Técnicas de Sistemas Inteligentes 2021-2022
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 2

Autor:
Brian Sena Simons.

Grupo:
3ºA Subgrupo A2

Brian,
Sena Simons

Satisfacción de Restricciones

Ingeniería Informática
8 de mayo de 2022

Índice

1. Ejercicio 1

En este apartado tenemos que diseñar una representación del problema de asignación de monedas para un importe dado. Para ello he optado por utilizar un vector con dónde cada posición es una cantidad de moneda y otro vector de igual tamaño que para cada posición tiene el valor de esa moneda. De esta manera podemos alcanzar un importe N como la suma de las cantidades por sus valores.

La implementación inicial del apartado a) no impone ninguna restricción sobre el orden de las monedas y/o si se debe minimizar la cantidad. Por lo que como podemos verificar con la tabla, el tiempo de cómputo que utiliza "MiniZinc" crece de forma exponencial por la explosión combinatoria de soluciones sobre que cantidad de cada moneda puede utilizar para resolver el problema, ya que habría miles de combinaciones según el tamañó de importe.

Tabla 1.1: Apartado a).

Importe	Solución Encontrada	Total de Soluciones	Runtime (Segundos)
0.17	17	28	0.18
1.43	143	17952	1.319
2.35	235	150824	10.8634
4.99	499	6224452	427.12

Tabla 1.2: Apartado b).

Importe	Solución Encontrada	Total de Soluciones	Runtime (Segundos)
0.17	17	28	0.159
1.43	44	284	1.352
2.35	36	162	0.206
4.99	101	4366	6.352

Pregunta 1: Apartado c).

Importe	Solución Encontrada	Total de Soluciones	Runtime (Segundos)
0.17	3	15	0.162
1.43	5	40	0.156
2.35	4	32	0.171
4.99	8	94	0.722

Con los resultados obtenidos empiricamente he llegado a la conclusión que MiniZinc converge antes cuántas más restricciones impones sobre las variables libres. En otras palabras, el tiempo de cómputo para el apartado "a" es inviable para valores de importe muy elevados. Si estuvieramos hablando de importe de miles de millones de euros igual tardaría cuestión de días, ya que sigue como un esquema de fuerza bruta empezando por el valor de menor restricción.

Incluso si utilizaramos las versiones "b" y "c" vemos que aún es notable el crecimiento del tiempo de cómputo pero a menor medida. Una posible solución para este problema para importes muy elevados consiste en utilizar un criterio de elección de variables al revés de como hace MiniZinc. En pocas palabras, la idea clave sería empezar por aquellas monedas con mayor valor hasta que nos pasemos del importe, retroceder y probar con la siguiente mayor. De esta manera nos aseguramos que la solución obtenida será la óptima y además se encontrará en muchas menos iteraciones.

