UNIVERSIDAD COMPLUTENSE
MADRID

Práctica 1

Alberto García Doménech & David Gody Ruiz

Consideramos una fábrica que nunca para la producción. Por esta razón, es necesario cubrir cada día los tres turnos diarios asignando a cada uno de ellos los trabajadores que sean necesarios. Suponemos que tenemos que hacer la planificación de turnos para D días consecutivos con los T trabajadores que tenemos y con N1, N2 y N3 trabajadores respectivamente en cada turno.

Hemos codificado la solución mediante una matriz de tamaño DXT, con valores posibles entre 0 y 3, significando el valor 0 que el trabajador ese día libra, 1 que está en el turno 1, 2 en el turno 2 y 3 en el turno 3.
Solución de ejemplo, donde las filas son los días y las columnas son los trabajadores.

TRABJ |1 |2 |3 |4 |5 |6 |7 |8 |9 |10
DIA 1:|2 |3 |0 |0 |3 |0 |1 |2 |2 |1
DIA 2:|2 |2 |3 |3 |0 |2 |1 |0 |0 |1
DIA 3:|0 |3 |2 |2 |3 |2 |1 |0 |0 |1
DIA 4:|1 |2 |1 |2 |2 |0 |0 |3 |3 |0
DIA 5:|0 |3 |2 |1 |2 |1 |0 |2 |3 |0

Hemos implementado las siguientes restricciones en el apartado de satisfacción:

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

```
constraint forall (i in DIAS) ((sum(j in TRABAJADORES) (bool2int(sol[i,j] = 1)))=N1);
constraint forall (i in DIAS) ((sum(j in TRABAJADORES) (bool2int(sol[i,j] = 2)))=N2);
constraint forall (i in DIAS) ((sum(j in TRABAJADORES) (bool2int(sol[i,j] = 3)))=N3);
```

2. Un trabajador solo puede estar en un turno cada día:
   Debido a la representación de nuestra solución esta restricción está implementada de manera implícita ya que cada trabajador solo puede tener un valor entre 0 y 3.

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

```
constraint forall (i in TRABAJADORES) ((forall (j in 1..(D-MaxDT+1)) ((sum (k in 0..(MaxDT-1)) (bool2int(sol[j+k,i] = 0))) >= 1)));
```

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

```
constraint forall (i in TRABAJADORES) ((forall (j in 1..(D-MaxDL+1)) ((sum (k in 0..(MaxDL-1)) (bool2int(sol[j+k,i] != 0))) >= 1)));
```

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

```
constraint forall (i in TRABAJADORES) ((sum (j in DIAS) (bool2int((sol[j,i] != 0)))) >= MinDT);
```

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

```
constraint forall (j in TRABAJADORES) (forall (i in 1..D-3) (sum (k in 0..2) (bool2int(sol[i+k, j] = 3 /\ sol[i+k+1, j] = 1)) < 2));
```

7. Dada una serie de parejas de trabajadores afines y un número A, cada trabajador de un turno tiene que estar con al menos A trabajadores afines en ese turno.

```
constraint forall(i in DIAS) ((forall (k in TRABAJADORES) (sum(w in TRABAJADORES where sol[i,w] == sol[i, k] /\ w!=k /\ sol[i, k] != 0)(afin[k,w])>= A)));
```

   afin es una matriz de tamaño NxN simétrica donde dos trabajadores son afines si su intersección es 1. Un trabajador no puede ser afin consigo mismo por lo que la diagonal principal tiene que estar llena de ceros.

8. Sea C el número de categorías. Dada la categoría de los trabajadores. En cada turno debe haber al menos numcateg[k] de cada categoría, siendo numcateg un array que indica el número de trabajadores de cada categoría que debe haber cada turno, y k el turno.

```
constraint forall (i in DIAS) (forall(k in TURNOS) (forall(c in CATEGORIAS)(sum(j in TRABAJADORES where categ[j] = c)(bool2int(sol[i,j]=k)) >=numcateg[k])));
```

   categ es un array de tamaño N donde a cada trabajador se le asigna una categoría (valor entre 1 y C)

Hemos implementado las siguientes restricciones en el apartado de optimización:

1. Añadid la posibilidad de que los trabajadores pidan días que no quieren trabajar:

```
solve minimize (sum (i in DIAS) (sum(j in TRABAJADORES) (bool2int(sol[i,j]!=0 /\ diasPedidos[j,i]=1))));
```

   diasPedidos es una matriz de tamaño TXD con valores 0 o 1, donde 1 significa que el trabajador pide no trabajar ese día y 0 que no lo pide

