Progetto del corso
Automated Reasoning 2017/18
Esercizio 9

Gabriele Venturato (125512)

23 luglio 2018

Sommario

1 Materiale Prodotto

Nel materiale consegnato è possibile trovare uno script `create_instances.py` per generare le istanze di input per i due modelli in ASP e MiniZinc — le istanze sono le stesse per entrambi. Lo script è stato scritto in Python (versione 3.6) e genera le istanze nella cartella `instances/`. Ogni file di input generato ha il nome $in\_n\_k\_h\_i.ext$, dove n, k, e h sono i parametri di input, e i è un indice per distinguere le diverse istanze con gli stessi parametri di input. Le tessere del domino di ogni istanza sono generate casualmente quindi ogni istanza è (idealmente) diversa dalle altre.

Sono presenti poi le cartelle `asp/` e `minizinc/`. All'interno di ognuna si trovano:

- una cartella `input/` che contiene una copia delle istanze del rispettivo modello generate (sono in una cartella diversa da quella di generazione per evitare sovrascritture involontarie).
- una cartella `output/` che contiene gli output dei test effettuati. Il nome dei file è analogo a quello degli input. Epresente una sottocartella *sample* che contiene una porzione di output (vedi punto successivo).
- uno script bash `benchmark.sh` che può essere lanciato senza specificare niente, oppure con l'opzione `-s` che testa il modello sul 10 delle istanze disponibili scelte in maniera pseudo-casuale e distribuita tra tutte le possibilità.

- e infine i modelli del problema.

Inoltre, nella cartella di minizinc è possibile trovare un modello *max-sequence.mzn* che è un'estrapolazione dal modello completo, che permette di trovare la sequenza più lunga possibile di tessere, non vincolate dalla scacchiera.

2 Il problema

Oltre a quanto descritto nel testo del problema, ho assunto che una tessera può essere ruotata in verticale solo di 90 gradi in senso orario. In questo modo il valore destro della tessera può essere o a destra o in alto.

2.1 Difficoltà

Ho dovuto modificare i parametri di input dopo aver testato diverse soluzioni. Ho lasciato le dimensioni di n invariate, ho invece definito $k = n/2$ invece che $n/5$, e ho diminuito leggermente le dimensioni della scacchiera testando $h = 6, 7, 8, 9, 10$.

Tale scelta non è casuale ma è frutto di un'analisi dei risultati prodotti dai modelli. Il modello ASP non ha presentato gravi problemi. Quello con cui ho avuto più difficoltà è stato il modello MiniZinc che non è stato performante fin da subito e mi ha fornito le indicazioni su come diminuire i parametri. Prima di tutto la dimensione della scacchiera è stata diminuita per abbassare il numero di possibilità in cui si possono disporre le tessere ovviamente. Per quanto riguarda k invece ho voluto abbassarlo per diminuire il numero di diverse sequenze di tessere possibili. Questo mi ha permesso di "allontanarmi" dal valore $n!$ a cui tende se le tessere sono spesso ripetute, cioè se $k << n$.

