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

%Parámetros cuyo valor sera adquirido por fichero externo

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

%Fichero datos1 ("datos1.dzn")

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

```

int: butter;    %no. grams of butter available
int: cocoa;    %no. grams of cocoa available

% -----

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= flour;

% bananas
constraint 2*b <= banana;

% sugar
constraint 75*b + 150*c <= sugar;

% butter
constraint 100*b + 150*c <= butter;

% cocoa
constraint 75*c <= cocoa;

% maximize our profit
solve maximize 400*b + 450*c;

output ["banana cakes = ", show(b), "\n", "chocolate cakes = ",
show(c), "\n"];

```

```

%Fichero datos2 (include "datos2.dzn")
flour = 8000;
banana = 11;
sugar = 3000;
butter = 1500;
cocoa = 800;

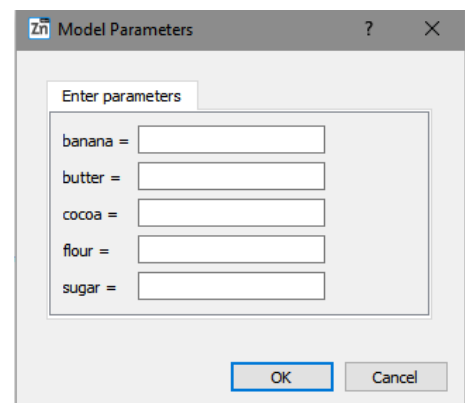
```

```

%Fichero datos3 (include "datos3.dzn")
flour = 6000;
banana = 16;
sugar = 4500;
butter = 2000;
cocoa = 600;

```

%Sin indicar fichero



En estos casos de obtención de datos externos, pueden ponerse unas restricciones especiales en el modelo (**constraint assert**) sobre los valores leídos, tal que se produzca un mensaje de error si la restricción indicada sobre los datos adquiridos no se cumple.

Ejemplo: `constraint assert (flour >= 0,"Invalid datafile! " ++ "Amount of flour is non-negative");`

4.- Vectores y Conjuntos

4.1.- Definición de Conjuntos

MiniZinc permite la definición de **conjuntos**, aunque solo formados por enteros, reales o booleanos. La definición, que incluye la asignación de los valores iniciales del conjunto, es de la forma:

```
set of int|float|bool : <var-name> = <expresion> | {<exp1> <exp2>, ...<expn>} ;
```

Ejemplos:

```

int: P;                                % P es un parámetro, cuyo valor se indicaría en cada ejecución
set of int: Conjunto1 = 1 ..P;         % Se define un conjunto de enteros, formado por el subrango 1..P
set of float: Conjunto2 = {2.5, 6.5, 10.5}; % Se define un Conjunto2 con los números reales indicados

```

Las principales operaciones específicas sobre conjuntos son: Pertenencia (in, subset, superset), Union (union), Interseccion (inter), Diferencia (diff) y Cardinalidad (card).

Una vez se ha definido, una variable conjunto puede ser utilizada como un nuevo tipo: `Var Conjunto1 : Variable1;`

4.2.- Definición u Operativa de Vectores (Arrays)

MiniZinc permite la definición y uso de **vectores n-dimensionales**, donde el tipo de sus elementos (y su dimensión) se indica en la correspondiente declaración del vector.

La declaración de una variable del tipo array (de dimensión n) es de la forma (donde <index-i> es un subrango o conjunto de enteros):

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

Ejemplos de declaración de vectores:

int: N;	% N es un parámetro, cuyo valor se indicaría en cada ejecución
int: M;	% M es un parámetro, cuyo valor se indicaría en cada ejecución
int: k=10;	% k es un parámetro con un valor ya indicado en el modelo
array [1..N, 1..N] of var int: celda1;	% Array bi-dimensional de enteros. Cada valor en celda1[i, j] es un entero
array [1..k] of var 1..100: celda2;	% Array uni-dimensional. Cada valor en celda2[i] es un valor entero 1..100
array [1..10, 1 .. 5, 1..15] of var bool: celda3;	% Array 3-dimensional, cada valor en celda3[i,j,k] es un booleano
array [Conjunto1] of int: celda4;	% Array unidimensional cuyo índice es el Conjunto1, definido previamente
array [1..N, 1..M] of var 0..1: Asignacion;	% Matriz de asignación de N-trabajos a M-máquinas (0/1 representa F/T)

