Automated Reasoning A.A. 2021/2022

Relazione di progetto

Francesco Bombassei De Bona 144665

1 Outline

In questa relazione analizzerò il problema proposto per lo svolgimento del progetto di Automated Reasoning, proporrò una soluzione al problema codificata in Minizinc e una soluzione codificata in ASP. Inoltre, analizzerò i risultati ottenuti con entrambe le soluzioni e le differenze tra i due approcci.

2 Struttura del progetto

La struttura del progetto è la seguente:

- **report**: contiene il report del progetto;
- **asp**: contiene il codice ASP;
- **minizinc**: contiene il codice MiniZinc;
- **python**: contiene il codice Python per la generazione di grafici e per la generazione delle visualizzazioni delle soluzioni;
- **results**: contiene i risultati dei benchmark salvati in formato *json*;
- **benchmark.c**: contiene il codice per lanciare i benchmark;
- **setup.c**: contiene il codice per generare le istanze casuali del problema.

In particolare le cartelle *asp* e *minizinc* contengono le seguenti sottocartelle:

- **data**: contiene le istanze del problema;
- **results**: contiene le visualizzazioni create con Python dei risultati dei benchmark.

3 Problema

Sia data una griglia quadrata di lato n e sia dato un certo numero di carte di cui si consideri il segno ♣, ♡, ♠, ♢. Il numero di carte per ogni seme può essere diverso da quello standard (tredici per seme). Alcune celle sono già occupate, alcune celle sono bloccate/vietate (indicate con il simbolo □), ogni cella libera può contenere al più una carta, e non vi possono essere più di tre carte adiacenti dello stesso seme. L'obiettivo è

disporre le carte disponibili in modo da massimizzare il punteggio ottenuto dalle seguenti regole:

- due carte adiacenti dello stesso seme (orizzontali e/o verticali): 1 punto;
- tre carte adiacenti dello stesso seme (orizzontali e/o verticali): 3 punti;
- se si conta la terna non si contano le due coppie;
- quattro carte adiacenti disposte in verticale di quattro semi diversi: 4 punti;
- quattro carte adiacenti disposte in orizzontale di quattro semi diversi: 6 punti.

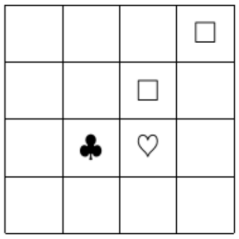

Figure 1: Esempio di griglia iniziale

4 Modello codificato in MiniZinc

Nel modello MiniZinc i semi e le celle vietate vengono codificati utilizzando degli interi: al seme di cuori corrisponde il numero 1; al seme di fiori corrisponde il numero 2; al seme di picche corrisponde il numero 3; al seme di quadri corrisponde il numero 4; le celle vietate vengono occupate con il numero 5.

Il modello MiniZinc è costruito in modo da ricevere in input:

- la dimensione della griglia;
- il numero di carte disponibile per ogni seme;
- il numero di celle bloccate;
- una matrice 2D di variabili che rappresenta la griglia iniziale, popolata dalle carte iniziali e dai blocchi.

verb:input

```
3     int: n;
4     int: hearts;
5     int: clubs;
```

```
6       int: spades;
7       int: diamonds;
8       int: blocked;
9       array[1..n, 1..n] of var 1..5: m;
```

Una variabile viene utilizzata per salvare il valore del punteggio ad ogni passaggio del processo di ottimizzazione:

verb:score

```
10      var int: p;
```

Per imporre il vincolo di cardinalità su ogni seme (numero di carte disponibile per ogni seme, dato in input) viene utilizzato il vincolo *global_constraint_low_up* con minimo 0 e massimo il valore della variabile corrispondente al seme d'interesse:

verb:cardinality

```
12      constraint global_cardinality_low_up([m[i,j] | i,j in 1..n], 1..1, [0
        ↪  | k in 1..1], [hearts | k in 1..1]);
13      constraint global_cardinality_low_up([m[i,j] | i,j in 1..n], 2..2, [0
        ↪  | k in 2..2], [clubs | k in 2..2]);
14      constraint global_cardinality_low_up([m[i,j] | i,j in 1..n], 3..3, [0
        ↪  | k in 3..3], [spades | k in 3..3]);
15      constraint global_cardinality_low_up([m[i,j] | i,j in 1..n], 4..4, [0
        ↪  | k in 4..4], [diamonds | k in 4..4]);
```

