Elaborato assegnato per l'esame finale di AI
Modellazione di Tetravex

Beatrice Vangi

21 gennaio 2022

L'elaborato assegnatomi mira a sviluppare un modello per formulare il gioco Tetravex come problema di soddisfacimento dei vincoli. Questo progetto è stato realizzato usando il liguaggio di modellizzazione MiniZinc

1 Problemi di soddisfacimento dei vincoli

I problemi di soddisfacimento dei vincoli (CSP) sono una particolare classe di problemi per i quali si ha un insieme noto di *variabili* con *domini* noti e un insieme di *vincoli* che pongono delle restrizioni sui valori che le suddette variabili possono assumere. Si deve assegnare un valore a ciascuna variabile in modo tale da rispettare tutti i vincoli.

2 Tetravex

Tetravex è un gioco di edge-matching. In una griglia $n \times n$ si devono sistemare delle tessere quadrate con un numero su ogni lato. Due tessere possono essere posizionate l'una accanto all'altra solo se i numeri su facce adiacenti combaciano.

3 MiniZinc

MiniZinc è un linguaggio open-source sviluppato per modellare problemi di soddisfacimento dei vincoli e di ottimizzazione. Una volta scritte le specifiche tramite modello (in un file .mzn) e dati (in un datafile .dzn), Minizinc le interpreta e le converte in linguaggio FlatZinc. Questo è progettato in modo tale da essere facilmente tradotto nella forma richiesta da risolutori di CSP, dunque le specifiche in questo formato possono poi essere risolte da più solver.

4 Modellizzazione

Vediamo ora in dettaglio le parti che compongono il progetto: i file nxn.dzn e il file del modello Tetravex.mzn.

4.1 nxn.dzn

In questi file sono presenti i dati che il modello necessita in ingresso. Ci sono otto file .dzn, ognuno contenente la dimensione della griglia e un'istanza del problema.

Ecco i parametri in dettaglio:

1. **N** rappresenta il numero di tessere su ogni riga e su ogni colonna da disporre sulla griglia. Questo parametro può assumere i valori da 3 a 6 compresi.

2. **puzzle** è un array bidimensionale con $N \times N$ righe e 4 colonne. Ogni riga rappresenta una tessera e i numeri al suo interno sono ordinati in modo tale che il primo numero si riferisca al numero in alto, il secondo a quello in basso, il terzo a quello a destra e il quarto a quello a sinistra [Nord, Sud, Est, Ovest].

L'array puzzle è stato riempito osservando le tessere generate randomicamente dal gioco al sito: `https://smart-games.org/en/tetravex/game/`. Pertanto tutte le istanze di puzzle presenti nei datafile hanno soluzione.

4.2 Tetravex.mzn

Questo file, dove è contenuto il modello, è il vero cuore del programma. È qui infatti che vengono definiti le variabili, i loro domini e i vincoli.

Nel caso in esame le variabili sono le posizioni che ciascuna tessera deve occupare e sono memorizzate in un array unidimensionale di $N \times N$ elementi. Ciascuna tessera è identificata da un numero diverso che va da 1 a $N \times N$. Sostanzialmente l'array indica l'ordine in cui devono essere disposte le tessere nella griglia. L'elemento nella i-esima posizione dell'array andrà disposto nella casella i-esima della griglia (contate da destra a snistra e poi a scendere).

Per quanto riguarda i vincoli, ce ne sono tre:

- il primo garantisce che due tessere possono essere posizionate accanto orizzontalmente solo se le facce a contatto hanno lo stesso numero. Per fare questo ho sfuttato un doppio ciclo che garantisse questa proprietà per ogni riga della griglia finale, poi memorizzata nell'array *res*.

- il secondo vincolo opera in modo analogo al primo, con la differenza che garantisce la proprietà veriticalmente. Ossia: due tessere possono essere posizionate l'una sotto l'altra se i lati a contatto coincidono in numero.

- il vincolo *alldifferent* è necessario per fare in modo che le tessere siano posizionate tutte. Per certi input particolari, infatti, potrebbe essere collocata due volte una stessa tessera, lasciandone un'altra inutilizzata.

Per risolvere il problema così modellato si deve quindi scrivere solve satisfy; questa riga è perfetta per i CSP nei quali ci basta trovare una qualsiasi soluzione che soddisfi i vincoli.

5 Interpretazione dell'output e Guida all'utilizzo

Per interpretare l'output è importante ricordare che ogni tessera è identificata da un numero corrispondente alla riga in cui è memorizzata nell'array in input *puzzle*.

Se il problema è insoddisfacibile, il programma mostra in uscita
"=====UNSATISFIABLE=====".

Nel caso in cui invece il problema abbia soluzione, in output viene restituito l'array *res*, ovvero un array di numeri compresi tra 1 e $N \times N$ ordinati in base alla posizione che deve occupare la tessera corrispondente nella griglia. Ovvero, la tessera in posizione i-esima nell'array dovrà essere posta in posizione i-esima nella griglia (contando da sinistra verso destra e poi dall'alto verso il basso).
Più formalmente, la tessera identificata dal numero in posizione i-esima nell'array stampato a schermo sarà da collocare nella griglia nella posizione di coordinate:
(r,c) = $(1+\lfloor\frac{i-1}{N}\rfloor, i-\lfloor\frac{i-1}{N}\rfloor\times$N).

Per riprodurre i risultati si deve digitare include "" (tra le virgolette si scriva il nome del datafile) nel file .mzn. Il risolutore utilizzato è stato Gecode, consiglio il suo impiego nella configurazione del solver.

6 Fonti

Per la realizzazione del progetto ho consultato i seguenti siti: `https://www.minizinc.org/doc-2.5.5/en/index.html` `http://www.hakank.org/minizinc/`
Per generare le istanze: `https://smart-games.org/en/tetravex/game/`