Informe de la Pràctica de Minizinc

Guillem Vidal

20 de gener de 2025

1 Explicació del model

Tinc dues variables principals:

schedule `array[days, stadiums] of var set of teams`

place `array[teams, days] of var stadiums`

La primera contindrà el calendari final dels partits, limitant que la cardinalitat dels seus conjunts hagi de ser o bé de 0, o bé de 2. I la segona descriu a quin estadi juga un equip en un dia concret, cosa que ajuda a limitar que els equips només puguin jugar un sol partit al dia.

`place` és com una variable auxiliar, ja que pren el seu valor únicament de `schedule` i simplement ajuda a restringir encara més la diversitat de valors que l'última pot prendre.

1.1 Restriccions

Amb aquesta restricció limito que el paràmetre d'entrada `fixes` només pugui tenir conjunts de cardinalitats menors o igual a 2, a més de limitar que la cardinalitat dels conjunts de `schdule` a 0 o 2:

```
constraint forall(d in days, s in stadiums)(
  count(t in fixes[d, s])(t >= 0) <= 2 /\
  (count(t in schedule[d, s])(t >= 0) == 2
  \/ count(t in schedule[d, s])(t >= 0) == 0)
);
```

Prosseguim amb la inicialització[1] de la variable `schedule` mitjançant `fixes`, on l'únic que requereixo és que el valor a la posició de `fixes` sigui un subconjunt del valor a la mateixa posició de `schedule`:

```
constraint forall(d in days, s in stadiums)(
  schedule[d, s] union fixes[d, s] == schedule[d, s]
);
```

[1] Dir inicialitzar en aquest context declaratiu i per restriccions és bastant incorrecte, però crec que fa bé la feina a l'hora d'explicar la intenció darrere la restricció.

Amb la següent restricció comparo els conjunts directament, per assegurar-me que no hi ha combinacions d'equips repetides en el calendari. No he pogut fer servir la funció predefinida `all_different` aquí, perquè els conjunts buits sí que s'han de poder repetir:

```
constraint forall(d1 in days, s1 in stadiums)(
  forall(d2 in days, s2 in stadiums)(
    (d1 == d2 /\ s1 == s2) \/
    schedule[d1, s1] == {} \/
    schedule[d1, s1] != schedule[d2, s2]
  )
);
```

Ara comencem a veure la variable `place`, la qual inicialitzem amb els continguts trobats a `schedule`:

```
constraint forall(d in days, s in stadiums)(
  forall(t in schedule[d, s])(
    place[t, d] == s
  )
);
```

En tenir un sol possible valor per cada equip i dia, estem limitant que un equip no pugui jugar dos cops o més el mateix dia.

Per acabar, tenim la restricció que ens permet assegurar que cap equip jugarà dues vegades o més al mateix estadi:

```
constraint forall(t in teams)(
  all_different([place[t, d] | d in days])
);
```

2 Eficiència

Aquestes proves s'han executat en un Lenovo ThinkPad E15 amb una CPU Intel Core i7 de 8 nuclis i 16 GB de memòria RAM.

Totes les proves s'han realitzat amb el `solve satisfy`, perquè tan sols retorni la primera solució trobada. Les altres optimitzacions per cada solució no m'han sortit prou bé com per fer-hi gaires proves.

He utilitzat el nivell d'optimització `-O5`.

Prova	*Temps (s)*
`t0`	1
`t1fix`	4
`t1`	6
`t2fix`	21
`t3fix`	121
`t2`	∞
`t3`	∞
`t4`	∞

Taula 1: Temps que ha tardat cada prova en arribar a una solució.