I giochi con avversario

Alessio Micheli

a.a. 2015/2016

Credits: Maria Simi

Russell-Norvig

I giochi con avversario

- Cavallo di battaglia storico dell'AI, natura astratta
  - Regole semplici e formalizzabili
  - deterministici, ambiente accessibile
- due giocatori, turni alterni, *a somma zero*, informazione perfetta [e.g. 1 vince, avversario perde]
- ambiente multi-agente competitivo:
  - la presenza dell'avversario rende l'ambiente *strategico* ⇒ più difficile rispetto ai problemi di ricerca visti finora
  - Interesse per quelli «difficili»: *scacchi* b medio 35, 50 mosse per giocatore → $35^{100} = 10^{154}$ nodi nell'albero di ricerca (10^{40} distinti)
- Impone attenzione all'efficienza: vincoli di tempo reale:
  - si può solo cercare di fare la mossa migliore nel tempo disponibile
  - il più efficiente…. Vince !!!

Sommario

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

Giochi come problemi di ricerca

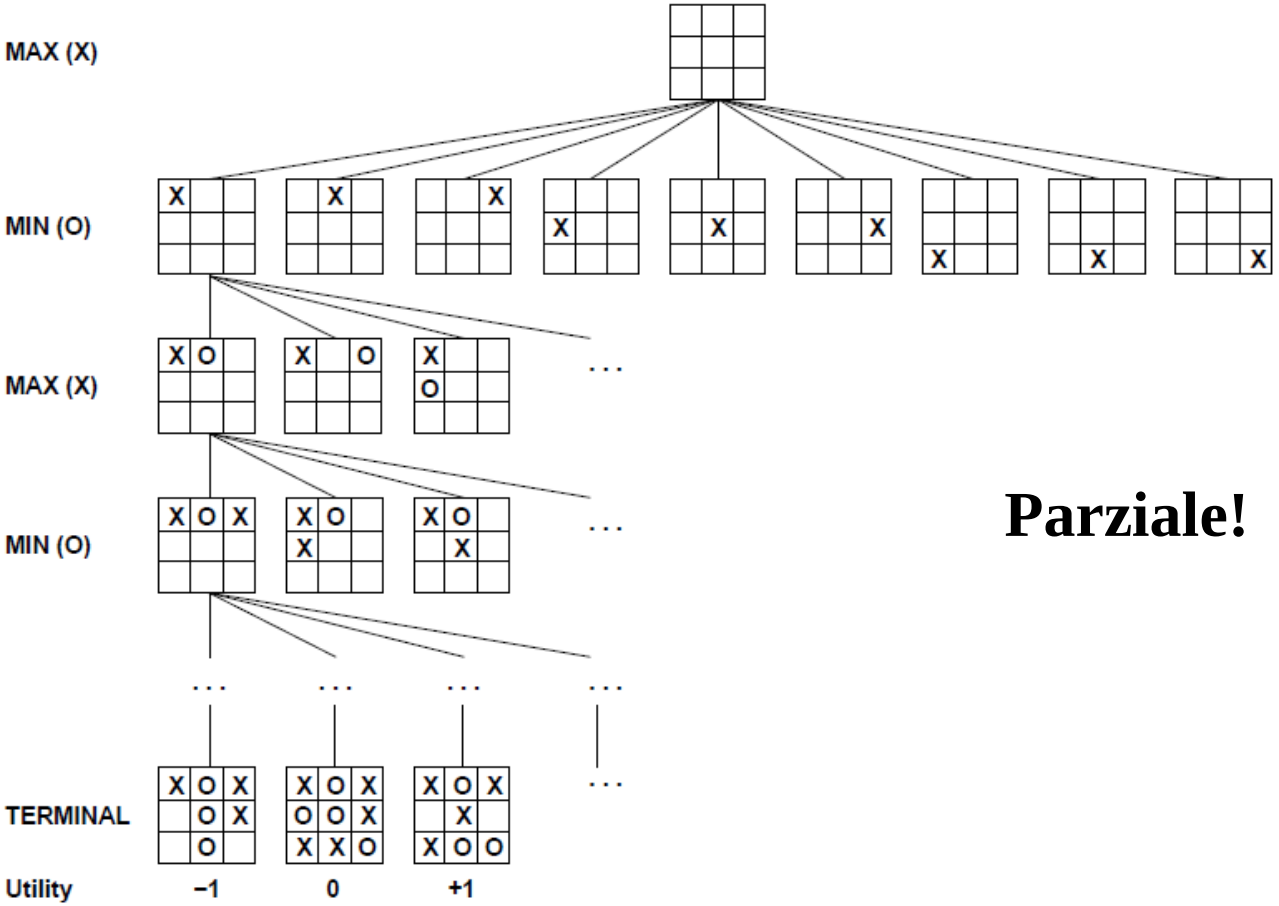
- *Stati*: configurazioni del gioco
  - Giocatore(s): a chi tocca muovere in s
- *Stato iniziale*: configurazione iniziale
- *Azioni*(*s*): le mosse legali in *s*
- Risultato(*s*, *a*): lo stato risultante da una mossa
- *Test-terminazione*(*s*): determina la fine del gioco
- Utilità(*s*, *p*): funzione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