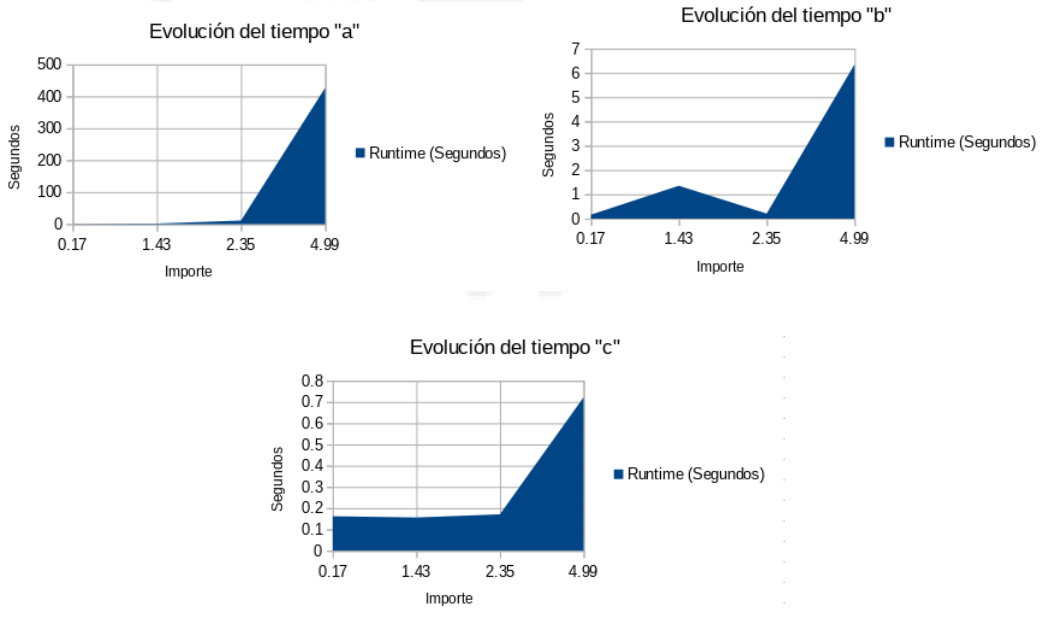

Figura 1.1: Evolución del tiempo de cómputo según restricciones.

2. Ejercicio 2

Tenemos que planificar los horarios de un conjuntos de asignaturas acorde con las restricciones de horas semanales, de disponibilidad de los profesores y de días asignados para cada asignatura entre otras. Los horarios obtenidos para este ejercicio son los siguientes:

Asignación de horarios Instituto					
Horario	**Lunes**	**Martes**	**Miercoles**	**Jueves**	**Viernes**
08:00:00	A4	A4	A8	A5	A5
09:00:00	A4	A4	A8	A5	A5
10:00:00	A9	A7	A6	A2	A6
11:00:00	Recreo	Recreo	Recreo	Recreo	Recreo
12:00:00	A1	A1	A3	A3	A2
13:00:00	A1	A1	A3	A3	A7

Asignación de horarios Instituto					
Horario	**Lunes**	**Martes**	**Miercoles**	**Jueves**	**Viernes**
08:00:00	A4	A4	A8	A5	A5
09:00:00	A4	A4	A8	A5	A5
10:00:00	A9	A7	A6	A2	A6
11:00:00	Recreo	Recreo	Recreo	Recreo	Recreo
12:00:00	A1	A1	A3	A3	A7
13:00:00	A1	A1	A3	A3	A2

Apenas he obtenido 2 soluciones, que son simétricas ya que semánticamente son exactamente lo mismo, el mismo profesor da las mismas clases pero intercambiadas de hora. Este es uno de los problemas que nos podemos encontrar con MiniZinc. Que aunque haya encontrado una solución no sabe diferenciar entre soluciones simétricas y esto puede ser un problema, en el ejercicio 5 fue mucho más notable por el increible aumento del tiempo de cómputo.

Por ello se recomienda tomar iniciativas para saber diferenciar cuando estamos ante una solución simétrica e intentar obviar su exploración. En cualquier caso, en este ejercicio quizás por la implementación apenas he obtenido 1 solución y otra simétrica a ella. En el ejercicio 5 se vuelve hablar del tema.

3. Ejercicio 3

En este ejercicio tenemos que resolver un problema lógico de las cinco casas, quiénes viven en ellas, con qué animales, cuáles son sus bebidas y cuáles son sus profesiones. Para ello he optado por tomar el camino de asignar una variable para cada característica perteneciente a un individuo. Es decir, si el brasileño bebe whiskey, entonces brasileño = whiskey = 10/10.

La idea a partir de esto es ir definiendo una a una las igualdades provenientes del enunciado. De tal manera que, en menos de 0.1 segundos, he podido fácilmente obtener que **la cebra pertenece al gallego** y el **andaluz bebe agua**.

4. Ejercicio 4

Este sin duda fue el ejercicio más dificil para mi, la representación de lo que es la concurrencia fue un tremendo reto. Sin embargo, tras leer el manual oficial de MiniZinc, y replantear el ejercicio miles de veces he logrado obtener resultados optimistas.

El ejercicio nos impone una tabla de tareas, nos indica que tenemos tres trabajadores tales que cada uno tiene un tiempo asignado para poder completar tal tarea, y entre ellos pueden trabajar concurrentemente. Además, ciertas actividades tienen que realizarse antes de las demás.

