

# I giochi con avversario

Maria Simi  
a.a. 2011/2012

## I giochi con avversario

- Regole semplici e formalizzabili
- Deterministici, due giocatori, turni alterni, a somma zero, informazione perfetta (ambiente accessibile)
- ambiente multi-agente competitivo: la presenza dell'avversario rende l'ambiente *strategico*  $\Rightarrow$  piú difficile rispetto ai problemi di ricerca visti fin'ora
- complessità e vincoli di tempo reale: si può solo cercare di fare la mossa migliore nel tempo disponibile
- $\Rightarrow$  i giochi sono un po' piú simili ai problemi reali

## Sommario

- La soluzione teorica: come si sceglie la mossa migliore in un gioco con uno spazio di ricerca limitato
- Estensione a giochi piú complessi, in cui non è possibile una esplorazione esaustiva
- Tecniche di ottimizzazione della ricerca
- Giochi con casualità

## Giochi come problemi di ricerca

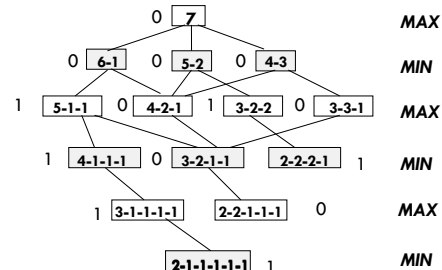
- Stati:** configurazioni del gioco
  - Giocatore(s): a chi tocca muovere in  $s$
- Stato iniziale:** configurazione iniziale
- Azioni(s):** le mosse legali in  $s$
- Risultato(s, a):** lo stato risultante da una mossa
- Test-terminazione(s):** determina la fine del gioco
- Utilità(s, p):** funzione di *utilità* (o *pay-off*), valore numerico che valuta gli stati terminali del gioco per  $p$   
Es.  $1 | -1 | 0$ , conteggio punti, ... somma costante

## Il gioco del NIM

- Regole:**
  - un certo numero di fiammiferi, 7, allineati sul tavolo
  - una mossa consiste nel separare una fila in due file aventi un numero diverso di fiammiferi.
  - Perde il giocatore che non può piú muovere
- Stati:**  $||| ||| | \Rightarrow 6-1$
- Mosse:**  $6-1 \begin{cases} \rightarrow 5-1-1 \\ \rightarrow 4-2-1 = 2-4-1 \end{cases}$

## Il gioco del NIM

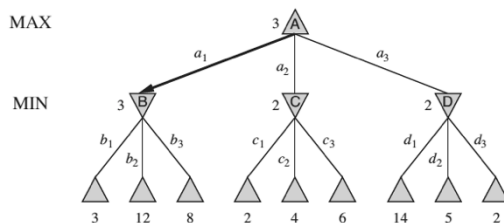
Il computer si chiama MAX, MIN è l'avversario. Quale mossa conviene fare a MAX nello stato iniziale?



## Valore Minimax

$$\text{Minimax}(n) = \begin{cases} \text{Utilità}(s, \text{MAX}) & \text{se Test-terminale}(s) \\ \max_{a \in \text{Azioni}(s)} \text{Minimax}(\text{Risultato}(s, a)) & \text{se Giocatore}(s) = \text{MAX} \\ \min_{a \in \text{Azioni}(s)} \text{Minimax}(\text{Risultato}(s, a)) & \text{se Giocatore}(s) = \text{MIN} \end{cases}$$

