Práctica 2:
TSI

UNIVERSIDAD
DE GRANADA

Pablo Martín Palomino
DNI: 7593097H
E-mail: pablomarpa@correo.ugr.es

Respecto uso de la

Como comentario general he usado ia en general para el formateo de las salidas de los ejercicios de manera acorde a como se pide en los enunciados, pues me daban bastantes errores poniendo outputs. En cada ejercicio iré comentando donde y para que he usado ia.

1.Problema del cambio de monedas

Previo a explicar un poco como hice el ejercicio he de aclarar que en general las entregas tanto de este ejercicio como de todos los ejercicios están preparadas para ejecutar directamente y no tener que introducir nada en hora de ejecución. Si se quisiese introducir los datos del problema a la hora de ejecutar lo único que habría que hacer es borrar la asignación de valor en el código. Por ejemplo en este ejercicio int: total =147; borrar el = 147.
La representación para el problema será un array con la cantidad de monedas de cada tipo necesarias para llegar a la cantidad indicada([nº monedas 1 céntimos,nº monedas 3 céntimos,nº monedas 5 céntimos,nº monedas 10 céntimos,nº monedas 25 céntimos,nº monedas 50 céntimos,nº monedas 1 unidad,nº monedas 2 unidades,nº monedas 5 unidades]).
Tenemos como constraint que la cantidad de monedas tomada tiene que sumar la cantidad de importe introducida.
a)**Se desea encontrar un conjunto de monedas cuyo importe sea exactamente una cantidad dada. Para ello, partimos de una divisa imaginaria (que llamaremos Unidad), en donde se dispone de monedas de 1, 3, 5, 10, 25 y 50 céntimos, así como de las de 1, 2 y 5 Unidades (asumiendo que 1 Unidad equivale a 100 céntimos, 2 Unidades a 200 céntimos, y 5 Unidades a 500 céntimos)**

Importe	Primera solución encontrada y número de monedas en la misma	Número total de soluciones	Runtime (en milisegundos)
0,13 Unid	[13, 0, 0, 0, 0, 0, 0, 0, 0] número de monedas: 13	12	94
1,47 Unid	[147, 0, 0, 0, 0, 0, 0, 0, 0] número de monedas: 147	11555	2140
2,30 Unid	[230, 0, 0, 0, 0, 0, 0, 0, 0] número de monedas: 230	77206	8985
2,99 Unid	[299, 0, 0, 0, 0, 0, 0, 0, 0] número de monedas: 299	258664	29176

b) **Modifique el código del apartado anterior de forma que la parte entera de los importes sea asignada únicamente a monedas de una o dos Unidades (dado que la de 5 Unidades, para los importes solicitados, no es necesaria). Por ejemplo, para el importe de 1,47 Unidades se deberá forzosamente asignar una moneda de 1 Unidad y, posteriormente, completar los 47 céntimos restantes con monedas de 1, 3, 5, 10, 25 o 50 céntimos. Complete la tabla anterior con esta nueva codificación**
He de decir que para este apartado se puede abordar de dos maneras,

1- creo que es la más eficiente,que es directamente quitar el tipo de moneda de 5 unidades. En la entrega he mandado también el código correspondiente con el nombre Ejercicio1b_CambioRep.mzn.

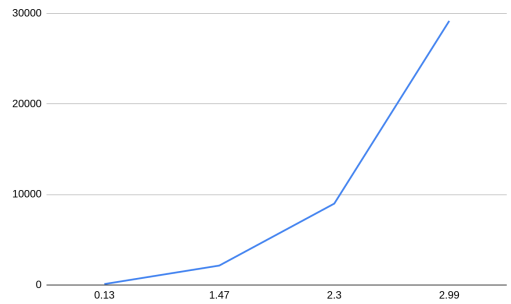
2- Añadir la restricción de que la cantidad de monedas de 5 unidades sea 0.

