MCTS e videogiochi: un'applicazione per le Gare Pokémon Live

Matteo Silvestro

10 ottobre 2015

1 MCTS
- Applicazione alle Gare Pokémon Live
- L'algoritmo
- Tipo di retropropagazione

2 UCT
- Multi-armed bandit problem
- UCB1
- UCT e convergenza al minimax
- Caratteristiche

3 Analisi delle prestazioni
- Simulazioni interne e contro il 3DS
- Prestazioni al variare delle iterazioni

4 Considerazioni finali
- Cosa si può migliorare
- Programma e codice sorgente

Le Gare Pokémon Live

La scelta è ricaduta sulle Gare Pokémon Live per diversi motivi:

- sono un buon compromesso tra giochi deterministici a turni e videogiochi stocastici in tempo reale;
- sono un gioco a 4 giocatori con scelta delle mosse simultanea;
- sono interessanti dal punto di vista strategico senza essere troppo complicate.

Sono state completamente implementate la Gare di Bellezza di livello Master, con i cinque avversari che può schierare il 3DS, ovvero Tropica, Plumy, Macy, Betta e Trod, più uno per il giocatore umano, Speranza.

Monte Carlo Tree Search

Il Monte Carlo Tree Search (MCTS) è una tecnica di ricerca *best-first* (ovvero un algoritmo di ricerca che espande i nodi più promettenti in base a una regola specifica) che si basa su simulazioni casuali, in cui sia il giocatore che gli avversari giocano mosse in modo casuale.
Da una sola partita si può dedurre poco, ma, simulando molte partite casuali, si può trarre una buona strategia.
L'algoritmo si compone di quattro fasi, che vengono eseguite per un certo numero di iterazioni.

