Elaborato di Intelligenza Artificiale. Un esempio di modellazione di problemi CSP: Quadrati Colorati.

Lorenzo Pisaneschi

Agosto 2018

1 Introduzione

In generale, possiamo definire un problema di soddisfacimento dei vincoli (CSP, Costraints Satisfaction Problem), un problema che si pone come obiettivo quello di individuare un assegnamento coerente per delle variabili. Tale assegnamento può essere eseguito prendendo degli opportuni valori da domini assegnati, rispettando determinati vincoli, i quali dettano i valori che possono essere assegnati alle variabili. Per fare questo esistono molte tecniche, euristiche ed algoritmi che, lavorando sui dati in input, riescono a trovare una soluzione o a dire con certezza che questa non esiste per specifici dati in ingresso; tuttavia, lo scopo di questa relazione, è quello di concentrarsi su un particolare problema, studiarne la soluzione ed i risultati ottenuti.

1.1 Quadrati colorati

Cominciamo dalla descrizione del problema. In questo esercizio si costruisce un modello Minizinc (di cui sono stati dati dettagli in classe) 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 m in modo tale che i lati adiacenti siano sempre dello stesso colore. Si sviluppino due varianti, una nella quale i quadrati possono essere ruotati e l'altra dove le rotazioni non sono ammesse.

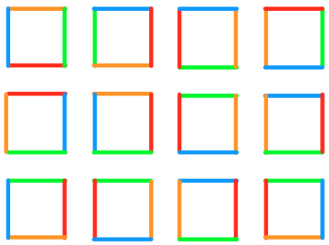

Figura 1: Esempio di problema risolto con successo.

2 Il modello Minizinc

Dopo aver descritto in maniera sommaria cosa sono i CSP e di quale problema andiamo occupandoci, passiamo alla descrizione della modellizzazione dello stesso e allo studio dei risultati ottenuti. Sono doverose delle premesse, utili per capire meglio gli esiti di queste procedure:

- **Cos'è Minizinc**: Minizinc è un linguaggio ad alto livello per la modellizzazione di CSP. Permette di costruire modelli con file **.mzn** e set di dati da sottomettere con file **.dzn**.
- **Quale macchina viene utilizzata**: gli esperimenti vengono eseguiti su un MacBook Pro che monta un processore Intel Core i5 a 2,6GHZ e una memoria RAM da 8GB DDR3 a 1600MHz. Il sistema operativo è "OS X El Captitan".

2.1 Strategie implementative

L'esercizio richiede lo sviluppo di due varianti: una dove non è possibile cercare una soluzione ruotando i quadrati, l'altra dove lo è.
In entrambi i casi si utilizzano *rows, columns e colours*, interi, con la loro evidente funzionalità di descrivere la configurazione della griglia in cui sono disposti i quadrati ed il numero di colori utilizzati; fondamentale è il ruolo di *setSquares*, un array bidimensionale di interi contenente l'input dei quadrati, la cui colorazione è indicata mediante numeri identificativi: ad ogni riga di setSquares corrisponde un quadrato ed ad ogni colonna è associata una faccia dei quadrati, nell'ordine top, right, bottom e left.
Nel caso della variante dove non è permessa la rotazione, è presente anche l'array di interi *Position*, che contiene l'ordine di presentazione dei quadrati, inizialmente dato in ingresso dal file .dzn, e modificato nel corso del processo di soddisfacimento dei vincoli.
Per concludere questa parte sulle variabili e i parametri, nel caso della variante dove è permessa la rotazione, è presente un altro array di interi, *Rotation*, che conserva un valore di rotazione definito nella fase di risoluzione (il valore indica un numero di passi in senso antiorario).
Consideriamo adesso i vincoli, che sono il cuore del problema e di questo elaborato. Il modello si basa su due vincoli principali:

- Il primo visita tutti i quadrati, occupandosi di far combaciare la faccia inferiore di un quadrato i-esimo (setSquares[i,3]) e la faccia superiore di un altro quadrato j-esimo (setSquares[j,1]), registrandoli rispettivamente in Position[i] e Position[i+columns].
- Il secondo vincolo visita i quadrati ed effettua un accoppiamento fra la faccia destra di un quadrato i-esimo (setSquares[i,2]) e la faccia sinistra di un altro quadrato (setSquares[j,4]), registrandoli rispettivamente in Position[i] e Position[i+1]. Nel caso della possibilità di ruotare i quadrati, il valore della faccia viene modificato dal valore di Rotation[h], dove h corrisponde al valore presente in Position[i].

