Práctica 2

Programación con Restricciones

Borja Aday Guadalupe Luis
Alejandro Toledo Viñoly

Índice

1. Problema.

El problema que tenemos que resolver está basado en la organización de la fabricación de un producto final mediante el refinado de distintos aceites. Nuestro objetivo es conseguir una solución que cumpla todas las restricciones pedidas y maximizar los beneficios finales de la empresa.

1.1. Variables de entrada.

Las variables de entrada son:

- NAceitesVegetales: Cantidad de aceites vegetales.
- NAceitesNoVegetales: Cantidad de aceites no vegetales.
- Meses: Número de meses para los que hay que organizar la producción.
- K: Máximo número de aceites que puedo mezclar mensualmente.
- T: Mínimas toneladas de aceite que podemos refinar de cada aceite que mezclemos.
- Precios: Tabla que nos indica los precios de cada aceite mensualmente.
- PreciosMensual: Lista de precios a los que podemos vender nuestro producto.
- MAXV: Máximas toneladas de aceite vegetal que podemos refinar.
- MAXN: Máximas toneladas de aceite no vegetal que podemos refinar.
- MCAP: Capacidad máxima del almacén de cada aceite.
- MCAPR: Capacidad máxima del almacén de aceite refinado.
- CA: Costes de almacenaje de aceite no refinado.
- CAR: Costes de almacenaje de aceite refinado.
- MinD: Mínima dureza de nuestro aceite final.
- MaxD: Máxima dureza de nuestro aceite final.
- Durezas: Lista que indica la dureza de cada aceite no refinado.
- RestanteFinal: Lista que nos indica la cantidad final que debe quedar en el almacén de cada aceite.
- AceitesAUsar: Tabla que nos indica las restricciones de uso de aceites en función de otros.
- MinB: Mínimo beneficio deseado por la empresa.

1.2. Restricciones del problema.

Las restricciones que se deben cumplir son:

1. Como máximo refinamos la cantidad disponible de cada aceite.

2. Como máximo se refinan MAXV toneladas de aceites vegetales y MAXN toneladas de aceites no vegetales.

3. Como máximo se almacenan MCAP toneladas de cada aceite no refinado.

4. Al final del último mes de producción deben quedar las cantidades exactas en el almacén de aceites no refinados.

5. La dureza de nuestro producto final debe estar entre MinD y MaxD.

6. No usamos más de K aceites mensualmente.

7. Si un aceite es seleccionado para refinar, debe usarse al menos T toneladas.

8. Cumplimos las restricciones de uso de aceites en función de otros.

9. Alcanzamos el beneficio mínimo: MinB.

1.3. Restricciones extra del problema.

Como añadido al problema hemos decidido aportar información extra, ahora también sabemos a que precio se vende nuestro producto dependiendo del mes del año. Por ello, tenemos la capacidad de guardarnos una cierta cantidad en nuestro almacén a cambio de un pequeño coste con el fin de vender este aceite refinado en el mes que más caro se venda nuestro producto.

Este extra es muy interesante para muchas empresas, pues si el producto final es un producto codiciado en algunas fechas determinadas del año, el programa calculará cuando almacenar y cuando vender para maximizar los beneficios finales. Por ejemplo, imaginemos que somos una empresa de ensamblado de PCs. Sabemos que los PCs se venden en gran medida a un precio mayor en navidad por el efecto de la alta demanda, entonces, podemos decidir sacar a la venta cierta cantidad en noviembre y guardarnos parte de ella para venderlos en diciembre a un precio mayor y ganar más dinero.

