La strategia MiniMax e le sue varianti

Paolo Boldi

29 gennaio 2002

1 Preliminari

In questa dispensa studieremo una strategia per l'analisi di alcuni giochi deterministici (cioè, giochi di pura abilità in cui il caso non gioca alcun ruolo). Considereremo solo giochi:

- *a due giocatori*, nei quali cioè si affrontano solo due contendenti, indicati con 0 e 1;
- *a mosse alternate*, nei quali cioè i giocatori muovono a turno, alternandosi (se una certa mossa spetta al giocatore x, la mossa successiva spetta al giocatore $1 - x$).

In qualunque gioco, si può definire il concetto di *stato*: cosa sia uno stato di gioco dipende dal gioco stesso; per esempio, nel caso degli scacchi lo stato è rappresentato dalla disposizione dei pezzi sulla scacchiera. Una *configurazione* di gioco è data dallo stato in cui il gioco si trova e dal giocatore cui tocca muovere (detto il "giocatore di turno"). Se c indica una configurazione, indicheremo con $c.s$ lo stato e con $c.t$ il giocatore di turno ($c.t \in \{0, 1\}$).

Alcune configurazioni sono dette *finali*: quando si arriva ad una configurazione finale, il gioco termina. Se c è una configurazione non finale, il giocatore $c.t$ può in genere scegliere fra molte mosse possibili, arrivando in diverse nuove configurazioni $c_1, \ldots, c_k$: tali configurazioni sono dette *configurazioni successive* a c (e, ovviamente, $c_i.t = 1 - c.t$, essendo il gioco a mosse alternate).

Quando si analizza un gioco, in genere si ha l'obiettivo di far vincere uno dei due giocatori, che chiameremo *protagonista* (l'altro giocatore sarà detto *avversario*). Il gioco inizia da una certa configurazione iniziale, a partire dalla quale i giocatori muovono a turno fino ad arrivare ad una configurazione finale: a questa configurazione corrisponde un *esito*. L'esito è un numero (positivo, nullo o negativo) che rappresenta quanto il protagonista guadagna alla fine. Ad esempio, nel gioco degli scacchi ci sono solo tre esiti possibili (1, che indica che il protagonista ha vinto; -1 che indica che l'avversario ha vinto; oppure 0 che indica una patta).

2 La strategia MiniMax

Per il momento, restringiamo la nostra attenzione a giochi con due ulteriori vincoli:

- *giochi aciclici*: cioè, a partire dalla configurazione iniziale, comunque i giocatori decidano di muovere si arriva prima o poi ad una configurazione finale;
- *giochi privi di memoria*: cioè, giochi in cui l'esito dipende solo dalla configurazione finale raggiunta, e non dalla sequenza di mosse fatte.

La strategia MiniMax è in grado, almeno in linea di principio, di permettere a ciascun giocatore di giocare sempre in modo ottimale, posto che le precedenti restrizioni siano rispettate. L'idea della strategia è quella di assegnare, ad ogni possibile configurazione c, un *punteggio* $p(c)$ con il seguente significato: $p(c)$ è il massimo valore ottenibile dal protagonista se il gioco parte dalla configurazione c ed entrambi i giocatori giocano in modo ottimale per loro.

L'idea è semplice: se c è una configurazione finale, $p(c)$ è semplicemente l'esito della configurazione c. Se c non è una configurazione finale, il calcolo di $p(c)$ dipende da $c.t$:

- se il giocatore di turno è il protagonista, lui sceglierà, fra le configurazioni successive $c_1, \ldots, c_k$ quella che gli permette di massimizzare il proprio punteggio, e quindi

$$p(c) = \max\{p(c_1), \ldots, p(c_k)\}$$

- se il giocatore di turno è l'avversario, lui sceglierà, fra le configurazioni successive $c_1, \ldots, c_k$ quella che gli permette di massimizzare il proprio punteggio, ovvero di minimizzare il punteggio dell'avversario, e quindi

$$p(c) = \min\{p(c_1), \ldots, p(c_k)\}$$

Ora, per decidere come muovere si può procedere come segue: se c è una configurazione e $c.t$ è il protagonista, sceglierà fra le configurazioni successive quella (o una di quelle) che realizza il massimo, cioè sceglierà una configurazione c_i tale che $p(c) = p(c_i)$.