  Es. 1|-1|0, conteggio punti, … somma costante

Una mossa a testa

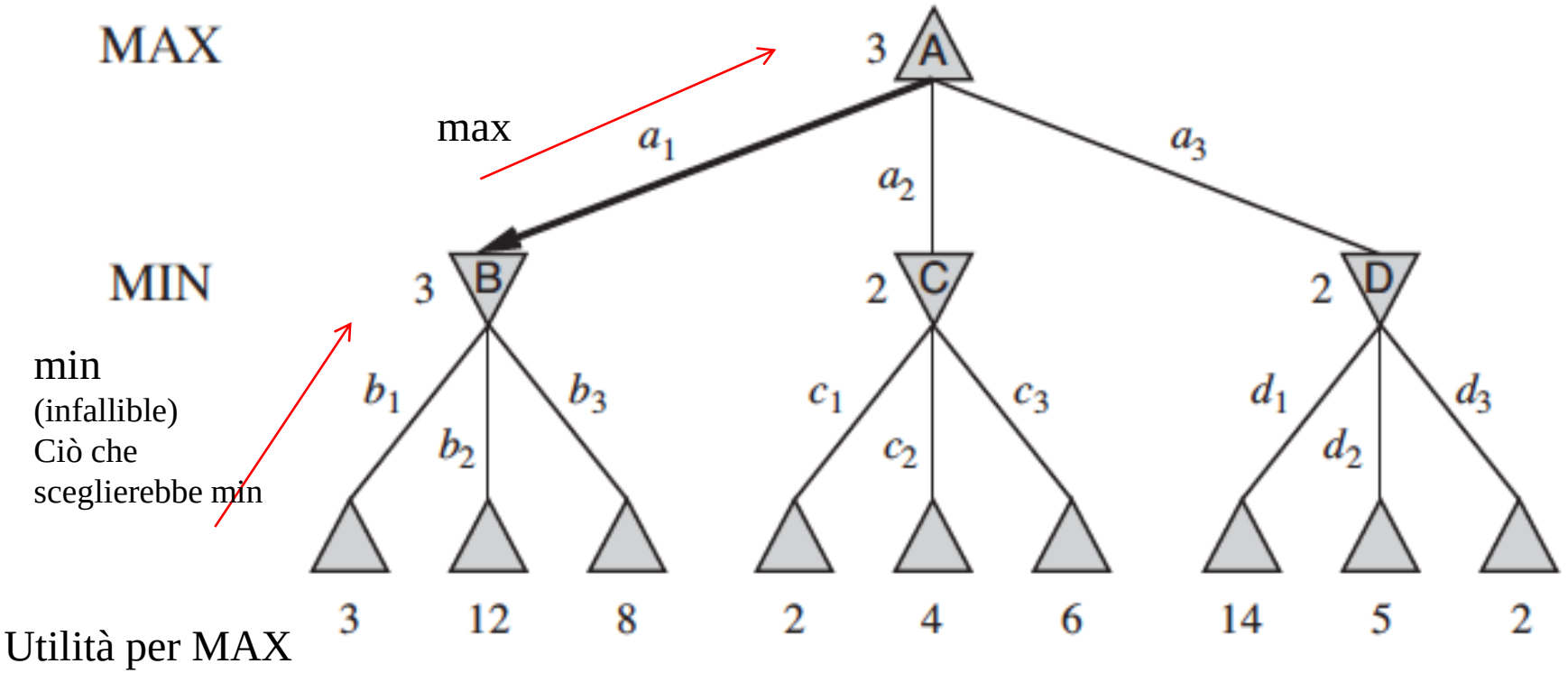
- Strategia contingente
- Dovremmo considerare le mosse possibili per "MAX"
- Poi le mosse possibili dell'avversario "MIN"
- Poi la risposta per *tutte* le possibili mosse dell'avversario "MIN"
- Ricorda l'albero AND-OR
- MIN assunto infallibile (che giochi in modo ottimo)

Tris/Filetto (Tic-tac-toe): albero di ricerca



Parziale!

Gioco Ply: etichettatura con utilità minimax



Albero di gioco profondo una mossa, due strati o *ply*

MAX seguirà a_1 (minimax più alta), (MIN b_1)

Caso pessimo (MIN infallibile) trovo mossa per cui qualunque mossa di MIN mi fa avere massima utilità.

Valore Minimax

- Etichettatura di albero con utilità

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

stato

L'algoritmo MIN-MAX (ricorsivo → DF)

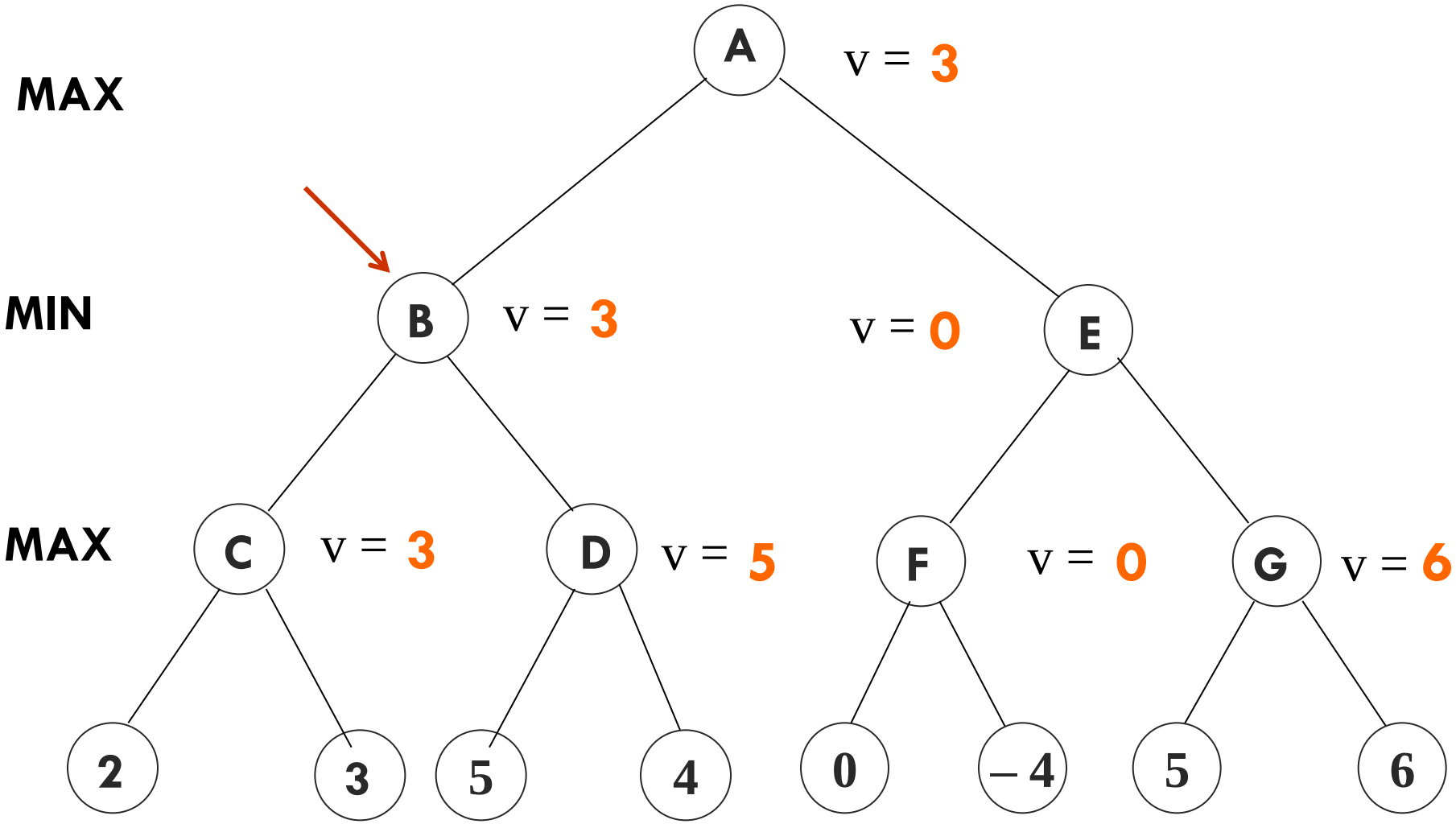
```
function Decisione-Minimax(stato) returns un'azione
  return argmax a ∈ Azioni(s) ValoreMin(Risultato(stato, a))

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
```

Valori minimax backed-up dalle foglie
Costo?