Operativa y Asignación de valores:

Una vez declarado un vector, puede operarse sobre sus celdas individuales (por ejemplo, celda1[i, j]), con los operadores correspondientes al tipo del vector. También pueden asignarse valores a todo el vector completo. Ejemplos:

- En el caso de vectores unidimensionales: celda2 = [3, 5, 6, 7,..... 76, 66];
- En el caso de vectores n-dimensionales, el símbolo | separa las filas: celda1 = [| 3, 5, | 6, 7, |];

Fichero de datos para la inicialización de vectores

Un vector n-dimensional puede declararse como variable o como parámetro, en cuyo caso permite su inicialización mediante un fichero externo de datos:

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

Ejemplos:

int: N=5;	
set of int: Conjunto = 1..N;	
array [1..N] of int: Vector1;	% Como parámetro. Equivalente a: array [Conjunto] of int: Vector1;
array [1..N, 1..N] of int: Vector2;	% Equivalente a: array [Conjunto, Conjunto] of par int: Vector2;
	% Equivalente a: array [Conjunto, Conjunto] of int: Vector2;

La inicialización de los elementos de un vector, *definido como un parámetro*, se puede hacer mediante un fichero externo de datos. En caso de más de una dimensión, los valores de cada dimensión se separan por |.

Ejemplo: Vector1 = [250, 2, 75, 100, 0]
Vector2 = [| 250, 2, 75, 100, 0, | 200, 0, 150, 150, 75 |];

Impresión de Vectores: En los ejemplos finales del boletín pueden verse diversos ejemplos para la impresión de los elementos de un vector.

5.- Restricciones Especiales

Todas las restricciones de un modelo van precedidas de la palabra clave **constraint**. Consultad el manual para conocer todas las restricciones disponibles.

Algunas de las restricciones especiales más utilizadas son:

Restricción	Ejemplos
Restricción OR y XOR: Se separan por \vee : <expr ₁ > \vee <expr ₂ > \vee ... \vee <expr _n > Análogamente se define la restricción xor , como una disyunción exclusiva.	constraint s1 + d1 <= s2 \vee s2 + d2 <= s1; constraint s1 + d1 <= s2 xor s2 + d2 <= s1;

Expresiones AND: Aunque las restricciones de un modelo son conjuntivas, puede indicarse una conjunción de restricciones, separadas por \wedge	<pre>constraint s1 + d1 <= s2 /\ s2 + d2 <= s1;</pre>
Condicional: if (boolexp) then (exp1) else (exp2) endif; Se cumple (exp1) o (exp2) expresión booleana, dependiendo de la condición (boolexp).	<pre>constraint if a > b then c > 10 else c < 10 endif; constraint if b > c then d > 10 endif; constraint if (s1 + d1 <= s2 /\ s2 + d2 <= s1) then (s1 + d1 >= s3 /\ s2 + d2 >= s4) else c < 10 endif;</pre>
Expresiones de Implicación: Si: (boolexp) -> (exp2) Solo-si: (exp2) <- (boolexp) Si-y-solo-si: (exp1) <-> (exp2)	<pre>constraint s1 + d1 <= s2 -> s2 + d2 <= s1; constraint s1 + d1 <= s2 <- s2 + d2 <= s1; constraint s1 + d1 <= s2 <-> s2 + d2 <= s1;</pre>
Negación: not((exp1))	<pre>constraint not (s1 + d1 <= s2 /\ s2 + d2 <= s1);</pre>
forall: Considera un conjunto de expresiones booleanas indizadas y retorna su conjunción lógica.	<pre>constraint forall (i,j in 1..3 where i < j) (a[i] != a[j]);</pre> <p>es equivalente a: $a[1] \neq a[2] \wedge a[1] \neq a[3] \wedge a[2] \neq a[3]$;</p> <p>También:</p> <pre>constraint forall (i in 1..3, j in 1..3 where i < j) (a[i] != a[j]);</pre> <p>Otro ejemplo (trabajando con datos fijos):</p> <pre>constraint forall (i in 1..3) (a[i] != b[2]); % b[2] está fijo</pre>
exists: Considera un conjunto de expresiones booleanas indizadas y retorna su disyunción lógica.	<pre>constraint exists (i,j in 1..3 where i < j) (a[i] != a[j])</pre> <p>es equivalente a: $a[1] \neq a[2] \vee a[1] \neq a[3] \vee a[2] \neq a[3]$.</p>
alldifferent: Todas las variables toman un valor diferente. El uso de esta restricción requiere incluir el código externo "alldifferent.mzn".	<pre>include "alldifferent.mzn"; constraint alldifferent ([S,E,N,D,M,O,R,Y]); constraint alldifferent (Q); % Los valores de las celdas del vector Q son todos diferentes constraint alldifferent (j in 1..N) (Q[j]); % q es un vector, tamaño >N, y los valores de sus N primeras celdas son todos diferentes</pre>
sumatorio: Considera el sumatorio de un conjunto de variables.	<pre>constraint forall (i in 1..N) (sum (j in 1..M) (uso[i,j] * duracion[j]) <= DuracionMaxima);</pre>