## Esempio



Albero di gioco profondo una mossa, due strati o ply

## L'algoritmo MIN-MAX

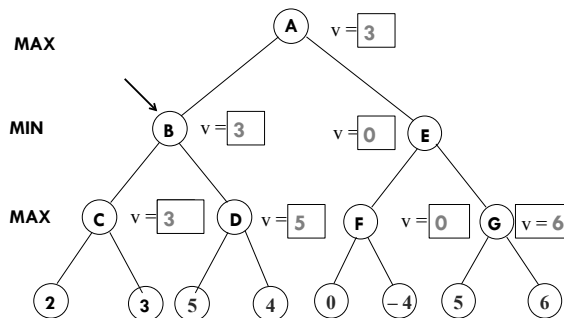
```

function Decisione-Minimax(stato) returns un'azione
    return argmaxa ∈ Azioni(s) ValoreMin(Risultato(stato, a))

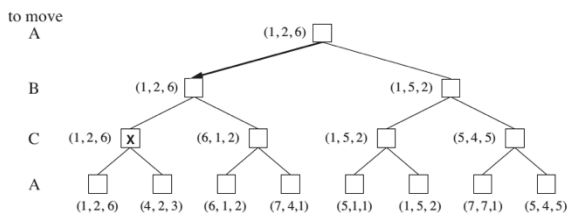
function Valore-Max(stato) returns un valore di utilità
    if TestTerminazione(stato) then return Utilità(stato)
    v = -∞
    for each a in Azioni(stato) do
        v = Max(v, ValoreMin(Risultato(stato, a)))
    return v

function Valore-Min(stato) returns un valore di utilità
    if TestTerminazione(stato) then return Utilità(stato)
    v = +∞
    for each a in Azioni(stato) do
        v = Min(v, ValoreMax(Risultato(stato, a)))
    return v
    
```

## Min-max: algoritmo in azione



## Giochi multiplayer



- $(v_a=1, v_b=2, v_c=6)$  valutazioni per A, B, C
- Il valore backed-up in x è il vettore migliore per C

## Min-max con decisione imperfetta

- In casi più complessi occorre una funzione di valutazione euristica dello stato  $Eval(s)$ .
- Strategia:** guardare avanti d mosse
  - Si espande l'albero di ricerca un certo numero di livelli d (compatibile col tempo e lo spazio disponibili)
  - si valutano gli stati ottenuti e si propaga indietro il risultato con la regola del MAX e MIN
- Algoritmo MIN-MAX come prima ma ...**

```

if TestTerminazione(stato) then return Utilità(stato)
→ if TestTaglio(stato, d) then return Eval(stato)
            
```

## Min-Max con euristica

Se  $d$  è la profondità massima consentita ...

$H\text{-Minimax}(s, d) =$

$Eval(s)$  se  $TestTaglio(s, d)$

$\max_{a \in Azioni(s)} H\text{-Minimax}(Risultato(s, a), d+1)$

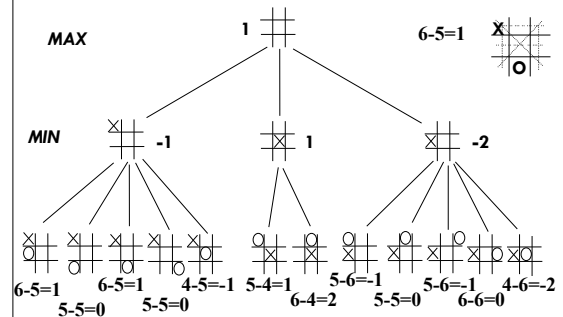
se  $Giocatore(s)=MAX$

$\min_{a \in Azioni(s)} H\text{-Minimax}(Risultato(s, a), d+1)$

se  $Giocatore(s)=MIN$

## Il filetto

$f(n) = X(n) - O(n)$   
 $X(n)$  righe aperte per X  
 $O(n)$  righe aperte per O



## La funzione di valutazione

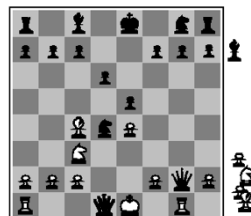
- La funzione di valutazione  $f$  è una stima della utilità attesa a partire da una certa posizione nel gioco.
  - es. il vantaggio in materiale negli scacchi: pedoni 1, cavallo o alfiere 3, torre 5, regina 9 ...
- Determinante la qualità della  $f$ :
  - deve essere consistente con l'utilità se applicata a stati terminali del gioco (indurre lo stesso ordinamento).
  - deve essere efficiente da calcolare;
  - deve riflettere le probabilità effettive di vittoria (maggiore valore ad A piuttosto che a B se da A ci sono più probabilità di vittoria che da B)