Un vincolo analogo viene utilizzato per impedire a MiniZinc di riempire le celle con il valore 5 (cella vietata):

verb:blocked_cardinality

```
12      constraint global_cardinality_low_up([m[i,j] | i,j in 1..n], 5..5, [0
        ↪  | k in 5..5], [blocked | k in 5..5]);
```

Per imporre il vincolo relativo al numero massimo di carte adiacenti dello stesso seme (tre) viene utilizzato un vincolo *forall* unito al costrutto if-then. In particolare, si impone che per ogni riga (e colonna), data una cella e due celle consecutive che contengono lo stesso valore della prima, la quarta cella consecutiva alla prima deve contenere un valore diverso (questo non vale per le celle vuote e quelle bloccate):

verb:four_different

```
17      constraint forall(i in 1..n, j in 1..n-3)(
18        if m[i,j] == m[i,j+1] /\ m[i,j] == m[i,j+2] /\ m[i,j] != 5
19        then m[i,j] != m[i,j+3]
20        endif
21       );
22
23      constraint forall(j in 1..n, i in 1..n-3)(
24        if m[i,j] == m[i+1,j] /\ m[i,j] == m[i+2,j] /\ m[i,j] != 5
25        then m[i,j] != m[i+3,j]
26        endif
27      );
```

Per il calcolo del punteggio è stato creato un vincolo sulla variabile scelta per racchiudere il punteggio: per ogni possibile combinazione di semi viene verificata un'espressione booleana associata alla combinazione stessa e per ogni occorrenza viene sommato il valore associato (ad esempio sei per il poker orizzontale). Di seguito si fornisce un esempio di calcolo del punteggio per la quaterna orizzontale:

verb:horiz_four

```
30      sum(i in 1..n)(
31          sum(j in 1..n-3)(
32            if
33              m[i,j] != 5 /\
34              m[i,j+1] != 5 /\
35              m[i,j+2] != 5 /\
36              m[i,j+3] != 5 /\
37              m[i,j] != m[i,j+1] /\
38              m[i,j] != m[i,j+2] /\
39              m[i,j] != m[i,j+3] /\
40              m[i,j+1] != m[i,j+2] /\
41              m[i,j+1] != m[i,j+3] /\
42              m[i,j+2] != m[i,j+3]
43            then 6
44            else 0
45            endif
46          )
47      )
```

Si rimanda al codice sorgente per avere una visione del modello completo e del calcolo del punteggio.

5 Modello codificato in ASP

Nel modello ASP si è adottata la stessa codifica per i semi e le celle bloccate presentata per il modello codificato in MiniZinc.

Per il modello ASP sono stati creati i predicati:

- *lato* di arità 1 con range [1,n];
- *simbolo* di arità 1 con range [1,5];
- *griglia* di arità 3 che rappresenta l'assegnazione alla cella di coordinate (x,y) del simbolo s.

Per definire le assegnazioni del predicato *griglia* è stato definito il seguente vincolo di cardinalità:

verb:asp_allocation

```
4    {griglia(X,Y,S) : simbolo(S), S != 5} 1:- lato(X), lato(Y), not
     ↪  griglia(X, Y, 5).
```

Questo garantisce che a ogni step di ottimizzazione venga fatta un'assegnazione (con S diverso da 5) per ogni cella che non è bloccata od occupata in partenza (per queste celle viene fornito un apposita assegnazione del predicato *griglia* prima della valutazione del modello).