Las restricciones añadidas serían:

10. No podemos vender más aceite refinado del que tenemos.

11. El almacén de aceite refinado no puede superar su máxima capacidad: MCAPR.

2. Solución en MiniZinc.

2.1. Representación del problema.

Para representar el problema en el MiniZinc, hemos hecho uso de 3 variables de decisión más una cuarta que nos permite tener guardado el beneficio para poder mostrarlo y usarlo en las restricciones de forma sencilla.

```
# Tabla que nos indica cuanto comprar de cada aceite en cada mes.
array[1..Meses, 1..NAceites] of var 0..MCAP: compras;

# Tabla que nos indica cuanto refinar de cada aceite cada mes.
array[1..Meses, 1..NAceites] of var 0..max(MAXV, MAXN): refinado;

# Lista que nos indica que cantidad vender cada més.
array[1..Meses] of var 0..(MAXN + MAXV + MCAPR): ventaMensual;
```

De este modo, una representación de la solución en MiniZinc puede ser:

```
Beneficio
96238

Tabla de compras
0 200 0 202 48
0 0 7 702 0
147 0 0 0 156
0 200 0 0 0
0 0 0 0 0
11 203 0 211 0

Tabla de refinado
0 200 0 202 48
0 0 0 250 0
147 0 0 202 48
0 200 0 202 48
0 0 0 48 0
0 200 0 202 48

Lista de ventas
450
217
397
483
15
483

Tabla de restantes
0 0 0 0 0
0 0 7 452 0
0 0 7 250 108
0 0 7 48 60
0 0 7 0 60
11 3 7 9 12
```

2.2. Explicación de las restricciones.

En las restricciones que explicaremos a continuación, encontraremos llamadas a funciones que nos permiten o bien calcular el restante de un aceite concreto no refinado un determinado mes o la cantidad de aceite refinado en el almacén un cierto mes.

```
function var int: restanteMes(...) =
if mes = 0 then 0
else sum(i in 1..mes)(compras[i, aceite] - refinado[i, aceite]) endif;
```

Esta función se encarga de calcular la cantidad restante de un aceite determinado en un mes determinado.

```
function var int: restanteMesRefinado(...) =
if mes = 0 then 0
else sum(i in 1..mes, j in 1..NAceites)(refinado[i, j]) –
     sum(i in 1..mes)(ventaMensual[i]) endif;
```

Función que se encarga de calcular el aceite refinado restante no vendido hasta un mes concreto.

Restricción 1.

```
constraint forall(i in 1..Meses, j in 1..NAceites)(refinado[i, j] <=
(compras[i, j] + restanteMes(compras, refinado, i - 1, j)));
```

Para cada mes y cada aceite, comprobamos que no se ha refinado más toneladas que las disponibles, teniendo como disponibles lo comprado ese mismo mes más lo que nos sobró de ese aceite en el mes anterior.

Restricción 2.

```
constraint forall(i in 1..Meses)(sum(j in 1..NAceitesVegetales)(refinado[i,
j]) <= MAXV /\ sum(j in NAceitesVegetales + 1..NAceites)(refinado[i, j]) <=
MAXN);
```

Para cada mes, comprobamos que la suma de la cantidad refinada de los aceites vegetales no supera MAXV, lo mismo hacemos para los aceites no vegetales.

Restricción 3.

```
constraint forall(i in 1..Meses, j in 1..NAceites)((compras[i, j] +
restanteMes(compras, refinado, i - 1, j)) <= MCAP);
```

Para comprobar que nunca tenemos en el almacén de aceites no refinados más de MCAP de cada aceite, comprobamos para todos los meses y aceites que lo que nos sobró del mes anterior de ese aceite concreto más lo que compramos este mes no supera MCAP.

Restricción 4.

```
constraint forall(i in 1..NAceites)(restanteMes(compras, refinado, Meses, i)
= RestanteFinal[i]);
```

En este caso, comprobamos que para cada aceite, el restante del último mes de producción sea exactamente lo que nos dicen por la entrada.

Restricción 5.

```
constraint forall(i in 1..Meses)(sum(j in 1..NAceites)(refinado[i, j] *
Durezas[j]) <= MaxD * sum(j in 1..NAceites)(refinado[i, j]) /\ sum(j in
1..NAceites)(refinado[i, j] * Durezas[j]) >= MinD * sum(j in
1..NAceites)(refinado[i, j]));
```

Para comprobar que la dureza de nuestro producto final se encuentra dentro de los límites, tenemos que hacer la media ponderada y comprobar que es menor que MaxD y MinD. Si quisiéramos reproducir tal cual la fórmula de la media ponderada nos daríamos cuenta de que estropearíamos la linearidad del cálculo, por lo que despejando podemos obtener una fórmula equivalente que si sea lineal.