## Esempio

- Per gli scacchi si potrebbe pensare di valutare caratteristiche diverse dello stato:
  - Valore del materiale (pedone 1, cavallo o alfiere 3, torre 5, regina 9 ...)
  - Buona disposizione dei pedoni
  - Protezione del re
- Funzione lineare pesata
  - $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
  - o anche combinazioni non lineari di caratteristiche

## Problemi con MIN-MAX: stati non quiescenti

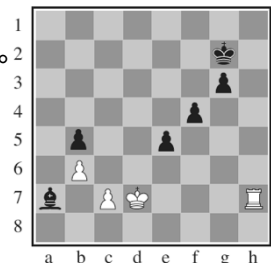
- Stati non quiescenti:** l'esplorazione fino ad un livello può mostrare una situazione molto vantaggiosa: alla mossa successiva la regina nera viene mangiata.
- Soluzione:** applicare la valutazione a stati *quiescenti* stati in cui la funzione di valutazione non è soggetta a mutamenti repentini (ricerca di quiescenza)



## Problemi con MIN-MAX: effetto orizzonte

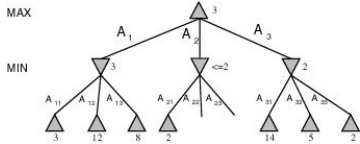
**Effetto orizzonte:** può succedere che vengano privilegiate mosse diversive che hanno il solo effetto di spingere il problema oltre l'orizzonte

L'alfiere in a7, catturabile in 3 mosse dalla torre, è spacciato. Mettere il re bianco sotto scacco con il pedone in e5 e poi con quello in f4 evita il problema temporaneamente, ma è un sacrificio inutile di pedoni.



## Potatura alfa-beta: l'idea

- Tecnica di *potatura* per ridurre l'esplorazione dello spazio di ricerca in algoritmi MIN-MAX.



$$\begin{aligned} \text{MINMAX}(\text{radice}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{con } z \leq 2 \\ &= 3 \end{aligned}$$

## Potatura alfa-beta: implementazione

- Si va avanti in profondità fino al livello desiderato e propagando indietro i valori si decide se si può abbandonare l'esplorazione nel sotto-albero.

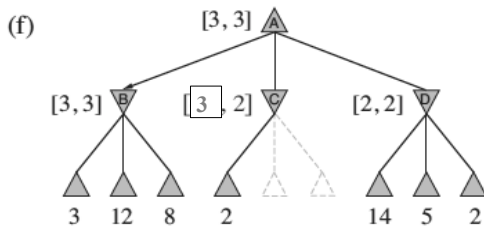
- MaxValue e MinValue vengono invocate con due valori di riferimento:  $\alpha$  (inizialmente  $-\infty$ ) e  $\beta$  (inizialmente  $+\infty$ ): rappresentano rispettivamente la migliore alternativa per MAX e per MIN fino a quel momento.

- I tagli avvengono quando nel propagare indietro:

$$v \geq \beta \quad \text{per i nodi MAX (taglio } \alpha)$$

$$v \leq \alpha \quad \text{per i nodi MIN (taglio } \beta)$$

## Alfa-beta: funzionamento



## L'algoritmo Alfa-Beta: max

**function** Alfa-Beta(stato) **returns** un'azione

$v = \text{ValoreMax}(\text{stato}, -\infty, +\infty)$

**return** l'azione in Azioni(stato, a) con valore v

**function** Valore-Max(stato,  $\alpha$ ,  $\beta$ ) **returns** un valore di utilità

**if** TestTerminazione(stato) **then return** Utilità(stato)

$v = -\infty$

**for each** a in Azioni(stato) **do**

$v = \text{Max}(v, \text{ValoreMin}(\text{Risultato}(\text{stato}, a), \alpha, \beta))$