Min-max: algoritmo in azione

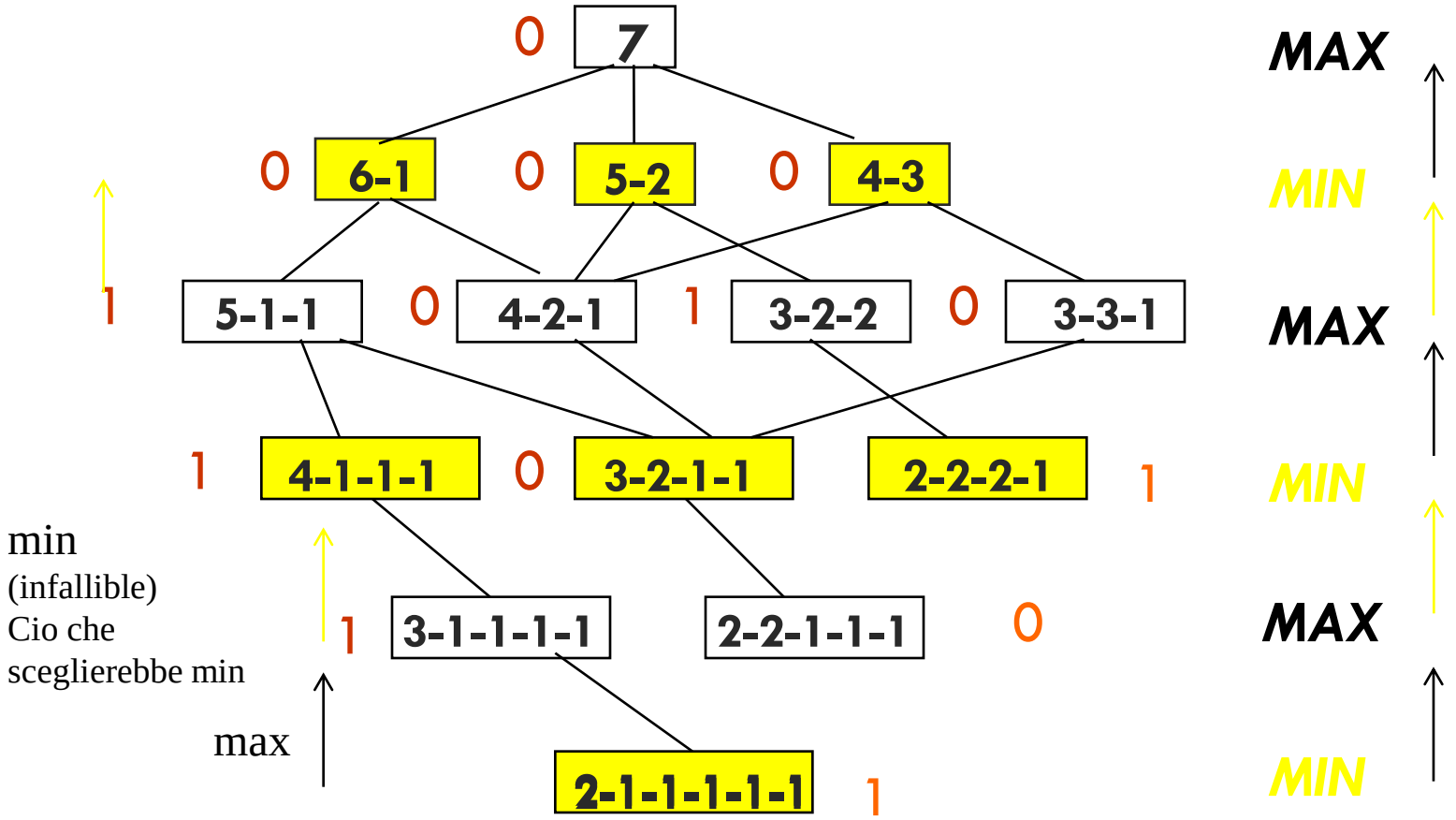


Il gioco del NIM

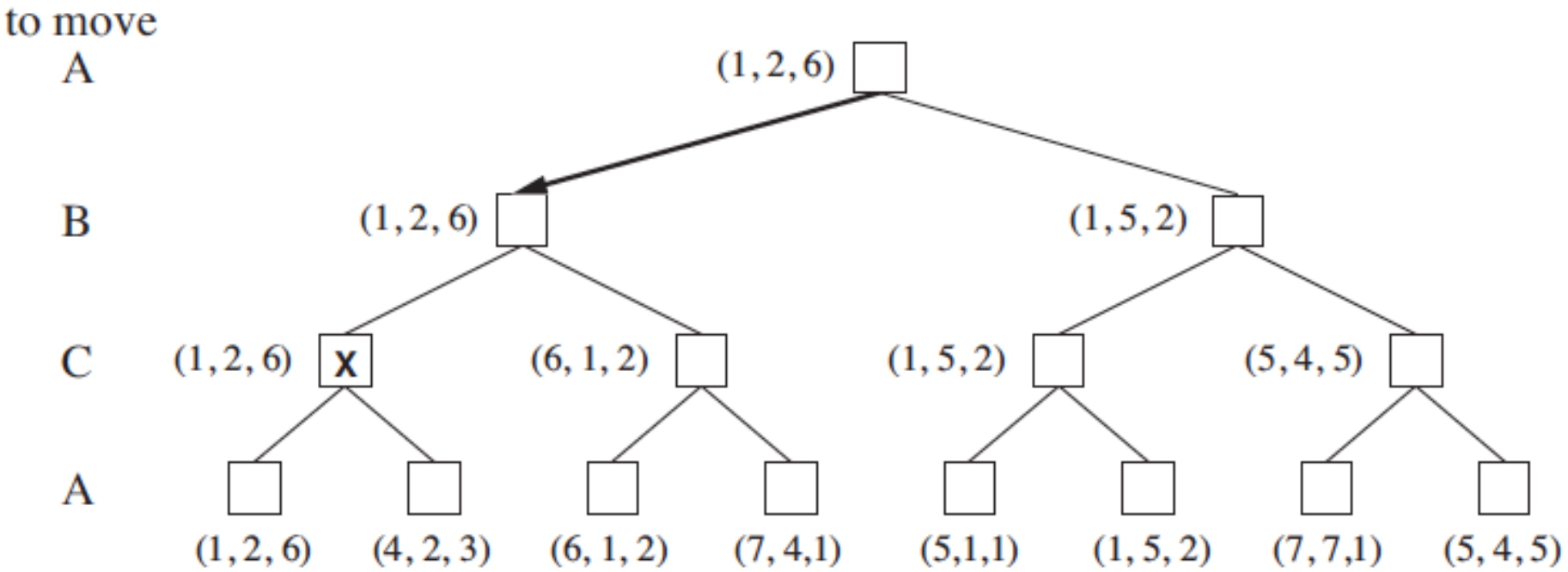
- Regole:
  - un certo numero di fiammiferi, 7, allineati sul tavolo
  - una mossa consiste nel separare una fila in due file aventi un numero diverso di fiammiferi (ogni mossa).
  - Perde il giocatore che non può più muovere
- Stati: |||||| | $\Rightarrow$ 6-1
- Mosse: 6-1 — 5-1-1
  4-2-1 = 2-4-1

Il gioco del NIM

La prima mossa tocca a MAX, MIN è l'avversario. Quale mossa conviene fare a MAX nello stato iniziale?



