Alberto Alpe – Gianmario Cuccuru

Prolog
Clingo

Presentazione del progetto d'esame per la parte di programmazione logica

Università degli Studi di Torino – Dipartimento di Informatica

Alberto Alpe – Gianmario Cuccuru

I parte - Prolog

Università degli Studi di Torino – Dipartimento di Informatica

Scelte implementative

Dominio scelto: gioco del 15

- Versione 4X4
- Versione 3X3 (gioco dell'8)

Algoritmo di ricerca: IDA* con memoria

Codice organizzato in tre moduli:

- regole.pl
- euristiche.pl
- idastar.pl

regole.pl

Regole per individuare:

- la casella ‘vuota’ (tilezero/2)
- le tessere a lei adiacenti(adjacent/4)
- le azioni possibili
- le configurazioni derivanti dalla loro applicazione(trasforma/3)

Due modi per calcolare le azioni possibili:

- actions/2 → più semplice, trova le azioni applicabili a una configurazione
- testabili/3 → trova le azioni e le ordina in base al valore di euristica (crescente) della configurazione risultante.

euristiche.pl

Regole relative al calcolo delle euristiche.

Dato uno stato S (una configurazione) è possibile:

- aggiornarne il valore della G(G/2)
- calcolarne la F(F/3)
- calcolare l'euristica corrispondente(manhattan/2, hamming/2), tramite regole che trovano le caselle fuori posto (incorrects/2) e le coordinate desiderate per ogni valore (correct/2).

euristiche.pl

Euristiche usate:

- **manhattan/2 → somma delle distanze di Manhattan delle caselle in posizione errata**
- **hamming/2 → calcola il numero di caselle in posizione errata rispetto alla soluzione**

Alberto Alpe – Gianmario Cuccuru

idastar.pl

idastar.pl implementa l’algoritmo di ricerca IDA*

- Il numero di mosse necessarie a risolvere il gioco del 15 nelle configurazioni medio/difficili può essere relativamente elevato;
- A seconda della posizione della casella vuota, ogni stato può generare 2, 3 o 4 stati figli
- → la dimensione del problema può aumentare molto aumentando la profondità
- → per questo motivo è preferibile la ricerca in profondità rispetto a quella in ampiezza (es. A*)
- → la variante con memoria dell’algoritmo, tenendo traccia degli stati visitati, può alleggerire il ‘peso’ computazionale e ridurre i tempi di calcolo della soluzione.

Alberto Alpe – Gianmario Cuccuru

Test

Configurazioni di test:

4X4 facile

1	2	3	4
5	6	7	8
10	11	12	15
	9	13	14

4X4 medio

1	2	3	4
5	6	7	8
15		14	11
10	12	9	13

4X4 difficile

11	7	1	5
3	9	4	8
2	13	12	15
6	14	10	

3X3 facile

1	2	3
	6	8
4	7	5

3X3 medio

4		5
1	8	2
6	7	3

Alberto Alpe – Gianmario Cuccuru

Risultati

L’algoritmo risolve correttamente le configurazioni più semplici del puzzle 4X4 e 3X3

4X4, euristica Manhattan, regole ‘testabili’

```
% 87,614 inferences, 0.000 CPU in 0.097 seconds (0% CPU, Infinite Lips)
Lista di azioni: tile(2,4,9),left | tile(3,4,13),left | tile(4,4,14),left | tile(4,3,15),down | tile(3,3,12),right | tile(4,3,12),left | tile(2,3,11
),right | tile(3,3,11),left | tile(1,3,10),right | tile(1,4,9),up | tile(2,4,13),left | tile(3,4,14),left | tile(4,4,15),left.
S = [(tile(2, 4, 9), left),  (tile(3, 4, 13), left),  (tile(4, 4, 14), left),  (tile(4, 3, 15), down),  (tile(3, 3, 12), right),  (tile(4, 3, 12), l
eft),  (tile(2, 3, 11), right),  (tile(..., ..., ...), left),  (..., ...)|...] .
```

