Università degli Studi di Padova
Corso di Laurea Magistrale in Informatica
Insegnamento di Intelligenza Artificiale
A.A. 2013/2014

Algoritmo Minimax, varianti ed euristiche: applicazione al gioco Dots and Boxes

Riccardo Angius, matricola 1074854

1 Introduzione

Minimax è un algoritmo per il calcolo della strategia ottima per un agente inserito in un ambiente multiagente competitivo: la sua versione più semplice comporta l'esplorazione completa in profondità dell'albero di gioco, in modo che l'agente possa selezionare l'azione che lo porti verso lo stato terminale che ottimizzi la sua funzione di utilità.

Di seguito vengono discusse l'implementazione e la sperimentazione di Minimax e delle sue varianti nell'ambito del gioco Dots and Boxes, un esempio di gioco a somma zero con informazione perfetta.

2 Regole del gioco

Il gioco avviene su una griglia quadrata di punti come mostrato in Figura 1. I giocatori a turno scelgono e tracciano una linea tra due punti adiacenti (verticalmente o orizzontalmente) che non siano già collegati. Quando la linea è l'ultima delle quattro necessarie per disegnare un quadrato che non include punti nella regione delimitata dal proprio perimetro, il giocatore che l'ha tracciata guadagna un punto e continua tracciandone un'altra. Se questo non si verifica, il turno passa all'avversario.

Figura 1 Tavoliere di gioco 3x3.

Il gioco termina quando sono state tracciate tutte le linee possibili e il vincitore è colui che ha tracciato l'ultima linea per il maggior numero di quadrati.

3 Lavoro svolto

È stato deciso l'uso di un linguaggio moderno come Python sia per la disponibilità di librerie, quali NetworkX e WxPython, che permettessero di astrarre facilmente da dettagli eccessivamente a basso livello, sia per la chiarezza del codice che permette di produrre.

Dopo un primo periodo in cui è stata sviluppata un'implementazione del modello di gioco contestualmente ad un'applicazione che permettesse a due utenti di sfidarsi, si è proceduto all'implementazione di un agente giocatore che utilizzasse l'algoritmo Minimax.

I test preliminari durante lo sviluppo sono avvenuti usando una griglia di punti 3x3, che comporta una profondità massima dell'albero di gioco di $m = 12$ mezze mosse, deputando all'utente la scelta della prima mezza mossa e all'agente quella di risposta. Di seguito i riferimenti al numero di nodi visitati descrivono dunque un'esplorazione dell'albero a 11 livelli di distanza dagli stati terminali.

L'agente Minimax naive ottenuto ha subito mostrato le proprie limitazioni, richiedendo la visita di 108.505.112 nodi per la scelta dell'azione migliore in riposta alla prima mezza mossa dell'utente. Un netto miglioramento è stato apportato dall'uso della variante Alfa-beta, che ha permesso di ridurre a 405.294 il numero di nodi visitati per prendere una decisione nelle medesime condizioni.

Anche in Dots and Boxes, come in molti altri giochi, possiamo verificare la presenza di stati

ripetuti nell'albero di gioco: data una sequenza di mezze mosse, lo stato risultante a partire da quello iniziale risulta infatti a volte raggiungibile anche da alcune permutazioni della stessa sequenza.

L'utilizzo di una tabella delle trasposizioni ha permesso di ridurre sensibilmente il numero di nodi generati dall'agente, rendendo quasi trascurabile il tempo necessario per la scelta della mossa migliore nei turni successivi al primo (in quanto effettivamente precalcolata). Nonostante le aspettative, il suo uso nelle istanze di gioco più grandi non ha apportato un overhead significativo tale da diventare controproducente in termini di efficienza dell'agente. Nell'esempio base ha ridotto a 14.300 il numero di nodi visitati dal Minimax naive (circa 20,17s) e a 6.331 quelli con la potatura Alfa-beta (circa 8,97s).

Come noto, a parità di tempo un ordinamento ottimo dei successori di un nodo permette all'algoritmo alfa-beta di percorrere più in profondità l'albero di gioco permettendo la potatura tempestiva dei rami inutili ai fini della decisione: è stato dunque introdotta la possibilità di specificare un criterio di ordinamento per i successori.