Per imporre il vincolo di cardinalità sul numero di carte per ogni seme vengono utilizzati i seguenti vincoli:

verb:asp_cardinality

```
4    :- hearts < #count{X,Y : lato(X), lato(Y), griglia(X,Y,1)}.
5    :- clubs < #count{X,Y : lato(X), lato(Y), griglia(X,Y,2)}.
6    :- spades < #count{X,Y : lato(X), lato(Y), griglia(X,Y,3)}.
7    :- diamonds < #count{X,Y : lato(X), lato(Y), griglia(X,Y,4)}.
```

Mentre per imporre il vincolo sul numero massimo di carte adiacenti dello stesso seme (tre) vengono utilizzati i seguenti vincoli, rispettivamente per le colonne e per le righe:

verb:asp_four_different

```
4    :- simbolo(S), lato(X), lato(Y1), lato(Y2), lato(Y3), lato(Y4),
5       Y1+1 == Y2, Y1+2 == Y3, Y1+3 == Y4,
6       S != 5,
7       griglia(X,Y1,S), griglia(X,Y2,S), griglia(X,Y3,S),
        ↪  griglia(X,Y4,S).
```

```
8
9       :- simbolo(S), lato(Y), lato(X1), lato(X2), lato(X3), lato(X4),
10         X1+1 == X2, X1+2 == X3, X1+3 == X4,
11         S != 5,
12         griglia(X1,Y,S), griglia(X2,Y,S), griglia(X3,Y,S),
           ↪ griglia(X4,Y,S).
```

Per il calcolo del punteggio è stato creato il predicato *score* di arità 1 con variabile P che rappresenta il punteggio totale. Si rimanda al codice sorgente per avere una visione del modello completo e del calcolo del punteggio.

6 Test

Per supportare la fase di test sono stati creati due programmi in C per la generazione d'istanze casuali del problema nei due formati d'input e per raccogliere i risultati dell'esecuzione dei modelli sulle due istanze.

È stata apportata anche una modifica all'output del modello MiniZinc in modo che coincidesse con quello del modello ASP e il programma C potesse interpretare i risultati in modo uniforme.

verb:output_mzn

```
4       output["griglia(" ++ show(i) ++ "," ++ show(j) ++ "," ++ show(m[i,j])
        ↪ ++ ") " | i,j in 1..n] ++
5       ["\nOptimization: " ++ show(p)]
```

6.1 Generazione delle istanze

Per la generazione delle istanze casuali è stato creato il programma *setup.c* che prende in input una sequenza di interi corrispondenti alle dimensioni delle griglie da generare e per ogni dimensione genera 20 istanze casuali del problema andando a generare casualmente le celle bloccate e le carte iniziali. Per ogni seme viene generato, in modo indipendente, un intero casuale che al più può essere pari al numero di celle totali della griglia. Le celle bloccate o occupate possono essere al più un terzo del totale delle celle della griglia.

Il programma genera un file per ogni istanza sia nel formato MiniZinc che nel formato ASP e li salva nelle sottocartelle dedicate per ogni formato.

6.2 Esecuzione dei test

Per l'esecuzione dei test è stato creato il programma *benchmark.c* che prende in input una sequenza di interi corrispondenti alle dimensioni delle griglie da testare e per ogni

