**Técnicas, Entornos y Aplicaciones de Inteligencia Artificial**

# Evaluación Práctica-3: CSPs - MiniZinc. 2023-2024.

**Nombre:** César Martínez Chico 04659582N

**MUY IMPORTANTE. NORMAS DEL EXAMEN**

1) **Poned el nombre y leed bien cada una de las preguntas.**
2) **Subid a Poliformat todos los ficheros correspondientes a la práctica (habrá distintas versiones: versión original y cada uno de los ejercicios). Se puede subir un archivo .zip.**
3) **Contestad a las preguntas siguientes de forma razonada, rellenando los huecos con las respuestas. Se deberá subir también este archivo con las respuestas en formato .RTF o .PDF.**
4) **En cada ejercicio se debe partir del apartado e) de la práctica original. Los ejercicios NO son incrementales.**
5) **Puedes cambiar de solver si lo consideras necesario. En tal caso, indícalo en la solución.**
6) **ÚNICAMENTE se admitirán los trabajos subidos a la tarea de Poliformat y dentro del plazo de la tarea.**
7) **No está permitida la utilización de aplicaciones de mensajería.**

**Tiempo: 75 minutos.**

---

**1.** **(2 *puntos, Tiempo estimado: 10')*** Añadir la siguiente nueva información:

- Hay dos servidores más: Servidor9 y Servidor10, con capacidad máxima 90 y 100, respectivamente.
- Hay tres clientes más: Cliente7, Cliente8 y Cliente9. Su demanda es [150, 30, 70] y el máximo de unidades de cómputo es [40, 20, 60].
- El Servidor9 debe atender al Cliente8 o 9 de forma exclusiva (**xor**). Obviamente, también puede atender a otros clientes.

Indica las modificaciones realizadas:

Básicamente he añadido los nuevos datos al fichero de datos.

model_base.mzn | par_base.dzn

```
NumServidores = 10; % 8 + 2
NumClientes = 9; %6 + 3

CapacidadServidores = [100, 100, 100, 90, 90, 80, 80, 80, 90, 100]; %su longitud ha de ser = NumServidores
DemandaClientes = [100, 100, 100, 50, 80, 100, 150, 30, 70]; %su longitud ha de ser = NumClientes
SeguridadClientes = [75, 60, 50, 40, 50, 80, 40, 20, 60]; %su longitud ha de ser = NumClientes
```

Y además he añadido esta restricción :

```
%pregunta 1 del examen -> el servidor9 no puede atender al cliente 8 y 9 a la vez
constraint Solucion[9,8]!=0 xor Solucion[9,9]!=0;
```

Indica el resultado para los siguientes servidores:

| Servidores | Clientes y unidades de cómputo por cliente |
|---|---|

| | |
|---|---|
| Servidor8 | Servidor 8 atiende a cliente 6 la cantidad 20 / 100 |
| Servidor9 | Servidor 9 atiende a cliente 1 la cantidad 29 / 100<br>Servidor 9 atiende a cliente 2 la cantidad 60 / 100<br>Servidor 9 atiende a cliente 9 la cantidad 1 / 70 |
| Servidor10 | Servidor 10 atiende a cliente 1 la cantidad 71 / 100<br>Servidor 10 atiende a cliente 2 la cantidad 10 / 100 |

**En el apartado b) original se produce un cambio**. Ahora, se desea garantizar que todos los servidores ejecuten un mínimo de **20 unidades** de cómputo cada uno

Indica las modificaciones realizadas:

Basicamente cambiar la restriccion del apartado b para que aumente de 10 a 20 la carga minima

```
%apartado b -> comprobar que todos los servidores al menos manejan 20 unidades de carga. ejer1b
constraint forall(servidor in 1..NumServidores) (sum(cliente in 1..NumClientes)(Solucion[servidor, cliente]) >= 20);
```

Indica el resultado para los siguientes servidores:

| Servidores | Clientes y unidades de cómputo por cliente |
|---|---|
| Servidor8 | Servidor 8 atiende a cliente 6 la cantidad 20 / 100 |
| Servidor9 | Servidor 9 atiende a cliente 1 la cantidad 60 / 100<br>Servidor 9 atiende a cliente 2 la cantidad 10 / 100<br>Servidor 9 atiende a cliente 9 la cantidad 1 / 70 |

**2.** ***(3.5 puntos, Tiempo estimado: 25')*** Se desea gestionar un coste por uso de los 8 servidores y 6 clientes originales. Por simplicidad, cada servidor define el siguiente coste de cómputo (independientemente del cliente que esté atendiendo): [1, 2, 3, 4, 4, 3, 2, 1]. Cada cliente define el siguiente coste de cómputo (independientemente del servidor en el que se esté ejecutando): [2, 1, 2, 1, 2, 2]. Así, el coste por uso depende del servidor, del cliente y de las unidades de cómputo que se estén atendiendo. Por ejemplo, si el servidor 1 atiende 25 unidades del cliente 1, el coste asociado será: 1*25*2=50. Si, además, el servidor 1 atiende 25 unidades del cliente 2, el coste asociado será: 50 (el que ya había) + 1*25*1=75. Evidentemente, ahora nos interesa minimizar el coste de uso de todos los servidores al atender a todos los clientes.

