B003725 Intelligenza Artificiale (2020/21)
Quadrati colorati

Mistretta Marco 7005065

Gennaio 2021

Contents

1 Descrizione Problema

In questo esercizio si costruisce un modello MiniZinc per risolvere il seguente puzzle:
Sono dati nm quadrati con lati colorati (con colori scelti da un insieme di dimensione k). Il problema consiste nel disporre i quadrati su una griglia n per m in modo tale che i lati adiacenti siano sempre dello stesso colore.
Sono state sviluppate due varianti, una nella quale i quadrati possano essere ruotati e l'altra dove le rotazioni non sono ammesse.

Esempio: Se nella riga i e colonna j viene posto un quadrato il cui lato inferiore è blu e il cui lato sinistro è rosso, allora il quadrato nella riga i e colonnaj deve avere lato destro rosso mentre il quadrato nella riga i+1 e colonna j deve avere lato superiore blu.

2 La Teoria

Ai fini di una migliore comprensione della relazione ho riportato, nelle due sezioni sottostanti, alcuni riferimenti teorici.

2.1 CSP

Un problema di soddisfacimento di vincoli presuppone un assegnamento iniziale, ovvero un insieme di variabili già vincolate. L'assegnamento iniziale può anche essere vuoto. La risoluzione del problema prosegue estendendo l'assegnamento iniziale, ovvero assegnando via via valori alle variabili ancora libere. La soluzione di un CSP è un assegnamento completo e coerente di valori alle variabili (ovvero un assegnamento che soddisfi tutti i vincoli e non lasci variabili libere), ottenuto estendendo l'assegnamento iniziale.
Sarà proprio l'assegnamento iniziale la chiave del nostro programma.

2.2 MiniZinc

Come linguaggio per descrivere il problema è stato utilizzato il linguaggio MiniZinc: un apposito linguaggio per descrivere problemi di soddisfacimento di vincoli (e problemi di ottimizzazione), che ruota attraverso la sostanziale classificazione delle entità in Parametri e Variabili.
Le specifiche scritte in MiniZinc (nei file .mzn) vengono convertite in linguaggio FlatZinc tramite un convertitore mzn2fzn.
Le specifiche FlatZinc possono poi essere risolte tramite molti solver.
Durante l'esecuzione è stata utilizzata la versione 6.3.0 del Solver Geocode.
N.B.: E' presente inoltre un particolare tipo di file (il file .dzn) che permette l'inizializzazione dei parametri del problema.

3 Documentazione

Per l'esercizio sono stati scritti due diversi file .mzn:

- Il file **senzaRotazione.mzn**
- Il file **conRotazione.mzn**

Il primo implementa la risoluzione con il vincolo aggiuntivo che i quadrati non possano essere ruotati ai fini della risoluzione del problema stesso.
Il secondo invece permette che i quadrati possano essere ruotati a piacimento
Entrambi i file .mzn sopra citati si avvalgono di vari file .dzn di appoggio presenti nella cartella dataset.
In ciascun file .dzn sono presenti due istanze di problema:

- la prima sempre risolvibile da entrambi i problemi (il soddisfacimento non richiede necessariamente la rotazione di alcuni quadrati)
- la seconda risolvibile solo ed esclusivamente dalla versione di programma che permette le rotazioni (rotazioni in questo caso necessarie ai fini della risoluzione)

4 Modellizzazione del problema

Come sopra citato, MiniZinc distingue la dichiarazione di un entità in due soli tipi base: parametri e variabili. Per quanto riguarda **il primo problema** (quello che non permettava rotazioni) si è deciso di implementare e modellizzare il problema come segue:

- **i parametri**: erano ovviamente costituiti da n (numero di righe della matfrice), m (numero di colonne della matrice), k (numero di colori a disposizione), ma soprattutto l'assegnamento iniziale dei colori e della rotazione (l'ordine dei colori e la rotazione in questo primo problema non potevano cambiare).
  La colorazione di tutti i quadrati è stata modellizzata tramite un array di nm righe e quattro colonne, ovvero in cui la riga i-esima rappresentava in ciascuna colonna il colore del corrispettivo lato del quadrato i-esimo.
  Per convenzione: colonna1: lato superiore, colonna2: lato destro, colonna3: lato inferiore, colonna4: lato sinistro
- **le variabili**: le variabili erano ovviamente le posizioni che ciascuno dei nm quadrati iniziali avrebbe dovuto assumere nella matrice risultato. Ho deciso dunque dopo svariati tentativi di implementare tale informazione con l'ausilio proprio di una matrice n per m dove in posizione i-j-esima sarebbe stato salvato un numero identificativo che avrebbe identificato il quadrato che avrebbe dovuto assumere tale posizione (i quadrati sono stati numerati con gli interi da 1 a nm)

