UNIVERSIDAD
COMPLUTENSE
MADRID

Práctica 1
Programación con Restricciones

Andrés Marí Piqueras
Gabriel Fernández Sacristán

15/03/2024

Andrés Marí Piqueras
Gabriel Fernández Sacristán

Problema de satisfacción

Considerad una fábrica que nunca para la producción. Por esta razón, es necesario cubrir cada día los tres turnos diarios (identificado con el 1, el 2 y el 3) asignando a cada uno de ellos los trabajadores que sean necesarios. Suponed que tenemos que hacer la planificación de turnos para “D” días consecutivos con los “T” trabajadores (cada trabajador tiene un número distinto asignado entre 1 y “T”) que tenemos y con “N1”, “N2” y “N3” trabajadores respectivamente en cada turno.

Variables

- *(int)* **T**: trabajadores.
- *(int)* **D**: días laborables.
- *(int)* **A**: afines mínimos.
- *(int)* **N1**, **N2**, **N3**: trabajadores mínimos por turno.
- *(int)* **MaxDT**: máximo de días trabajados consecutivos.
- *(int)* **MinDT**: mínimo de días trabajados.
- *(int)* **MaxDL**: máximo de días libres.
- *(array [1..T, 1..T] of bool)* **afines**: relaciones entre trabajadores.
- *(set of 1..T)* **R**: el conjunto de encargados.
- *(array [1..D, 1..T] of var 0..3)* **turnos**: (solución) calendario de turnos de trabajadores.

Restricciones

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

```
constraint forall(d in 1..D)(sum(t in 1..T)(turnos[d,t]=1)=N1);
constraint forall(d in 1..D)(sum(t in 1..T)(turnos[d,t]=2)=N2);
constraint forall(d in 1..D)(sum(t in 1..T)(turnos[d,t]=3)=N3);
```

Se comprueba para cada día que la suma de los trabajadores de cada turno es igual al número de trabajadores necesarios para cada turno proporcionado por los datos del problema.

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

```
array [1..D, 1..T] of var 0..3: turnos;
```

Se declara un array bidimensional, donde sus filas son los días y las columnas los trabajadores, cuyo contenido es el turno del trabajador en su día correspondiente (0 son vacaciones, 1 es el primer turno, 2 el segundo turno y 3 el tercer turno).

3. Dado un número “MaxDT”, garantizar que nadie trabaja “MaxDT” días consecutivos.

```
constraint forall(t in 1..T)(forall(d in 1..D-MaxDT)(sum(m in 0..MaxDT)(turnos[d+m,t]!=0)<=MaxDT));
```

Para cada trabajador, se comprueban los días en tramos de longitud *D-MaxDT* que en cada uno de ellos los trabajadores poseen al menos un día de vacaciones.

4. Dado un número “MaxDL”, garantizar que nadie tiene “MaxDL” días libres consecutivos.

```
constraint forall(t in 1..T)(forall(d in 1..D-MaxDL)(sum(m in 0..MaxDL)(turnos[d+m,t]=0)<=MaxDL));
```

Para cada trabajador, se comprueban los días en tramos de longitud *D-MaxDL* que en cada uno de ellos los trabajadores no poseen más de *MaxDL* días libres consecutivos.

5. Dado un número “MinDT”, garantizar que todos trabajan como mínimo “MinDT” en los “D” días.

```
constraint forall(t in 1..T)(sum(d in 1..D)(turnos[d,t]!=0)>=MinDT);
```

Se comprueba para cada trabajador que cumpla un mínimo de días trabajados en cualquier turno de la fábrica.

6. Si un trabajador hace el último turno de un día entonces no puede tener el primero del día siguiente.

```
constraint forall(t in 1..T)(forall(d in 1..D-1)(turnos[d,t]=3 -> turnos[d+1,t]!=1));
```

Se verifica la restricción que garantiza que si un trabajador ocupa el último turno de un día, entonces no estará asignado al primer turno del día siguiente.

7. Si un trabajador hace el último turno dos días seguidos entonces tiene que librar el día siguiente.

```
constraint forall(t in 1..T)(forall(d in 1..D-2)((turnos[d,t]=3 /\ turnos[d+1,t]=3) -> turnos[d+2,t]=0));
```

Nos aseguramos de que cada trabajador que no tenga descanso dos días seguidos esté libre al día siguiente.