Giochi multiplayer



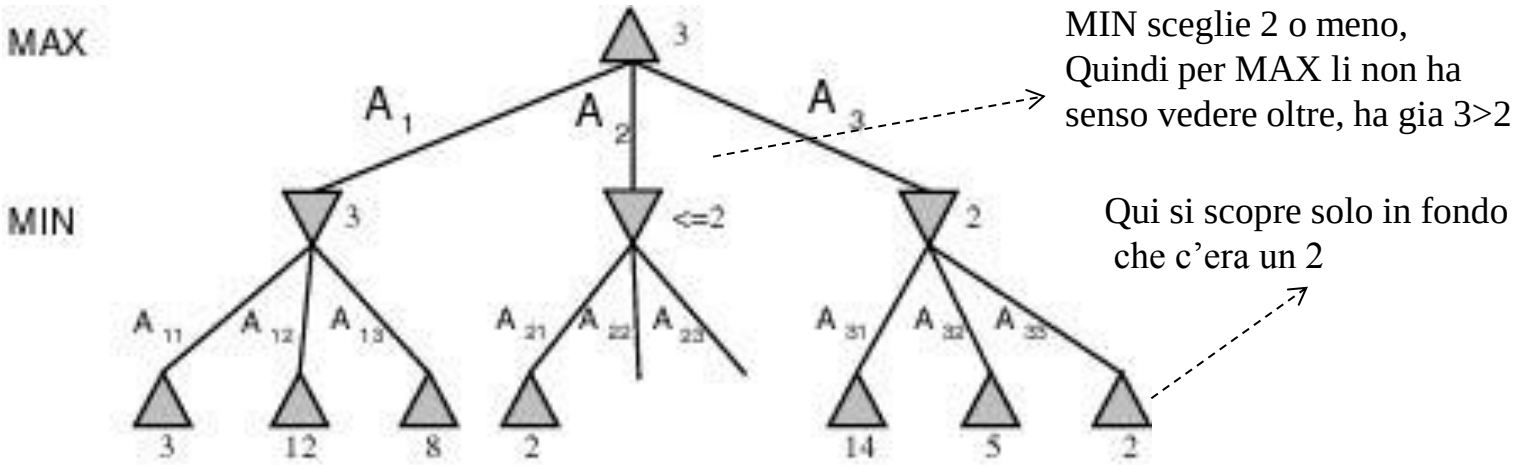
1. $(v_a=1, v_b=2, v_c=6)$ valutazioni per A, B, C
2. Il valore backed-up in x è il vettore migliore per C

Costo

- Alto: come DF $O(b^m)$
- Schacchi: 35^{100} → infeasible!
- Ma dobbiamo necessarimente esplorare ogni cammino?
- NO, esiste modo di (intanto) dimezzare la ricerca pur mantendo la decisione minmax corretta
- Pruning alfa-beta (McCarthy 1956, per scacchi)

Potatura alfa-beta: l'idea su ply

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



MINMAX(*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) con $z \leq 2$

= 3

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: α (inizialmente $-\infty$) e β (inizialmente $+\infty$): rappresentano rispettivamente la migliore alternativa per MAX e per MIN fino a quel momento.
  - I tagli avvengono quando *nel propagare indietro*:

    $v \geq \beta$ per i nodi MAX (taglio β)

    $v \leq \alpha$ per i nodi MIN (taglio α) [caso esempio prima su ply]

L'algoritmo Alfa-Beta: max

Schema del Minimax

function Alfa-Beta(*stato*) **returns** un'azione
  v = ValoreMax(*stato, -∞, +∞,*)
  return l'azione in Azioni(*stato*, *a*) con valore *v*

function Valore-Max(*stato, α, β*) **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*), *α, β*))
    if *v* ≥ *β* **then return** *v* ← *taglio β* **termina le chiamate ricorsive**
    α = Max(*α, v*)
  return *v*

Aggiorna il migliore per MAX

Si ferma tra i figli di v:
il migliore per MIN (beta) è più basso di v,
MIN non lo sceglierà

L'algoritmo Alfa-Beta: min

function Valore-Min(*stato, α, β*) **returns** un valore di utilità
  if TestTerminazione(*stato*) **then return** Utilità(*stato*)
  $v = +\infty$
  for each *a* in Azioni(*stato*) **do**
    v = Min(v, ValoreMax(Risultato(*stato, a*), α, β))
    if $v \leq \alpha$ **then return** v ← *taglio* α **termina le chiamate ricorsive**
    β = Min(β, v)
  return v

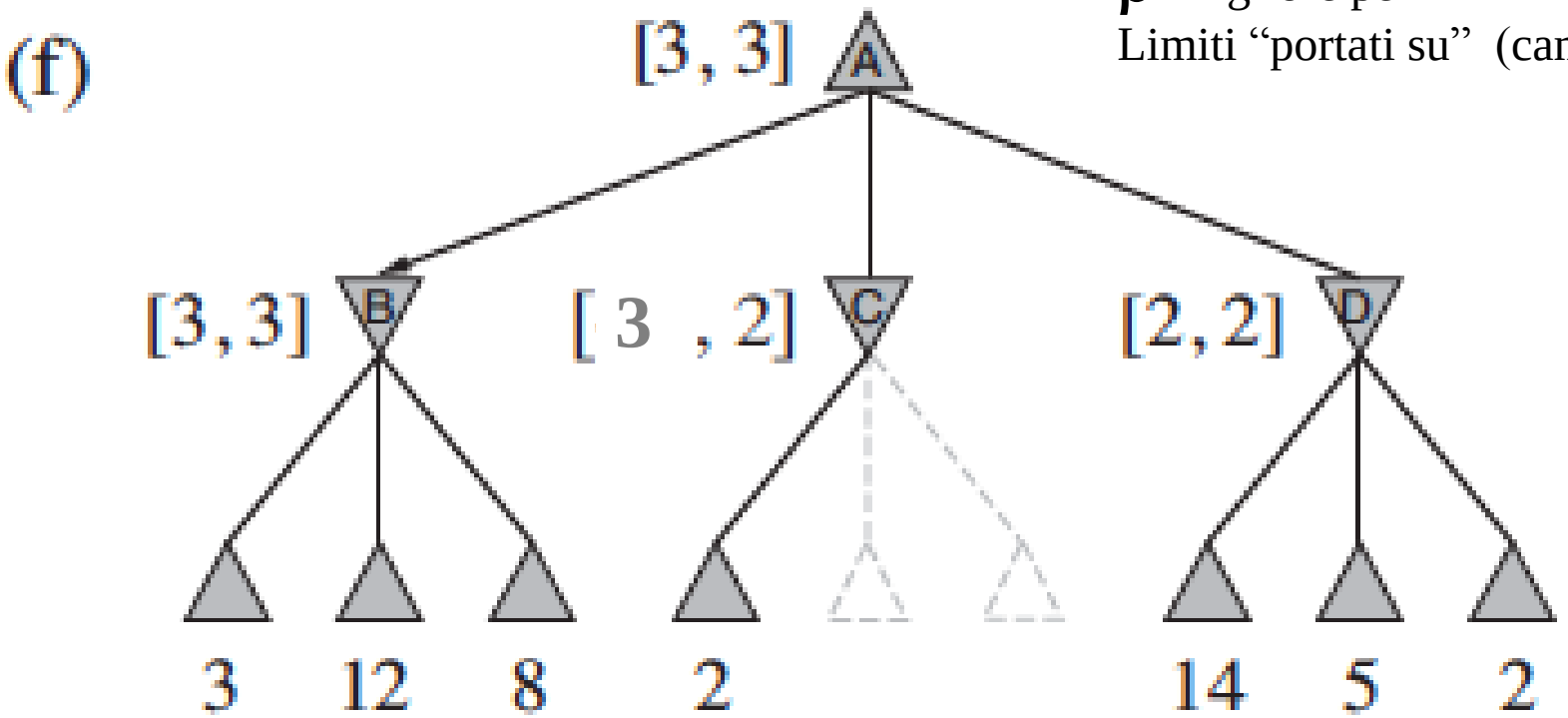
Si ferma tra i figli di v:
il migliore per MAX (alfa) è più alto di v,
MAX non lo sceglierà

