Práctica 1

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 los trabajadores en una fábrica que nunca para la producción. Nuestro objetivo es conseguir una solución que cumpla todas las restricciones pedidas o maximizar las restricciones pedidas que hayamos podido cumplir.

Las variables de entrada son:

- D: Cantidad de días que hay que organizar.
- T: Cantidad de trabajadores disponibles.
- MaxDT: Máximo de días consecutivos trabajados.
- MaxDL: Máximo de días libres consecutivos.
- MinDT: Mínima cantidad de días trabajados en los D días.
- N1: Número de trabajadores necesarios en el turno 1.
- N2: Número de trabajadores necesarios en el turno 2.
- N3: Número de trabajadores necesarios en el turno 3.
- A: Mínima afinidad que debe tener un trabajador con el resto de trabajadores de su mismo turno.
- Afinidad: Matriz de tamaño TxT que define el grado de afinidad entre dos trabajadores.
- C: Número de categorías de trabajadores que tiene la empresa.
- Categorías: Array de tamaño T que nos indica la categoría para cada trabajador.
- MinTC: Array de tamaño 3xC que nos indica cuantos trabajadores son necesarios de cada categoría para cada turno.
- DiasLibres: Matriz de tamaño TxD que nos indica los días libres pedidos por cada trabajador.
- TurnosNoDeseados: Matriz de tamaño TxD que nos indica los turnos no deseados pedidos por cada trabajador.

Las restricciones que se deben cumplir son:

1. Cada turno tiene el número de trabajadores (N1, N2 o N3) que le corresponde.

2. Un trabajador solo puede estar en un turno cada día.

3. Garantizar que nadie trabaja MaxDT días consecutivos.

4. Garantizar que nadie tiene MaxDL días libres seguidos.

5. Garantizar que todos trabajan como mínimo MinDT días en los D días.

6. Un trabajador no puede hacer el último turno de un día y el primero del día siguiente dos veces seguidas en cuatro días consecutivos.

7. Dada la matriz Afinidad y un número A, para cada trabajador de un turno, la suma de las afinidades con el resto de trabajadores de ese mismo turno debe ser de al menos A.

8. En cada turno debe haber el número de trabajadores que indique MinTC de cada categoría.

9. Los trabajadores no trabajan los días libres que se han pedido en DiasLibres.

10. Los trabajadores no trabajan los turnos que han pedido no trabajar en TurnosNoDeseados.

2. Primera solución.

2.1. Representación del problema.

En esta primera solución, se ha representado el problema en una matriz de TxD, donde se indica para cada trabajador y cada día, si trabaja en el turno 1, turno2, turno3 o libra.

De este modo, una matriz solución para 10 trabajadores y 10 días podría ser la siguiente:

```
L1 T2 L1 T3 T2 L1 T1 T2 L1 L1
T1 L1 T1 T1 L1 T1 T2 L1 T3 T1
T3 T3 L1 T2 T3 L1 T3 L1 T2 T3
T2 L1 T3 L1 T1 T3 L1 T3 L1 T2
L1 T1 T2 L1 L1 T2 L1 T1 T1 L1
T1 T3 L1 T2 T3 L1 T1 T1 L1 T1
T2 T1 L1 T1 L1 T1 L1 T3 T3 L1
L1 L1 T3 L1 T1 T3 L1 L1 T1 T3
L1 T2 T1 L1 T2 L1 T2 T2 L1 T2
T3 L1 T2 T3 L1 T2 T3 L1 T2 L1
```

2.2. Explicación de las restricciones.

Restricción 1.

```
constraint forall(i in 1..D)(sum(j in 1..T)(bool2int(sol[j,i] = T1)) = N1);
constraint forall(i in 1..D)(sum(j in 1..T)(bool2int(sol[j,i] = T2)) = N2);
constraint forall(i in 1..D)(sum(j in 1..T)(bool2int(sol[j,i] = T3)) = N3);
```

Para cada día, sumamos los trabajadores que trabajan en el turno 1 y eso debe ser igual a N1. Hacemos lo mismo con los demás turnos.

Restricción 2.

Esta restricción se cumple directamente por la forma en la que hemos representado el problema. Pues un trabajador en un día solo podrá tener asignado T1, T2, T3 o L1 pero nunca más de uno a la vez.

Restricción 3.

```
constraint forall(i in 1..T, j in 1..D - (MaxDT - 1))(sum(k in j..j + (MaxDT
- 1))(bool2int(sol[i,k] != L1)) < MaxDT);
```

Para comprobar que nadie trabaja MaxDT días seguidos, para cada trabajador creamos una ventana de tamaño MaxDT y sumamos los días en los que su asignación es distinto de L1, es decir, que trabaja, y por tanto, esa suma debe ser menor que MaxDT.

