AlphaBetaPrugna

Enrico Emiro - matricola N. 0000975012
Giulio Billi - matricola N. 0000970522

11 Febbraio 2022

Corso di Algoritmi e Strutture Dati dell'Università di Bologna
Relazione progetto

Indice

Lista Algoritmi

1 Problema computazionale

Il progetto mira ad implementare, con la soluzione più efficiente, un "giocatore ottimo" nel m,n,k-game (versione generalizzata del gioco "tic-tac-toe").
In un m,n,k-game ogni giocatore per vincere deve allineare, prima dell'avversario, K simboli in una matrice $M \times N$.
Il problema computazionale che emerge è quello relativo alla ricerca delle mosse migliori da effettuare ad ogni turno.
Un approcio naïf è quello di controllare e valutare ogni singola mossa possibile nella matrice, scegliendone la migliore; lo svantaggio di questo approccio è dato dall'impossibilità di applicarlo in matrici molto grandi a causa dell'elevato numero di configurazioni generabili.
La radice dell'albero di gioco ha infatti $M \times N$ figli che a loro volta hanno $(M \times N) - 1$ figli. Ne consegue che l'albero di gioco avrà un numero complessivo di nodi pari a:

$$O((M \times N) \cdot ((M \times N) - 1) \cdot \ldots \cdot 1) = O((M \times N)!)$$

A questo punto, è evidente che l'approccio naïf non rappresenta la migliore soluzione per risolvere il problema.
Di conseguenza, è necessario porre un limite alla profondità di esplorazione dell'albero e utilizzare una euristica per valutarne i nodi.

2 Soluzione adottata

Per risolvere il problema sopra citato, il "giocatore" è stato implementato utilizzando "alpha-beta pruning" come algoritmo di ricerca.
La profondità massima dell'alpha-beta viene gestita applicando la strategia di "iterative deepening".
Le configurazioni di gioco sono valutate attraverso un'euristica basata sul numero di celle consecutive di ogni giocatore.
Le celle che vengono esplorate sono ordinate dando priorità a quelle più vicine all'ultima mossa e con una posizione migliore.
Infine, per evitare di valutare le stesse configurazioni di gioco più volte, esse vengono salvate all'interno di una hash table nota come "tabella delle trasposizioni".

2.1 Select Cell

Il metodo "Select Cell" si occupa di scegliere la mossa che il giocatore deve eseguire. Esso, nel caso in cui il nostro giocatore sia il primo a giocare, sceglie

direttamente la cella centrale della matrice di gioco, poichè permette di avere il maggior numero di possibilità di movimento.
Se invece non siamo i primi a giocare, la scelta della cella è affidata all'algoritmo Alpha-beta pruning.
Nel caso in cui l'Alpha-beta non riesca a calcolare la mossa entro il tempo limite, viene scelta una mossa casuale tra quelle disponibili.

2.2 Alpha-beta pruning

Alpha-beta pruning è una variante dell'algoritmo minimax, che riduce drasticamente il numero di nodi da valutare nell'albero di ricerca e si basa su due valori: α e β.
α e β rappresentano, in ogni punto dell'albero, la posizione migliore e peggiore che è possibile raggiungere a partire dalla posizione (i, j).
Se A è il giocatore massimizzante, B è il giocatore minimizzante e (i, j) è la posizione di partenza:

- α: è il punteggio minimo che A può raggiungere a partire da (i, j).
- β: è il punteggio massimo che B può raggiungere a partire da (i, j).

Se $\beta \leq \alpha$, allora il sottoalbero relativo al nodo verrà "potato" perchè sicuramente la soluzione non sarà ottima.

2.3 Iterative deepening depth-first search (IDDFS)

L'iterative deepening gestisce la ricerca della mossa migliore eseguendo l'algoritmo Alpha-beta con una profondità sempre maggiore, realizzando così una visita in ampiezza dell'albero di gioco.
Il maggior numero di esecuzioni dell'Alpha-beta non peggiora la complessità computazionale in quanto il costo della ricerca a massima profondità sovrasta quelle a profondità minore.
Inoltre l'aumento del numero di iterazioni complessivo viene attenuato dall'ordinamento delle mosse - il quale permette un numero maggiore di potature - e dal salvataggio delle configurazioni di gioco già esplorate.
Questa strategia permette di sfruttare al meglio l'algoritmo Alpha-beta in quanto consente di ottenere la valutazione di un numero maggiore di configurazioni rispetto a quelle che si ottengono con una profondità fissata.

