AI-Project

Federico Galli

Aprile 2024

1 Introduzione

Il seguente elaborato tratterà della modellazione di un *CSP* inerente lo scheduling di un calendario di partite di pallavolo, mirato a minimizzare il numero di partite in una giornata in una giornata. La minimizzazione verrà fatta prendendo come dati di riferimento un numero di squadre da 10 a 14 e un numero di città da 2 a 4.
Per semplicità ogni squadra gioca nel proprio palazzetto

2 Problema e generazione dei dati

Il quesito, come detto nell'introduzione, è un problema di soddisfacimento di vincoli. Il testo già ci dà i primi vincoli sui dati, si hanno:

- **n** squadre
- **m** palazzetti
- **c** città
- $n = m \ \wedge \ c < n$

L'assegnazione delle squadre ai palazzetti e dei palazzetti alle città è randomizzata attraverso il file *RandomData.py*.
Il file richiede di inserire in input:

- il numero delle squadre del campionato n. *(Il numero di palazzetti m sarà uguale a n di default)*
- il numero delle città c

Attraverso i dati assegna i valori a:

- n, m, c
- un array, di dimensione m, il cui indice indicherà il palazzetto i-esimo e il valore al suo interno la squadra di appartenenza
- una matrice di dimensioni $c \times m$ che tiene traccia della distribuzione dei palazzetti alle città

RandomData.py scriverà quindi i dati creati in un file, *RandomData.dzn*, con la giusta formattazione e verrà incluso in Playground.mzn così da fornirgli i dati del problema e permettergli la risoluzione del *CSP*.

3 Modellazione problema

Il problema è stato modellato attraverso una matrice *"calendario"*, di valori $\{0,1\}$, di dimensioni $N \times N \times C \times G$. Di queste N e C sono rispettivamente il numero di squadre e il numero di città, mentre G è il numero di giornate, che viene calcolato in due modi distinti:

- se N è pari allora $G = (N-1) \cdot 2$
- se N è dispari allora $G = N \cdot 2$

Ora procederò con la spiegazione dell'utilizzo della matrice *"calendario"* e dei vincoli imposti su di essa.
Le variabili *SQUADRE*$= 1 \ldots n$, *PALAZZETTI*$= 1 \ldots m$, *GIORNATE*$= 1 \ldots \{(n-1) \cdot 2 \ \vee \ n \cdot 2\}$ se n pari o dispari rispettivamente e *CITTA*$= 1 \ldots c$, sono set di interi che servono nei *forall* e i loro valori massimi corrispondono alle effettive dimensioni della matrice *calendario*.
Per memorizzare la randomizzazione dei palazzetti alle città ho utilizzato una matrice di dimensioni $[M \times C]$, *"pal_pc"* che viene generata nel file di dati *"RandomData.dzn"* dall'omonimo script python.
Per controllare il numero di partite giocate in ogni giornata in ogni città ho usato una matrice di dimensioni $[C \times M]$.

3.1 Vincoli

Ogni elemento c_{ijkg} della matrice *"calendario"* corrisponde ad una partita giocata tra le squadre i e j nella città k nella giornata g.
L'indice i, che serve a scorrere la prima dimensione, N, della matrice, ci dice quale squadra sta giocando in casa.

- Il primo vincolo inserito è quello di controllo sulle diagonali di ogni matrice $n \times n$. Poiché una squadra non può mai giocare contro se stessa si deve avere $calendario_{ijkg} = 0 \ \forall i = j$

```
constraint forall(i in SQUADRE, j in SQUADRE, k in CITTA, g in GIORNATE where i=j)
    (calendario[i,j,k,g]=0);
```

- Una squadra può giocare al massimo una volta per giornata quindi la somma sulle righe e sulle colonne di ogni sottomatrice $N \times N$ deve essere sempre minore o uguale a uno.

```
constraint forall(g in GIORNATE, k in CITTA, j in SQUADRE)
    (sum(i in SQUADRE)(calendario[i,j,k,g])<=1);
constraint forall(g in GIORNATE, k in CITTA, i in SQUADRE)
    (sum(j in SQUADRE)(calendario[i,j,k,g])<=1);
```