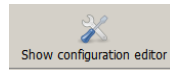
Nota: debe resaltarse que la expresión if es una restricción en su conjunto. Es decir, si se cumple (o no) la condición del if, se cumplirá la condición then (o la condición else). Pero puede haber soluciones que no cumplan la condición del if pero sí las del then/else.

6.- Configuración del resolutor

La elección y configuración del resolutor a utilizar en la resolución del modelo CSP se selecciona en:

"MiniZinc > Solver configurations > Show configurations editor"

o bien, directamente en el icono:



Principalmente, podemos seleccionar:

- Resolutor (solver) a utilizar. Podemos cambiarlo e instalar otros, pero optaremos por Gecode.
- Tiempo límite de resolución para obtener una solución. Por defecto no hay límite.
- User defined behavior: número de Soluciones que se presentan:
 - en optimización, solo la mejor solución (en el tiempo indicado) o las sucesivas mejores optimizadas,
 - en satisfactibilidad, soluciones antes de detenerse.
- Nivel de optimización o preproceso previo a la resolución:
 - básico (O1 – O3)
 - nodo consistencia (O4)
 - arco consistencia (O5)

Adicionalmente a la configuración del resolutor en la interfaz, en el propio modelo CSP pueden indicarse diversos parámetros para la búsqueda de soluciones (ver manual). Particularmente, para problemas de satisfactibilidad, podemos indicar diversos parámetros al resolutor (sobre el orden de instanciación de variables, orden de selección de valores en sus dominios, etc.). Esto se indica tras la palabra clave 'solve' con el conector '::'

```

solve :: int_search(q, first_fail,
indomain_min) satisfy;

```

Por ejemplo, en esta expresión se indica que la búsqueda debe hacerse seleccionando, de los elementos de la matriz de enteros 'q', aquel con el dominio actual más pequeño (regla *first_fail*), y prefiriendo el valor más pequeño del dominio (selección del valor *indomain_min*).

Como criterios para la ordenación de variables, pueden indicarse:

- *first_fail*: elección de la variable con el tamaño de dominio más pequeño,
- *smallest*: elección de la variable con el valor más pequeño en su dominio
- *dom_w_deg*: elección de la variable con el menor 'tamaño de dominio dividido por el número de restricciones en la que participa.'