Restricción 6.

```
constraint forall(i in 1..Meses)(sum(j in 1..NAceites)(bool2int(refinado[i,
j] > 0)) <= K);
```

En este caso, para comprobar que nunca usamos mas de K aceites en un mismo mes, para todos los meses comprobamos que lo suma de todos aquellos aceites en los que haya refinado al menos una tonelada sea menor que K.

Restricción 7.

```
constraint forall(i in 1..Meses, j in 1..NAceites)(refinado[i, j] > 0 ->
refinado[i, j] >= T);
```

Para que no podamos usar menos de T toneladas de un aceite, hemos propuesto que si has usado más de 0 toneladas, entonces debas usar al menos T toneladas.

Restricción 8.

```
constraint forall(i in 1..Meses, j in 1..NAceites, k in
1..NAceites)(refinado[i, j] > 0 /\ AceitesAUsar[j, k] = 1 -> refinado[i, k] >
0);
```

En la tabla de AceitesAUsar, nos encontramos para cada aceite, cuales tenemos que usar si el primero de estos es utilizado. Entonces, para cada mes y cada aceite si hemos usado ese aceite, obligamos a que todos los marcados como true en su fila también sean utilizados.

Restricción 9.

```
constraint beneficio >= MinB;
```

Como explicamos anteriormente, beneficio es una variable de decisión que igualamos al beneficio final de la producción tras todos los meses para tener un acceso fácil a esa variable, por ello esta restricción es muy sencilla de escribir indicando solamente que beneficio tiene que ser mayor o igual a MinB.

Restricción 10.

```
constraint forall(i in 1..Meses)(ventaMensual[i] <= sum(j in
1..NAceites)(refinado[i, j]) + restanteMesRefinado(refinado, ventaMensual, i
- 1));
```

En esta restricción comprobamos para cada mes que la venta del aceite refinado sea menor o igual al disponible, siendo el disponible la cantidad refinada de todos los aceites en ese mes concreto más el aceite restante del mes anterior que fue refinado pero no vendido.

Restricción 11.

```
constraint forall(i in 1..Meses)(restanteMesRefinado(refinado, ventaMensual,
i) <= MCAPR);
```

Dado que el almacén de aceite refinado tiene una capacidad de MCAPR, tenemos que comprobar que lo refinado del mes menos lo vendido sea menor que MCAPR, por el contrario, no tendríamos espacio para guardar el sobrante no vendido.

2.3. Tiempos de ejecución.

En esta tabla se pueden ver reflejados los tiempos de ejecución que maximiza los beneficios en función de las entradas. Disponemos de 3 inputs de cada tamaño, por lo que veremos que para inputs del mismo tamaño pero con restricciones más o menos ajustadas varían su tiempo de ejecución tanto como el tamaño del problema en sí.

	NAceites = 5 Meses = 6	NAceites = 11 Meses = 8	NAceites = 30 Meses = 12
Input 0	401msec	772msec	29s 652msec
Input 1	768msec	812msec	2m 30s
Input 2	1s 185msec	2s 254msec	3m 2s

3. Solución en SMT.

3.1. Representación del problema.

En esta segunda solución, la solución al problema se ha representado mediante variables enteras. Como no podemos usar matrices ni listas, ahora se ha creado una variable para cada celda de las matrices anteriores, de modo que refinado_0_0 representaría la cantidad del aceite 0 que se ha refinado en enero.