   Para tener en cuenta que estos días no cuentan en la restricción de días consecutivos libres:

```
constraint forall (i in TRABAJADORES) ((forall (j in 1..(D-MaxDL)) ((sum (k in 0..(MaxDL-1)) (bool2int(sol[j+k,i] != 0 \/ diasPedidos[i,j+k]=1))) >= 1)));
```

   Para descontarlos de MinDT:

```
constraint forall (i in TRABAJADORES) ((sum (j in DIAS) (bool2int((sol[j,i] != 0)))) >= (MinDT - (sum (j in DIAS) (diasPedidos[i,j]))));
```

   Formas de conseguir que los incumplimientos se distribuyan entre todos:

```
solve maximize trabj(sol,diasPedidos);
function var int: trabj(array[int,int] of var int: sol, array[int,int] of 0..1: diasPedidos) = (sum(j in TRABAJADORES) (bool2int(sum (i in DIAS)
(bool2int(sol[i,j]!=0 /\ diasPedidos[j,i]=1)) > 1)));
```

   En vez de minimizar el número de veces totales que se cumple que un trabajador que había pedido el día libre le toque trabajar, maximizamos el número de trabajadores a los que les ocurre eso al menos una vez (para que afecte al mayor número de trabajadores, dividiéndose entre todos más equitativamente).

2. Añadid la posibilidad de que los trabajadores pidan qué turno no quieren

```
solve minimize (sum (i in DIAS) (sum(j in TRABAJADORES) (bool2int(sol[i,j] = prefTurnos[j] /\ prefTurnos[j] != 0))));
```

   prefTurnos es un array de tamaño T donde cada trabajador indica que turno no quiere trabajar(de 1 a 3 o 0 si no tiene preferencia).

Si queremos que en vez de tener una preferencia única cada trabajador tenga una preferencia por día sería de este modo:

```
solve minimize (sum (i in DIAS) (sum(j in TRABAJADORES) (bool2int(sol[i,j] = prefDiasTurnos[j,i] /\ prefDiasTurnos[j,i] != 0))));
```

donde prefDiasTurnos es una matriz de tamaño TxD donde cada trabajador indica que turno no quiere cada día (de 1 a 3 o 0 si no tiene preferencia).

o

Para distribuir las veces que incumplimos las peticiones de los trabajadores maximizamos el número de trabajadores afectados por lo que las veces que les pase a cada uno será menor y estarán más repartidas.

```
solve maximize (sum(j in TRABAJADORES) (bool2int(sum (i in DIAS) (bool2int(sol[i,j] = prefTurnos[j] /\ prefTurnos[j] != 0)) > 0)));
```

En cuanto a las optimizaciones que tratan de repartir los incumplimientos entre los trabajadores, son buenas en una situación en la que los incumplimientos son elevados con la optimización base, ya que evitan que sufra solo un trabajador. Sin embargo, en situaciones con pocos incumplimientos, es decir, cuando los trabajadores no se pisan entre sí con sus peticiones es peor, ya que las incumple intencionadamente.

Tenemos 3 conjuntos de datos, cada cual con más trabajadores y más restrictivos.

Con el primer conjunto de datos (“*1.dzn*”) el programa tarda ~380 - ~400 msec en encontrar una solución. Si queremos que optimice el número de veces que hacemos trabajar a alguien que no quiere ese día conseguimos una solución óptima en ~420 msec. En cambio si queremos que distribuya esa optimización tarda ~400 msec consiguiendo una solución óptima donde denegamos más veces las peticiones de librar pero es más equitativa.
Si queremos optimizar el número de veces que hacemos trabajar a alguien un turno que no quiere conseguimos una solución donde no ignoramos ninguna petición en ~390-410 msec

Con el segundo conjunto de datos (“*2.dzn*”) el programa tarda ~560-600 msec en encontrar una solución. Con las diferentes optimizaciones tarda ~11 s en encontrar una solución óptima (donde ignoramos solo una vez la petición de librar un día) y ~620-640 msec en encontrar una solución óptima donde no ignoramos ninguna vez la petición de los turnos de los trabajadores.

Con el último conjunto de datos (“*3.dzn*”) el programa tarda ~750-800 msec en encontrar una solución válida. Para encontrar una solución donde solo ignoramos dos veces (la solución óptima) las peticiones de librar un día el programa tarda ~1’5 s y para encontrar una solución ignorando el mínimo de veces peticiones de turno tarda ~700 msec.