Diagramma delle fasi

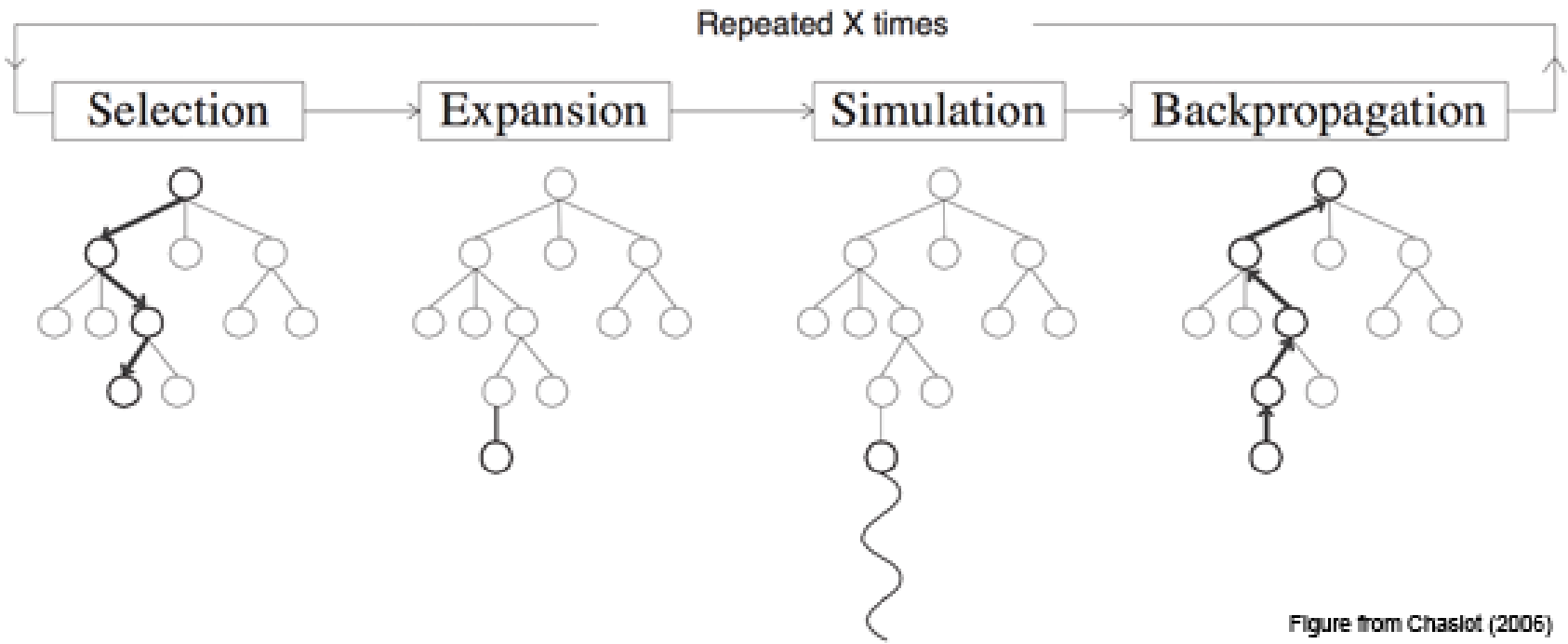


Figure from Chaslot (2006)

Fasi per iterazione

Selezione Viene usata una *strategia* per la selezione dei figli in modo ricorsivo.

Espansione Viene aggiunto un nodo figlio al nodo selezionato.

Simulazione Dal nodo espanso si esegue una simulazione finché non si raggiunge uno stato terminale del gioco e quindi un risultato.

Retropropagazione Il risultato della simulazione viene usato per aggiornare ogni nodo dell'albero attraversato.

Fasi per iterazione

Selezione Viene usata una *strategia* per la selezione dei figli in modo ricorsivo.

Espansione Viene aggiunto un nodo figlio al nodo selezionato.

Simulazione Dal nodo espanso si esegue una simulazione finché non si raggiunge uno stato terminale del gioco e quindi un risultato.

Retropropagazione Il risultato della simulazione viene usato per aggiornare ogni nodo dell'albero attraversato.

Fasi per iterazione

Selezione Viene usata una *strategia* per la selezione dei figli in modo ricorsivo.

Espansione Viene aggiunto un nodo figlio al nodo selezionato.

Simulazione Dal nodo espanso si esegue una simulazione finché non si raggiunge uno stato terminale del gioco e quindi un risultato.

Retropropagazione Il risultato della simulazione viene usato per aggiornare ogni nodo dell'albero attraversato.

Fasi per iterazione

Selezione Viene usata una *strategia* per la selezione dei figli in modo ricorsivo.

Espansione Viene aggiunto un nodo figlio al nodo selezionato.

Simulazione Dal nodo espanso si esegue una simulazione finché non si raggiunge uno stato terminale del gioco e quindi un risultato.

Retropropagazione Il risultato della simulazione viene usato per aggiornare ogni nodo dell'albero attraversato.

Tipo di retropropagazione

Nell'implementazione attuale, per il tipo di aggiornamento nella propagazione sono state analizzate diverse possibilità:

- $+1$ in caso di vittoria, 0 in caso di sconfitta;
- numero di cuori totali guadagnati durante la partita;
- somma del numero totale di cuori guadagnati e del numero totale di cuori totali guadagnati dal giocatore peggiore.

Questo ultimo sistema si è rivelato il più adatto, scegliendo mosse che ottimizzano il guadagno di cuori contando che anche gli altri giocatori sceglieranno mosse per massimizzare il guadagno.

Strategie di selezione

Un ruolo fondamentale per le prestazioni dell'algoritmo è svolto dalla *strategia di selezione*, che decide il modo in cui vengono selezionati i nodi nella prima fase.
La strategia banale di selezione in modo casuale uniforme è molto poco efficiente.
L'UCT, invece, si affermò subito come strategia di selezione, in quanto è stato dimostrato essere molto efficiente. Dopo la sua introduzione, l'MCTS cominciò a essere usato con grandissimo successo e superare altri algoritmi di AI in molti giochi, in particolare nel Go.