4X4, euristica Manhattan, regole ‘actions’

```
% 32,444 inferences, 0.000 CPU in 0.075 seconds (0% CPU, Infinite Lips)
Lista di azioni: tile(2,4,9),left | tile(3,4,13),left | tile(4,4,14),left | tile(3,4,14),right | tile(4,3,15),down | tile(3,3,12),right | tile(4,3,1
2),left | tile(2,3,11),right | tile(3,3,11),left | tile(1,3,10),right | tile(2,3,10),left | tile(1,4,9),up | tile(2,4,13),left | tile(3,4,14),left |
 tile(4,4,15),left.
S = [(tile(2, 4, 9), left),  (tile(3, 4, 13), left),  (tile(4, 4, 14), left),  (tile(3, 4, 14), right),  (tile(4, 3, 15), down),  (tile(3, 3, 12), r
ight),  (tile(4, 3, 12), left),  (tile(..., ..., ...), right),  (..., ...)|...]
                                                                                                   Ln 12, Col 6    Spaces: 4    U
```

Alberto Alpe – Gianmario Cuccuru

Risultati

L’algoritmo risolve correttamente le configurazioni più semplici del puzzle 4X4 e 3X3

4X4, euristica Hamming, regole ‘testabili’

```
% 87,614 inferences, 0.016 CPU in 0.094 seconds (17% CPU, 5607296 Lips)
Lista di azioni: tile(2,4,9),left | tile(3,4,13),left | tile(4,4,14),left | tile(4,3,15),down | tile(3,3,12),right | tile(4,3,12),left | tile(2,3,11
),right | tile(3,3,11),left | tile(1,3,10),right | tile(1,4,9),up | tile(2,4,13),left | tile(3,4,14),left | tile(4,4,15),left.
S = [(tile(2, 4, 9), left),  (tile(3, 4, 13), left),  (tile(4, 4, 14), left),  (tile(4, 3, 15), down),  (tile(3, 3, 12), right),  (tile(4, 3, 12), l
eft),  (tile(2, 3, 11), right),  (tile(..., ..., ...), left),  (..., ...)|...]
                                                                                                   Ln 37, Col 1   Spaces: 4   UTF-
```

4X4, euristica Hamming, regole ‘actions’

```
% 32,444 inferences, 0.016 CPU in 0.076 seconds (20% CPU, 2076416 Lips)
Lista di azioni: tile(2,4,9),left | tile(3,4,13),left | tile(4,4,14),left | tile(3,4,14),right | tile(4,3,15),down | tile(3,3,12),right | tile(4,3,1
2),left | tile(2,3,11),right | tile(3,3,11),left | tile(1,3,10),right | tile(2,3,10),left | tile(1,4,9),up | tile(2,4,13),left | tile(3,4,14),left |
 tile(4,4,15),left.
S = [(tile(2, 4, 9), left),  (tile(3, 4, 13), left),  (tile(4, 4, 14), left),  (tile(3, 4, 14), right),  (tile(4, 3, 15), down),  (tile(3, 3, 12), r
ight),  (tile(4, 3, 12), left),  (tile(..., ..., ...), right),  (..., ...)|...]
```

Alberto Alpe – Gianmario Cuccuru

Risultati

L'algoritmo risolve correttamente le configurazioni più semplici del puzzle 4X4 e 3X3

3X3, euristica Manhattan

```
% 18,488 inferences, 0.016 CPU in 0.039 seconds (40% CPU, 1183232 Lips)
Lista di azioni: tile(1,3,4),up | tile(2,3,7),left | tile(3,3,5),left | tile(3,2,8),down | tile(2,2,6),right | tile(2,3,5),up | tile(3,3,8),left.
S = [(tile(1, 3, 4), up), (tile(2, 3, 7), left), (tile(3, 3, 5), left), (tile(3, 2, 8), down), (tile(2, 2, 6), right), (tile(2, 3, 5), up), (tile(3, 3, 8), left)] .
```

