Técnicas de los Sistemas Inteligentes.

Práctica 2. Problemas de satisfacción de restricciones.

UNIVERSIDAD DE GRANADA

Ignacio Yuste López.

54141533Q

TSI. A1.

1. Problema de las monedas

1.1. Codificación.

Para codificar este problema he decido crear una lista con todas las monedas disponibles con su valor en céntimos (es decir, la moneda de euro se almacena como 100, cincuenta céntimos como 50 y así sucesivamente) y una lista variable que almacena el número de monedas de cada tipo necesarias para sumar la cantidad pedida, donde el máximo número es 2 ya que no es necesario más de dos monedas de cada tipo para calcular cualquier importe en céntimos.

He usado valores enteros ya que con la misma codificación pero en valores flotantes (es decir, en valores de euro) y con el mismo procedimiento y restricciones para algunos valores no encontraba ninguna solución. Creo que esto se debe al tratamiento que Minizinc le da a los números en coma flotante, siendo mucho más sencillo para él los números enteros.

En la última versión de la implementación se nos decía que calculásemos los euros por un lado y los céntimos por otro. Para ellos simplemente he calculado directamente los euros usando la división entera para las monedas de dos y el módulo para las monedas de uno.

1.2. Restricciones.

En esta ocasión he implementado una única restricción. Esta se asegura que la suma de cada moneda de céntimos por el número de veces que aparezca sea igual al importe total de céntimos.

Finalmente, hacemos que calcule las soluciones con el mínimo número total de monedas. Para esto simplemente es necesario sumar todos los valores de la lista del número de monedas.

1.3. Otras implementaciones.

Para el apartado A no se pedía que el número de monedas debía ser óptimo. La solución no contemplaba la separación de euros y céntimos. Además, a la hora de codificar la lista que almacena el número de monedas de cada tipo sus valores podían ser todos los números enteros positivos. De esta manera encontraba muchas más soluciones pero ninguna era óptima.

Para el apartado B simplemente se añadió la separación entre euros y céntimos, lo cual reducía considerablemente el número de monedas empleadas.

1.4. Tablas de soluciones.

Implementación A:

IMPORTE	Primera solución y número de monedas	Número total de soluciones	Runtime (s)
0,17€	{[0.01,17]} → 17 monedas	28	0,171
1,43€	{[0.01,143]} → 143 monedas	17952	4,391
2,35€	{[0.01,235]} → 235 monedas	150824	32,941
4,99€	{[0.01,499]} → 499 monedas	>410000	NA (se ha congelado el pc)

Implementación B:

IMPORTE	Primera solución y número de monedas	Número total de soluciones	Runtime (s)
0,17€	{[0.01,17]} → 17 monedas	28	0,152
1,43€	{[0.01,43], [1.0,1]} → 44 monedas	284	0,226
2,35€	{[0.01,35], [2.0,1]} → 44 monedas	162	0,217
4,99€	{[0.01,99], [2.0,2]} → 101 monedas	4366	0,858

Implementación C:

IMPORTE	Solución óptima	Número total de soluciones	Runtime (s)
0,17€	{[0.02,1],[0.05,1],[0.1,1]} → 3 monedas	2	0,134
1,43€	{[0.01,1],[0.02,1],[0.2,2],[1.0,1]} → 5 monedas	3	0,135
2,35€	{[0.05,1],[0.1,1],[0.2,1],[2.0,1]} → 4 monedas	4	0,144
4,99€	{[0.02,2], [0.05,1], [0.2, 2], [0.5,1], [2.0,2]} → 8 monedas	3	0,140

1.5. Cuestiones.

El espacio de búsqueda sería absurdamente grande. La complejidad del problema sería inmensa y encontrar su solución inviable. Tanto es, que mi ordenador (Intel i7 y nVidia 1060) se ha quedado congelado varias veces intentando ejecutar con 4,99€.

Con la implementación C se podría encontrar sin problema ya que para calcular el número de euros son dos cálculos sencillos. A continuación muestro una imagen de la salida con un importe de más de un millón de euros:

```
Número de  monedas de 0.01€ = 0
Número de  monedas de 0.02€ = 0
Número de  monedas de 0.05€ = 1
Número de  monedas de 0.1€ = 0
Número de  monedas de 0.2€ = 2
Número de  monedas de 0.5€ = 0
Número de  monedas de 1.0€ = 1
Número de  monedas de 2.0€ = 617283
Número total de monedas: 617287
---------
==========
Finished in 174msec.
```

El orden de tiempo empleado varía poco indiferentemente del ejemplo ya que la mayor parte del tiempo de cómputo está en el cálculo de los céntimos, por lo que la cantidad total de euros da igual.

2. Problema de los horarios

2.1. Codificación.

La codificación en este caso consiste en una matriz donde las columnas son los días de la semana y las filas las franjas horarias.

Para simplificar la escritura y lectura del código he definido una serie de *set*. Estos son:

- numAsignaturas: Códigos de las asignaturas ([0..9])
- días: Días lectivos de la semana ([1..5])
- franjas: Franjas horarias de un día lectivo ([1..6])
- profX: Un *set* por cada profesor, definiendo las franjas en las que puede dar clase.