Restricción 4.

```
constraint forall(i in 1..T, j in 1..D - (MaxDL - 1))(sum(k in j..j + (MaxDL
- 1))(bool2int(sol[i,k] = L1 /\ not DiasLibres[i,k])) < MaxDL);
```

En este caso realizamos exactamente lo mismo, para cada trabajador creamos una ventana de tamaño MaxDL y sumamos todas las asignaciones L1 en los que no se haya pedido el día libre (pues el día libre cuenta como trabajado) y esa suma debe ser menos que MaxDL.

Restricción 5.

```
constraint forall(i in 1..T)(sum(j in 1..D)(bool2int(sol[i,j] != L1 \/
DiasLibres[i,j])) >= MinDT);
```

Para comprobar que todos los trabajadores trabajan MinDT días en los D días, para cada trabajador, sumamos todos los días que ha trabajado o que se ha pedido libres (pues estos días cuentan como trabajados) y esa suma debe ser mayor o igual que MinDT.

Restricción 6.

```
constraint forall(i in 1..T, j in 1..D - 3)((sol[i,j] = T3 /\ sol[i,j + 1] =
T1 /\ sol[i,j + 2] = T3) -> sol[i,j + 3] != T1);
```

En este caso, para cada trabajador y cada día, comprobamos si el trabajador ha trabajado ese día en el turno 3, el siguiente en el turno 1 y el siguiente en el turno 3, entonces, el cuarto día no puede trabajar en el turno 1.

Como para cada día accedemos a 3 posiciones más, solo es necesario comprobar esta condición hasta D - 3.

Restricción 7.

```
constraint forall(i in 1..D, j in 1..T where sol[j, i] != L1)(sum(k in 1..T
where j != k /\ sol[j, i] = sol[k, i])(Afinidad[j,k]) >= A);
```

Para comprobar que cada trabajador tiene un grado de afinidad con sus compañeros de turno mayor que A, para cada día y cada trabajador que trabaje ese día, buscamos cualquier otro trabajador distinto a él mismo que trabaje en el mismo turno y sumamos sus afinidades, esa suma debe ser mayor o igual que A.

Restricción 8.

```
constraint forall(i in 1..D, c in 1..C)(sum(j in 1..T)(bool2int(sol[j, i] =
T1 /\ Categorias[j] = c)) >= MinTC[c]);
constraint forall(i in 1..D, c in 1..C)(sum(j in 1..T)(bool2int(sol[j, i] =
T2 /\ Categorias[j] = c)) >= MinTC[C*1 + c]);
constraint forall(i in 1..D, c in 1..C)(sum(j in 1..T)(bool2int(sol[j, i] =
T3 /\ Categorias[j] = c)) >= MinTC[C*2 + c]);
```

Para cada día, cada categoría y cada turno, recorro los trabajadores que trabajan en ese turno y tienen esa categoría, y luego comparamos con la cantidad mínima exigida que se encuentra en MinTC.

Por ejemplo, para el turno 2, categoría 2 si existieran 4 categorías, deberíamos comparar con MinTC[4*1+2].

Restricción 9.

Esta restricción es una restricción especial, pues solo se utilizará cuando el problema que queramos resolver sea de satisfacción. En el caso de que queramos minimizar el número de días libres pedidos no concedidos, debemos comentarla para no obligar a dar todos los días libres solicitados.

```
constraint forall(i in 1..T, j in 1..D)(DiasLibres[i,j] -> sol[i,j] = L1);
```

En este caso recorremos la matriz de DiasLibres y si encontramos un true (se ha pedido el día libre), asignamos a la matriz solución en la posición de ese día y ese trabajador un L1.

Restricción 10.

Esta restricción es una restricción especial, pues solo se utilizará cuando el problema que queramos resolver sea de satisfacción. En el caso de que queramos minimizar el número de turnos no deseados pedidos no concedidos, debemos comentarla para no obligar a no dar todos los turnos no deseados solicitados.

```
constraint forall(i in 1..T, j in 1..D)((TurnosNoDeseados[i,j] != L1) ->
sol[i,j] != TurnosNoDeseados[i,j]);
```

En este caso recorremos la matriz de TurnosNoDeseados y si encontramos un algo distinto a L1 (se ha pedido un turno libre), decimos que la solución, para ese trabajador y ese día debe ser distinto del turno que se ha pedido.

2.3. Tiempos de ejecución.

	T = 10	T = 15	T = 20
Satisfacción	0.69 s	22.40 s	2.38 s
Optimización (1ª)	5m 52s	5m 33s	∞
Optimización	22m 44s	∞	∞

3. Segunda solución.

3.1. Representación del problema.

En esta segunda solución, la solución al problema se ha representado con una matriz de Tx(D*3), en este caso, hemos pasado de una matriz de enums, a una matriz de booleanos, donde cada día ocupa 3 columnas, haciendo referencia la primera de ellas al turno 1, la segunda al turno 2 y la tercera al turno 3.