Inoltre si è deciso di implementare la ricerca con backtracking tramite l'uso di un iteratore, in modo da ridurre la complessità spaziale dell'algoritmo: la necessità di questa feature è stata giudicata superflua per le dimensioni delle istanze di gioco sperimentate e andrà rivalutata con dimensioni maggiori.

Mettendo da parte la necessità di un calcolo esatto della mossa migliore, sono state sperimentati varianti operanti entro dei limiti temporali ristretti e prestabiliti, facendo uso di funzioni di valutazione euristiche: l'agente sviluppato inizialmente richiedeva la specifica del livello di esplorazione massima possibile, rendendo necessario un calcolo a priori grossolano della corrispondenza tra questo e il tempo desiderato per la scelta dell'azione.

Successivamente l'uso di un timer ha permesso l'uso della ricerca ad approfondimento iterativo, imponendo all'agente di interromperla e dichiarare la mossa ritenuta migliore in base alla funzione di valutazione adottata.

In ultima istanza, si è deciso di permettere all'agente di far uso di una funzione di valutazione della quiescenza degli stati posti nel livello di cutoff, in modo da poter approfondire ulteriormente la ricerca solo nei sottoalberi di quelli *rumorosi*. Sperimentalmente si è scelto di permettere all'agente di approfondire la ricerca al massimo per 2 livelli oltre quello in cui dovrebbe essere stata interrotta.

4 Funzioni di valutazione

Sono state sperimentate diverse funzioni di valutazione, formulate come funzioni lineari pesate delle caratteristiche identificate da S_1, S_2, S_3, dove S_N indica il numero di gruppi di punti con esattamente N linee su 4 necessarie per la tracciatura di un quadrato, e da S_{4A} e S_{4B}, indicatori dei punteggi rispettivamente del giocatore A (che muove per primo) e del giocatore B.

$$Eval_1(P) = S_{4A} - S_{4B}$$

$$Eval_4(P) = 2(S_{4A} - S_{4B}) + s(P)(0.75S_3 - 0.5S_2)$$

$$s(P) = \begin{cases} +1 & P = A \\ -1 & P = B \end{cases}$$

La $Eval_1$ deriva direttamente dall'applicazione prematura della funzione di valutazione degli stati terminali.

La $Eval_4$ è stata riscontrata in un lavoro con obiettivi simili [1] al presente, e si poggia sulla seguente intuizione: avere uno o più gruppi di punti con una sola linea mancante si traduce in punteggio aggiuntivo immediatamente acquisibile per il giocatore corrente, mentre la presenza di soli gruppi con due linee mancanti comporta l'obbligo a tracciare una terza linea in uno di essi, permettendo all'avversario l'acquisizione di punteggio con una singola mossa. Vengono dunque privilegiati gli stati con gruppi del primo tipo penalizzando al contempo quelli contenenti gruppi del secondo, utilizzando dei coefficienti adattati empiricamente. Per quanto possa sembrare ragionevole, questa intuizione in realtà assume che S_2

e S_3 siano indipendenti, un'ipotesi abbastanza implausibile.

Rifuggendo queste ipotesi è stato sperimentato il comportamento delle euristiche che considerano le S_4, a cui viene dato un peso maggiore, in congiunzione a una sola tra S_2, S_3 e S_4.

$$Eval_5(P) = 2(S_{4A} - S_{4B}) + s(P)(S_2)$$

$$Eval_6(P) = 2(S_{4A} - S_{4B}) + s(P)(S_3)$$

$$Eval_7(P) = 2(S_{4A} - S_{4B}) + s(P)(S_4)$$

5 Risultati sperimentali

La sperimentazione si è concentrata sul confronto delle euristiche: tutte le diverse tipologie finali di agente condividono l'uso di Alfabeta e della tabella delle trasposizioni. La denominazione generale C_K indica che l'agente usa la $Eval_K$: seguita da T_1, indica che l'agente ordina i successori applicando direttamente ad essi la $Eval_K$, seguita da T_2 che l'ordinamento è secondo i valori calcolati al livello di cutoff nell'iterazione precedente. Una Q indica che l'agente opera ricerca di quiescenza fino ad un massimo di 2 livelli in più rispetto al livello di approfondimento corrente.

