Técnicas de los Sistemas Inteligentes grupo 2 (2021-2022)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 2
MiniZinc

José María Ramírez González
jmramirez@correo.ugr.es

8 de mayo de 2022

Índice

1. Introducción

En el desarrollo de esta práctica vamos a trabajar con MiniZinc, un software que resuelve problemas de satisfacción de restricciones.
Vamos a explorar cinco problemas, en los que abordaremos varias preguntas relativas a cada uno de ellos.
Estos problemas son:

- Problema de las monedas.
- Problema de los horarios.
- Un problema lógico.
- Problema de asignación de tareas.
- Problema de coloreado de grafos.

Entraremos más en detalle sobre cada problema en la sección del mismo.

2. Problema de las monedas

Este problema consiste en encontrar un conjunto de monedas cuyo importe sea exactamente una cantidad dada.
Tenemos disponibles todas las monedas disponibles aquí en España, tanto céntimos, como euros.
Una vez modelado el problema, no vamos a encontrar la solución óptima (menor número de monedas), si no que vamos a encontrar todas las posibles soluciones para las cantidades de 0.17€, 1.43€, 2.35€ y 4.99€. También anotaremos la primera solución encontrada y el tiempo total en encontrar las soluciones del problema.

Importe	Primera solución encontrada y número de monedas en la misma	Número total de soluciones	Runtime (segundos)
0.17 €	17 monedas de 1 céntimo, total 17 monedas	28	0.061
1.43 €	143 monedas de 1 céntimo, total 143 monedas	17952	2.142
2.35 €	235 monedas de 1 céntimo, total 235 monedas	150824	21.469
4.99 €	499 monedas de 1 céntimo, total 499 monedas		

Como vemos, no nos ofrece soluciones nada óptimas y el tiempo de ejecución crece de forma exagerada con la cantidad. Vamos a añadir restricciones para que la parte entera del importe se asigne únicamente a monedas de 1 y 2 euros.
Vamos a realizar una nueva tabla igual a la anterior a ver si hemos conseguido mejorar los resultados.

Importe	Primera solución encontrada y número de monedas en la misma	Número total de soluciones	Runtime (segundos)
0.17 €	17 monedas de 1 céntimo, total 17 monedas	28	0.057
1.43 €	43 monedas de 1 céntimo y 1 moneda de 1 euro, total 44 monedas	284	0.104
2.35 €	35 monedas de 1 céntimo y 1 moneda de 1 euro, total 36 monedas	162	0.096
4.99 €	99 monedas de 1 céntimo y 2 monedas de 2 euros, total 101 monedas	4366	0.509

Como vemos, hemos mejorado considerablemente el tiempo de resolución y los resultados obtenidos, no obstante, seguimos sin tener la solución óptima. Para esto, tenemos que minimizar el número de monedas a obtener cambiando `solve satisfy` por `solve`

`minimize cantidadMonedas`[1]

Vamos a generar una nueva tabla, esta vez con las soluciones óptimas, la cantidad de monedas de las mismas, y el tiempo de ejecución.

Importe	Solución óptima	Número de monedas	Runtime (s)
0.17 €	1 moneda de 10 cent, 1 moneda de 5 cent y 1 moneda de 2 cent	3	0.054
1.43 €	1 moneda de 1 euro, 2 monedas de 20 cent, 1 moneda de 2 cent y 1 moneda de 1 cent	5	0.053
2.35 €	1 moneda de 2 euros, 1 moneda de 20 cent, 1 moneda de 10 cent y 1 moneda de 5 cent	4	0.053
4.99 €	2 monedas de 2 euros, 1 moneda de 50 cent, 2 monedas de 20 cent,1 moneda de 5 cent y 2 monedas de 2 cent	8	0.057

Ahora si, con esta última modificación obtenemos soluciones óptimas.

Se nos plantean ahora las siguientes preguntas:

¿Qué ocurriría si usando la primera codificación, el importe buscado es mucho mayor?

Pues bien, si el importe buscado fuera exageradamente más grande que 4.99€, tendríamos un problema, ya que el número de combinaciones posibles sería enorme, por lo que el tiempo de ejecución y la memoria que consumiría el programa se volvería impracticable. Es decir, podríamos ejecutar el programa, pero este no acabaría nunca.

¿Se podría encontrar alguna solución usando la primera codificación con una cantidad del orden de millones de euros? ¿Cuál sería una estrategia prometedora para esto?