Como alternativas para la selección de valores:

- *indomain_min* (*indomain_median*/ *indomain_random*): elección del valor más pequeño (medio/aleatorio) en el dominio,

The screenshot shows the 'Configuration' window for MiniZinc. It is divided into several sections:

- Configuration:** A dropdown menu shows 'Gecode 6.3.0'. There are buttons for 'Clone', 'Set as default', and 'Reset to defaults'.
- Options:**
 - A checkbox 'Maintain these options across solver configurations' is checked.
 - Solving:**
 - 'Time limit' is set to '0,000s'.
 - 'Default behavior' is selected with a radio button.
 - 'User defined behavior' is unselected.
 - 'Optimization problems' is set to 'print all intermediate solutions'.
 - 'Satisfaction problems' is set to 'stop after first solution'.
 - Output:**
 - 'Verbose compilation' and 'Verbose Solving' are unchecked.
 - 'Output compilation statistics' and 'Output solving statistics' are unchecked.
 - 'Output timing information' is unchecked.
- Advanced options:**
 - Compiling:**
 - 'Optimization level' is set to '-O1 (default)'.
 - 'Additional data' is an empty text box.
 - Solving:**
 - 'Number of threads' is set to '1'.
 - 'Random seed' and 'Free search' are unchecked.
 - Extra configuration parameters:**
 - A table with columns: Parameter, Flag Type, Data Type, and Value.
 - Buttons for 'Add parameter' and 'Remove parameter' are at the bottom.

- `indomain_split`: dividir en dos partes el dominio de las variables y excluyendo la mitad superior.

Adicionalmente, como norma general para conseguir una mayor eficiencia, se sugiere limitar en lo posible el dominio de las variables enteras y limitar el uso de las variables reales.

7.- Ejemplos Resueltos

7.1.- Modelo para el juego Sudoku, con un tamaño indefinido $N \times N$.

El Sudoku es un rompecabezas matemático que se popularizó en Japón en 1986. El objetivo es rellenar una cuadrícula de 9×9 celdas, dividida en sub-cuadrículas de 3×3 con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. No se debe repetir ninguna cifra en una misma fila, columna o sub-cuadrícula. Un sudoku está bien planteado si la solución es única. Pueden plantearse casos para 9×9 , 8×8 , 10×10 , etc.

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

Consideremos el modelo para un sudoku general, de $N \times N$. Para ello, definimos una matriz $N \times N$ de números enteros. Esta matriz está compuesta de N^2 sub-matrices $S \times S$, tal que $N=S*S$.

Debemos especificar que las celdas en una fila (o columna) son distintas, así como las celdas de cada una de las sub-matrices. Finalmente, debemos presentar el resultado según un formato adecuado.

El modelo es:

```
%Modelo de un Sudoku N x N

%Declaracion de la dimensión (N) del Sudoku como parámetro, para ser un modelo general para cualquier tamaño;
%La matriz esta compuesta de submatrices S x S
par int: S; %este valor será pedido en la ejecución del modelo. Por ejemplo, un valor=3 indica un sodoku 9 x 9.
int: N = S*S; %parametro, para poder ser usado con índice de la matriz celda

%Declaracion de un vector N x N
array [1..N, 1..N] of var 1..N: celda; % Podemos acceder a cada 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]) ++

% Establecemos blancos separadores de submatrices
if j = N then % Nueva linea o dos-nuevas lineas en un nuevo conjunto de submatrices
if i mod S = 0 /\ i < N then "\n\n" else "\n" endif
```

```

else      % Establecemos blancos separadores de submatrices
  if j mod S = 0 then " " else "" endif
endif
| i,j in 1..N ];

```

Si quisiéramos asignar valores iniciales a algunas celdas, podemos incluir, por ejemplo:

```

constraint celda[1,2] = 3;
constraint celda[2,6] = 5;
% etc.

```

7.2.- Problema de las N-reinas

Modelo para el típico problema de colocar N reinas en un tablero N x N sin que se ataquen entre sí. La dimensión del tablero es genérica (se introduce como parámetro).

```

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

include "alldifferent.mzn";

constraint
  forall (i,j in 1..n where i!=j) (q[i] != q[j]); %No en misma columna

constraint
  forall (i,j in 1..n where i!=j) ((q[i] - q[j]) != (i - j)); %No en misma diagonal SE

constraint
  forall (i,j in 1..n where i!=j) ((q[j] - q[i]) != (i - j)); %No en misma diagonal SO

% search
solve satisfy;

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

```

7.3.- Cuadrado Mágico

Un cuadrado mágico es la disposición de una serie de números enteros en una matriz de forma tal que la suma de los números por columnas, filas y diagonales sea la misma 'constante mágica' $M_2(n)$.

$$M_2(n) = \frac{n(n^2 + 1)}{2}$$

Usualmente los números empleados para rellenar las casillas son consecutivos, de 1 a n^2 , siendo n el número de columnas y filas del cuadrado mágico (en el ejemplo, $n=3$).