2.2 Strategia di Ricerca

Concludiamo la parte descrittiva del programma Minizinc concentrandosi sull'ultimo passo: la ricerca di una soluzione per un determinato input nel rispetto dei vincli sopra descritti. Come detto nell'introduzione, esistono molti modi per risolvere un CSP; Minizinc in generale si preoccupa di risolvere il modello sottomessogli mediante l'istruzione *solve satisfy*, ma si è deciso di utilizzare altre srategie di scelta di variabili e domini che ci mette a disposizione; questo è possibile adottando la sintassi *search(vars,variable selection strategies, domain reduction strategies, explore)*.

Strategie di selezione delle variabili:

- firstfail: viene scelta la variabile con meno valori rimanenti nel dominio.
- inputorder: vengono scelte le variabili nell'ordine dell'input.
- smallest: viene scelta la variabile con il valore piu‘ piccolo nel dominio.
- largest: viene scelta la variabile con valore piu‘ grande nel dominio.

Strategie di riduzione del dominio:

- indomainmin: viene scelto il valore minore del dominio.
- indomainmax: viene scelto il valore maggiore del dominio.

Prima di passare al testing e alle conclusioni, facciamo un piccolo appunto sulle strategie di risoluzione dei CSP. La ricerca di una soluzione o della costatazione che questa non esiste per un dato problema CSP, può essere resa più veloce attraverso l'uso di inferenze, che possono lavorare sull'assegnazione di valori alle variabili e/o l'ordinamento dei domini dai quali quei valori vanno presi. Questa operazione ha lo scopo di rendere il problema consistente e quindi più facilmente risolvibile da un algoritmo. E' evidente che una certa inferenza possa essere più o meno efficiente per un determinato input piuttosto che per un altro. E' quindi interessante vedere quali sono migliori per un certo problema o per una sua determinata istanza.

3 Test e Conclusioni

Nel seguito vengono descritti i risultati dei test: per il codice dove è prevista la risoluzione del problema senza la possibilità di ruotare i quadrati sono validi solo due dei sei dataset utilizzati; nella versione del codice con possibilità di rotazione, sono validi tutti i sei dataset. Qui sotto, per comprendere meglio i risultati dei test, sono riportati i datasets, in seguito i risultati.

	Set1	Set2	Set3	Set4	Set5	Set6
Coulumns	4	4	5	4	4	4
Rows	2	4	4	3	8	2
Colors	4	4	5	5	5	4

Risultati test codice senza rotazione, INDOMAINMIN: norotation.mzn

INDOMAINMIN	Largest	Smallest	Input order	First fail
Set 3	87ms	11ms	82ms	91ms
Set 6	95ms	108ms	92ms	65ms

Risultati test codice senza rotazione, INDOMAINMAX: norotation.mzn

INDOMAINMAX	Largest	Smallest	Input order	First fail
Set 3	97ms	69ms	82ms	68ms
Set 6	75ms	69ms	92ms	81ms

Risultati test codice con rotazione, INDOMAINMIN: rotation.mzn

INDOMAINMIN	Largest	Smallest	Input order	First fail
Set 1	65ms	77ms	66ms	64ms
Set 2	143ms	163ms	144ms	146ms
Set 3	1077ms	81ms	83ms	104ms
Set 4	48s 126ms	1min 18s	47s 858ms	48s 750ms
Set 5	unknown	unknown	unknown	unknown
Set 6	66ms	93ms	62ms	80ms

Risultati test codice con rotazione, INDOMAINMAX: rotation.mzn

INDOMAINMAX	Largest	Smallest	Input order	First fail
Set 1	270ms	450ms	431ms	440ms
Set 2	217ms	231ms	219ms	221ms
Set 3	unknown	unknown	unknown	unknown
Set 4	1min 7s	1min 52s	1min 50s	1min 51s
Set 5	unknown	unknown	unknown	unknown
Set 6	107ms	104ms	104ms	109ms

Dove nelle tabelle appare la scritta "unknown" significa che il risultato del problema è sconosciuto in quanto il programma è stato interrotto (dopo 5 minuti di esecuzione il problema si considera unknown). Facendo un rapido esame dei risultati ottenuti, la strategia *Indomainmax* risulta essere più vantaggiosa rispetto a *Indomainmax*; per quanto riguarda le strategie di selezione delle variabili *first fail* risulta essere la migliore in termini di prestazioni, in generale.