Importe	Primera solución encontrada y número de monedas en la misma	Número total de soluciones	Runtime (en milisegundos)
0,13 Unid	[13, 0, 0, 0, 0, 0, 0, 0, 0] número de monedas: 13	12	104
1,47 Unid	[47, 0, 0, 0, 0, 0, 1, 0, 0] número de monedas: 48	230	336
2,30 Unid	[30, 0, 0, 0, 0, 0, 2, 0, 0] número de monedas: 32	142	146
2,99 Unid	[99, 0, 0, 0, 0, 0, 2, 0, 0] número de monedas 101	5146	439

c) **Realice una nueva modificación sobre la versión anterior (la del apartado (b)) para que el conjunto de monedas encontrados sea mínimo, es decir, que tenga el menor número de monedas posible**
Aquí la única diferencia es que haremos un solve minimize con respecto a la cantidad de monedas usadas.

Importe	Solución Óptima	Número de monedas de la solución óptima	Runtime (en milisegundos)
0,13 Unid	[0, 1, 0, 1, 0, 0, 0, 0, 0]	2	90
1,47 Unid	[2, 0, 0, 2, 1, 0, 1, 0, 0]	6	96
2,30 Unid	[0, 0, 1, 0, 1, 0, 0, 1, 0]	3	91
2,99 Unid	[1, 1, 0, 2, 1, 1, 0, 1, 0]	7	118

d)i)**A medida que vamos aumentando el importe en el apartado (a), ¿qué ocurre? ¿Cómo escala (a nivel de tiempos de resolución) nuestra codificación con importes progresivamente más elevados? Por ejemplo, ¿cuánto se tardaría en encontrar todas las soluciones para un importe de 3.5 Unid. y 5.27 Unid.?**
Conforme aumenta la cantidad de importe el tiempo aumenta de manera exponencial como se puede observar en la gráfica.



Para 3.50 ha tardado 50 segundos con 555724 soluciones.
Para 5.27 no me ha terminado y llevaba un rato haciendo una predicción por ajuste por una exponencial y(x)=75.8 · e^(2.07x) me sale que tardaría 4143279 ms..

ii)**En relación con el punto anterior, ¿qué ocurriría si, usando la codificación (a) para encontrar todas las soluciones, el importe buscado es mucho mayor (del orden de millones de Unidades)?**

Usando la codificación del ejercicio “a” podemos claramente ver que para ese caso encontrar todas las soluciones no es posible. Encontrar una única solución si lo sería pues la solución de tomar todo con la primera moneda de 1 céntimo es viable (siempre y cuando no desborde en memoria)

iii)**En caso de que haya algún problema con lo planteado en el punto anterior (es decir, si resulta problemático encontrar todas las soluciones para un importe de millones de Unidades), ¿cuál podría ser una estrategia prometedora para resolverlo?**

Efectivamente encontrar todas las soluciones con la codificación del ejercicio “a” en los casos del apartado anterior no es viable por eso propongo eliminar tipos de monedas para restringir el espacio de búsqueda quedándonos solo con las monedas de 1 céntimo, de 50 céntimos, de 1 unidad y de 5 unidades. Para el importe propuesto 122233367 me da 488934 soluciones en un tiempo de 48 segundos. Luego viendo que la casuística sigue siendo bastante larga en tiempo lo que voy a hacer es usar para la parte entera solo monedas de 1 unidad y para la decimal de 1 céntimo, de 10 céntimos y de 50 céntimos reduciendo así drásticamente el tiempo (implementado en ejercicio1d_v2.mzn). Así obtenemos 9 soluciones en 99ms. Otra posible idea sería intentar maximizar la cantidad de monedas grandes cogiendo siempre el máximo posible pero esa no la he implementado.

e)**Finalmente, el alumnado debe repetir la implementación del apartado (a) pero, en esta ocasión, con codificación flotante (tanto para el valor de las monedas, como para los importes y el número de monedas usadas). ¿Qué ocurre ahora y a qué se puede deber?**

He de decir que para este apartado tengo dos códigos:

- Código Ejercicio1e.mzn que convierte todo a float y Da error me imagino por desbordamiento del float. Sale out of limits puede que por tomar números demasiado pequeños o demasiado grandes
- Código Ejercicio1e_v2.mzn que convierte todo a float menos la cantidad de monedas que se toman. Aquí lo que ocurre es que se pierden soluciones con respecto a la codificación a, posiblemente por errores de redondeo.