Multi-armed bandit problem

Un *K-armed bandit problem* è definito dalle variabili casuali $X_{i,n}$ per $1 \leq i \leq K$ e $n \geq 1$, in cui i identifica una leva della slot machine. Abbassando la leva i si ottengono le ricompense $X_{i,1}, X_{i,2}, \cdots$ che sono indipendenti e indenticamente distribuite con un'attesa incognita μ_i. Si suppone anche l'indipendenza tra $X_{i,s}$ e $X_{j,t}$ per ogni $1 \leq i \leq K$ e $s, t \geq 1$
Una *strategia di allocazione* è un algoritmo che sceglie la prossima leva da abbassare basandosi sulle giocate precedenti e sulle ricompense ottenute.

Upper Confidence Bounds (UCB1)

L'UCB1 è una strategia di allocazione molto efficiente:

- inizialmente si gioca ogni leva una volta;
- si gioca la leva j che massimizza $\overline{X_j} + \sqrt{\frac{2 \log n}{n_j}}$ dove $\overline{X_j}$ è la ricompensa media ottenuta dalla leva j, n_j è il numero di volte che la leva j è stata giocata finora, e n è il numero totale di giocate.

Si ottiene così un compromesso tra *sfruttamento* (primo termine) e *esplorazione* (secondo termine).

UCT

Ogni volta che un nodo (raggiungibile tramite una mossa) viene scelto dall'albero esistente, la scelta può essere modellata come un *multi-armed bandit problem*. Viene scelto il nodo j che massimizza:

$$\text{UCT} = \overline{X_j} + 2C_p\sqrt{\frac{2\log n}{n_j}}$$

dove $\overline{X_j}$ è la ricompensa attesa approssimata dalle simulazioni Monte Carlo, n è il numero di volte che il nodo corrente (padre) è stato visitato, n_j è il numero di volte che il nodo figlio j è stato visitato e $C_p > 0$ è una costante.

Convergenza al minimax

Il *minimax* è un algoritmo di intelligenza artificiale che, a piena profondità, crea l'albero completo del gioco e assegna ad ogni stato del gioco un valore, basandosi sui risultati dei nodi terminali. Kocsis e Szepesvári mostrarono che la probabilità di fallimento alla radice dell'albero (ovvero la probabilità di scegliere una mossa sub-ottimale) converge a zero con un andamento polinomiale al tendere a infinito dei giochi simulati. Questa dimostrazione implica che, con abbastanza tempo (e memoria), l'UCT permette all'MCTS di convergere all'albero di minimax ed è, quindi, ottimale.

L'MCTS è una scelta popolare tra gli algoritmi di AI per alcune sue caratteristiche uniche:

Non euristico Non è necessaria conoscenza di dominio specifica, quindi nell'implementazione di base è sufficiente conoscere soltanto le regole del gioco

Sempre arrestabile In qualsiasi momento può essere interrotto, restituiendo un'azione alla radice dell'albero. Continuando le iterazioni spesso si migliora il risultato.

Asimmetrico L'UCT seleziona i nodi più promettenti, senza tralasciare nodi inesplorati, portano a un albero asimmetrico con il passare del tempo.

L'MCTS è una scelta popolare tra gli algoritmi di AI per alcune sue caratteristiche uniche:

Non euristico Non è necessaria conoscenza di dominio specifica, quindi nell'implementazione di base è sufficiente conoscere soltanto le regole del gioco

Sempre arrestabile In qualsiasi momento può essere interrotto, restituiendo un'azione alla radice dell'albero. Continuando le iterazioni spesso si migliora il risultato.

Asimmetrico L'UCT seleziona i nodi più promettenti, senza tralasciare nodi inesplorati, portano a un albero asimmetrico con il passare del tempo.

