Antonio

Relazione del progetto per l'insegnamento di Algoritmi e Strutture di Dati

Marco Milani (0000971467), Simone Tassi (0000971252)

Università di Bologna
Maggio 2023

Contents

1 Descrizione del problema

L'obiettivo del progetto è quello di implementare, con la soluzione più efficiente, un "giocatore con strategia ottima" per un CX-game, versione generalizzata del comune "*Connect 4*".
Ogni giocatore sceglie una colonna all'interno di una matrice M $\times$ N in cui far cadere il proprio disco che cade nella posizione più bassa disponibile nella colonna selezionata. Lo scopo del gioco è quello di allineare, per primo, X dischi in direzione verticale, orizzontale o diagonale.
La scelta della mossa ad ogni turno costituisce il problema computazionale principale nello sviluppo di questo progetto.
Per matrici di dimensioni ridotte si potrebbe procedere controllando e valutando tutte le mosse possibili. Ciò diventa però impossibile quando si prendono in esame matrici più grandi poiche' il numero di mosse disponibili sarebbe elevato.

2 Scelte progettuali

L'implementazione del giocatore si basa sull'algoritmo di ricerca *AlphaBeta Pruning*.
La potatura *AlphaBeta* è una tecnica di ottimizzazione per l'algoritmo *Minimax*, il quale suggerisce quanto sia convieniente per il giocatore fare una determinata scelta piuttosto che un'altra.

2.1 Select column

Il metodo *selectColumn* stabilisce la colonna che il player deve scegliere nel turno corrente. Esso si compone di una parte di valutazione preliminare in cui vengono esaminate situazioni in cui è particolarmente semplice determinare la miglior mossa da marcare e un'altra in cui si risolve il problema di scelta della mossa tramite l'algoritmo *AlphaBeta Pruning*.

Si procede dapprima controllando se c'è solo una colonna disponibile: in tal caso la scelta è obbligata.
Successivamente si itera su tutte le colonne selezionabili controllando se si ha la possibilità di vincere con una sola mossa o se può farlo l'avversario al turno successivo, in modo da bloccarlo anticipatamente.

Se non si presenta nessuno dei casi sopracitati si passa all'esecuzione ricorsiva di *AlphaBeta Pruning*, su ogni colonna a disposizione, utilizzando un parametro *depth* che ne determina la profondità limite in base alla dimensione della "board" di gioco. Confrontando i risultati dell'algoritmo si ottiene la colonna che massimizza la possibilità di vincita (e minimizza quella di sconfitta). Se il tempo a disposizione per controllare tutte le colonne libere non basta (se è stato superato il 99% del tempo a disposizione, ad esempio se il turno dura *10s*, dopo *9.9s*),

viene marcata la colonna considerata più vantaggiosa fino a quel momento.

2.2 AlphaBeta Pruning

Come accennato precedentemente *AlphaBeta Pruning* è un'ottimizzazione dell'algoritmo di ricerca *Minimax*, nel quale il numero di stati esaminati del gioco sarebbe esponenziale rispetto alla profondità dell'albero. Per ridurre il costo della ricerca si utilizza quindi *AlphaBeta Pruning* che senza controllare ogni nodo dell'albero di gioco permette di calcolare la decisione Minimax corretta. La potatura alfabeta può essere applicata a qualsiasi profondità di un albero e talvolta non solo pota le foglie dell'albero ma anche un intero sottoalbero. È necessario introdurre due parametri definiti come:

- α: La scelta migliore (con il valore più alto) trovata finora nel percorso di massimizzazione. Il valore iniziale di alfa viene settato a -∞.
- β: la scelta migliore (con il valore più basso) trovata finora nel percorso di minimizzazione. Il valore iniziale di beta viene settato a $+\infty$.

Se $\beta \leq \alpha$, allora il sottoalbero del nodo in questione viene "potato" perchè sicuramente non conduce a una soluzione ottima. Quando *AlphaBeta* va a controllare gli ultimi nodi, ovvero le foglie dell'albero di esplorazione, chiama il metodo *evaluate* che ritorna un valore intero che varia in base alla situazione in cui ci si trova. Ciò accade anche nel caso in cui venga raggiunta la profondità limite indicata dal parametro *depth* o quando il turno sta per finire (analogamente a ciò che succede in *selectColumn*).

2.3 Evaluate

Il metodo *evaluate*, come suggerisce il nome, valuta i vari stati delle foglie che vengono visitate da *AlphaBeta* assegnandogli un valore numerico. Esso prende in input lo stato attuale della "board" di gioco e nel caso di una situazione favorevole (vittoria) ritorna un valore positivo (100), nel caso contrario di situazione sfavorevole (sconfitta) ritorna un valore negativo (-100). Quando, si arriva a un nodo che porta al pareggio ritorna 0, mentre quando si raggiunge la massima profondità di esplorazione (quindi il nodo esplorato non è una foglia), viene chiamato il metodo *evalMaxDepth*.

2.4 EvalMaxDepth

Il metodo *evalMaxDepth* assegna un valore alla colonna nel caso in cui si sia giunti alla massima profondità di esplorazione, contando i dischi allineati in tutte le direzioni (verticale, orizzontale, diagonale e antidiagonale) e ritornandone la somma. Per fare ció vengono utilizzate *evalVertical, evalHorizontal, evalDiagonal* e *evalAntiDiagonal*.

- *evalVertical* conta i propri dischi, sottostanti al disco della mossa presa in esame. Se si é prossimi al limite verticale della matrice e non si ha la

possibilitá di allineare X dischi verso l'alto allora il metodo ritorna 0, in quanto la mossa non ha valore in senso verticale.

- *evalHorizontal* conta i propri dischi, adiacenti orizzontalmente al disco della mossa presa in esame. Se si é bloccati a destra e a sinistra dall'avversario o dal limite della matrice, la somma dei dischi è minore di X (perchè se fosse uguale a X sarebbe stata valutata nella prima parte di *selectColumn*) e quindi viene ritornato 0, in quanto la mossa non ha valore in senso orizzontale.
- *evalDiagonal* e *evalAntiDiagonal* sono analoghe a *evalHorizontal* ma in senso diagonale e antidiagonale.

2.5 Strutture dati utilizzate

- Array monodimensionali: utilizzati per iterare sulle colonne disponibili.
- Array bidimensionali: la "board" è implementata tramite un array bidimensionale.

3 Costo computazionale

Il costo computazionale di *selectColumn* corrisponde a $O(n)$ in quanto vengono visitate tutte le colonne disponibili nella "board", che sono al più n.

Il costo computazionale di *AlphaBeta Pruning* nel caso pessimo è $O((n)^d)$ (dove d indica la profondità massima di ricerca), questo quando non avviene nessuna potatura, infatti corrisponde al costo computazionale di *Minimax*.
Nel caso ottimo, ovvero quando le mosse migliori vengono sempre valutate per prime (e quindi i nodi visitati sono quelli strettamente necessari), il costo computazionale è $O(\sqrt{(n)^d})$.

4 Conclusioni

Nonostante le prestazioni di *"Antonio"* siano soddisfacenti potrebbero essere migliorate utilizzando ulteriori tecniche come quella dell'*Iterative deepening* che gestisce in modo migliore la profondità massima di ricerca in relazione al tempo disponibile.