Alfa-beta: funzionamento

α migliore per MAX

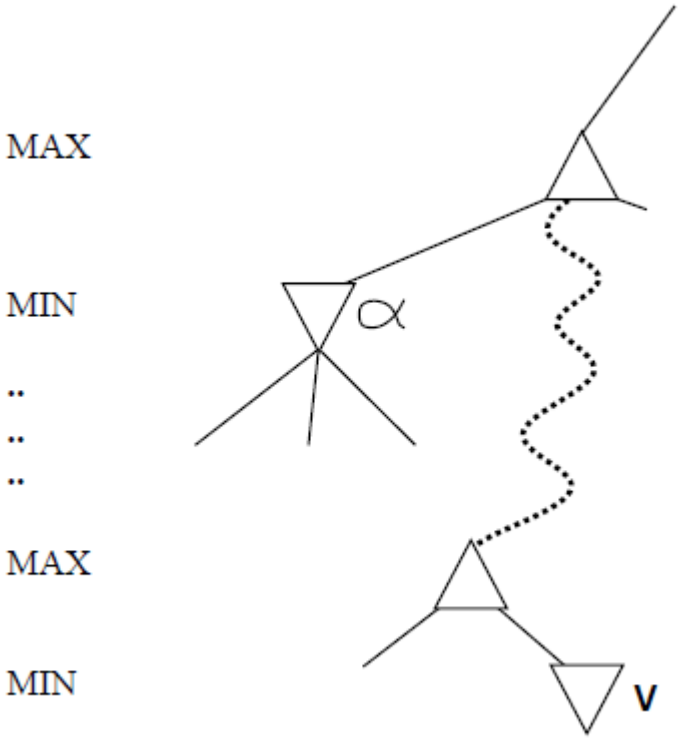
β migliore per MIN

Limiti "portati su" (cammino)



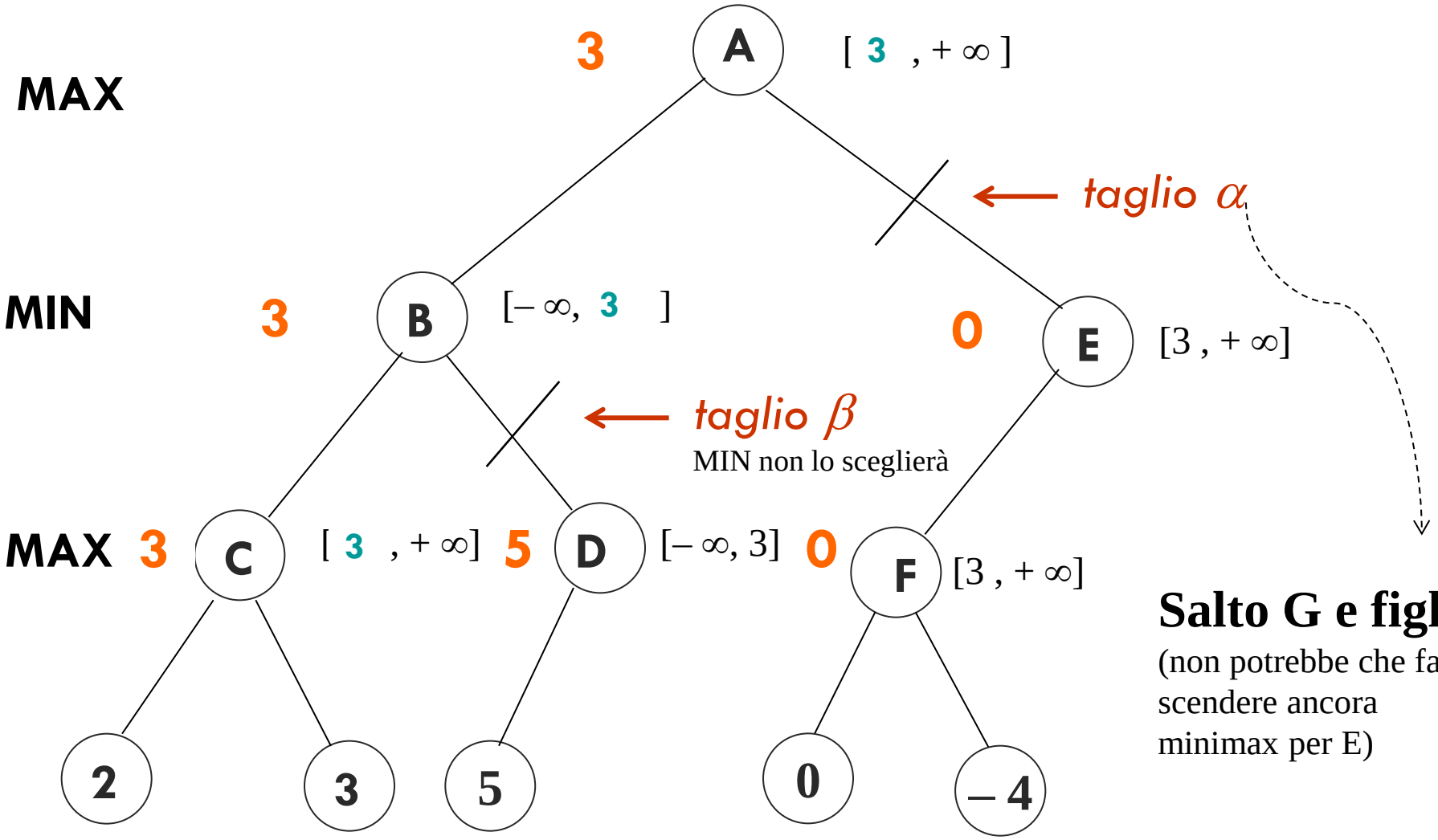
L'idea

- Consideriamo v
- Se c'è scelta migliore sopra, quel v non sarà mai raggiunto
- Tanto, ad esempio MAX, passerà piuttosto da alfa (o altro) che finire a v