L'MCTS è una scelta popolare tra gli algoritmi di AI per alcune sue caratteristiche uniche:

Non euristico Non è necessaria conoscenza di dominio specifica, quindi nell'implementazione di base è sufficiente conoscere soltanto le regole del gioco

Sempre arrestabile In qualsiasi momento può essere interrotto, restituiendo un'azione alla radice dell'albero. Continuando le iterazioni spesso si migliora il risultato.

Asimmetrico L'UCT seleziona i nodi più promettenti, senza tralasciare nodi inesplorati, portano a un albero asimmetrico con il passare del tempo.

Analisi delle prestazioni

Una volta implementato il programma si sono svolte della analisi per verificare le prestazioni dell'algoritmo, sia all'interno del programma che contro il 3DS, l'unica intelligenza artificiale disponibile come metro di confronto.
In tutte le analisi si sono presi in considerazione:

- lo *scarto di cuori*, ovvero la differenza tra il numero di cuori guadagnato in una partita e la media di cuori di tale partita;
- la *percentuale di vittorie* sul totale di partite giocate.

Campioni in esame

I campioni in esame sono i seguenti:

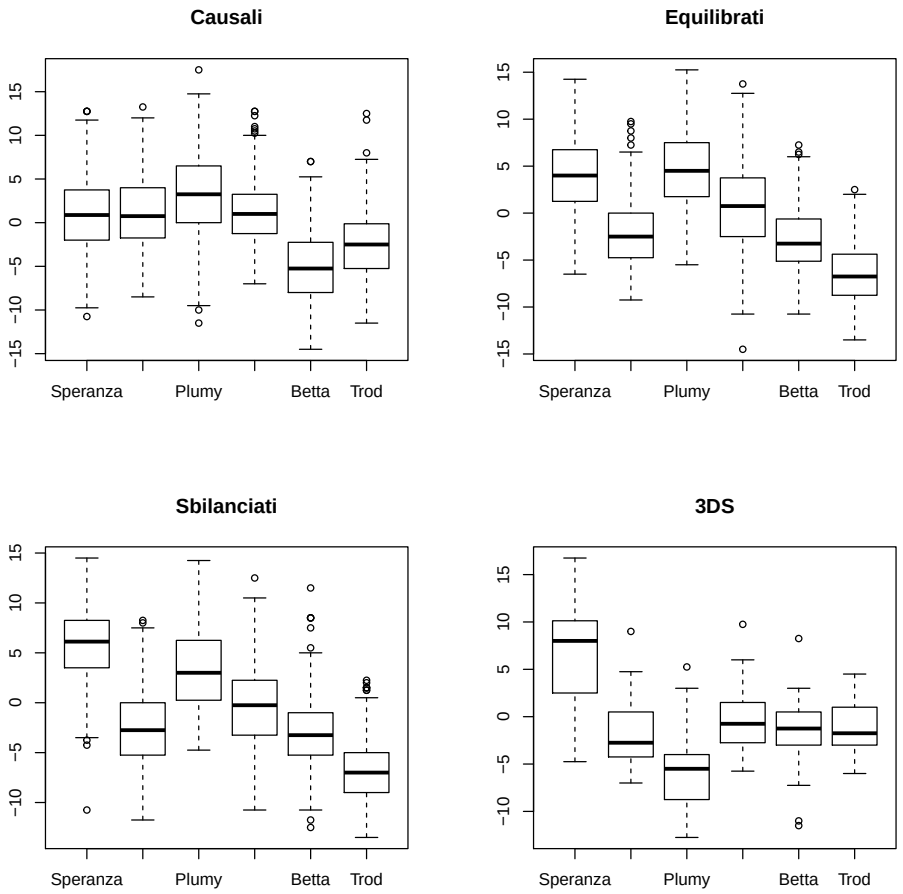
Casuali Tutti i giocatori eseguono mosse scegliendole in modo casuale uniforme.

Equilibrati Tutti i giocatori scelgono le mosse usando l'MCTS con 100 iterazioni.

Sbilanciati Speranza sceglie le mosse con l'MCTS a 10000 iterazioni contro gli altri con sole 100 iterazioni.