Ejemplo de salida para 4 días y 3 trabajadores:

```
1 0 0 1 0 0 1 0 0 1 0 0
0 1 0 0 1 0 0 1 0 0 1 0
0 0 1 0 0 1 0 0 1 0 0 1

En este caso el trabajador 1 trabaja siempre en el turno 1, el 2 en el turno
2 y el 3 en el turno 3.
```

3.2. Explicación de las restricciones.

En este caso las restricciones son todas exactamente igual que en el caso anterior, pero caben destacar 4 cosas importantes:

- Ahora cada día está representado en 3 columnas, es por ello que las ventanas creadas en las anteriores restricciones ahora multiplican por 3 su tamaño.

- Como ahora utilizamos la variable D desde 1 hasta 3*D, para saber a qué día estamos haciendo referencia en la matriz de DiasLibres por ejemplo, tenemos que trabajar con la división entera.

- Del mismo modo, tenemos que utilizar el módulo para comprobar a que turno estamos haciendo referencia.

- En esta representación de la solución, es importante comprobar que ningún trabajador trabaja dos turnos el mismo día, y por ello añadimos la siguiente restricción:

```
constraint forall(i in 1..T, j in 1..3*D where j mod 3 = 1)(sol[i,j] -> (not
sol[i, j + 1] /\ not sol[i, j + 2]));
constraint forall(i in 1..T, j in 1..3*D where j mod 3 = 2)(sol[i,j] -> (not
sol[i, j - 1] /\ not sol[i, j + 1]));
constraint forall(i in 1..T, j in 1..3*D where j mod 3 = 0)(sol[i,j] -> (not
sol[i, j - 2] /\ not sol[i, j - 1]));
```

Para cada trabajador y cada turno 1 de los D días, si trabaja, entonces no puede trabajar en el 2 y no puede trabajar en el 3. Lo mismo pasa con los turnos 2 y 3.

3.3. Tiempos de ejecución.

	T = 10	T = 15	T = 20
Satisfacción	1.93 s	2.59 s	∞
Optimización (1ª)	∞	∞	∞
Optimización	∞	∞	∞

4. Tercera solución.

4.1. Representación del problema.

En esta tercera solución, la solución al problema se ha representado con una matriz de 3xD pero ahora, cada posición de la matriz contiene un set de 1..T, de esta forma, cada posición de la matriz indica para ese día y ese turno, que trabajadores trabajan.

También utilizamos un array de tamaño D de set 1..T en el que nos indica para cada día, los trabajadores que han trabajado ese día en cualquier turno.

Ejemplo de salida para 10 días y 10 trabajadores:

```
{4,7} {3,6} {2,8} {5,7} {4,9} {2,7} {5,8} {1,6} {5,10} {1,6}
{2,6} {4,9} {5,10} {2,6} {1,6} {5,8} {1,10} {4,9} {2,7} {4,9}
{3,10} {1,7} {1,9} {3,8} {3,10} {4,9} {3,6} {2,7} {3,8} {3,8}
```

4.2. Explicación de las restricciones.

Las restricciones vuelven a ser prácticamente idénticas a las anteriores, pero en este caso hay algunas que se simplifican por la utilidad de los sets. Las restricciones beneficiadas son:

```
constraint forall(i in 1..D)(card(sol[1,i]) = N1 /\ card(sol[2,i]) = N2 /\
card(sol[3,i]) = N3);
```

Para comprobar que cada turno tiene la cantidad de trabajadores necesarios solo necesitamos hacer uso de la función cardinal sobre los sets.

```
constraint forall(i in 1..D, t1 in 1..3, t2 in 1..3 where t1 != t2)(sol[t1,i]
intersect sol[t2,i] = {});
```

Para comprobar que un trabajador no trabaja en más de un turno, comprobamos que la intersección entre todos los turnos diferentes de un día sea vacía, de este modo, sabemos que no aparece un mismo trabajador en dos turnos.

```
constraint forall(i in 1..D - (MaxDT -
1))(array_intersect(TrabajadoresDia[i..i + (MaxDT - 1)]) = {});
```

Para comprobar que ningún trabajador trabaja MaxDT días seguidos, hacemos la intersección de todos los trabajadores que han trabajado en una ventana de MaxDT días y como ninguno puede estar en los MaxDT sets a la vez, la intersección debe ser vacía.

4.3. Tiempos de ejecución.

	T = 10	T = 15	T = 20
Satisfacción	0.81 s	∞	∞
Optimización (1ª)	0.97 s	∞	∞
Optimiazción	∞	∞	∞

5. Conclusión.

Tras haber realizado la práctica mediante 3 aproximaciones diferentes, podemos concluir que la forma de programar mediante el uso de restricciones permite resolver problemas muy complejos de forma sencilla, pues se basa en hacer una traducción matemática de las restricciones a la representación del problema.

Sin embargo, algo que desconcierta un poco es la aleatoriedad en los tiempos de ejecución, tenemos 3 soluciones y 3 casos de prueba, y cada una de ella se comporta de forma diferente frente a las mismas soluciones con tiempos de ejecución muy dispares.