Un esempio: il gioco del Nim. La procedura MiniMax può essere rappresentata graficamente mediante l'albero di configurazioni del gioco. Per spiegare di cosa si tratta, consideriamo il "gioco del Nim". In questo gioco[1] a due giocatori, esiste un certo numero di pile;

[1] Abbiamo scelto questo gioco in quanto è didatticamente molto adatto, vista la sua estrema semplicità. In realtà, esiste una semplice strategia combinatoriale che consente al primo giocatore di vincere sempre e che non richiede l'analisi dell'albero di gioco.

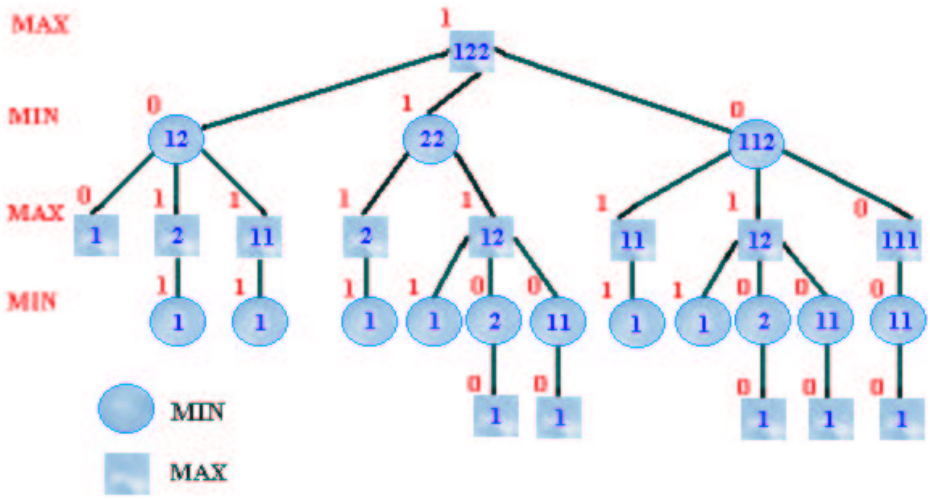

Figura 1: La procedura MiniMax nel gioco del Nim a partire dallo stato 122.

ciascuna pila può contenere uno o più dischi. Quando un giocatore deve muovere, può scegliere arbitrariamente una qualunque pila non vuota, e togliere dalla pila un qualunque numero positivo di dischi (al massimo, il numero di dischi presenti in quella pila, nel qual caso la pila si svuota completamente). I giocatori si alternano al gioco, muovendo a turno; assumiamo che il primo giocatore a muovere sia il protagonista. Il gioco termina quando tutte le pile si svuotano, e perde il giocatore che prende l'ultimo piatto (cioè, l'ultimo giocatore a muovere).

Rappresentiamo uno stato di gioco mediante una sequenza di interi positivi che corrispondono al numero di dischi presenti sulle pile, ordinata in ordine non decrescente. In Figura 1 mostriamo l'albero di gioco assumendo che lo stato iniziale sia 122. Ogni nodo nell'albero corrisponde a una possibile configurazione raggiungibile da quella iniziale: lo stato nella configurazione è indicato dalla stringa contenuta nel nodo, e il giocatore di turno è indicato dalla forma del nodo (i nodi quadrati corrispondono alle configurazioni in cui il protagonista deve muovere, quelli rotondi corrispondono alle configurazioni in cui è l'avversario a muovere). I numeri che compaiono in alto a sinistra su ciascun nodo rappresentano i valori di $p(c)$, calcolati dalla procedura MiniMax. Le configurazioni finali sono quelle in cui c'è una sola pila con un solo disco; il giocatore di turno viene sconfitto, quindi il punteggio assegnato dalla procedura MiniMax sarà 0 se il turno è del protagonista, e sarà 1 se il turno è dell'avversario[2].

[2] Poiché in questo gioco non è possibile il pareggio, scegliamo di valutare 1 l'esito di vittoria del protagonista, e 0 l'esito di sconfitta.