- Una giornata deve avere un numero di partite uguale alla parte intera inferiore di n/2, per considerare anche i casi in cui ci siano numero di squadre dispari.

```
constraint forall(g in GIORNATE)(sum(k in CITTA, i in SQUADRE, j in SQUADRE)
    (calendario[i,j,k,g])=n div 2);
```

  Quest'ultimo vincolo può sembrare ridondante rispetto al precedente ma dopo dei test mi sono reso conto che, anche se di poco, velocizza il processo di risoluzione del *CSP*. Di seguito due test fatti con 14 squadre in 4 città, il primo senza l'inserimento del vincolo e il secondo con l'inserimento del vincolo.

```
Running Playground.mzn                                   27s 500msec
Running Playground.mzn                                    24s 90msec
```

- Il seguente vincolo serve per evitare che ci possa essere sia l'andata che il ritorno della stesso scontro nella stessa giornata. Il vincolo è impostato in modo tale che quando viene trovato un elemento $c[i, j, k, g] = 1$ allora si controlla:
  - che la somma sulla j-esima colonna sia 0 all'interno delle k-esime città diverse da k
  - che la somma sulla j-esima riga sia 0 all'interno delle k-esime città diverse da k
  - che la somma sulla i-esima colonna sia 0 all'interno delle k-esime città diverse da k
  - che la somma sulla j-esima riga sia 0 all'interno delle k-esime città diverse da k
  - che la somma sulla j-esima colonna sia 0 all'interno della stessa k-esima città
  - che la somma sulla i-esima riga sia 0 all'interno della stessa k-esima città

```
constraint forall(i in SQUADRE, j in SQUADRE, k in CITTA, g in GIORNATE)(if calendario[i,j,
    k,g]=1 then
(forall(cit in CITTA where cit!=k)(sum(col in SQUADRE)(calendario[j,col,cit,g])=0)) /\
(forall(cit in CITTA where cit!=k)(sum(row in SQUADRE)(calendario[row,j,cit,g])=0)) /\
(forall(cit in CITTA where cit!=k)(sum(col in SQUADRE)(calendario[i,col,cit,g])=0)) /\
(forall(cit in CITTA where cit!=k)(sum(row in SQUADRE)(calendario[row,i,cit,g])=0)) /\
sum(row in SQUADRE)(calendario[j,row,k,g])=0 /\
sum(col in SQUADRE)(calendario[col,i,k,g])=0 /\
sum(gio in GIORNATE)(calendario[i,j,k,gio])=1
else true endif);
```

- Per assicurarmi che l'effettivo numero di partite sia $n * (n - 1)$ ho imposto un vincolo sul numero di $c_{ijkg} = 1$ nella matrice *"calendario"*

```
constraint count(i in SQUADRE, j in SQUADRE, k in PALAZZETTI, g in GIORNATE)
    (calendario[i,j,k,g]=1)=n*(n-1);
```

- Per controllare che le squadre, quando giocano in casa, giochino nella città giusta, ho fatto un controllo su quali c_{ijkg} possano prendere valore 1 in base alla città k. Per fare questo ho sfruttato l'array *"squad_pp"* in cui sono registrate quali squadre giocano in quali città e ho controllato che la somma sulla riga corrispondente alla squadra nelle altre città sia pari a zero.

```
constraint forall(p in SQUADRE, q in CITTA where pal_pc[p,q]!=0)(
    forall(g in GIORNATE, k in CITTA where k!=q)(
        sum(j in SQUADRE)(calendario[pal_pc[p,q],j,k,g])=0 ));
```

4 Risultati

Si sono eseguiti i test tramite lo script *VolleyballTournamentCalendar.py*. Per la randomizzazione dei dati si è usata la funzione di Python, *random.randint()* dalla libreria.
Il risultato che salta all'occhio per primo è che molto spesso il numero di partite nella città e nella giornata selezionata da minimizzare è pari a 0. Questo potrebbe essere dovuto al modo in cui la funzione *randint()* funziona. In generale si è riscontrato un aumento del numero di partite nella stessa città nella stessa giornata nei casi in cui le città sono solo 2. Questo, come ci si aspettava, è dovuto al fatto che è più probabile avere un caso tendente al caso pessimo di tutte le squadre tranne una, in una sola città quando si hanno molte squadre in poche città.

