5 settembre 2019 - laboratorio

Constraint Programming

Prof. Marco Gavanelli

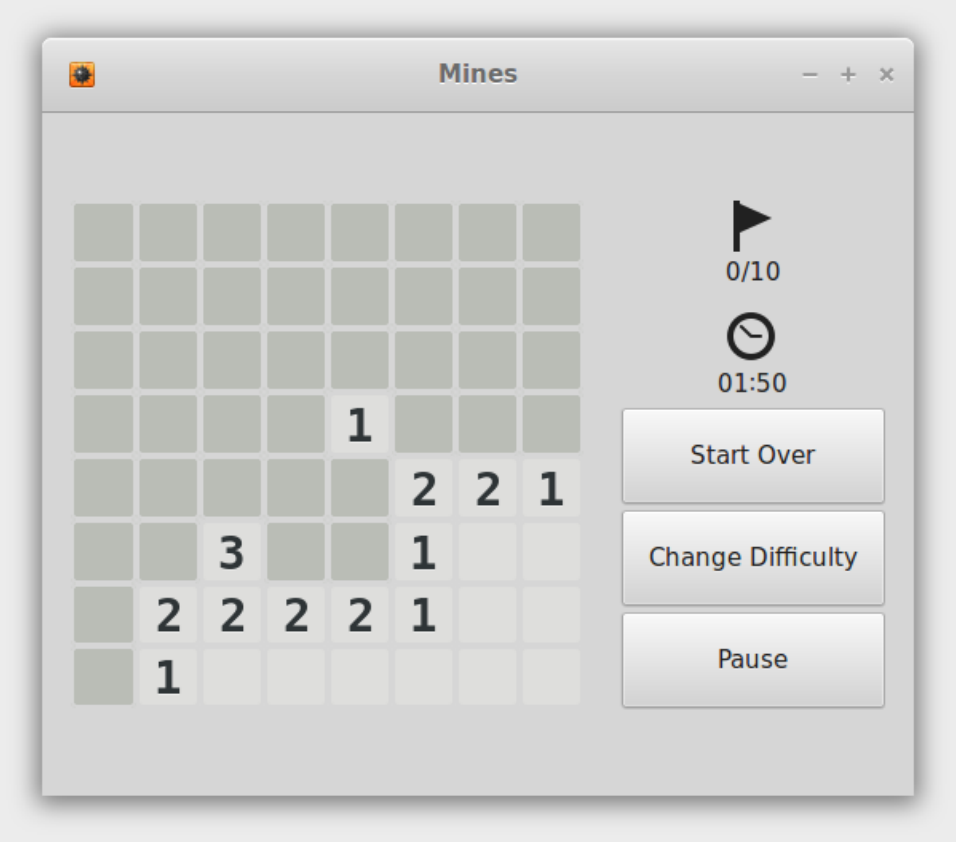
5 settembre 2019

Descrizione problema

Nel gioco *campo minato* viene fornita una griglia con N_R righe e N_C colonne. Cliccando su una cella della griglia, questa viene scoperta. Molte celle contengono mine: se la cella su cui si è cliccato contiene una mina, il gioco termina e si è persa la partita.

Altrimenti, nella cella cliccata viene mostrato un numero fra 0 e 8: questo numero indica il numero di celle adiacenti (in verticale, orizzontale o diagonale) che contengono mine.

Nel seguente schema



le celle scure rappresentano celle coperte (in cui non è stato cliccato), mentre quelle chiare sono scoperte. Le celle chiare senza numero rappresentano celle con il valore 0.

Si desidera scrivere un programma che aiuti a giocare a campo minato, fornendo indicazioni su dove su può cliccare avendo la certezza di non trovare una mina.

1 CLP

1.1 (punti 15)

Il file `schema1.pl` contiene uno schema (in questo caso, corrispondente a quello mostrato nella figura precedente). Esso contiene:

- un fatto *schema*(N_R,N_C,*Matrice*)dove
  - N_R e N_C rappresentano il numero di righe e colonne della griglia
  - *Matrice* rappresenta la griglia, nel formato lista di liste (come nella libreria `matrix_util`). Quando in una cella della matrice è presente una variabile, significa che quella cella non è stata scoperta, mentre un valore intero rappresenta il valore mostrato in quella cella.

Si scriva un predicato che, dati due valori *R* e *C*, verifica se nella cella di coordinate *R* e *C* è possibile che ci sia una mina.

Suggerimento: una possibilità è quella di usare la funzione obiettivo ...

Oppure si potrebbe dare fallimento se nella cella (*R*,*C*) è possibile che ci sia una mina e dare successo se non ci può essere ... (o viceversa).

Semplificazione. Per semplificare la soluzione, **si può usare** (non è obbligatorio usarlo) il predicato *estrai_confinanti*(++*Nriga*,++*Ncol*,+*M*,–*LConfinanti*), che, dati un numero di riga *Nriga*, uno di colonna *Ncol* e una matrice *M*, fornisce nella lista *LConfinanti* le celle della matrice che confinano con la cella di indici [*Nriga*,*Ncol*]. Ad esempio, *estrai_confinanti*(2,1,[[*A*,2,2,*B*],[*C*,*D*,*E*,1], [0,*F*,*G*,0]],*LConfinanti*) fornisce l'unificazione *LConfinanti* = [*A*,2,*D*,0,*F*].