8. Dada una serie de parejas de trabajadores afines, que se indicarán con una matriz 1..T x 1..T de Booleanos "afines", y un número "A", cada trabajador de un turno tiene que estar con al menos A trabajadores afines en ese turno.

```
constraint forall(d in 1..D)(forall(i in 1..3)
    (forall(t1 in 1..T where (turnos[d,t1]=i))
    (sum (t2 in 1..T  where (turnos[d,t2]=i))(afines[t1,t2])>= A)));
```

Comprueba que cada trabajador en cada turno está al menos con A trabajadores afines. Contando, gracias al array bidimensional "afines", los compañeros que trabajan a la vez que sean afines entre sí.

9. Sea "R" el conjunto de trabajadores (que se obtendrá como un set of números de trabajador) que tienen la categoría de encargados. En cada turno debe haber al menos un responsable.

```
constraint forall(d in 1..D)
    (forall(i in 1..3)
    (sum(t in R)(turnos[d, t]=i)>0));
```

Mira en cada turno de cada día que la suma de encargados sea mayor que 0.

Compilación

ejemplos1.dzn:

- Gecode 6.3.0:

```
turnos =
[| 2, 3, 2, 3, 2, 3, 1, 0, 1, 0, 3, 1
 | 1, 2, 1, 2, 3, 0, 0, 3, 3, 3, 2, 1
 | 3, 3, 3, 1, 2, 1, 1, 3, 2, 2, 0, 0
 | 2, 2, 2, 1, 1, 0, 3, 0, 3, 3, 3, 1
 | 2, 1, 2, 1, 1, 3, 3, 3, 3, 2, 0, 0
 |];
----------
Finished in 224msec.
```

- Chuffed 0.12.1:

```
turnos =
[| 0, 3, 0, 1, 1, 2, 3, 1, 3, 2, 3, 2
 | 1, 2, 1, 3, 1, 0, 2, 0, 2, 3, 3, 3
 | 1, 0, 1, 3, 2, 3, 3, 3, 1, 2, 0, 2
 | 1, 1, 1, 0, 0, 2, 2, 2, 3, 3, 3, 3
 | 1, 0, 0, 1, 1, 2, 2, 2, 3, 3, 3, 3
 |];
----------
Finished in 191msec.
```

- HiGHS 1.5.1:

```
turnos =
[| 2, 0, 2, 0, 3, 1, 1, 1, 3, 3, 3, 2
 | 3, 1, 3, 1, 3, 1, 2, 0, 2, 0, 2, 3
 | 2, 1, 0, 3, 0, 3, 1, 3, 2, 1, 3, 2
 | 3, 0, 1, 2, 2, 3, 1, 3, 3, 1, 0, 2
 | 3, 1, 3, 1, 3, 0, 2, 0, 2, 3, 1, 2
 |];
----------
Finished in 864msec.
```

Problemas de optimización

Variables

Optimización 1

```
array [1..D, 1..T] of bool: vacaciones;
```

En la matriz “vacaciones”, se emplea el valor 'true' para indicar la solicitud de vacaciones por parte del trabajador en la columna respectiva, mientras que se utiliza 'false' para indicar que el trabajador no solicita vacaciones en esas fechas.

Optimización 2

```
array [1..D, 1..T] of 0..3: turnos_no;
```

Por otro lado, en la matriz "turnos_no", cada trabajador específica en cada columna qué turno de cada día preferiría evitar, siendo el 1 el primer turno, 2 el segundo y 3 el tercero.

Restricciones

Optimización 1

```
constraint forall(t in 1..T)(forall(d in 1..D-MaxDL)
(sum(m in 0..MaxDL)(turnos[d+m,t]=0 /\ not vacaciones[d+m,t])<=MaxDL));
```

Asegurar que nadie disfrute de una racha de "MaxDL" días libres consecutivos, dada una cantidad máxima "MaxDL" e ignorando las vacaciones pedidas.

```
constraint forall(t in 1..T)(sum(d in 1..D)(turnos[d,t]!=0 \/ vacaciones[d,t])>=MinDT);
```

Dado un número “MinDT”, garantizar que todos trabajan como mínimo “MinDT” en los “D” días, salvo que se haya pedido vacaciones.

```
solve minimize (sum(t in 1..T, d in 1..D)(turnos[d,t]!=0 /\ vacaciones[d,t]));
```

Minimizar el número de veces que hacemos trabajar a alguien cuando ha solicitado vacaciones.