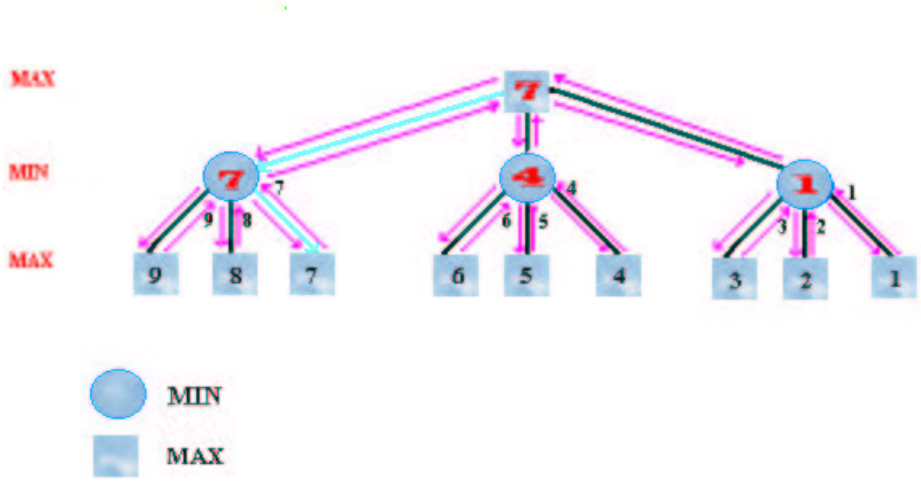

Figura 2: La procedura MiniMax.

Come si vede dall'albero, la prima mossa che conviene che il protagonista faccia per raggiungere l'ottimo consiste nel prendere il disco dalla pila 1. La Figura 2 mostra un altro esempio di applicazione della procedura MiniMax a un albero di gioco: come al solito il punteggio assegnato alle configurazioni in cui il turno è del protagonista viene calcolato come un massimo, mentre il punteggio assegnato alle configurazioni in cui il turno è dell'avversario viene calcolato come un minimo.

Il codice. Il seguente frammento di codice rappresenta la procedura MiniMax che calcola e restituisce il punteggio $p(c)$ della configurazione c; il secondo parametro, muovi indica se deve anche essere emessa la mossa da effettuare (ha senso solo se c.t==PROTAGONISTA.

```
int p( configurazione c, char muovi ) {
  if ( c finale )
    return esito(c);
    /* P.es. 1 se in c vince PROTAGONISTA, 0 se patta, -1 se
       vince AVVERSARIO */
  siano c[1],...,c[k] le configurazioni successive a c;
  if ( c.t == PROTAGONISTA ) {
    m = max( p(c[1],0), ..., p(c[k],0) );
    if ( muovi )
       fai la mossa che corrisponde all'indice i per cui
```

```
        si realizza il massimo (cioe', p(c[i],0)=m)
    }
    else
      m = min( p(c[1],0), ..., p(c[k],0) );
    return m;
  }
```

La procedura, così come è indicata, ricalcola il punteggio delle varie configurazioni ogni volta che viene invocata. Potrebbe essere utile, se il numero di configurazioni è "piccolo", realizzare una forma di caching, p.es. tenendo i valori calcolati in un array.

Estensione al caso di giochi con memoria. Non è difficile modificare la procedura precedente nel caso che il gioco non sia privo di memoria (cioè, l'esito non dipenda solo dalla configurazione finale raggiunta, ma anche dalla sequenza di mosse effettuate). È sufficiente passare un terzo parametro alla procedura che rappresenta la sequenza di mosse effettuate fino a quel momento (o un qualche valore che la riassuma e che sia sufficiente a determinare, l'esito del gioco una volta arrivati alla configurazione finale).

Estensione al caso di giochi con cicli. Se il gioco non è aciclico, la procedura MiniMax deve essere modificata intanto per evitare che si incarti su cicli infiniti. Il modo più semplice per farlo consiste nel mantenere un array globale che indica quali configurazioni si sono già visitate. Più precisamente, questo array dovrebbe contenere:

- per le configurazioni non visitate, uno speciale valore NONVISITED;
- per le configurazioni in corso di visita, il valore VISITING (questo valore viene messo nell'array, nella posizione corrispondente alla configurazione c, non appena si entra nella procedura MiniMax);
- per le configurazioni visitate, il valore $p(c)$ (questo valore viene messo nell'array alla fine della procedura di visita, prima del **return**).

Ora, la procedura non considera (quando calcola il minimo o il massimo) le configurazioni che compaiono nell'array come VISITING: in questo modo la procedura ignora i cicli, non considerandoli nel calcolo del minimo o del massimo. Si noti che, se tutte le configurazioni successive dovessero dare luogo a una situazione del genere, la procedura dovrebbe restituire $-\infty$ se sta calcolando un massimo, e $+\infty$ se sta calcolando un minimo, per indicare che il valore ottenuto per quella configurazione è ininfluente (se si arriva in quella configurazione, ogni sequenza di mosse porta il gioco a non terminare!).