3 Modelli

3.1 Modello ASP

```
1 %% Exercise 9 - (Custom) Domino
2 %% Gabriele Venturato (125512)
3 %% Automated Reasoning 2017/2018
4 %% tested with clingo 5.3.0
5
6 value(1..k).
7 tilename(1..n).
8 coord(1..h).
9
```

```
% Input: k, n, h, tile(n,l,r)

% ------------------------------------------------------------------------------ %
%% Placed
% a tile can be "placed on" the board at position (x,y)
% a tile is placed, if it is placed on a position
placed(T) :- placed_on(T,_,_).

% Left and Right values
% the value in (x,y) is the left (or top) one of the tile T
leftval(T,X,Y) :- placed_on(T,X,Y), placed_on(T,X+1,Y), coord(X+1).
leftval(T,X,Y) :- placed_on(T,X,Y), placed_on(T,X,Y+1), coord(Y+1).

% the value in (x,y) is the right (or bottom) one of the tile T
rightval(T,X,Y) :- placed_on(T,X,Y), not leftval(T,X,Y).

%% Compatible
% two tiles are compatible if and only if they are not the same tile
% and have the same value in position left and right
compatible(T1,T2) :- tile(T1,_,R), tile(T2,L,_), T1 != T2, R == L.
:- compatible(T,T).
:- compatible(T1,T2), tile(T1,_,R), tile(T2,L,_), R != L.


% ---------------------------- CONSTRAINTS ---------------------------------- %
%% Placed
% and can occupy exactly zero or exactly two positions
0 { placed_on(T,X,Y) : coord(X), coord(Y) } 2 :- tilename(T).
{ placed_on(T,X,Y) : coord(X), coord(Y) } != 1 :- tilename(T).

%% a tile can be only placed in two consecutive (no diagonal) cells of the board
:- placed_on(T,X1,Y1), placed_on(T,X2,Y2), X1 != X2, Y1 != Y2.
:- placed_on(T,X,Y1), placed_on(T,X,Y2), Y1 < Y2, Y2 != Y1 + 1.
:- placed_on(T,X1,Y), placed_on(T,X2,Y), X1 < X2, X2 != X1 + 1.

%% tiles can't overlap
0 { placed_on(T,X,Y) : tilename(T) } 1 :- coord(X), coord(Y).

%% Next
% define when a tile can be next to another
0 { next(T1,T2) : tilename(T2) } 1 :- tilename(T1), compatible(T1,T2).
0 { next(T1,T2) : tilename(T1) } 1 :- tilename(T2), compatible(T1,T2).
:- next(T1,T2), not compatible(T1,T2).
:- next(T,T).

% a tile can be next only to one another tile
:- next(T,T1), next(T,T2), T1 != T2.
:- next(T1,T), next(T2,T), T1 != T2.

%% Next and Placed
% a tile can't be placed and not next to anything
:- placed(T), placed(T1), T != T1, not next(T,_), not next(_,T).

% a tile can't be next to another if it is not placed
:- next(T,_), not placed(T).
:- next(_,T), not placed(T).

%% Where to place (if two tiles are next)
% if two tiles are on the same column the row distance must be 1
:- placed_on(T1,X1,Y1), placed_on(T2,X2,Y2),
    T1 != T2, next(T1,T2),
    rightval(T1,X1,Y1), leftval(T2,X2,Y2),
    X1 == X2,
    |Y1 - Y2| != 1.
```

```
74
75 % if two tiles are on the same row the column distance must be 1
76 :- placed_on(T1,X1,Y1), placed_on(T2,X2,Y2),
77     T1 != T2, next(T1,T2),
78     rightval(T1,X1,Y1), leftval(T2,X2,Y2),
79     Y1 == Y2,
80     |X1 - X2| != 1.
81
82 % they can't be on a different row and different column
83 :- placed_on(T1,X1,Y1), placed_on(T2,X2,Y2),
84     T1 != T2, next(T1,T2),
85     rightval(T1,X1,Y1), leftval(T2,X2,Y2),
86     |Y1 - Y2| != 0,
87     |X1 - X2| != 0.
88
89 % both different from 1 is not necessary because overlap is not allowed
90
91 %% Symmetry breaking
92 % if a sequence is long l, then there exist a sequence of length l which starts
93 % from top left corners
94 placed_on(T,1,1) :- firsttile(T).
95 placed_on(T,2,1) :- firsttile(T).
96
97 % firsttile is T if there are at least two tiles and T is the first
98 % or if T is the only tile placed.
99 firsttile(T) :- next(T,_), not next(_,T), placed(T1), T1 != T.
100 firsttile(T) :- not next(_,_), placed(T).
101
102 %% Sequence definition
103 sequence(S+1) :- S = #count{ T: next(T,_) }.
104
105 #maximize { S : sequence(S) }.
```