Il predicato *estrai_confinanti* è implementato nel file `estrai_confinanti.eco` , che è un file compilato. Per usare il file `estrai_confinanti.eco` ,

- in `tkeclipse`, dal menu `File` → `Compile` → selezionare il tipo di file `ECLiPSe precompiled files (*.eco)`
- nella versione a linea di comando di ECL$_i$PS$_e$, si può caricare il file con `['estrai_confinanti.eco'].` eventualmente inserendo il path completo del file.

1.2 (1 punto)

Si supponga ora che nell'invocazione del programma, i valori di *R* e *C* non vengano forniti al predicato, ma vengano passate due variabili. Il predicato dovrà

fornire i valori di *R* e *C* che corrispondono ad una cella coperta su cui è possibile cliccare, senza rischio che ci sia una bomba.

1.3 (1 punto)

Si implementi il predicato *estrai_confinanti*, oppure si risolva il problema senza usarlo.

2 ASP (8 punti)

Si risolva il problema in Answer Set Programming.

Per semplificare la lettura dei dati, lo schema viene fornito nel file `schema1.asp` ; esso contiene:

- un predicato *matr*(N_R,N_C) che fornisce il numero di righe N_R e il numero di colonne N_C dello schema
- un predicato *schema*(*R*,*C*,*V*) che fornisce, per ogni cella scoperta, la riga *R*, la colonna *C* in cui si trova la cella ed il valore *V* in essa contenuto
- un predicato *click*(*R*,*C*) che contiene la riga *R* e la colonna *C* in cui l'utente chiede di cliccare

Il programma dovrà

- fallire se nella cella identificata dal predicato *click* può esserci una mina
- avere successo se nella cella identificata dal predicato *click* si può cliccare liberamente (ovvero, se è impossibile che in quella cella ci sia una mina)

Soluzione

Definiamo un CSP in cui si ha una variabile per ogni cella, ciascuna con dominio 0..1. Il valore 0 significa che in quella cella non c'è la mina, il valore 1 vuol dire che c'è. Le variabili vengono rappresentate nella matrice `Mine`.

L'unico vincolo del problema è che se la cella (*i,j*) della matrice di ingresso (cioè quella data dal predicato `schema`) ha un valore *v*, allora delle celle confinanti a *Mine[i,j]*, esattamente *v* devono contenere la mina.

Facendo la search (ad esempio, con il `labeling`), può accadere che in alcune soluzioni la cella con le coordinate (*R,C*) data in ingresso (chiamiamola *Mine[R,C]*) abbia la mina ed in altre non ce l'abbia. In tal caso, non è una cella su cui si può cliccare liberamente. Per verificare questa condizione si può fare in diversi modi:

1. si possono trovare tutte le soluzioni (con `findall`) e verificare se in una di queste nella cella *Mine[R,C]* è presente una mina (ha valore 1)
2. oppure si può usare il valore della cella *Mine[R,C]* come funzione obiettivo: massimizzando il valore di tale cella, se &eagrave; possibile ottenere un 1 vuol dire che in quella cella è possibile mettere una mina
3. oppure si può imporre che *Mine[R,C]=1* (cioè imponiamo che nella cella ci sia la mina) e facciamo il labeling di tutte le altre celle. Se si ottiene una soluzione, vuol dire che non è possibile cliccare su quella cella liberamente, in quanto potrebbe esserci una mina. Se non si ottiene soluzione (il predicato fallisce), allora vuol dire che su quella cella si può cliccare liberamente, in quanto è impossibile che vi sia una mina.

Nella soluzione sottostante si sceglie la terza soluzione. Qui si è preferito dare successo se non può esserci la mina, per cui si usa il `not`.

```
:- lib(fd).
:- lib(fd_global).
:- lib(matrix_util).
:- lib(listut).
:- ['estrai_confinanti.eco'].
:- [schema1].

mine(ClickR,ClickC):-
 schema(Nrighe,Ncol,Vicini),
 matrix(Nrighe,Ncol,Mine),
 flatten(Mine,MineFlat),
 MineFlat :: 0..1,
 matrix(Nrighe,Ncol,Vicini),
 flatten(Vicini,ViciniFlat),
```

```
 vincolo_mine(Nrighe,Ncol,Mine,Vicini),

 nth1(ClickR,Mine,RigaClick),
 nth1(ClickC,RigaClick,ElClick),
 % Aggiungendo queste 3 righe si risolve l'esercizio 1.2
     nth1(ClickR,Vicini,RigaClickVicini),
     nth1(ClickC,RigaClickVicini,ElClickVicini),
     var(ElClickVicini),
 not(
    (ElClick=1,
     labeling(MineFlat)
    )
 ).

vincolo_mine(0,_Ncol,_Mine,_Vicini):- !.
vincolo_mine(Nrighe,Ncol,Mine,Vicini):-
 vincolo_mine_righe(Nrighe,Ncol,Mine,Vicini),
 Nrighe1 is Nrighe-1,
 vincolo_mine(Nrighe1,Ncol,Mine,Vicini).

vincolo_mine_righe(_Riga,0,_Mine,_Vicini):- !.
vincolo_mine_righe(Riga,Ncol,Mine,Vicini):-
 nth1(Riga,Vicini,RigaVicini),
 nth1(Ncol,RigaVicini,Num),
 estrai_confinanti(Riga,Ncol,Mine,Confinanti),
 occurrences(1,Confinanti,Num),
 Ncol1 is Ncol-1,
 vincolo_mine_righe(Riga,Ncol1,Mine,Vicini).
```