Il caso dei giochi ciclici con memoria. La tecnica spiegata serve per risolvere il caso dei giochi ciclici a patto che essi siano senza memoria, oppure a patto che i cicli non influenzino l'esito del gioco. Se questo non è vero, il gioco stesso è mal posto. Supponete, infatti, che ad esempio in un gioco vinca il giocatore che fa più mosse, e supponete che il gioco contenga cicli: ovviamente converrebbe (ad entrambi i giocatori) tentare di arrivare ad un ciclo e percorrerlo infinite volte!

3 La strategia MiniMax con lookahead limitato

La strategia MiniMax richiede, per ogni mossa, l'analisi di tutte le possibili configurazioni future (se si tengono i valori calcolati in una cache, ciò avviene solo per la prima mossa): questo è possibile solo a patto che il numero di configurazioni possibili sia piccolo. Se così non è (questo avviene nella maggior parte dei giochi reali, p.es. nel gioco degli scacchi) occorre adottare una strategia "meno dispendiosa", anche se non esatta.

Una variante che si adotta frequentemente è quella cosiddetta del "lookahead limitato". In pratica, a partire da una certa configurazione, si considerano le configurazioni future solo fino a un certo livello di profondità (detto *lookahead*): se si raggiungono configurazioni con quel livello di profondità e che non sono finali, se ne valuta il punteggio usando una qualche euristica che tenti di stabilire quanto la configurazione è vantaggiosa per il protagonista.

Ecco una possibile variazione della procedura MiniMax con lookahead limitato:

```
int p( configurazione c, char muovi, int livello ) {
  if ( c finale )
    return esito(c);
    /* P.es. 1 se in c vince PROTAGONISTA, 0 se patta, -1 se
       vince AVVERSARIO */
  if ( livello == MAX_LIVELLO )
    return euristica(c);
    /* Tenta di valutare euristicamente quanto promettente
       sia la configurazione c */
  siano c[1],...,c[k] le configurazioni successive a c;
  if ( c.t == PROTAGONISTA ) {
    m = max( p(c[1],0,livello+1), ..., p(c[k],0,livello+1) );
    if ( muovi )
       fai la mossa che corrisponde all'indice i per cui
       si realizza il massimo (cioe', p(c[i],0)=m)
  }
  else
```

```
    m = min( p(c[1],0), ..., p(c[k],0) );
  return m;
}
```

4 La strategia MiniMax con potatura alfa/beta

La *potatura alfa/beta* è una tecnica che consente di ridurre, in qualche caso in modo consistente, il numero di configurazioni da analizzare durante la procedura MiniMax. L'idea è piuttosto semplice: il calcolo del punteggio $p(c)$ associato ad una configurazione, nella procedura MiniMax, consiste nel calcolo di un minimo o di un massimo (a seconda di $c.t$) dei punteggi delle configurazioni successive a c.

Ora, supponiamo di invocare la funzione p con altri due parametri $\alpha \leq \beta$, con il seguente significato: $p(c, \alpha, \beta)$ deve restituire il valore $p(c)$, se esso è compreso nell'intervallo $[\alpha, \beta]$, ma non siamo interessati al suo vero valore se esso è fuori da questo range: in questo caso, ci accontentiamo di ottenere un qualunque valore $\leq \alpha$ (se $p(c)$ si trova a sinistra dell'intervallo, cioè $p(c) \leq \alpha$) e di ottenere un qualunque valore $\geq \alpha$ (se $p(c)$ si trova a destra dell'intervallo, cioè $p(c) \geq \beta$).

Supponiamo di dover valutare $p(c, \alpha, \beta)$, e che c comporti il calcolo di un massimo (cioè, che il giocatore $c.t$ sia il protagonista). Siano $c_1, \ldots, c_k$ le configurazioni successive. Innanzitutto valutiamo $v_1 = p(c_1, \alpha, \beta)$.

Ora, se $v_1 \geq \beta$ sarebbe inutile continuare a valutare le configurazioni successive, perché il massimo ottenuto alla fine sarebbe sicuramente maggiore o uguale a v_1 (e quindi a β): quindi, ci fermiamo e restituiamo v_1. Supponiamo, invece, che $v_1 \leq \beta$; a questo punto dovremmo calcolare il valore di v_2, ma non ci interessano valori minori di v_1 (perché questo è il candidato massimo corrente): quindi chiameremo $p(c_2, \max(\alpha, v_1), \beta)$, e valuteremo il risultato come prima.