2.4 Ordinamento delle mosse

Per aumentare le potature e valutare prima le celle più promettenti, è stato effettuato un ordinamento delle mosse in base alla loro posizione e alla loro distanza dall'ultima cella marcata. In particolare, le mosse vengono prima ordinate circolarmente rispetto all'ultima cella marcata, allontanandosi progressivamente fino a raggiungere una distanza di K.
Successivamente vengono riordinate in base al valore associato alla loro posizione che è contenuto in una LinkedHashMap.
Una volta ordinate le mosse relative a un giocatore, vengono ordinate nello stesso modo quelle dell'avversario e infine vengono unite alle restanti celle libere.
L'ordinamento delle celle dei due giocatori viene eseguito K volte nelle 8 direzioni possibili, con un costo computazionale pari a:

$$O(8 * k) = O(K)$$

per ogni giocatore, mentre l'inserimento delle restanti celle libere ha un costo di:

$$O((M * N) - (8 * K))$$

2.5 Euristica

L'euristica utilizzata per valutare i nodi intermedi si basa principalmente sulla valutazione delle serie, in particolare valutiamo, rispetto all'ultima cella marcata dall'alpha-beta, le seguenti caratteristiche:

- Numero di celle consecutive che compongono una serie, in ogni direzione possibile e la loro distanza rispetto a K;
- Per ogni serie trovata prendiamo in considerazione le celle che hanno provocato la terminazione della serie, sui suoi due lati;
- La posizione della cella di partenza.

2.5.1 Serie

Le serie vengono valutate partendo dall'ultima cella marcata dall'Alpha-beta, nelle 4 direzioni che attraversano la cella considerata (verticale, orizzontale, diagonale e antidiagonale).
In particolare, ogni direzione viene esplorata contando le celle consecutive su un lato e poi sull'altro, e l'esplorazione si ferma una volta raggiunta una cella con uno stato diverso rispetto a quello di partenza.
Una volta terminata l'esplorazione sui due lati, i risultati vengono uniti e

valutati in base a quanto si avvicina la serie a K.
Di seguito riportiamo lo pseudo-codice per la valutazione delle serie.

Algoritmo 1 Valutazione delle serie

```
1: function EvalSeries(alignments)
2:     if K > 2 ∧ alignments == K − 1 then return 100
3:     else if K > 3 ∧ alignments == K − 2 then return 30
4:     else if K > 4 ∧ alignments == K − 3 then return 10
5:     end if
6:     return 0
7: end function
```

2.5.2 Incrementatore (increaser)

Oltre a contare il numero di celle consecutive, viene considerato lo stato delle celle che terminano ogni serie e in base ad esso viene aumentato o diminuito lo score della serie. Abbiamo definito 5 diversi gruppi di celle terminali e a ogni gruppo abbiamo assegnato un valore diverso:

- **Gruppo 1**: se le celle terminali sono entrambe nulle (fuori dalla board) o dell'avversario, lo score della serie rimane invariato;
- **Gruppo 2**: se le celle sono una nulla e una libera allora lo score viene incrementato di 1;
- **Gruppo 3**: se le celle sono una nulla e una dell'avversario allora lo score rimane invariato;
- **Gruppo 4**: se le celle sono una libera e una dell'avversario lo score viene incrementato di 1;
- **Gruppo 5**: se le celle sono entrambe libere lo score viene incrementato di 2;

In questo modo abbiamo un valore leggermente più alto per le serie che hanno più probabilità di produrre una vittoria.

2.5.3 Board scores

Analizzando le partite, abbiamo visto come le celle centrali della matrice danno al giocatore più opzioni per arrivare alla vittoria e in alcuni casi, permettono di creare serie che, se non vengono bloccate in partenza, portano a una vittoria assicurata (ad esempio in una matrice $5 \times 5 \times 4$ se in una delle direzioni un giocatore riesce a concatenare tre celle al centro ha una vittoria

assicurata perchè l'avversario non ha possibilità di bloccarlo).
Di conseguenza, abbiamo assegnato a ogni cella uno score che diventa maggiore più ci si avvicina al centro e che viene sommato alle celle che vengono valutate.