Potatura alfa-beta: un esempio

α migliore per MAX
β migliore per MIN
Limiti "portati su" (cammino)



Salto G e figli
(non potrebbe che far scendere ancora minimax per E)

Nota: questi (F etc) sono pero' stati esplorati

Ordinamento delle mosse

- *Se in Ply avevo visto prima 2 sotto D, tagliavo anche li!*
- Potatura ottimale (teorico):
  - 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!
- Fattore di ramificazione: sqrt(b), e.g. 35 → ~6 (schacchi)
- Ma come avvicinarsi all'ordinamento ottimale?

Ordinamento dinamico (cenni)

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 (per non ripetere calcoli)
   - Situazione tipica: $[a_1, b_1, a_2, b_2]$ e $[a_1, b_2, a_2, b_1]$ portano nello stesso stato
   - Analoga a gestione della lista degli esplorati

Decisioni imperfette in tempo reale

- Torniamo a Minimax
- Cerchiamo modo di tagliare la ricerca prima di arrivare alle foglie
  - Valutando in modo euristico una *stima dell'utilità della posizione raggiunta*
  - Decidendo quando applicarla: *cutoff-test* o *test di taglio*

Ossia: costruire stime di utilità anche senza arrivare in fondo, perché spesso è troppo oneroso, fermandosi ad un profondità limite (dovuta al tempo massimo per mossa)

Min-max con decisione imperfetta

- Nei casi complessi occorre una funzione di valutazione euristica dello stato *Eval*(*s*).
- *Strategia*: guardare avanti *d* livelli
  - 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

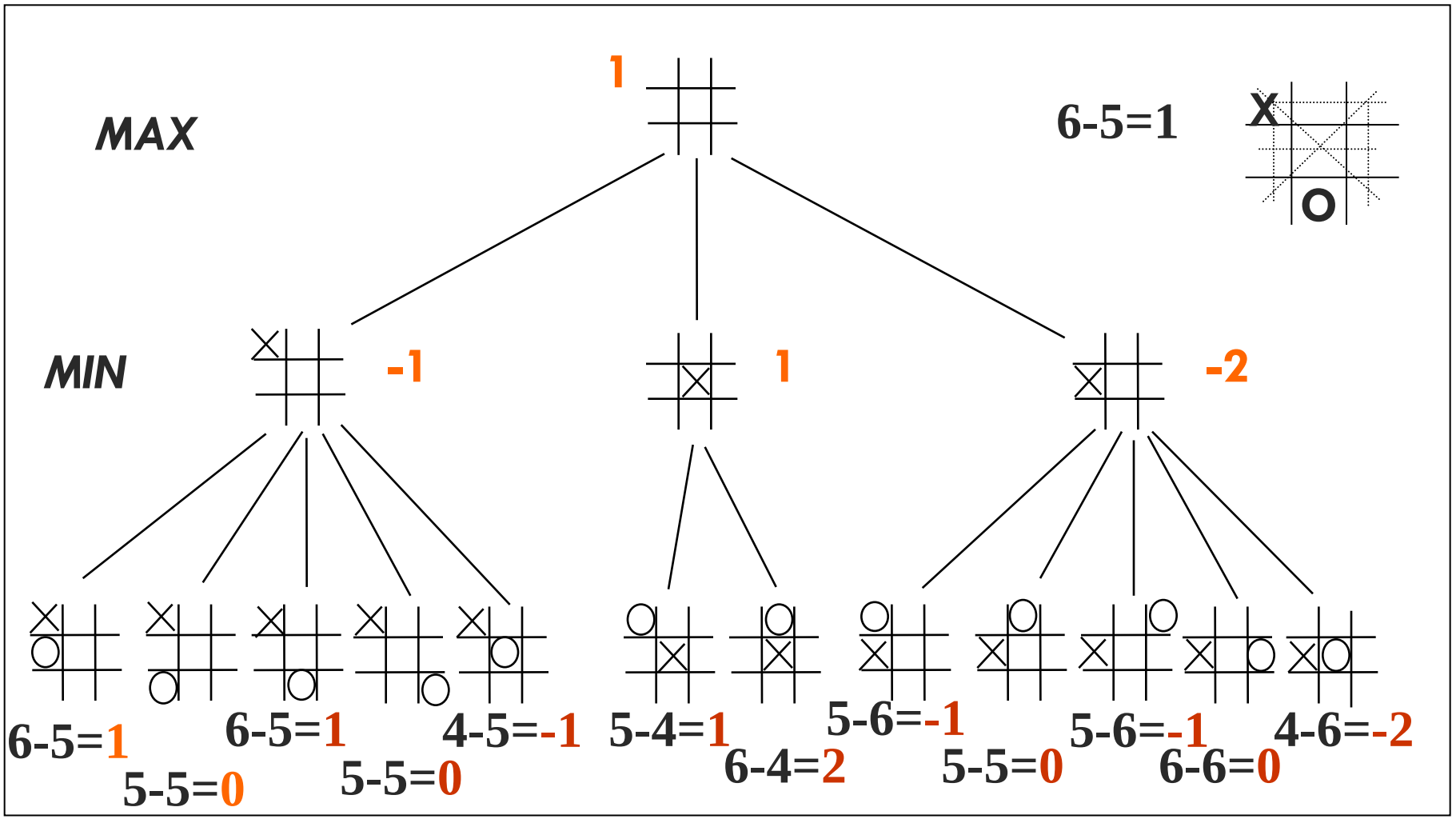
Se *D* è la profondità massima consentita …

H-Minimax(*s*, *d*) =

Eval(*s*)	se TestTaglio(*s*, *d*)
$\max_{a \in \text{Azioni}(s)}$ H-Minimax(Risultato(s, a), d+1)	se Giocatore(s)=MAX
$\min_{a \in \text{Azioni}(s)}$ H-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.
- 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