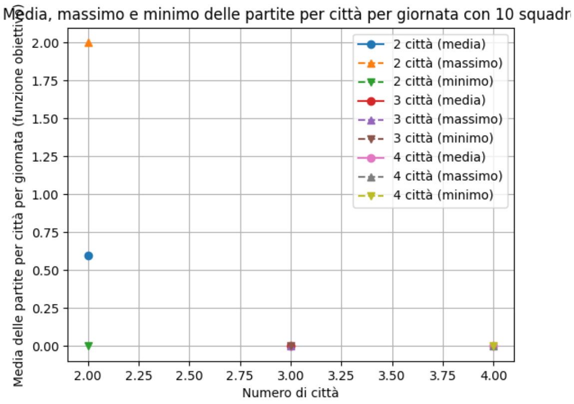

Figure 1: Risultati 10 squadre

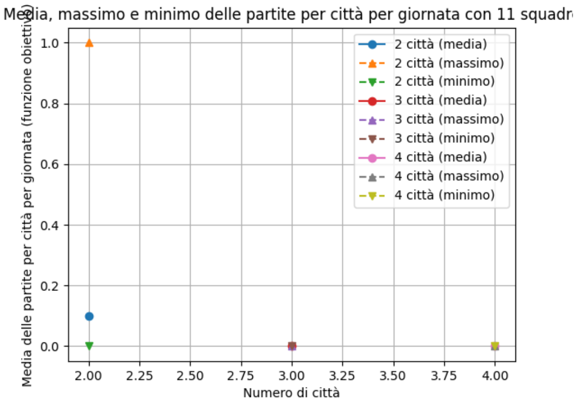

Figure 2: Risultati 11 squadre

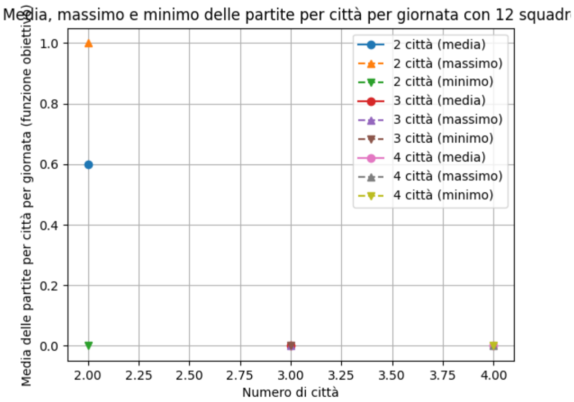

Figure 3: Risultati 12 squadre

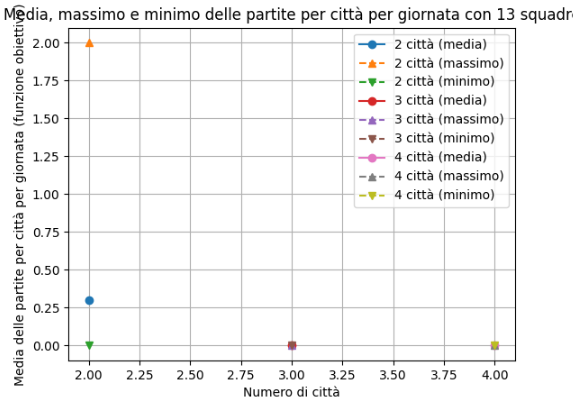

Figure 4: Risultati 13 squadre

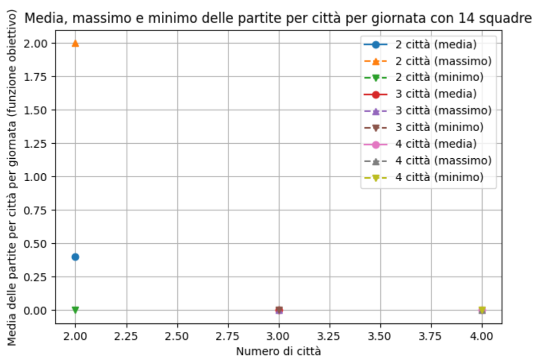

Figure 5: Risultati 14 squadre