4	9	2
3	5	7
8	1	6

La dimensión del cuadrado mágico es genérica (se introduce como parámetro)

```

int: n; %dimensión del cuadrado mágico, leída como parámetro
var int: p;
array [1..n, 1..n] of var 1..n*n: Z;
include "alldifferent.mzn";
constraint p = (n * (pow(n,2) + 1)) / 2;
constraint alldifferent (Z);
constraint forall (j in 1..n) (sum (i in 1..n) (Z[i, j]) == p);
constraint forall (i in 1..n) (sum (j in 1..n) (Z[i, j]) == p);

```

```

constraint sum (i,j in 1..n where i=j) (Z[j, i]) = p;
output [show(p), " ", show (Z)];

```

Una salida más visual sería con:

```

output [show(p), "\n" ] ++
  [show(Z[i,j]) ++
    if j = n then "\n" else " " endif % Nueva línea en cada fila
    | i,j in 1..n ];

```

7.4.- Acabados de carrocería

Un fabricante de coches tiene 4 posibles tipos de acabado de la carrocería:

- Tipo 0: Tipo base, sin “Pintura” ni “Metalizado”,
- Tipo 1: Con “Pintura” (se asume una pintura especial),
- Tipo 2: Con “Metalizado”,
- Tipo 3: Tipo completo, con “Pintura” y “Metalizado”.

Se fabrican 5 modelos, M1..M5 y, por razones económicas y comerciales, se cumplen las siguientes reglas de acabado:

1. El modelo M1 tiene acabado “Pintura”, si y solamente si, el modelo M2 tiene el acabado “Pintura” o “Metalizado”. El modelo M1 tiene acabado de “Metalizado” si y solamente si, o bien el modelo M3 tiene el acabado base (sin “Pintura” ni “Metalizado”) o bien el modelo M5 tiene el acabado completo “Pintura” y “Metalizado”.
2. El modelo M2 tiene acabado “Pintura” si y solamente si, o el modelo M1 tiene acabado “Pintura” o “Metalizado”, o bien, el Modelo M5 tiene el acabado base (sin “Pintura” ni “Metalizado”). El modelo M2 tiene acabado “Metalizado” si y solamente si el modelo M3 o el M4 tienen el acabado completo (“Pintura y Metalizado”).
3. EL modelo M1 no tiene el mismo acabado que el modelo M2. Los modelos M3, M4 y M5 tienen también acabados distintos entre ellos.

El modelo CSP correspondiente para la obtención de los posibles acabados de los diferentes modelos sería:

```

include "alldifferent.mzn";
var 0..3: M1;
var 0..3: M2;
var 0..3: M3;
var 0..3: M4;
var 0..3: M5;

constraint M1=1 <-> (M2=1 \/ M2=2);
constraint M1=2 <-> (M3=0 \/ M5=3);
constraint M2=1 <-> (M1=1 \/ M1=2 \/ M5=0 );
constraint M2=2 <-> (M3=3 \/ M4=3);
constraint M1 != M2;
constraint alldifferent ([M3, M4, M5]);

solve satisfy;

output ["Modelo1 ", show(M1), "\n", "Modelo2 ", show(M2), "\n", "Modelo3 ", show(M3), "\n",
"Modelo4 ", show(M4), "\n", "Modelo5 ", show(M5), ];

```

Si quisiéramos minimizar el coste del acabado de los modelos, suponiendo que “Una capa adicional de acabado respecto al acabado base (sea pintura o metalizado) tiene un coste adicional de 5, 10, 15 o 20 euros para los modelos M1, M2, M3, M4 y M5, respectivamente”, podríamos poner:

```

solve minimize (5*M1 + 10*M2 + 15*M3 +20*M4 + 25*M5);

```

que obtendría los acabados consistentes y más económicos posibles para los diferentes modelos M1..M5.