  $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

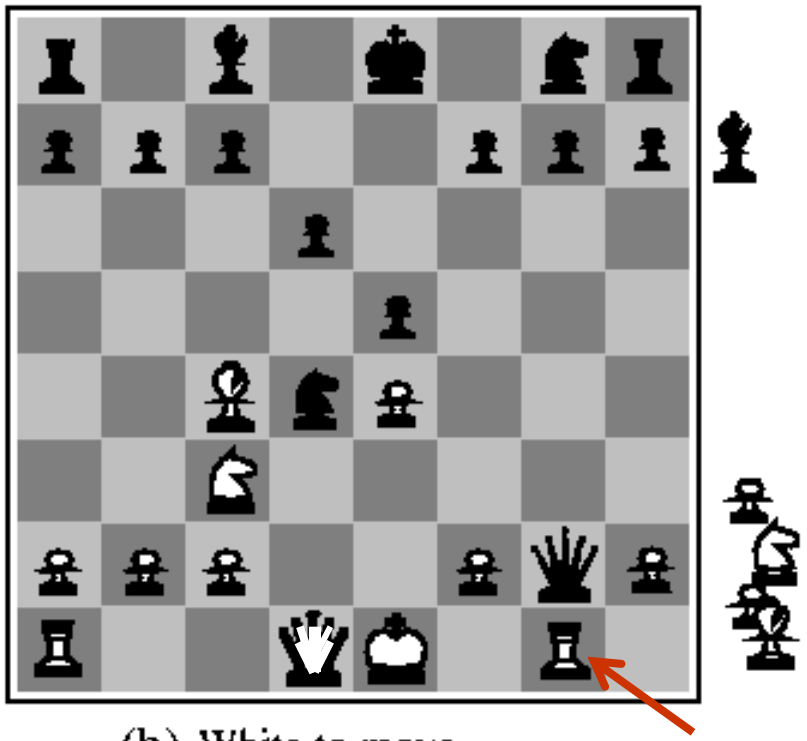
  o anche combinazioni non lineari di caratteristiche

  e.g. coppia di alfieri vale piu del doppio del singolo, che vale di piu nel finale…

  Via Learning? Si, e conferma di molte euristiche degli esperti (su schacchi)….
- Nota: conta l'ordine su Eval, non il valore assoluto

Problemi con MIN-MAX: stati non quiescienti

- *Stati non quiescienti*: l'esplorazione fino ad un livello può mostrare una situazione molto vantaggiosa
- Alla mossa successiva la regina nera viene mangiata.
- *Soluzione*: test di taglio diverso che la mera profondità raggiunta: applicare la valutazione a stati *quiescenti* stati in cui la funzione di valutazione non è soggetta a mutamenti repentini (ricerca di quiescenza)
- Altrimenti proseguire fino a stati quiescenti (continua a valutare dove ci sono molte catture)

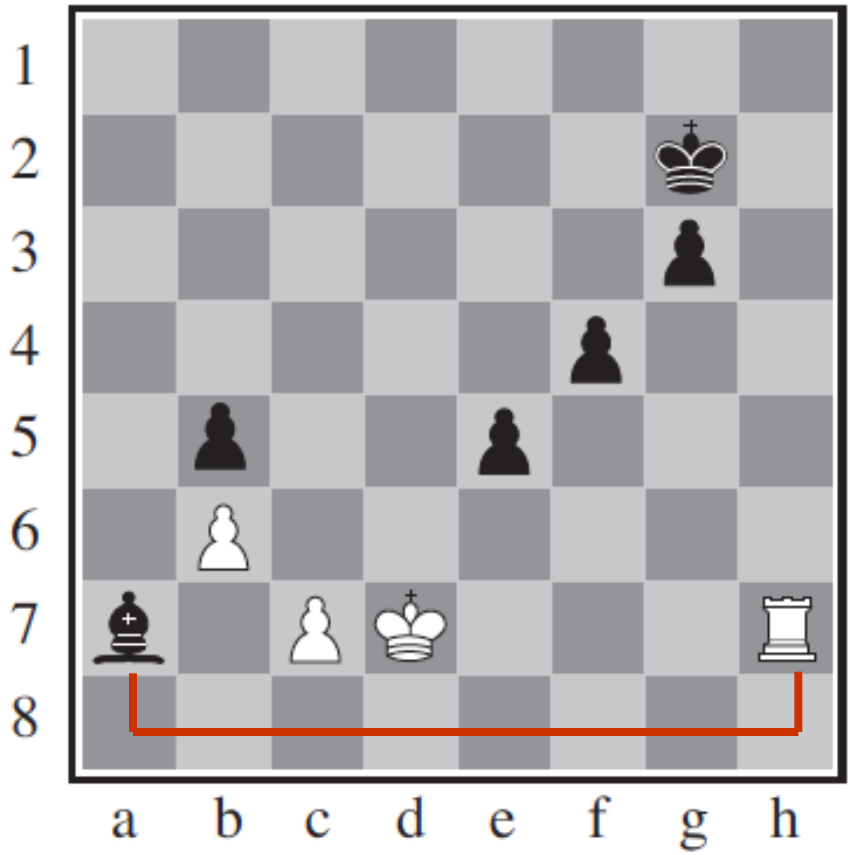


(b) White to move

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 (per pezzi a cattura inevitabile)

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.
Ma per il suo livello limitato sarebbe scelta buona…



Altri miglioramenti

1. Potatura in avanti: esplorare solo alcune mosse ritenute promettenti e tagliare le altre
   - Beam search (solo le prime k, con f piu alta): rischioso
   - Tagli probabilistici (basati su esperienza). Miglioramenti notevoli ad alfa-beta in Logistello [Buro, 1995]
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 …)

Scacchi

- Pronti?
- Con il meglio detto sinora, un PC che generi 1 milione di nodi al sec (2010), si arriva a circa 10 livelli, ossia 5 mosse che è un livello da giocatore esperto
- Si può arrivare a 14 livelli, usare funzioni di valutazioni accurate e sfruttare aperture e chiusure per raggiungere i livelli di una grande maestro
- Deep Blue (1997) sconfisse il campione del mondo
  - multiprocessore, 14 livelli e fino a 40 ove fosse «interessante», 8000 features nella f di valutazione, grande DB di aperture e chiusure
- Ma oggi basta un PC (e software migliore basato su euristiche raffinate)…