2.Puzzle lógico

Para la representación de este problema decidí usar dos arrays de tamaño el número de parejas donde por posiciones indicará la pareja , luego decidí tener otro array que exactamente igual por posición de array indica la ciudad a la que pertenece una pareja y finalmente lo mismo con los años. A parte de esto comentar que a parte de las restricciones del enunciado añadí una que es que se ordene de forma creciente en años y ordenar las parejas para romper simetrías.

Solución:

Sevilla: Claudia y Marcos - 5 años
Madrid: Cristina y Marta - 10 años
Parma: Sofía y Pablo - 20 años
Barcelona: Lucía y Pedro - 25 años
Lyon: Juan y José - 30 años

3.Problema de la tabla binaria

Este problema realmente no era muy complicado y para lo único que use ia fue cuando tenía el código, como lo veía bastante feo , le pedí a ver si se podía hacer de forma más compacta y me dijo que podía introducir predicate para no repetir sentencias lógicas. La representación de este problema es una matriz 12x12 de 0s e 1s.

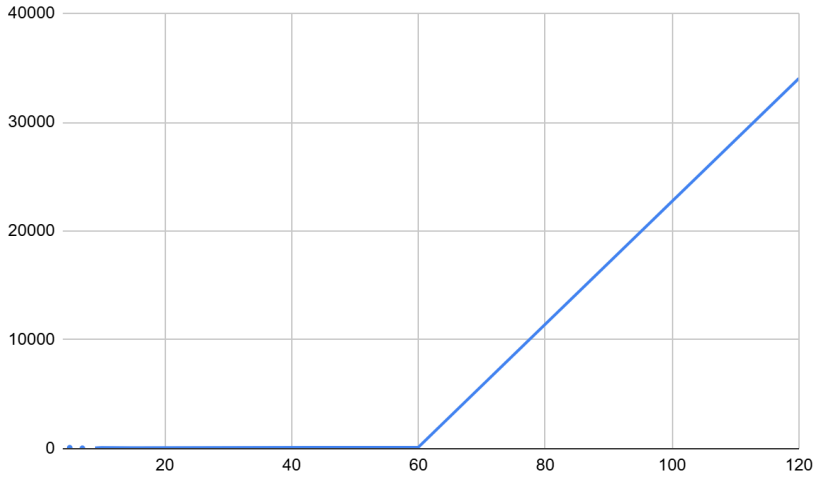
El problema tiene una única solución que es la siguiente:

1	1	0	0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	0	1	0	0	1	1
1	0	0	1	1	0	1	0	1	0	1	0
1	1	0	0	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	0	1	1
0	1	1	0	0	1	1	0	0	1	1	0
1	0	0	1	1	0	0	1	1	0	0	1
1	0	1	1	0	0	1	0	0	1	0	1
0	1	0	0	1	1	0	1	0	1	1	0
0	0	1	1	0	1	1	0	1	0	1	0
1	0	1	0	1	0	0	1	0	1	0	1
0	1	0	1	0	1	0	1	1	0	0	1

4.Problema de los cuadrados

X	S	|S|	Runtime
4	UNSATISFIABLE		
5	{[3, 4]}	2	97
6	UNSATISFIABLE		
7	{[2, 3, 6]}	3	49
8	UNSATISFIABLE		
9	{[2, 4, 5, 6]}	4	81
10	{[1, 3, 4, 5, 7]}	5	103
15	{[1, 2, 3, 4, 5, 7, 11]}	7	91
30	{[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15]}	13	99
60	{[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19,	21	142

	21, 22, 23]}		
120	{[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 33, 34, 38]}	34	34038



se debe crear una gráfica que muestre en el eje de abscisas los valores de X y en el eje de ordenadas el runtime. ¿Qué conclusión se puede extraer del comportamiento observado? ¿Se trata de un problema escalable? Intente reflexionar sobre la complejidad computacional del problema en cuestión, y relaciónelo con los resultados obtenidos.

No es escalable, de hecho es exponencial la subida de tiempo respecto a la subida de número a encontrar. Por lo que es fácil ver que no es un problema escalable. La complejidad computacional de este problema es O(2^n) pues se puede evaluar el conjunto partes del conjunto, es decir, todos los posibles subconjuntos del conjunto {1,....x-1}.