Indica las modificaciones realizadas:

Lo primero es añadir los datos:

```
NumServidores = 8;
NumClientes = 6;

CosteServidores = [1, 2, 3, 4, 4, 3, 2, 1];
CosteClientes = [2, 1, 2, 1, 2, 2];
CapacidadServidores = [100, 100, 100, 90, 90, 80, 80, 80]; %su longitud ha de ser = NumServidor
DemandaClientes = [100, 100, 100, 50, 80, 100]; %su longitud ha de ser = NumClientes
SeguridadClientes = [75, 60, 50, 40, 50, 80]; %su longitud ha de ser = NumClientes
```

Sigue cambiar el solve a:

```
solve minimize sum(servidor in 1..NumServidores, cliente in 1..NumClientes) (CosteServidores[servidor] *
CosteClientes[cliente] * Solucion[servidor, cliente]);
```

Calculando así la suma para cada combinación de servidor y cliente el coste de computo de servidor * cliente

Indica el resultado para los siguientes servidores:

| Servidores | Clientes y unidades de cómputo por cliente |
|---|---|
| Servidor 6 | Servidor 6 atiende a cliente 3 la cantidad 1 / 100<br>Servidor 6 atiende a cliente 6 la cantidad 9 / 100 |
| Servidor 7 | Servidor 7 atiende a cliente 6 la cantidad 11 / 100 |
| Servidor 8 | Servidor 8 atiende a cliente 6 la cantidad 80 / 100 |

**CosteTotal**: 2256

Nos hemos dado cuenta de que el Cliente3 y el Cliente5 consumen mucha memoria y no deben ejecutarse en el mismo servidor.

Indica las modificaciones realizadas:

Añado la restriccion:

```
constraint forall(servidor in 1..NumServidores) (Solucion[servidor, 3] * Solucion[servidor, 5] == 0);
```

de forma que si o si, si en uno de los dos hay carga, en el otro no.

Indica el resultado para los siguientes servidores:

| Servidores | Clientes y unidades de cómputo por cliente |
|---|---|
| Servidor 6 | Servidor 6 atiende a cliente 3 la cantidad 30 / 100 |
| Servidor 7 | Servidor 7 atiende a cliente 3 la cantidad 50 / 100<br>Servidor 7 atiende a cliente 6 la cantidad 30 / 100 |
| Servidor 8 | Servidor 8 atiende a cliente 3 la cantidad 10 / 100<br>Servidor 8 atiende a cliente 6 la cantidad 70 / 100 |

**CosteTotal**: 1830

Cabe mencionar que a mas tiempo dejas ejecutando mejores encuentra pues va encontrando muchas soluciones factibles, probablemente con más tiempo encontraría soluciones mejores.

**3.** ***(3.5 puntos, Tiempo estimado: 25')*** Añadir la siguiente nueva información al problema original:

- Hay dos clientes más: Cliente7 y Cliente8. Su demanda es [120, 100] y el máximo de unidades de cómputo es [80, 100].
- El Cliente8 se tiene que ejecutar íntegramente en el Servidor1.
- El Cliente7 se tiene que ejecutar íntegramente en el Servidor4 + 5 (obligatoriamente en los dos).

- Ahora hay más demanda que capacidad de cómputo total, por lo que nos interesa atender tanta demanda como sea posible.

Indica las modificaciones realizadas:

Lo primero que debemos hacer es añadir los nuevos datos:

```
NumServidores = 8;
NumClientes = 8;

CapacidadServidores = [100, 100, 100, 90, 90, 80, 80, 80]; %su longitud ha de ser = NumServidor
DemandaClientes = [100, 100, 100, 50, 80, 100,120,100]; %su longitud ha de ser = NumClientes
SeguridadClientes = [75, 60, 50, 40, 50, 80,80,100]; %su longitud ha de ser = NumClientes
```

Para el segundo punto basta con comprobar que la suma de los servidores 2..N sea 0 para el cliente 1

```
%apartado a del ejercicio 3 -> cliente8 solo en el server 1
constraint sum(servidor in 2..NumServidores) (Solucion[servidor, 8]) == 0 ;
```

Para el tercer punto añadimos que no se puedan otros que no sean los dos mencionados

```
%apartado b del ejercicio 3 -> cliente 7 solo en servidores 4 y 5
constraint forall(servidor in 1..NumServidores) (servidor != 4 /\ servidor != 5 -> Solucion[servidor, 7] == 0);
```

Además debemos cambiar la restricción del ejercicio original que nos comprobaba que todas las demandas estuviesen cubiertas al 100%, por un menor o igual.

```
%compruebo que ningun cliente da más carga de la que demanda, sino exactamente la que demanda
constraint forall(cliente in 1..NumClientes) (sum(servidor in 1..NumServidores) (Solucion[servidor, cliente]) <= DemandaClientes[cliente]);
%compruebo que ningun servidor almacene demás para cada cliente segun la seguridad
```

Y por ultimo el nuevo solver:

```
% Solucionar el modelo y ultimo apartado
solve maximize sum(cliente in 1..NumClientes, servidor in 1..NumServidores) (Solucion[servidor, cliente]);

```

Indica el resultado para los siguientes servidores:

| Servidores | Clientes y unidades de cómputo por cliente |
|---|---|
| Servidor 1 | Servidor 1 atiende a cliente 8 con carga 100 |
| Servidor 4 | Servidor 4 atiende a cliente 7 con carga 80 |
| Servidor 5 | Servidor 5 atiende a cliente 7 con carga 40 |

**DemandaCubierta**: 395

Cabe mencionar que a mas tiempo dejas ejecutando mejores encuentra pues va encontrando muchas soluciones factibles, probablemente con más tiempo encontraría soluciones mejores.

**4.** ***(1 punto, Tiempo estimado: 5')*** Supongamos que queremos modelar que cada cliente se ejecute como máximo en 4 servidores. Explica qué modificaciones serían necesarias para ello. **Nota**: no es necesario implementar nada; basta con explicar las variables y restricciones de forma textual.

Podríamos añadir una variable decisión por cada servidor que tome de valor 0 o 1, dependiendo de si se usa o no ese servidor en cada cliente, y comprobar que la suma de esas variables decisión para cada cliente sea menor o igual que 4,

garantizando así que no se supere el número máximo de 4 por cliente.