3DS Speranza sceglie le mosse con l'MCTS a 10000 iterazioni contro i giocatori del 3DS.

Boxplot dello scarto di cuori



Confronto della percentuale di vittorie

	Speranza	Tropica	Plumy	Macy	Betta	Trod
Casuali	24.40%	32.21%	49.17%	25.80%	7.82%	10.36%
Equilibrati	44.60%	7.74%	52.19%	27.27%	7.74%	0.63%
Sbilanciati	67.00%	4.25%	29.12%	17.79%	4.68%	0.64%
3DS	70.91%	9.68%	5.71%	16.22%	6.90%	9.09%

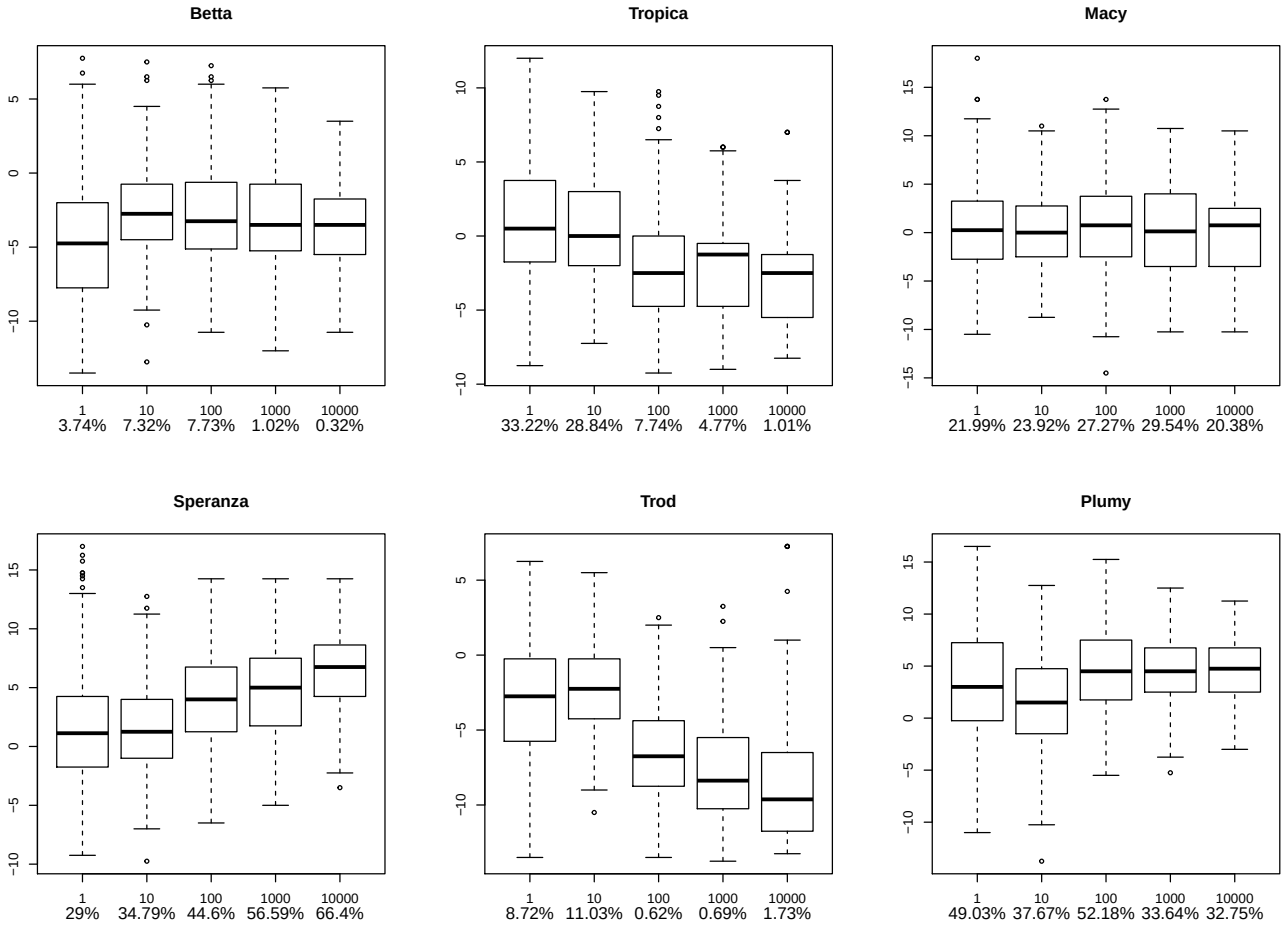
Commenti

Come si nota da queste analisi, l'algoritmo funziona bene con i giocatori che hanno mosse che possono far guadagnare molti cuori, che sono in particolare Speranza e Trod, mentre invece è meno adatto a giocatori come Trod, che possiede quasi solo mosse che danno al massimo 3 cuori.
Inoltre l'algoritmo, con la sua strategia che punta a massimizzare i cuori guadagnati in ogni partita, si mostra molto forte contro l'AI del 3DS, con la sua strategia più aggressiva. Infatti Speranza, il giocatore scelto per le simulazioni, raggiunge una percentuale di vittoria del 70.91% contro i giocatori del 3DS.

Campioni in esame

Si è scelto di analizzare, successivamente, lo scarto di cuori dalla media della partita all'aumentare delle iterazioni, supponendo che tutti giocatori usino l'MCTS con ugual numero di iterazioni.

Boxplot riassuntivo



Analisi qualitativa

Il risultato interessante è che non si notano differenze significative al crescere delle iterazioni. Questo significa che tutti i giocatori hanno un miglioramento della strategia equilibrato al rafforzarsi dell'algoritmo.
A ogni modo, anche in questo caso si nota un leggero miglioramento di Speranza e Plumy e un leggero peggioramento di Trod, in linea con le analisi precedenti.

Cosa si può migliorare?