**if**  $v \geq \beta$  **then return** v ← taglio  $\alpha$

$\alpha = \text{Max}(\alpha, v)$

**return** v

## L'algoritmo Alfa-Beta: min

**function** Valore-Min(stato,  $\alpha$ ,  $\beta$ ) **returns** un valore di utilità

**if** TestTerminazione(stato) **then return** Utilità(stato)

$v = +\infty$

**for each** a in Azioni(stato) **do**

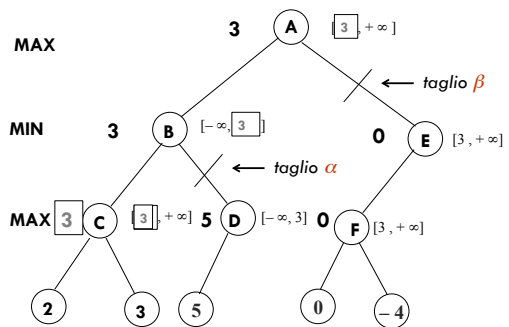
$v = \text{Min}(v, \text{ValoreMax}(\text{Risultato}(\text{stato}, a), \alpha, \beta))$

**if**  $v \leq \alpha$  **then return** v ← taglio  $\beta$

$\beta = \text{Min}(\beta, v)$

**return** v

## Potatura alfa-beta: un esempio



## Ordinamento delle mosse

- Potatura ottimale:
  - Per nodi MAX sono generate prima le mosse migliori
  - Per i nodi MIN sono generate prima le mosse peggiori per MAX (migliori per MIN)
- Complessità:  $O(b^{m/2})$  anziché  $O(b^m)$
- Alfa-Beta può arrivare a profondità doppia rispetto a Min-Max!
- Ma come avvicinarsi all'ordinamento ottimale?

## Ordinamento dinamico

1. Usando approfondimento iterativo si possono scoprire ad una iterazione informazioni utili per l'ordinamento, da usare in una successiva iterazione (mosse killer).
2. Tabella delle trasposizioni: per ogni stato incontrato si memorizza la sua valutazione
  - Situazione tipica:  $[a_1, b_1, a_2, b_2]$  e  $[a_1, b_2, a_2, b_1]$  portano nello stesso stato

## Altri miglioramenti

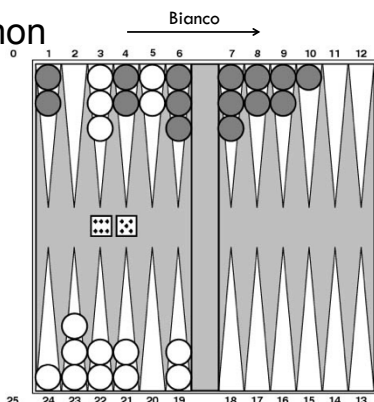
1. Potatura in avanti: esplorare solo alcune mosse ritenute promettenti e tagliare le altre
  - Beam search
  - Tagli probabilistici (basati su esperienza). Miglioramenti notevoli in Logistello [Buro]
2. Database di mosse di apertura e chiusura
  - Nelle prime fasi ci sono poche mosse sensate e ben studiate, inutile esplorarle tutte
  - Per le fasi finali il computer può esplorare off-line in maniera esaustiva e ricordarsi le migliori chiusure (già esplorate tutte le chiusure con 5 e 6 pezzi ... )

## Giochi stocastici

- Sono ad esempio i giochi in cui è previsto un lancio di dadi
- Ancora più reale: la realtà è spesso imprevedibile non solo complessa.
- *Backgammon*: ad ogni turno il giocatore deve tirare due dadi per decidere quali mosse sono lecite.