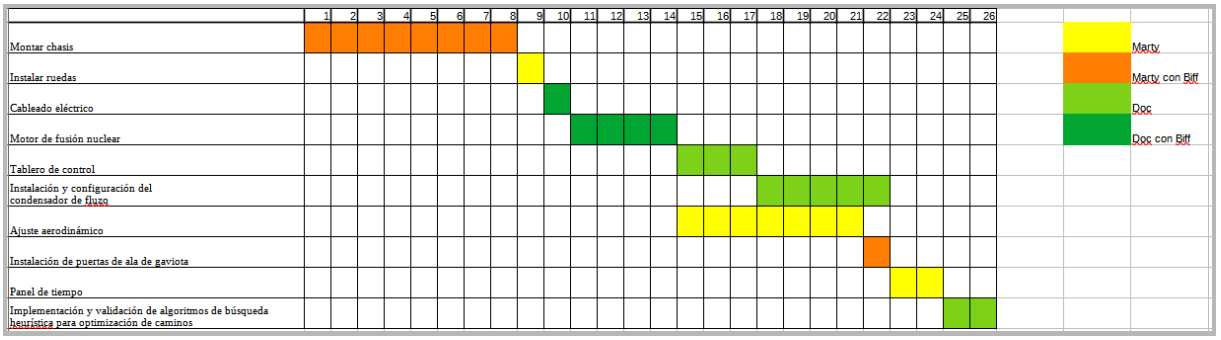
5. Problema de planificación de tareas

Sobre la implementación use diversos arrays donde uno guardaba el inicio de cada tarea, otro el final, otro que guardaba si una tarea está asignada a marty o a doc, otro booleano que decía si una tarea recibía ayuda de Biff o no. También use array para guardar las duraciones de las tareas y las reducciones (haciendo escalable así el problema si se quisiese introducir más tareas). Tuve un poco de lío con el tema restricciones con esta representación y para alguna recurrí a ayuda de ia pues me daba solución distinta a compañeros y no veía exactamente el por qué (tenía menor igual y era menor estricto).

i) La duración óptima es de 26 días.

"Montar chasis"
Inicio día 1, Fin día 8, Asignada a: Marty (con Biff)
"Instalar ruedas"

Inicio día 9, Fin día 9, Asignada a: Marty
"Cableado eléctrico"
Inicio día 10, Fin día 10, Asignada a: Doc (con Biff)
"Motor de fusión nuclear"
Inicio día 11, Fin día 14, Asignada a: Doc (con Biff)
"Tablero de control"
Inicio día 15, Fin día 17, Asignada a: Doc
"Instalación y configuración del condensador de fluzo"
Inicio día 18, Fin día 22, Asignada a: Doc
"Ajuste aerodinámico"
Inicio día 15, Fin día 21, Asignada a: Marty
"Instalación de puertas de ala de gaviota"
Inicio día 22, Fin día 22, Asignada a: Marty (con Biff)
"Panel de tiempo"
Inicio día 23, Fin día 24, Asignada a: Marty
"Implementación y validación de algoritmos de búsqueda heurística para optimización de caminos"
Inicio día 25, Fin día 26, Asignada a: Doc
Tiempo total: 26



ii) El número de soluciones son dos que son las siguientes:

"Montar chasis"
Inicio día 1, Fin día 8, Asignada a: Marty (con Biff)
"Instalar ruedas"
Inicio día 9, Fin día 9, Asignada a: Marty
"Cableado eléctrico"
Inicio día 10, Fin día 10, Asignada a: Doc (con Biff)
"Motor de fusión nuclear"
Inicio día 11, Fin día 14, Asignada a: Doc (con Biff)
"Tablero de control"
Inicio día 15, Fin día 17, Asignada a: Doc
"Instalación y configuración del condensador de fluzo"
Inicio día 18, Fin día 22, Asignada a: Doc
"Ajuste aerodinámico"
Inicio día 15, Fin día 21, Asignada a: Marty
"Instalación de puertas de ala de gaviota"
Inicio día 22, Fin día 22, Asignada a: Marty (con Biff)
"Panel de tiempo"
Inicio día 23, Fin día 24, Asignada a: Marty
"Implementación y validación de algoritmos de búsqueda heurística para optimización de caminos"
Inicio día 25, Fin día 26, Asignada a: Doc