8.- Problema a resolver

Utilizando técnicas de satisfacción de restricciones, deseamos automatizar la asignación del personal de quirófano para la realización de operaciones en un hospital. Concretamente, utilizaremos MiniZinc para modelar y resolver el problema de una forma inteligente.

En nuestro hospital, toda operación requiere un equipo formado por, al menos, 6 personas: 1 cirujano/a, 1 anestesta, 2 enfermeros/as y 2 técnicos/as de quirófano. En función de su dificultad, cada operación tiene una duración.



Por la normativa del hospital, hay una serie de restricciones que hay que cumplir:

- Una misma persona no puede acumular más de 6 horas (360 minutos) de operaciones en una misma jornada.
- Una misma persona no puede participar en más de 3 operaciones en una misma jornada, independientemente de su duración.

Actualmente, el hospital dispone de una plantilla de 16 trabajadores, distribuidos de la siguiente forma: 3 cirujanos (C1..C3), 3 anestestas (A1..A3), 5 enfermeros (E1..E5) y 5 técnicos (T1..T5).

Resolvid los siguientes apartados de **forma incremental** (cada apartado es la base para el siguiente), obteniendo y comparando los resultados:

- a) Se desea encontrar una asignación de personal para las siguientes 6 operaciones:

Operación	Duración (min)
O1	60
O2	120
O3	180
O4	30
O5	45
O6	60

- b) Se desea añadir la siguiente restricción: un cirujano no puede realizar dos operaciones consecutivas. Es decir, un mismo cirujano no puede realizar las operaciones {O1, O2}, {O2, O3}, {O3, O4}, {O4, O5} ni {O5, O6}.

- c) Según la operación, se contemplan las siguientes restricciones:

- La operación O1 la debe realizar, al menos, el cirujano C2 o C3 de forma exclusiva (xor).
- La operación O2 la debe realizar, al menos, el cirujano C1 o C2 de forma exclusiva (xor).
- La operación O3 la debe realizar, al menos, el cirujano C1 o C3 de forma exclusiva (xor).
- La operación O4 requiere al menos a dos cirujanos.

- d) Para facilitar la asistencia del cirujano C3 a un congreso, deseamos minimizar sus horas de trabajo. Por lo tanto, pasaremos de un problema de satisfacción de restricciones (CSP) a un problema de optimización de restricciones (COP). **Nota:** recordad la indicación “=====” cuando estamos en modo optimización.

Algunos **resultados orientativos** (las asignaciones no tienen que coincidir necesariamente con todos los valores propuestos) son:

a)

Operación	Plantilla
O1	C2, A2, E4, E5, T4, T5
O2	C2, A2, E2, E4, T2, T4
O3	C2, A2, E2, E3, T2, T3
O4	C1, A1, E1, E2, T1, T2
O5	C1, A1, E1, E3, T1, T3
O6	C1, A1, E1, E3, T1, T3

b)

Operación	Plantilla
O1	C1, A2, E4, E5, T4, T5
O2	C2, A2, E2, E4, T2, T4
O3	C1, A2, E2, E3, T2, T3
O4	C2, A1, E1, E2, T1, T2
O5	C1, A1, E1, E3, T1, T3
O6	C2, A1, E1, E3, T1, T3

c)

Operación	Plantilla
O1	C3, A2, E4, E5, T4, T5
O2	C1, A2, E2, E4, T2, T4
O3	C3, A2, E2, E3, T2, T3
O4	C1, C2, A1, E1, E2, T1, T2
O5	C3, A1, E1, E3, T1, T3
O6	C1, A1, E1, E3, T1, T3

d)

Operación	Plantilla
O1	C3, A2, E1, E4, T1, T4
O2	C2, A2, E1, E3, T1, T5
O3	C1, A2, E2, E3, T2, T5
O4	C2, C3, A1, E3, E5, T3, T5
O5	C1, A1, E2, E5, T2, T3
O6	C2, A1, E2, E5, T2, T3