2.2. Restricciones.

1. Todas los días a 4ta hora es el recreo, codificado como asignatura 0.
2. Las asignaturas que imparte el profesor 1 (A1, A3) solo podrán ponerse los días especificados en bloques de dos horas, dentro del horario del profesor.
3. Las asignaturas que imparte el profesor 2 (A4, A5) solo podrán ponerse los días especificados en bloques de dos horas, dentro del horario del profesor.
4. Las asignaturas que imparte el profesor 3 (A6, A9) solo podrán ponerse los días especificados en bloques de una hora, a 3ra hora.
5. Las asignaturas que imparte el profesor 4 (A8, A7) solo podrán ponerse los días especificados en bloques de dos horas, a cualquier hora.

2.3. Solución:

```
-------------------------------------
| L  | M  | X  | J  | V  |
-------------------------------------
| A4 | A4 | A8 | A5 | A5 |
| A4 | A4 | A8 | A5 | A5 |
| A9 | A7 | A6 | A2 | A6 |
| R  | E  | CR | E  | O  |
| A1 | A1 | A3 | A3 | A7 |
| A1 | A1 | A3 | A3 | A2 |
-------------------------------------
----------
Finished in 172msec.
```

2.4. Cuestiones.

Con esta codificación se consiguen únicamente dos soluciones. La diferencia entre ellas es que las asignaturas A7 y A2 pivotan de hora los viernes, ya que son las dos asignaturas del profesor 4 y por tanto pueden darse a cualquier hora y sin la restricción de una asignatura al día por profesor.

Por tanto, en esta codificación no se observa ningún tipo de simetría. Puesto que esta es la única idea de codificación he probado no puedo comentar resultados en los que se de simetría.

3.Problema lógico.

3.1. Codificación.

Considero el color de la casa, la profesión, el animal y la bebida como atributos asignados a una persona. Por ello, defino un tipo de enumerado para cada persona y uno por cada tipo de atributo, acompañados de un array del tipo de enumerado correspondiente, del 1 al 5. De esta manera Minizinc trabaja con datos enteros pero la escritura y la lectura del código se hace sencilla, ya que se puede acceder al atributo de una persona con el enumerado, haciéndolo fácilmente legible y entendible.

La solución toma forma de matriz, donde las filas son los atributos y las columnas las personas, de tal forma que en cada columna aparecen todos los atributos de una persona.

3.2. Restricciones.

Se implementa una restricción por cada una de las 14 condiciones del enunciado, traducidas al idioma de Minizinc.

Además, se implementan tres restricciones más. La primera consiste en que un atributo en concreto (p.e. la casa es verde) no puede aparecer en dos personas distintas. Las dos siguientes son necesarias por las dos últimas condiciones del enunciado, que dicen que el animal está a un lado de una casa, pero no a qué lado. Debido a la codificación propuesta, si, por ejemplo, la casa del diplomático está en la posición 1 (izquierda) cuando el código comprueba la posición de la casa de la izquierda a esta devuelve un 1, ya que no existen valores menores que 1 en el array y por lo tanto devuelve una solución incorrecta donde estas dos últimas condiciones no se cumplen en algunas de las soluciones.

3.3. Solución:

ATRIBUTOS	**VASCO**	**CATALÁN**	**GALLEGO**	**NAVARRO**	**ANDALUZ**
CASA	Roja	Blanca	Verde	Azul	Amarilla
TRABAJO	Escultor	Violinista	Pintor	Médico	Diplomático
ANIMAL	Caracol	Perro	Cebra	Caballo	Zorro
BEBIDA	Leche	Zumo	Café	Té	Agua
POSICIÓN	3	4	5	2	1

La cebra la tiene el gallego.

El andaluz bebe agua.

Runtime(s) = 0,131s.

5. Problema del coloreado de grafos

5.1. Codificación.

Para este ejercicio es necesario introducir los datos manualmente para cada grafo. Defino el número de nodos y aristas del grafo para calcular los posibles valores de estos y los colores con el uso de sets.

Defino un lista que almacenará los colores que tendrá cada arista. Para encontrar las aristas con nodos comunes defino dos listas, donde cada elemento se refiere a un nodo y el nodo al que está conectado a través de una arista está en el mismo índice de la otra lista.

5.2. Restricciones.

Implemento una restricción por cada posible ubicación del nodo parejo, forzando a que la otra arista que conecta al nodo sea distinta de la que queremos comprobar. De tal manera que las restricciones siguen esta forma:

- Se encuentran dos aristas con nodos parejos en caso en que las aristas se almacenan de la forma x,y - t,x = 3,4 5,3
- Se encuentran dos aristas con nodos parejos en caso en que las aristas se almacenan de la forma x,y - x,t = 3,4 3,5
- Se encuentran dos aristas con nodos parejos en caso en que las aristas se almacenan de la forma y,x - t,x = 4,3 1,3

Por último pedimos al solucionador que busque la solución cuya cantidad de colores diferentes sea la mínima.

5.3. Soluciones.

Para obtener los grafos he usado las seeds {23, 46, 69}.

Tamaño del grafo	Número de colores mínimo	Runtime (s)
N=4, M=6	3	0,127
N=6, M=15	5	0,144
N=8, M=28	6	0,302
N=10, M=45	8	25,6
N=12, M=66	10	NA
N=14, M=91		

5.4. Cuestiones.

Con la codificación implementada este problema no es para nada escalable. Sin ir más lejos, no he podido comprobar el tiempo de ejecución para 12 nodos ya que pasaba de los 35 minutos cuando lo he parado.

Esto se debe a que el orden de complejidad es de $0(n^2)$ ya que debemos recorrer las dos listas al completo una vez por cada restricción. Para un grafo realmente grande esta aproximación no es válida en absoluto.