Tiempo total: 26

"Montar chasis"
Inicio día 1, Fin día 8, Asignada a: Marty (con Biff)
"Instalar ruedas"
Inicio día 9, Fin día 9, Asignada a: Marty
"Cableado eléctrico"
Inicio día 10, Fin día 10, Asignada a: Doc (con Biff)
"Motor de fusión nuclear"
Inicio día 11, Fin día 14, Asignada a: Doc (con Biff)
"Tablero de control"
Inicio día 15, Fin día 17, Asignada a: Doc
"Instalación y configuración del condensador de fluzo"
Inicio día 18, Fin día 22, Asignada a: Doc
"Ajuste aerodinámico"
Inicio día 15, Fin día 21, Asignada a: Marty
"Instalación de puertas de ala de gaviota"
Inicio día 22, Fin día 22, Asignada a: Marty (con Biff)
"Panel de tiempo"
Inicio día 23, Fin día 24, Asignada a: Doc
"Implementación y validación de algoritmos de búsqueda heurística para optimización de caminos"
Inicio día 25, Fin día 26, Asignada a: Doc
Tiempo total: 26
----

6. Problema de conformación de tribunales de TFG

Para este ejercicio mencionar que al principio no tenía mucha idea de cómo representarlo y recurrí a una ia para que me ayudara a decidir cómo realizar la representación.

¿Cuál es el número de soluciones válidas obtenidas? En caso de ser menos de 10, lístelas explícitamente

Hay tres soluciones que son las siguientes:

Tribunal 1: 3(DECSAI), 4(LSI), 8(ICAR) (Día: Miércoles)
TFGs a evaluar: 1, 3, 5, 9, 10
Tribunal 2: 2(DECSAI), 6(LSI), 7(ICAR) (Día: Viernes)
TFGs a evaluar: 2, 4, 6, 7, 8

Tribunal 1: 3(DECSAI), 4(LSI), 8(ICAR) (Día: Miércoles)
TFGs a evaluar: 1, 2, 3, 9, 10
Tribunal 2: 2(DECSAI), 6(LSI), 7(ICAR) (Día: Viernes)
TFGs a evaluar: 4, 5, 6, 7, 8

Tribunal 1: 3(DECSAI), 4(LSI), 8(ICAR) (Día: Miércoles)
TFGs a evaluar: 1, 2, 3, 5, 10

Tribunal 2: 2(DECSAI), 6(LSI), 7(ICAR) (Día: Viernes)
TFGs a evaluar: 4, 6, 7, 8, 9

Discuta la posible existencia de soluciones simétricas

Teóricamente existen soluciones simétricas pero estas las he descartado con las siguientes restricciones:

```
% Rompemos simetrías: tribunal 1 con día menor que tribunal 2
constraint tribunal_dia[1] < tribunal_dia[2];
```

Evita que el modelo explore soluciones que sean idénticas pero con los tribunales intercambiados, reduciendo el número de soluciones equivalentes

```
% Ordenar los profesores por ID dentro del tribunal para romper simetría
constraint forall(t in TRIBUNALES)(
 tribunal_profesores[t,1] < tribunal_profesores[t,2] /\
 tribunal_profesores[t,2] < tribunal_profesores[t,3]
);
```

evita que el modelo explore combinaciones de los mismos profesores en distinto orden

```
% Romper simetría en asignación de TFGs No se si esta es relevante
constraint min([i | i in TFGS where tfg_tribunal[i] = 1]) < min([i | i in
TFGS where tfg_tribunal[i] = 2]);
```

Fuerza que el menor índice de TFG asignado al tribunal 1 sea más pequeño que el del tribunal 2.

Estás restricciones las elimina pues estamos forzando un orden en los días, también estamos forzando un orden en los profesores y finalmente también forzamos un orden en la asignación de tfgs.