De este modo, un ejemplo de salida en SMT para 5 aceites y 6 meses sería la siguiente:

```
((beneficio 96238))
((compras_0_0 0))
((compras_0_1 200))
((compras_0_2 0))
((compras_0_3 202))
((compras_0_4 48))
((compras_1_0 0))
((compras_1_1 0))
((compras_1_2 7))
((compras_1_3 702))
((compras_1_4 0))
((compras_2_0 147))
((compras_2_1 0))
((compras_2_2 0))
((compras_2_3 0))
((compras_2_4 156))
((compras_3_0 0))
((compras_3_1 200))
((compras_3_2 0))
((compras_3_3 0))
((compras_3_4 0))
((compras_4_0 0))
((compras_4_1 0))
((compras_4_2 0))
((compras_4_3 0))
((compras_4_4 0))
((compras_5_0 11))
((compras_5_1 203))
((compras_5_2 0))
((compras_5_3 211))
((compras_5_4 0))
((venta_0 450))
((venta_1 217))
((venta_2 397))
((venta_3 483))
((venta_4 15))
((venta_5 483))
((refinado_0_0 0))
((refinado_0_1 200))
((refinado_0_2 0))
((refinado_0_3 202))
((refinado_0_4 48))
((refinado_1_0 0))
((refinado_1_1 0))
((refinado_1_2 0))
((refinado_1_3 250))
((refinado_1_4 0))
((refinado_2_0 147))
((refinado_2_1 0))
((refinado_2_2 0))
((refinado_2_3 202))
((refinado_2_4 48))
((refinado_3_0 0))
((refinado_3_1 200))
((refinado_3_2 0))
((refinado_3_3 202))
((refinado_3_4 48))
((refinado_4_0 0))
((refinado_4_1 0))
((refinado_4_2 0))
((refinado_4_3 48))
((refinado_4_4 0))
((refinado_5_0 0))
((refinado_5_1 200))
((refinado_5_2 0))
((refinado_5_3 202))
((refinado_5_4 48))
```

3.2. Explicación de las restricciones.

Para la conversión de restricciones de MiniZinc a SMT, hemos realizado un fichero Python que se encarga de traducir las restricciones con un fichero de entrada con datos a el nuevo fichero en lenguaje SMT2. Para ello simplemente se ha realizado la traducción de las restricciones en MiniZinc a código SMT, por lo que no hace falta volver a explicar lo que hace cada restricción.

3.3. Tiempos de ejecución.

	NAceites = 5 Meses = 6	NAceites = 11 Meses = 8	NAceites = 30 Meses = 12
Input 0	570msec	10s 800msec	12m 48s
Input 1	9s 970msec	5s 20msec	> 30m
Input 2	910msec	14s 160msec	> 50m

4. Comparación de soluciones.

	NAceites = 5 Meses = 6	NAceites = 11 Meses = 8	NAceites = 30 Meses = 12
Input 0 - MiniZinc	401msec	772msec	29s 652msec
Input 0 - SMT	570msec	10s 800msec	12m 48s

	NAceites = 5 Meses = 6	NAceites = 11 Meses = 8	NAceites = 30 Meses = 12
Input 1 - MiniZinc	1s 185msec	2s 254msec	3m 2s
Input 1 - SMT	9s 970msec	5s 20msec	> 30m

	NAceites = 5 Meses = 6	NAceites = 11 Meses = 8	NAceites = 30 Meses = 12
Input 2 - MiniZinc	0.69 s	22.40 s	2.38 s
Input 2 - SMT	910msec	14s 160msec	> 50m

5. Conclusión.

Como conclusión cabe destacar que tras haber realizado múltiples pruebas con ambas soluciones, la solución de MiniZinc nos da mejores resultados que la solución propuesta en SMT.

A pesar de ello, SMT es una herramienta muy buena si se usa de forma correcta, en nuestro caso, hemos hecho las tablas de tiempos de ejecución con la maximización de cada uno de los problemas y es por eso que SMT tarda más, ya que hemos usado la función maximize en lugar de usar solo Max-SMT. Cuando quitamos en ambos la opción de maximizar, SMT iguala e incluso mejora los tiempos de MiniZinc.

Al fin y al cabo ambas son herramientas muy potentes que resuelven problemas muy complejos de forma rápida y sencilla.