2.6 Tabella di trasposizione

Le configurazioni di gioco già valutate vengono salvate all'interno di una Hash Table nota come "tabella di trasposizione" in modo da non doverne ricalcolare il loro valore.
Le chiavi per accedere alle configurazioni salvate nella Hash Table vengono calcolate attraverso la funzione di hashing "Zobrist", che in base al Player che sta giocando e alle coordinate delle celle, calcola la chiave e il valore di hash associato alla configurazione presa in considerazione.
Questa funzione di hashing viene inizializzata come una matrice tridimensionale $2 \times M \times N$ con valori casuali.
Nella tabella di trasposizione i valori che vengono salvati per ogni configurazione sono:

- **Score**: punteggio associato alla configurazione;
- **Depth**: profondità a cui è stata valutata la configurazione;
- **Flag**: indica se lo score salvato è un valore di bound oppure è un valore ottenuto tramite una valutazione euristica.

Per evitare di mantenere nella tabella di trasposizione configurazioni che non sono più raggiungibili (ossia già giocate) queste vengono eliminate dalla Hash Table durante l'operazione di "cleanup" che avviene durante il turno dell'avversario.

2.7 Strategie considerate

In questa sezione vengono mostrate le varie strategie prese in considerazione durante la realizzazione del "giocatore".

Nota: Il termine "fork" che viene utilizzato in questa sezione, indica una particolare configurazione di serie che è possiblie creare in alcune matrici e che permette una vittoria assicurata.
Queste configurazioni si verificano quando un giocatore riesce ad allineare $K - 1$ celle nella parte centrale in una delle 4 direzioni; l'avversario non può evitare la sconfitta poichè qualsiasi lato della serie venga bloccato si può vincere dall'altro.

2.7.1 Analisi delle serie

Un primo approccio alla realizzazione dell'euristica prendeva in considerazione soltanto il numero di celle consecutive che componevano una serie, senza considerare le caratteristiche delle celle che ne provocavano la terminazione. I risultati ottenuti erano buoni, tuttavia la probabilità di creare delle "fork" si riduceva di molto.

2.7.2 Evaluate circolare

Per cercare di creare e contrastare le "fork" è stato cambiato il modo in cui venivano esplorate le celle nelle varie direzioni.
Analizzando i casi in cui si generavano le "fork", abbiamo osservato che venivano create o sulla "X" - formata dalle due diagonali - o sulla croce formata dalla riga orizzontale e verticale.
Di conseguenza in questa euristica vengono valutate queste due casistiche; per ogni disposizione si fissano nelle 4 direzioni le quattro celle più vicine a quella di partenza e per ogni cella fissata in una direzione, si valutano le altre celle nelle rimanenti 3 direzioni.
Con questo metodo, nelle matrici con dimensioni grandi i risultati peggiorano perchè prima di concatenare le celle si cerca di creare la disposizione per la "fork", creando così uno svantaggio.

3 Costo computazionale

Il costo computazionale del metodo Select Cell è determinato principalmente dal costo del metodo Iterative Deepening.
Come già detto il suo costo viene sovrastato dal costo dell'Alpha-beta pruning che nel caso peggiore ha un costo pari a: $O((M \times N))$.
Assumendo un numero medio di mosse "m" e una massima profondità "d" si ha un costo pari a $O(m^d)$ nel caso pessimo, mentre nel caso ottimo (in cui la prima scelta è sempre la migliore) si ha un costo di $O(\sqrt{m^d})$.
Grazie all'utilizzo della strategia di iterative deepening e all'ordinamento delle mosse si aumentano le probabilità di cut-off dei rami e si diminuiscono quelle di entrare nel caso pessimo. Inoltre l'integrazione della tabella di trasposizione all'interno dell'Alpha-beta pruning permette di diminuire il numero di chiamate al metodo di valutazione delle celle, diminuendo il numero di operazioni da eseguire.

4 Possibili miglioramenti

1. Assegnando punteggi più accurati alle serie e agli increaser probabilmente si potrebbero ottenere mosse migliori in alcune situazioni, in particolare quando non si parte come primo giocatore.
2. Utilizzare una variante dell'algoritmo Alpha-beta pruning with memory chiamata MTDF che permette di effettuare un numero maggiore di cut-off utilizzando una ricerca zero-window ovvero con una differenza tra alpha e beta pari a 1.
3. Nella valutazione delle celle si può trovare una strategia più efficace per valutare le mosse anche in base all'utilità che hanno per l'avversario.

5 Fonti utilizzate

- Alpha-beta pruning:
  - Virtuale
  - Wikipedia
- Zobrist hashing
- MTD(f) + tabella di trasposizione