Realmente, podríamos encontrar una solución usando la primera codificación, el problema reside en que no podríamos encontrarlas todas, ya que en términos de tiempo y memoria sería imposible. No obstante, si limitamos el número de soluciones que queremos encontrar a "1", si que encontraríamos la primera solución disponible[2], pero esta no sería ni siquiera cercana a la óptima.

[1] En nuestro caso hemos llamado *cantidadMonedas* a la variable que almacena el número total de monedas

[2] Por cómo está diseñada la primera codificación, el conjunto solución solo tendría millones de monedas de 1 céntimo

3. Problema de los horarios

En este problema se nos pide encontrar una asignación de horarios que satisfaga unas condiciones dadas.
Con una primera implementación sencilla obtendríamos algunas posibles soluciones.
Se nos plantean ahora las siguientes preguntas:
¿Existen soluciones simétricas?
Si, existen soluciones simétricas con la codificación inicial, ya que ahora mismo, esta codificación no comprende que, por ejemplo, si tenemos una asignatura x y 2 horas seguidas, x_1 y x_2, es lo mismo tener x_1 y x_2 que tener x_2 y x_1. Aunque sean horas distintas, al ser de la misma asignatura, da igual y, por tanto, semánticamente son lo mismo.
¿Cómo se podrían evitar las soluciones simétricas y cuál sería el número de soluciones no simétricas?
Se podría evitar añadiendo nuevas restricciones que tuvieran en cuenta el número de horas que tiene una asignatura y no las horas como concepto propio, es decir, que contaran el número de horas de asignatura y no asignaran la franja a una hora concreta de la asignatura, si no a la asignatura propiamente. Usando esta codificación obtendríamos un menor número de posibles soluciones. En el fichero MZN se puede observar comentado el código del que hemos prescindido y se han señalado las nuevas restricciones.

4. Problema lógico

En este problema, en el que se nos presentan cinco personas, cada una de un lugar y con unos gustos concretos, se nos pide encontrar de quién es un animal concreto(cebra) y quién bebe una bebida concreta (agua).
Aparentemente, no hay una solución directa, pues tenemos un gran número de condiciones que tenemos que comprobar antes de suponer una cosa u otra en base a la información inicial.
Finalmente, tras implementar todas las restricciones dadas, podemos afirmar la respuesta a la pregunta que se nos plantea.
¿Dónde está la cebra y quién bebe agua?
Pues bien, obtenemos que la cebra está en la casa del gallego y que la persona que bebe agua es el andaluz.

5. Problema de asignación de tareas

En este problema, tenemos disponibles tres trabajadores y una lista de 9 tareas a realizar, cada trabajador tarda un tiempo determinado en realizar cada tarea. Además hay tareas que tienen que realizarse antes de que puedan comenzar otras.
Tenemos que buscar una asignación de tareas que minimice el tiempo en realizar todas las tareas.
Se nos plantea la cuestión de **¿Cuál es la duración mínima de construcción de la casa?**
Pues bien, con el problema planteado como hemos dicho, tardaremos un total de 11 días, con una asignación de tareas, siendo 1 el trabajador uno y 3 el trabajador 3 como sigue:

Tarea	Día inicio	Día final	Trabajador asignado
A	0	4	1
B	4	7	1
C	7	8	2
D	7	9	1
E	9	11	2
F	9	10	3
G	9	10	1
H	4	7	2
I	10	11	1

6. Problema de coloreado de grafos

En este problema se nos pide colorear las aristas de un grafo con el menor número de colores posible tal que no haya dos aristas que compartan un nodo que tengan el mismo color. A su vez, nos tenemos que asegurar que aquellas aristas que sean similares (es decir, que unan los mismos nodos) tengan el mismo color.
Vamos a rellenar la siguiente tabla con el número de nodos(N), el número de aristas(M), el tiempo total en segundos y el número mínimo de colores necesarios para colorearlo.
Hemos ejecutado cada una de las configuraciones 3 veces y hemos sacado el tiempo y colores medios para rellenar de la forma más precisa posible la tabla.

Tamaño grafo	Número de colores	Runtime(s)
N=4 ,M=6	2.66	0.1
N=6 ,M=15	4	0.11
N=8 ,M=28	5.66	0.115
N=10 ,M=45	7.33	0.145
N=12 ,M=66	8.66	0.756
N=14 ,M=91	11.33	348.3

A la vista de los resultados en tiempo de ejecución obtenidos, podemos responder a la pregunta de **¿Es este problema escalable?**. No, este problema no va a ser practicable para grafos con muchos nodos y aristas, ya que el tiempo de ejecución volvería imposible obtener un resultado.