Ci sono ampi margini di miglioramento per il programma, di seguito alcune idee:

- focalizzarsi su tutti i giocatori, riuscendo a sfruttare meglio, per esempio, un giocatore come Trod;
- implementare il gioco nella sua interezza, non solo limitato alle Gare di Bellezza di livello Master;
- ottimizzare l'algoritmo MCTS;
- aggiungere un algoritmo di *machine learning*.

Cosa si può migliorare?

Ci sono ampi margini di miglioramento per il programma, di seguito alcune idee:

- focalizzarsi su tutti i giocatori, riuscendo a sfruttare meglio, per esempio, un giocatore come Trod;
- implementare il gioco nella sua interezza, non solo limitato alle Gare di Bellezza di livello Master;
- ottimizzare l'algoritmo MCTS;
- aggiungere un algoritmo di *machine learning*.

Cosa si può migliorare?

Ci sono ampi margini di miglioramento per il programma, di seguito alcune idee:

- focalizzarsi su tutti i giocatori, riuscendo a sfruttare meglio, per esempio, un giocatore come Trod;
- implementare il gioco nella sua interezza, non solo limitato alle Gare di Bellezza di livello Master;
- ottimizzare l'algoritmo MCTS;
- aggiungere un algoritmo di *machine learning*.

Cosa si può migliorare?

Ci sono ampi margini di miglioramento per il programma, di seguito alcune idee:

- focalizzarsi su tutti i giocatori, riuscendo a sfruttare meglio, per esempio, un giocatore come Trod;
- implementare il gioco nella sua interezza, non solo limitato alle Gare di Bellezza di livello Master;
- ottimizzare l'algoritmo MCTS;
- aggiungere un algoritmo di *machine learning*.

Programma e codice sorgente

È possibile scaricare il programma e il codice sorgente, rilasciato sotto licenza MIT, dal mio sito personale.
Per maggiori informazioni, consultare l'appendice della tesi.
Grazie per l'attenzione!