Per quanto riguarda **il secondo problema** l'assenza del vincolo che i quadrati non potessero ruotare mi ha dato non pochi problemi inzialmente. Ma dopo svariati tentativi è stato semplicemnte necessario aggiungere una variabile al problema: la rotazione finale di ciascun quadrato. In una matrice n per m, proprio come l'altra variabile in gara, sarebbe stato salvato un numero che avrebbe identificato in modo univoco il livello di rotazione che avrebbe dovuto presentare il quadrato che avrebbe preso tale posizione nella matrice risultato finale del problema
Per convenzione: 0: no rotazione, 1: rotazione di pigreco mezzi antiorario, 2: rotazione di pigreco, 3: rotazoine di tre mezzi pigreco antiorario

5 Risultati Sperimentali Ottenuti

La prima fase di sperimentazione si è svolta tramite l'ausilio di un generatore pseudo-casuale di nuumeri fornito dal linguaggio. Tale generatore non permetteva la randomizzazione della generazione ad ogni esecuzione, è stato per questo motivo rimosso dal programma. Sono stati successivamente utilizzati i .dzn sopra citati:

- 1: 3x3.dzn
- 2: 4x4.dzn
- 3: 5x5.dzn

Per facilitare la stesura di tali dataset ciascuna matrice non era altro che sottomatrice diagonale-superiore-sinistra della successiva. Questo ha permesso di risparmiare tempo in fase di scrittura e di assicurarmi al 100 per 100 che i problemi che il programma identificava come risolvibili fossero effetivamente risolvibili.

6 Osservazioni e Conclusioni

Nelle fasi iniziali di sperimentazione (quando ancora il programma presentava il generatore casuale di numeri) sono rimasto davvero stupito da quanto il tempo di esecuzione crescesse in modo esponenziale in relazione al numero di righe e colonne della matrice. Inoltre, il problema che non permetteva rotazioni era raramente risolvibile per ovvi motivi.
L'idea inziale (concettualmete sbagliata) era quella di modellizzare la variabile come un unico array-3D di dimenzione n per m per 5. Il "piano terra" di tale matrice era ciò che poi è diventata la matrice sopra citata n per m utilizzata in entrambi i problemi. I "4 piani superiori" sarebbero serviti ad immagazzinare i colori dei lati del rispettivo quadrato presente in tale posizione. Tutto ciò era sbagliato su più fronti poichè introduceva i colori dei lati, ovvero parametri del problema per definizione stessa di parametri secondo MiniZinc, all'interno del dominio delle variabili.

Ho avuto non pochi problemi con l'implementazione del secondo problema. Inizialmente avevo trascurato l'opozione di aggiungere una variabile per codificare le rotazioni dei quadrati, e avevo in mente di aggiungere vincoli opzionali nel caso in cui fosse necessaria effettuare almeno una rotazione. Questo aveva portato alla stesura di uno statement di più di 100 linee di codice di soli if-then-else-condition.
In ultima osservazione posso affermare che il programma è sicuramente ottimizzabile con l'aggiunta di vincoli ridondanti. I vincoli utilizzati sono banali vincoli di dominio e banali di ovvia dimostrazione direttamente discendenti dalla natura del problema. Osservando però la risoluzione di tale problema effettuata da un semplice umano su carta, ho intuito vincoli di non banale scrittura ed implementazione, che porei implementare in futuro, e penso miglioreranno non poco le performance di tale esecuzione. Le performance non erano tra gli obbiettivi di questo elaborato.
In conclusione mi ritengo soddisfatto del lavoro svolto e della modelizzazione che è stata fatta tramite MiniZinc, di un problema in prima apparenza banale, che così banale non si è poi dimostrato.