Para ello he utilizado un vector de tareas con valores que especifican cuando empieza cada tarea. (vector[1] = 0 ->Tarea 1 empieza día 0). Luego además tengo un vector de tareas con sus respectivas duraciones, una matriz con los tiempos y otra que representa las tareas, que trabajador la hace y cuando. Además he utilizado funcionalidades de MiniZinc como es “alternative” y “disjunctive” que me ayudan a decidir que tarea realizar y con que trabajador y evitar solapamiento. El resultado obtenido se verifica en el siguiente diagrama:

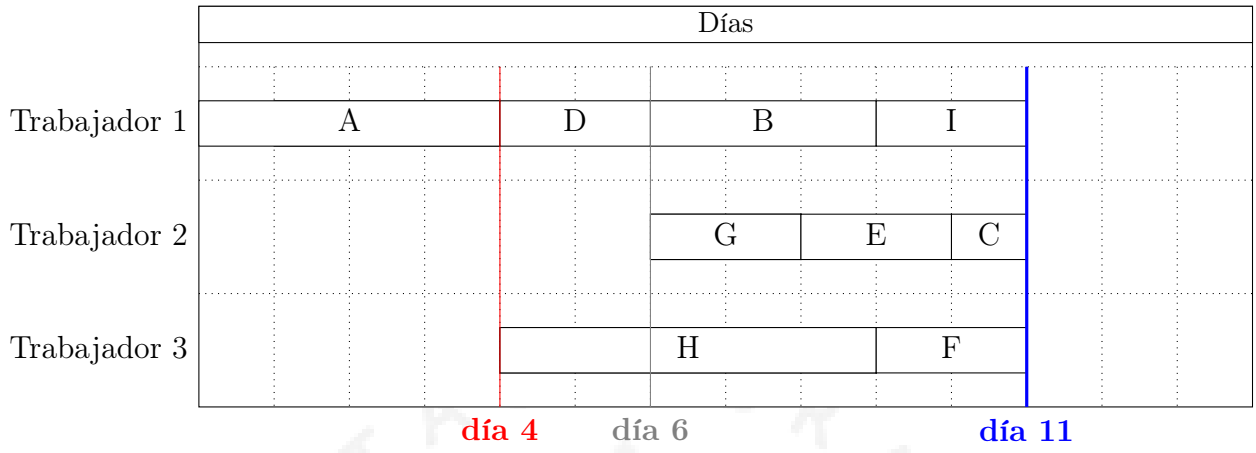

Figura 4.1: Diagrama para 3 trabajadores.

Como podemos observar en nuestro diagrama de Gantt, vemos que forzosamente empezamos por la actividad A, y a continuación disparamos dos actividades de forma simúltanea a parti del día 4. En el día 6 tenemos dónde se alcanza la máxima eficiencia con los 3 trabajadores activos. Esta es la mejor combinación que nos indica Minizinc en alrededor de 12 segundos. **El tiempo mínimo para terminar todas las tareas es de 11 días.**