Optimización 2

```
solve minimize (sum(t in 1..T,d in 1..D)(
        (turnos[d,t]=turnos_no[d,t])*2^sum(dd in 1..d-1)(turnos[dd,t]=turnos_no[dd,t])

));
```

Reduciendo la frecuencia de rechazos a solicitudes y aplicando penalizaciones exponenciales por rechazos recurrentes en un trabajador.

Compilación

Optimización 1

ejemplos1b.dzn:

- Gecode 6.3.0:

```
turnos =
[| 2, 3, 1
 | 3, 2, 1
 | 2, 3, 1
 | 3, 2, 1
 | 3, 2, 1
 |];
----------
==========
Finished in 124msec.
```

- Chuffed 0.12.1:

```
turnos =
[| 1, 3, 2
 | 1, 2, 3
 | 1, 3, 2
 | 1, 2, 3
 | 1, 2, 3
 |];
----------
==========
% [X_INTRODUCED_15_ > 14]
Finished in 118msec.
```

- HiGHS 1.5.1:

```
turnos =
[| 3, 1, 2
 | 2, 1, 3
 | 1, 3, 2
 | 1, 2, 3
 | 1, 2, 3
 |];
----------
==========
Finished in 130msec.
```

ejemplos2b.dzn:

- Gecode 6.3.0:

```
turnos =
[| 2, 1, 2, 3, 1
 | 3, 1, 2, 2, 1
 | 2, 2, 3, 1, 1
 | 1, 3, 2, 2, 1
 | 2, 2, 3, 1, 1
 | 3, 1, 2, 2, 1
 | 2, 3, 1, 2, 1
 |];
----------
==========
Finished in 4m 11s.
```

- Chuffed 0.12.1:

```
turnos =
[| 1, 3, 1, 2, 2
 | 2, 2, 1, 3, 1
 | 3, 1, 2, 2, 1
 | 2, 1, 3, 1, 2
 | 1, 3, 2, 1, 2
 | 1, 2, 1, 3, 2
 | 2, 1, 2, 3, 1
 |];
----------
==========
% [X_INTRODUCED_35_ > 34]
Finished in 142msec.
```

- HiGHS 1.5.1:

```
turnos =
[| 3, 2, 2, 1, 1
 | 2, 3, 1, 1, 2
 | 1, 2, 1, 3, 2
 | 1, 1, 3, 2, 2
 | 3, 1, 2, 1, 2
 | 2, 3, 2, 1, 1
 | 3, 2, 1, 2, 1
 |];
----------
==========
Finished in 165msec.
```

Optimización 2

ejemplos1c.dzn:

- Gecode 6.3.0:

```
----------
Turnos:
[|2, 1, 3
 |3, 1, 2
 |2, 3, 1
 |3, 2, 1
 |3, 2, 1
 |];
Peticiones denegadas: 5
----------
==========
Finished in 124msec.
```

- Chuffed 0.12.1: (2^n da error)

```
arithmetic.cpp:237: The constraint int_pow is not yet supported for non-negative base and exponent integer!=====ERROR=====
Process finished with non-zero exit code 1.
Finished in 1s 877msec.
```

- HiGHS 1.5.1:

```
Turnos:
[|1, 3, 2
 |1, 2, 3
 |3, 1, 2
 |2, 3, 1
 |3, 2, 1
 |];
Peticiones denegadas: 5
----------
==========
Finished in 316msec.
```

ejemplos2c.dzn:

- Gecode 6.3.0:

```
Turnos:
[|2, 1, 2, 3, 1
 |1, 3, 1, 2, 2
 |3, 2, 1, 2, 1
 |2, 3, 2, 1, 1
 |3, 2, 1, 1, 2
 |2, 2, 3, 1, 1
 |1, 1, 3, 2, 2
 |];
Peticiones denegadas: 7
----------
==========
Finished in 36s 442msec.
```

- Chuffed 0.12.1:(2^n da error)

```
arithmetic.cpp:237: The constraint int_pow is not yet supported for non-negative base and exponent integer!=====ERROR=====
Process finished with non-zero exit code 1.
Finished in 1s 877msec.
```

- HiGHS 1.5.1:

```
Turnos:
[|2, 1, 1, 3, 2
 |2, 1, 3, 2, 1
 |2, 3, 2, 1, 1
 |2, 2, 1, 3, 1
 |1, 3, 1, 2, 2
 |3, 2, 1, 1, 2
 |3, 2, 1, 2, 1
 |];
Peticiones denegadas: 7
----------
==========
Finished in 596msec.
```