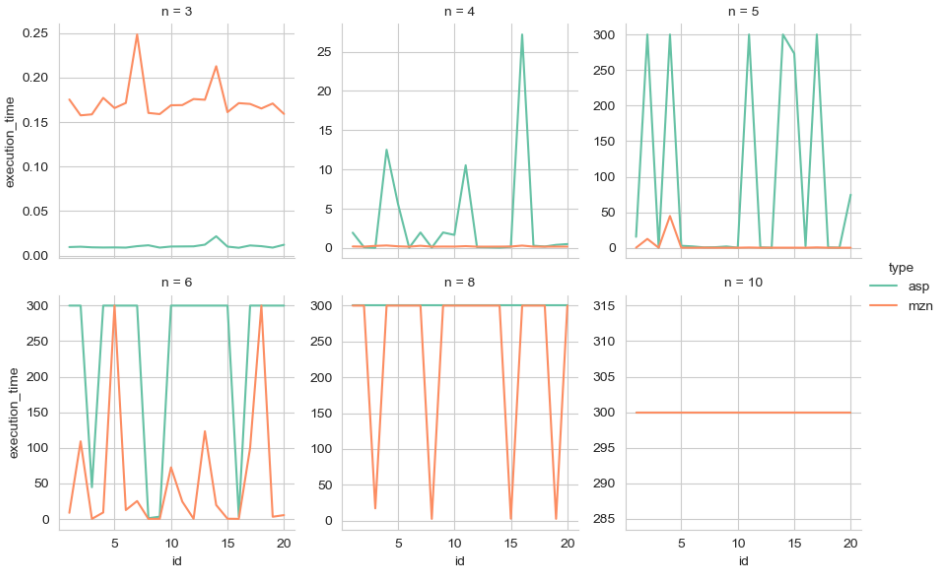

Figure 2: Confronto tra ASP e MiniZinc

dimensione esegue i modelli MiniZinc e ASP su tutte le istanze generate per quella dimensione.

L'output generato dai due modelli viene salvato in un file .json appositamente creato per ogni esecuzione dello script.

6.3 Parametri di esecuzione

Per l'esecuzione dei test con il modello ASP è stato utilizzato il solver *clingo* con i seguenti parametri: *–time-limit 300 –configuration=jumpy –restart-on-model -t 8,split*.

Per l'esecuzione dei test con il modello MiniZinc è stato utilizzato il solver *Chuffed* con timeout a 300 secondi, esecuzione parallela su 8 core e mostrando le soluzioni intermedie. Tutti i parametri sono disponibili nel file *minizinc/config.mpc*.

I parametri sono stati scelti dopo una fase di test preliminare in cui sono state analizzate le performance con diversi parametri in particolare variando il parametro *configuration* in ASP e il solver utilizzato da MiniZinc. Nella relazione sono presentati i risultati delle configurazioni migliori per questioni di spazio.

7 Risultati

L'esecuzione dei test è stata effettuata su griglie quadrate di dimensione: 3, 4, 5, 6, 8, 10. La dimensione massima della griglia è stata posta a 10 in quanto sia la codifica in MiniZinc che quella in ASP hanno mostrato il raggiungimento della soglia dei 300 secondi di esecuzione in tutte e 20 le griglie generate.

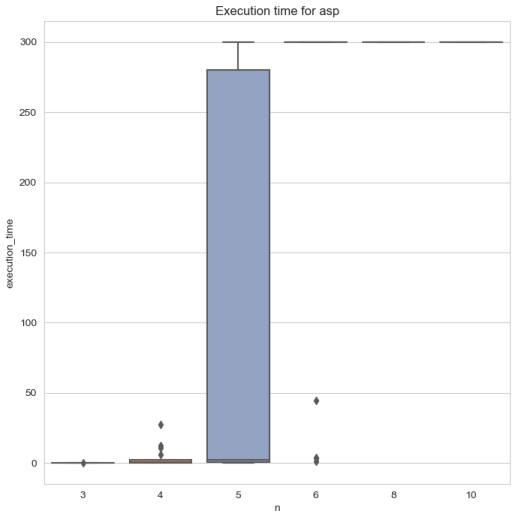

Figure 3: Boxplot dei tempi di esecuzione in ASP

Come visibile dalla figura 2, la codifica in MiniZinc si è rilevata essere più performante rispetto a quella in ASP per le griglie di dimensione superiore a 3, con alcune eccezioni per le griglie di dimensione 4.

La codifica in ASP ha iniziato a saturare il tempo di esecuzione a partire dalla griglia di dimensione 5, mentre la codifica in MiniZinc ha iniziato a saturare il tempo di esecuzione a partire dalla griglia di dimensione 6.

I boxplot delle figure 3 e 4 mostrano come la codifica in ASP sia meno stabile rispetto a quella in MiniZinc, in quanto i tempi di esecuzione sono molto meno concentrati attorno alla mediana (vedi griglia di dimensione 5 in ASP e griglia di dimensione 6 in MiniZinc).