3.2 Modello MiniZinc

```
1  % Exercise 9 - (Custom) Domino
2  % Gabriele Venturato (125512)
3  % Automated Reasoning 2017/2018
4  % tested with mzn-gecode v2.0.2 (i.e. fzn-gecode 5.1.0)
5
6  % ------------------------------- DATA ------------------------------------- %
7  % Input parameters
8  int: h; % board dimension
9  int: n; % number of tiles
10 int: k; % max value on tiles
11
12 % tiles[t,1] is the left value of tile t (2 is the right one)
13 array[1..n, 1..2] of 1..k: tiles;
14
15 % board[x,y] = p means that the tile t is in position p on the sequence,
16 % and that it is placed on position (x,y) on board
17 % (you can forget about this detail and think that a tile is placed on board,
18 % and not it's corresponding position in the sequence)
19 array[1..h, 1..h] of var 0..n: board;
20
21 var 1..n: s; % length of the sequence
22 array[1..n] of var 0..n: sequence;
23
24
25 % ---------------------------- CONSTRAINTS ---------------------------------- %
26 % SEQUENCE
```

```
% two tiles to be in sequence must respect values in them
constraint forall (i in 1..s-1) (
  tiles[ sequence[i] , 2 ] = tiles[ sequence[i+1] , 1 ]
);

% after last tile, can't have other tiles in the sequence
constraint forall (i in s+1..n) (
  sequence[i] = 0
);

% at least one tile in the sequence (avoid empty sequence)
constraint sequence[1] > 0;

% can use each tile at most once in the sequence
constraint forall(t in 1..n) (
  sum( [ bool2int( sequence[i] == t ) | i in 1..n ] ) <= 1
);

% BOARD
% a tile placed on the board must occupy at least two consequent cells
constraint forall( i,j in 1..h, p in 1..s ) (
  board[i,j] = p -> exists( a,b in {-1,0,1} where
      i+a >= 1 /\ j+b >= 1 /\
      i+a <= h /\ j+b <= h /\
      abs(a) + abs(b) = 1
    ) ( board[i+a,j+b] = p )
);

% for each row a tile can occupy at most two consequent cells
constraint forall( p in 1..s ) (
  sum( [ bool2int(board[i,j] == p) | i,j in 1..h ] ) <= 2
);

% defining where a tile can be placed on board
constraint forall( i in 1..s-1 ) (
  exists( t1x,t1y,t2x,t2y in 1..h ) (
    board[t1x,t1y] = i /\ rightval(board[t1x,t1y],t1x,t1y) /\
    board[t2x,t2y] = i+1 /\ leftval(board[t2x,t2y],t2x,t2y) /\
  abs(t1y - t2y) <= 1 /\
  abs(t1x - t2x) <= 1 /\
  abs(t1y - t2y) + abs(t1x - t2x) = 1)
);

% a tile can be on board iff it is in the sequence
constraint forall( p in 1..s ) (
  exists(x,y in 1..h) (board[x,y] = p)
);

% SYMMERTRY BREAKING
% the longest sequence will start from top left corner with the tile
% placed horizontally
constraint board[1,1] = 1 /\ board[1,2] = 1;

% try only sequences with length compatible with the board dimension
constraint s <= (h*h div 2);

% sequence must be > h (imposed by assignment)
% this could lead to unsatifiable situation if there are not enough tiles
% compatilble to build a sequence length at least h
constraint s > h;

% don't consider permutation of tiles with the same values
% this can be activated with high "n" and low "k", otherwise it is not worth it
%constraint forall( i in 1..s-1, j in i+1..s ) (
```

```
%  ( tiles[ sequence[i], 1 ] ==  tiles[ sequence[j], 1 ] /\
%  tiles[ sequence[i], 2 ] ==  tiles[ sequence[j], 2 ] )
%  -> sequence[i] < sequence[j]
%);

constraint forall( i,j in 1..h ) ( board[i,j] <= s );

% ------------------------------ OUTPUT ------------------------------------ %
solve :: seq_search([
    int_search([s], input_order, indomain_min, complete),
    int_search(sequence, first_fail, indomain_min, complete),
    int_search([ board[i,j] |i,j in 1..h], first_fail, indomain_min, complete)])
  maximize s;

output [
  "Sequence length: " ++ show(s) ++
  "\nSequence: " ++ show(sequence) ]
++ ["\nBoard:\n"] ++
[
  show(board[i,j]) ++ "\t" ++
  if (j mod h == 0) then "\n" else "" endif
  | i,j in 1..h
];


% ----------------------------- PREDICATES ---------------------------------- %
% true if position (x,y) is the right (resp. bottom) value of tile t
predicate rightval(var int: p, 1..h: x, 1..h: y) =
  exists(i,j in 1..h) (
    board[i,j] == p /\ board[x,y] == p /\ ( x > i \/ y > j )
  );

% true if position (x,y) is the left (resp. top) value of tile t
predicate leftval(var int: p, 1..h: x, 1..h: y) =
  exists(i,j in 1..h) (
    board[i,j] == p /\ board[x,y] == p /\ ( x < i \/ y < j )
  );
```

4 Risultati

Riporto in tabella i risultati dei benchmark. Non ho prodotto grafici perchè i risultati non si prestano ad una visualizzazione di questo tipo.

Come anticipato, il modello ASP risulta più performante...