Modello Tetravex

Marco Vignozzi

March 2023

1 Introduzione

In questa relazione viene descritto un modello del rompicapo Tetravex da me sviluppato in ambiente Minizinc.
In questo gioco sono presenti due scacchiere $N * N$: una che ha in ciascuna cella un tassello con un numero da 0 a 9 su ogni lato, l'altra vuota. Lo scopo del gioco è riempire la scacchiera vuota utilizzando tutti i tasselli presenti nell'altra, facendo in modo che le facce a contatto dei pezzi adiacenti abbiano lo stesso valore.
La Figura 1 mostra una rappresentazione del rompicapo per un problema con scacchiere 3x3.

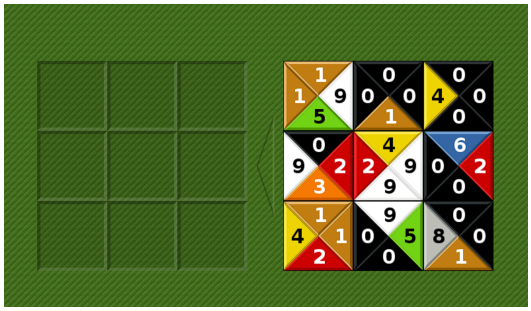

Figure 1: rappresentazione del gioco Tetravex per un problema 3x3

2 Modello

Il modello che ho creato permette di risolvere problemi per qualsiasi dimensione della scacchiera. Quest'ultimo valore deve essere fornito per ogni problema, insieme a ciascuno dei tasselli a disposizione (come spiegheremo in seguito). Non è possibile utilizzare più volte uno stesso tassello o ruotare gli stessi per ottenere la soluzione.

2.1 Struttura dati

La struttura non è complessa, sono presenti i seguenti elementi:

- `N` : rappresenta la larghezza (e l'altezza) della scacchiera
- `pezzi` : una matrice in cui ogni riga è un *array* di 4 valori da 0 a 9 che rappresenta un tassello
- `range` : set di valori da 1 al numero di tasselli (numero di righe di `pezzi`)
- `griglia` : matrice $N * N$ che rappresenta la scacchiera soluzione e associa ad ogni cella l'indice di riga nella matrice `pezzi` del tassello utilizzato.

Come ho già detto ciascun tassello è rappresentato da un *array* di 4 elementi. Questi rappresentano i valori sulle facce a partire da sinistra e andando in senso orario. I valori per ciascun pezzo saranno quindi dati nell'ordine:

```
TASSELLO = [SINISTRA, SU, DESTRA, GIU']
```

come nell'esempio in Figura 2.

Figure 2: un tassello della scacchiera. L'*array* che rappresenta questo tassello è nella forma [5, 3, 0, 4]

2.2 Variabili e vincoli

2.2.1 Variabili

Ho scelto di utilizzare come variabili le celle della scacchiera soluzione `griglia`. Il dominio di ciascuna variabile è il range di valori che va da 1 al numero di tasselli, rappresentato da `range`. Nel programma sviluppato su Minizinc ho quindi definito la matrice `griglia` come un *array* $N * N$ di variabili di tipo `range`.

2.2.2 Vincoli

I vincoli del problema possono essere espressi in questo modo:

Se due pezzi sono adiacenti allora le facce a contatto devono avere lo stesso numero **and** non è possibile usare due volte lo stesso pezzo.

La prima affermazione stabilisce la presenza di vincoli binari di uguaglianza sui valori delle facce a contatto di pezzi adiacenti. Questi sono stati definiti con il comando `costraint forall()` che ritorna la congiunzione logica dei vincoli binari nel corpo dell'istruzione, uno per ogni valore fornito tra parentesi.
La seconda affermazione implica la necessità di un vincolo globale di unicità dei valori inseriti nella `griglia`. Questo vincolo è stato espresso con il comando `costraint alldifferent()` che si assicura che ogni variabile specificata assuma un valore diverso.
Per maggiore chiarezza riporto di seguito le righe di codice che corrispondo alla definizione dei vincoli.

```
constraint forall(i in 1..N, j in 1..N-1)(
  pezzi[griglia[i,j], 3] = pezzi[griglia[i,j+1], 1]);

constraint forall(i in 1..N-1, j in 1..N)(
  pezzi[griglia[i,j], 4] = pezzi[griglia[i+1,j], 2]);

constraint alldifferent([griglia[i,j] | i,j in 1..N]);
```

2.3 Input e Output

Ciascun problema ha bisogno che vengano inizializzati due elementi: il valore di `N` e i valori degli $N * N$ *array*, uno per ogni riga di `pezzi`. Nel caso in cui il problema così fornito sia soddisfacibile il programma stamperà in output due rappresentazioni: la scacchiera iniziale, dove ogni pezzo appare nell'ordine in cui è stato scritto nella matrice `pezzi`, e la scacchiera soluzione.
Se il problema non è soddisfacibile l'output sarà quello standard di Minizinc, ovvero la stringa `"UNSATISFIABLE"`.
Se vengono forniti valori errati per i parametri di input il programma lo segnalerà con un output di errore.

3 Test

Per verificare la consistenza del modello ho condotto un totale di 8 test, verificando il corretto funzionamento con 2 istanze del problema per ciascun valore di `N` $\in [3, 6]$. Nelle Figure 3, 4, 5, 6 sono mostrate le scacchiere di input (*PEZZI*) e di output (*SOLUZIONE*) per tutti i test effettuati, che possono essere ripetuti con i dataset forniti insieme al programma.

(a) (b)

Figure 3: (a) e (b) mostrano i test per N=3

(a) (b)

Figure 4: (a) e (b) mostrano i test per N=4

(a) (b)

Figure 5: (a) e (b) mostrano i test per N=5

(a) (b)

Figure 6: (a) e (b) mostrano i test per N=6

4 Conclusioni

All'aumentare del parametro `N` e, in conseguenza di ciò, del numero di variabili e vincoli, i tempi necessari a terminare il programma sono rimasti molto vicini, dimostrando, seppur con scarso livello di significatività, l'efficacia del software nell'ottimizzazione della ricerca di una soluzione per un CSP.