È stato generato anche un grafico di correlazione per indagare la dipendenza tra il tempo di esecuzione e la dimensione della griglia, il numero di caselle occupate e il numero di caselle libere.

Come riportato in entrambi i grafici di correlazione (figure 5 e 6), il tempo di esecuzione è fortemente correlato con la dimensione della griglia, mentre c'è un moderato livello di correlazione con il numero di caselle occupate e il numero di caselle libere. Questo può essere dovuto al fatto che aumentando la dimensione della griglia aumenta notevolmente il livello di difficoltà del problema mentre il numero di caselle occupate rimane una frazione (massimo 1/3) del numero di caselle totali.

A scopo di completezza, è stata creata una breve animazione che mostra una comparativa della risoluzione della griglia numero 15 di dimensione 5 nelle due codifiche proposte. L'animazione è disponibile su YouTube al seguente indirizzo.

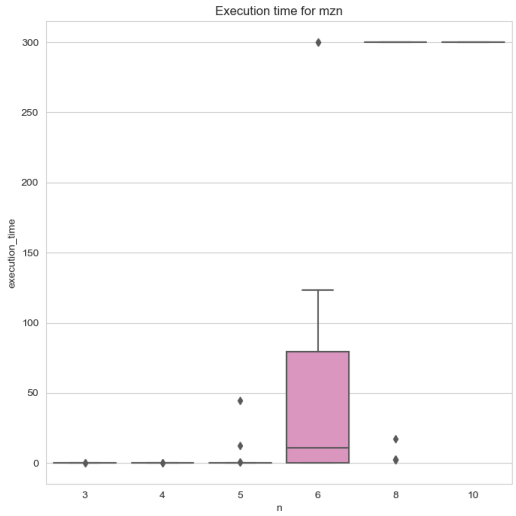

Figure 4: Boxplot dei tempi di esecuzione in MiniZinc

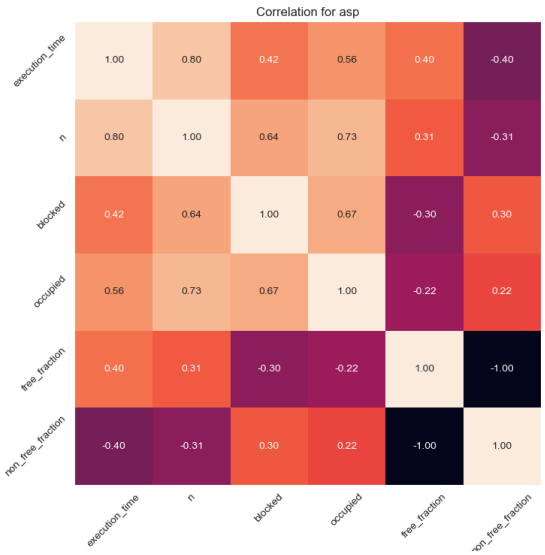

Figure 5: Grafico di correlazione per ASP

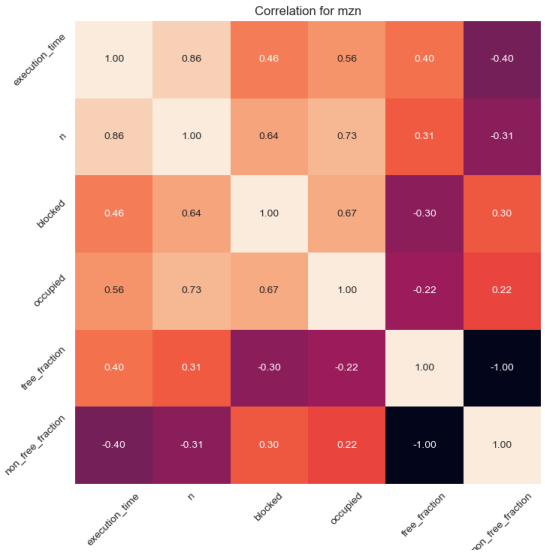

Figure 6: Grafico di correlazione per MiniZinc