Analogo discorso vale per le configurazioni in cui dobbiamo calcolare un minimo. Ecco il codice per il calcolo di p con potatura alfa/beta:

```
int p( configurazione c, int alfa, int beta, char muovi ) {
  if ( c finale )
    return esito(c);
    /* P.es. 1 se in c vince PROTAGONISTA, 0 se patta, -1 se
       vince AVVERSARIO */
  siano c[1],...,c[k] le configurazioni successive a c;
  if ( c.t == PROTAGONISTA ) {
    for ( i = 1; i <=k; i++ ) {
```

```
        v = p(c[i], alfa, beta, 0);
        alfa = max( alfa, v );
        if ( alfa >= beta ) break;
      }
      if ( muovi )
         fai la mossa che corrisponde all'indice i per cui
         si realizza il massimo
      return alfa;
    }
    else {
      for ( i = 1; i <=k; i++ ) {
        v = p(c[i], alfa, beta, 0);
        beta = min( beta, v );
        if ( alfa >= beta ) break;
      }
      return beta;
    }
  }
```

Naturalmente, all'inizio la funzione dovrà essere invocata con $\alpha = -\infty$ e $\beta = +\infty$. È importante notare che l'efficacia della potatura alfa/beta dipende dall'ordine con cui si valutano le configurazioni successive: per sfrondare il maggior numero possibile di configurazioni conviene visitare per prime le configurazioni successive che hanno maggior possibilità di essere "vincenti" per il giocatore di turno. Per questo motivo la potatura alfa/beta viene spesso implementata usando una funzione euristica che decide in che ordine visitare le configurazioni successive.

La Figura 3 mostra come risulta modificata la valutazione della procedura MiniMax sull'albero di gioco di Figura 2 nel caso che si applichi la potatura alfa/beta.

Inizialmente dobbiamo valutare un massimo (radice) e tutto ciò che sappiamo è che tale massimo è $\geq -\infty$. Dopo aver valutato (completamente) il primo figlio, scopriamo che il massimo è ≥ 7. Quando passiamo a valutare il secondo figlio della radice (quello etichettato con 6) dovremo valutare un minimo, e passeremo alla procedura $\alpha = 7$ (non ci interessano valori minori di 7) e $\beta = +\infty$. La procedura ora deve calcolare un minimo, ma già al primo passo (cioè, quando valuta il primo dei tre figli) scopre che il minimo sarà ≤ 6: è inutile proseguire a valutare gli altri figli, che potrebbero solo ridurre ulteriormente il minimo: la procedura restituisce 6, valore che comunque viene ignorato dalla radice (il massimo corrente per la radice è 7). Una situazione analoga avviene all'atto della valutazione del terzo e ultimo figlio della radice.

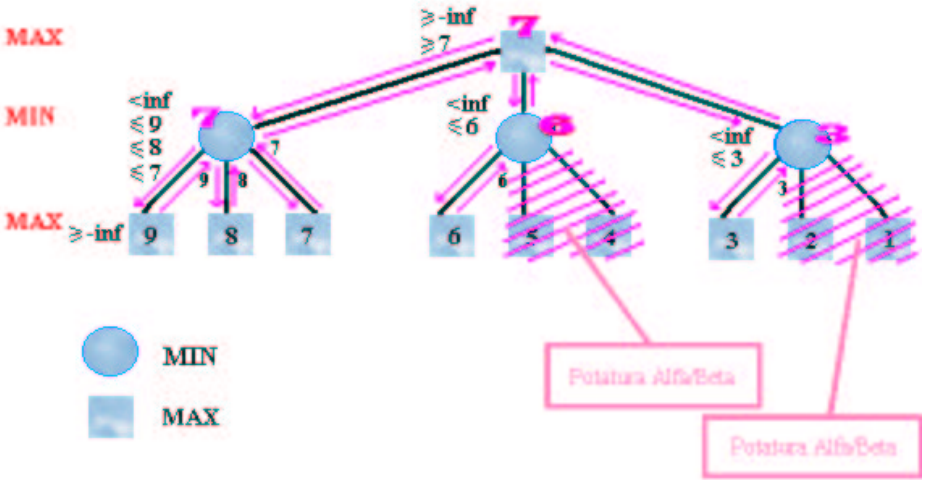

Figura 3: Potatura alfa/beta dell'albero di Figura 2.