Esperimento	1	2	X
C_1 vs C_1T_1	30	40	30
C_1 vs C_1T_2	27	32	41
C_1T_1 vs C_1T_2	40	27	33
C_1 vs C_1Q	26	45	29
C_1 vs C_4	38	24	38
C_1Q vs C_5 e	33	36	31
C_1Q vs C_6	28	33	39
C_1Q vs C_7	29	32	39
C_5 vs C_6	36	29	35
C_5 vs C_7	34	28	38
C_6 vs C_7	31	30	39

Tabella 1 Sperimentazione con istanza di gioco 3x3, 1 secondo per decidere.

Ogni riga nella tabella degli esperimenti descrive vittorie, sconfitte e pareggi per 100 partite tra i due sfidanti menzionati nella prima colonna, scegliendo ognuno come giocatore iniziale per 50 partite.

Dall'analisi della tabella non è possibile fare affermazioni che caratterizzino in modo forte le varie euristiche: con maggior tempo a disposizione sarà necessario verificare l'ipotesi che alcune varianti risultino avvantaggiate rispetto ad altre quando il tempo per la decisione permette di analizzare più complessamente e in modo più proficuo ogni stato prima di passare ad analizzare i livelli successivi. Uno studio più a lungo termine dovrà prendere necessariamente in considerazione la ripetizione degli esperimenti aumentando sia il tempo per ogni partita che la dimensione del tavoliere.

Con queste premesse in mente, possiamo comunque notare che nonostante la C_1Q permetta un netto miglioramento rapportato al semplice C_1 (riga 4), il suo vantaggio viene ridotto in occasione di un confronto con agenti che non fanno ricerca di quiescenza e usano differenti funzioni euristiche. Sembrerebbe dunque che le $Eval_5$, $Eval_6$ e $Eval_7$ siano dei veri miglioramenti rispetto alla semplice $Eval_1$, mentre la $Eval_4$ addirittura operi un ordinamento degli stati che non rispecchia il loro vero valore di utilità.

6 Lavoro futuro

Oltre quanto rimarcato nella sezione precedente, nel lavoro futuro, con gli strumenti fin qui sviluppati sarebbe sicuramente di interesse investigare la trasposizione delle strategie descritte in [2] in funzioni utilizzabili come euristiche per la valutazione.

Ulteriori miglioramenti nell'ambito dell'efficienza includono la modifica incrementale della copia dello stato corrente usata dall'agente, in modo da non operare una copia completa per ogni successore da generare.

Per quanto riguarda la riduzione del numero di nodi generati, sarebbe di interesse verificare l'applicabilità dell'uso di certificati per identificare isomorfismi nei grafi rappresentanti il tavoliere di gioco, in modo da poter trascurare l'esplorazione di rami di gioco equivalenti in termini di possibilità di vincita per entrambi gli avversari.

Riferimenti bibliografici

[1] O'Flaherty R. e Jay Wu, "Dots-and-Boxes AI Algorithm", `http://www.rowlandoflaherty.com/wp-content/uploads/2012/04/Project1_Paper_DotsAndBoxes_OFlaherty_Wu.pdf`,
ultimo accesso: 15 luglio 2014.

[2] Berlekamp, E., *The Dots-and-Boxes Game*. A K Peters, 2000.

Appendice

Attività	Tempo
Sviluppo applicazione per gioco tra due utenti-sfidanti.	30 ore
Sviluppo avversario minimax semplice, con uso di thread.	10 ore
Sviluppo alfa beta come generalizzazione di minimax; cutoff e valutazione.	10 ore
Ricerca quiescenza.	10 ore
Refactoring codice prototipale.	20 ore
Verifica correttezza codice tramite rilettura e test.	10 ore
Ricerca letteratura e studio delle caratteristiche per funzioni di valutazione.	30 ore
Sperimentazione e raccolta risultati.	20 ore
Produzione relazione e presentazione finali.	10 ore

Tabella 2 Consuntivo ore del lavoro svolto.