## Backgammon

Lancio dadi 6-5,  
4 mosse legali  
per il bianco:  
(5-10, 5-11)  
(5-11, 19-24)  
(5-10, 10-16)  
(5-11, 11-16)

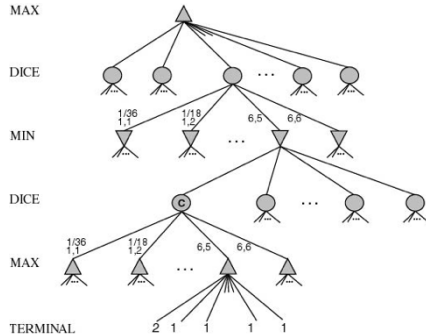


## Min-max con nodi di casualità

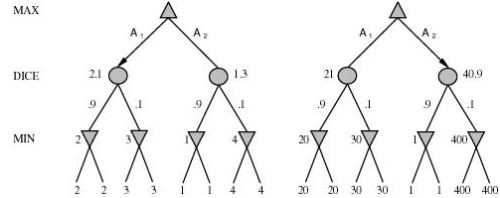
- Non si sa quali saranno le mosse legali dell'avversario: accanto ai nodi di scelta dobbiamo introdurre i nodi *casualità* (chance)
- Nel calcolare il valore dei nodi MAX e MIN adesso dobbiamo tenere conto delle probabilità dell'esperimento casuale
- Si devono calcolare il valore massimo e minimo attesi

## Min-max con nodi *chance* (esempio)

21 lanci diversi: i lanci doppi con  $p=1/36$ , gli altri con  $p=1/18$



## Min-max con nodi *chance* (cont.)

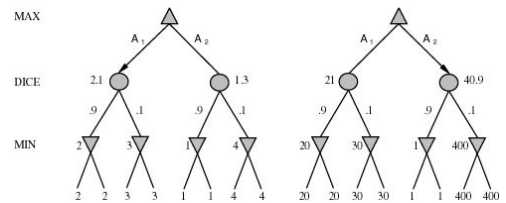


MIN con probabilità 0.9 farà 2 e con probabilità 0.1 farà 3 ...  
 $0.9 \times 2 + 0.1 \times 3 = 2.1$   
 $0.9 \times 1 + 0.1 \times 4 = 1.3$   
 La mossa migliore è la prima.

## Expectiminmax: la regola

$$\text{Expectiminmax}(n) = \begin{cases} \text{Eval}(n) & \text{se } n \text{ è uno stato sul livello di taglio} \\ \max_{a \in \text{Azioni}(s)} \text{Expectiminmax}(\text{Risultato}(s, a)) & \text{se } n \text{ è un nodo MAX} \\ \min_{a \in \text{Azioni}(s)} \text{Expectiminmax}(\text{Risultato}(s, a)) & \text{se } n \text{ è un nodo MIN} \\ \sum_{a \in \text{Azioni}(s)} P(r) \text{Expectiminmax}(\text{Risultato}(s, a)) & \text{se } n \text{ è un nodo casualità} \\ & r \text{ un esito dell'esperimento casuale} \end{cases}$$

## Min-max con nodi *chance*: nota



- Non è sufficiente l'ordinamento relativo dei successori; nel secondo caso la mossa scelta è diversa.
- La funzione di valutazione deve essere una trasformazione lineare positiva della probabilità di vincere a partire da una certa posizione

## Giochi parzialmente osservabili

- Giochi parzialmente osservabili *deterministici*
  - Le mosse sono deterministiche ma non si conoscono le risposte dell'avversario
  - Es. Kriegspiel
- Giochi parzialmente osservabili *stocastici*
  - Le carte distribuite a caso in molti giochi di carte
  - Es. Bridge, whist, peppa, briscola ...

## Lo stato dell'arte

- Stato dell'arte in
  - Scacchi (DeepBlue, Hydra, Rybka-campione mondiale nel 2008-09)
  - Dama (Chinook gioca in maniera perfetta)
  - Otello (Logistello, campione dal 1997)
  - Backgammon, Go, Bridge, Scarabeo ...
- Gli scacchi stanno all'IA come la Formula 1 sta all'industria automobilista