3X3, euristica Hamming

```
% 18,488 inferences, 0.000 CPU in 0.036 seconds (0% CPU, Infinite Lips)
Lista di azioni: tile(1,3,4),up | tile(2,3,7),left | tile(3,3,5),left | tile(3,2,8),down | tile(2,2,6),right | tile(2,3,5),up | tile(3,3,8),left.
S = [(tile(1, 3, 4), up), (tile(2, 3, 7), left), (tile(3, 3, 5), left), (tile(3, 2, 8), down), (tile(2, 2, 6), right), (tile(2, 3, 5), up), (t
ile(3, 3, 8), left)]
```

Riflessioni

- Configurazioni più semplici → ‘actions’ funziona meglio di ‘testabili’, dovendo effettuare meno controlli, inoltre la soluzione si trova particolarmente ‘’a sinistra’’ nello spazio degli stati;
- Test con trace e output di debug → sembra che la regola dfs_aux non effettui il backtracking correttamente;
- Testate diverse versioni alternative della regola, ma nessuna effettua la ricorsione correttamente per configurazioni più complicate;
- La risoluzione del gioco non richiede più di 80 mosse → implementare un ‘upper bound’ di profondità circa 80: manterrebbe la ricerca depth-first, esplorando lo spazio degli stati anche in ampiezza in modo più efficiente

Alberto Alpe – Gianmario Cuccuru

II parte - Clingo

Università degli Studi di Torino – Dipartimento di Informatica

Scelte progettuali

Inseriti prima i vincoli ‘fondamentali’ per un campionato:

- ogni squadra gioca esattamente una partita a giornata
- ogni partita coinvolge due squadre e si gioca in casa di una delle due
- una squadra non può giocare contro se stessa

Aggiunti poi gli altri vincoli richiesti

Per il caso del derby tra squadre che condividono la struttura di gioco, implementate due versioni dello stesso vincolo (una specifica con i nomi delle squadre in questione e una più generica con le variabili, in caso si aggiungessero altre squadre che condividono la struttura).

Alberto Alpe – Gianmario Cuccuru

Predicati aggiuntivi

- **totale_partite/2** → calcola e dimostra che si giocano 10 partite a giornata
- **incontri_andata/3** → calcola e dimostra che ogni coppia si incontra una volta sola nel girone di andata
- **incontri_ritorno/3** → calcola e dimostra che ogni coppia si incontra una volta sola nel girone di ritorno
- **incontri_S_A/3** → calcola e dimostra che ogni coppia si incontra esattamente due volte tra andata e ritorno

Nota: mostrare i risultati di questi predicati rende molto confusionario l'output; tuttavia questi, se letti insieme, dimostrano che i vincoli del calendario sono rispettati (consultare screenshots nella cartella del progetto)

Risultati

Campionato a 20 squadre, tutti vincoli rispettati:

5271.178 s → ~87min

Campionato a 20 squadre, senza i vincoli facoltativi:

5001.008 s → ~83min

Campionato a 20 squadre, senza i facoltativi e senza il doppio incontro A/R per ogni coppia di squadre:

3551.282 s → ~59min

Campionato a 10 squadre(tutti i vincoli):

4.640 s

Riflessioni

Appare evidente come sia la dimensione del problema a influire maggiormente sui tempi di computazione (a parità di vincoli inseriti, 4sec per 10 squadre e 87min per 20squadre).

È comunque interessante notare che alcuni vincoli restringono maggiormente l'insieme delle soluzioni possibili aumentando considerevolmente i tempi.
Nello specifico, imporre che due squadre si incontrino una sola volta all'andata e una sola al ritorno fa passare i tempi di computazione da 60 a quasi 90 minuti.

Grazie per l’attenzione