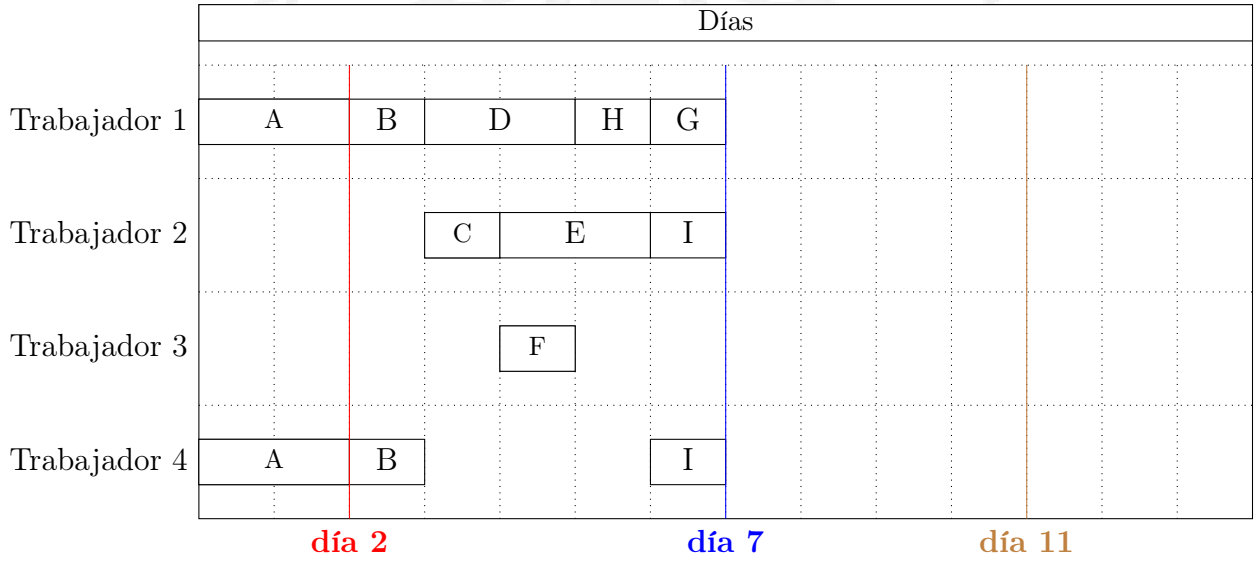

Figura 4.2: Diagrama para 4 trabajadores.

Ya en el caso de emplear un cuarto trabajador que reduce el tiempo de ejecución de una tarea en 2 días para todas tareas de igual o más de 3 días de duración el resultado es de **7 días**. Esto es debido a que puede llegar mucho antes a alcanzar la máxima eficiencia entre los trabajadores y además permite mayor flexibilidad a la hora de elegir qué trabajador hace qué tarea obteniendo así menores tiempos.

5. Ejercicio 5

Para este ejercicio disponemos de un código de python que nos genera un grafo de N nodos y M aristas sobre una semilla S. Estos datos debemos pasarlos a MiniZinc y colorear, ojo, a las aristas no a los nodos. Antes de empezar con el código realizé algunos ejercicios sencillos sobre un grafo de 4 nodos y conexiones secuenciales del 1 al 4... luego probé con los casos esquinas como lo son los grafos completos... y llegué a la conclusión obvia que el máximo número de colores sería equivalente al número de nodos que hay en nuestro grafo, ya que sería el máximo número de aristas salientes posibles de un nodo concreto.

Con esta información podemos establecer una cota superior y ayudar a MiniZinc a que converja antes. La implementación consiste en utilizar un esquema de matriz de adyacencia que se accede mediante los valores que corresponde a la pareja arista. Cada posición de la matriz que haga falta le asignaremos un color de 1..MAX. Si accedemos a la matriz en la fila 1 y columna 2, asignamos el mismo valor a fila 2, columna 1. Con esto, con simplesmente hacer uso de la librería "alldifferent_except_0", por fila, aseguramos que todas aristas tengan colores diferentes. Para evitar soluciones simétricas he impuesto que los colores van en orden creciente con la ayuda de "value_precede" según el manual de MiniZinc.

Tamaño del Grafo	Número de Colores	Runtime (segundos)
N=4, M=6	3	0.161
N=6, M=15	5	0.161
N=8, M=28	6	0.159
N=10, M=45	8	0.172
N=12, M=66	8	0.175
N=14, M=91	11	0.192
N=100, M=4950	75	11.941
N=125, M=7750	90	30.452

El ejercicio en si parece bastante escalable en tiempo de cómputo para el problema de coloreo de aristas en el caso de mi implementación. Sin embargo, es evidente observar que en complejidad espacial estoy utilizando una cantidad enorme de memoria. Una pequeña mejora sería modificar la representación para una matriz triangular y reducir a la mitad el uso de memoria.

De hecho, la razón por la que he parado en N=125, es debido a que con valores más grandes como {1000,500,250,200,150} me he quedado rápidamente sin memoria y eso que dispongo de 16GB de RAM. Por lo que es evidente, que incluso si utilizaramos la matriz triangular no sería escalable en esta versión. Habría que buscar otra implementación.