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

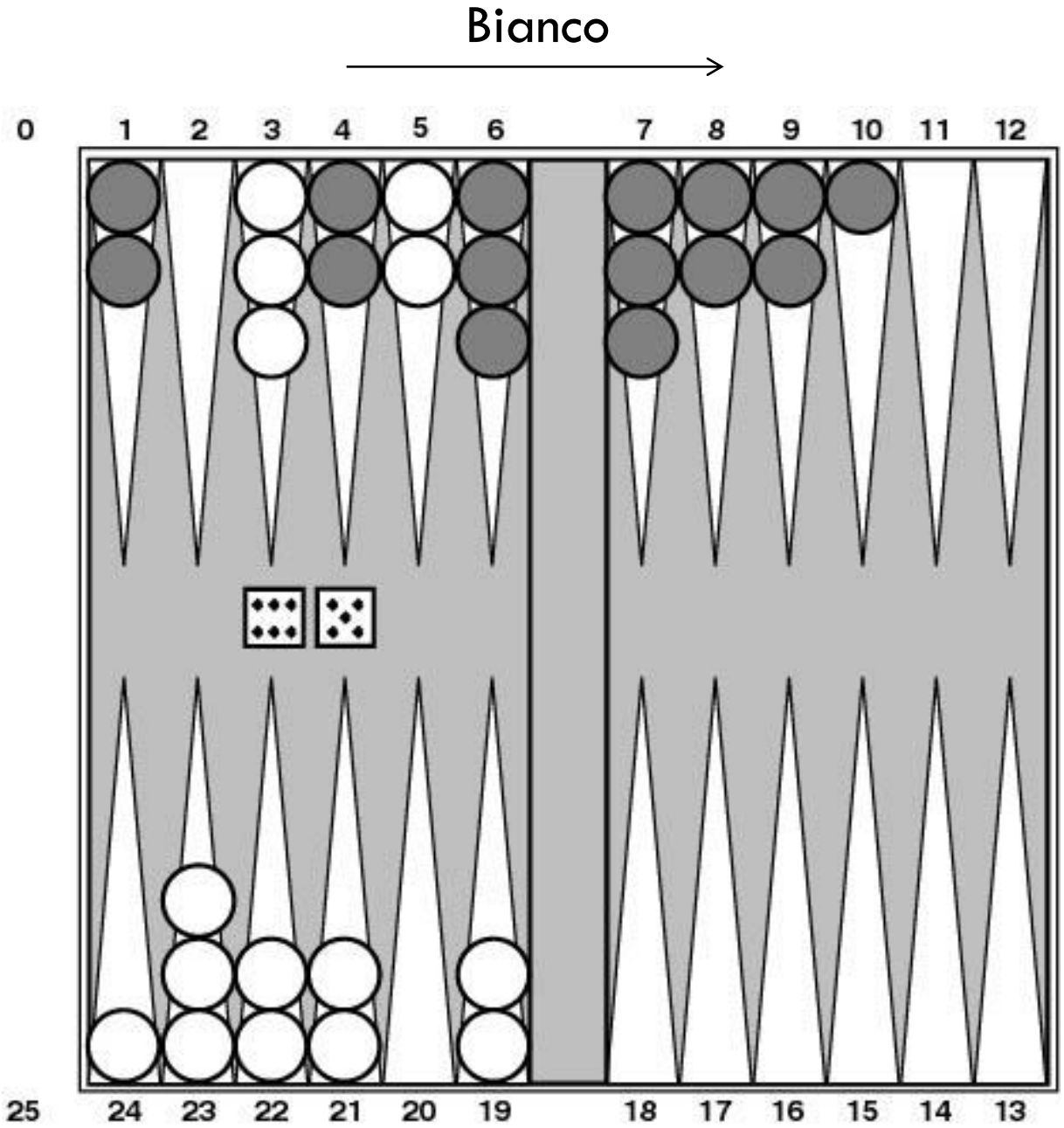
Lancio dadi 6-5,
4 mosse legali
per il bianco:

(5-10, 5-11)

(5-11,19-24)

(5-10, 10-16)

(5-11, 11-16)



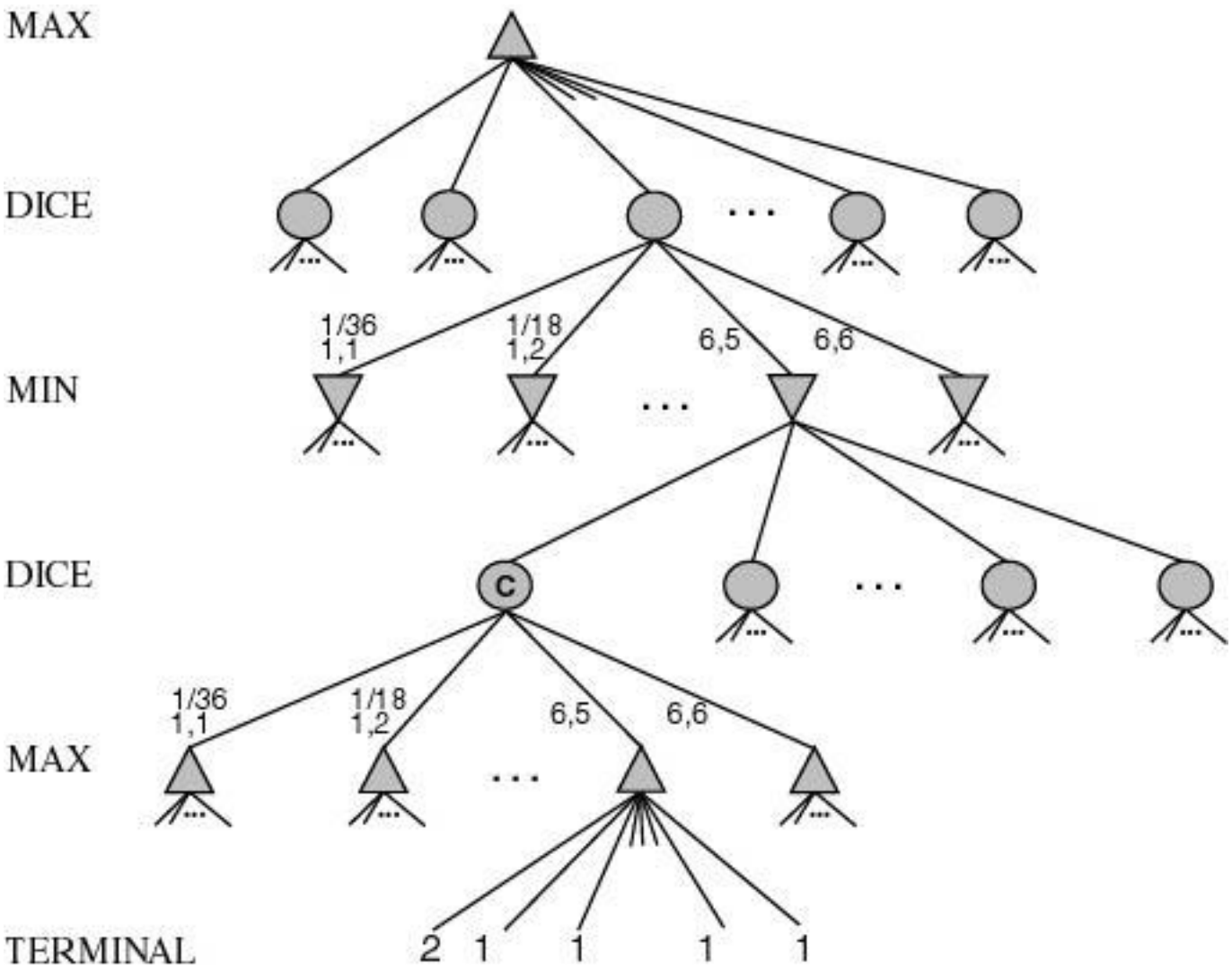
Min-max con nodi di casualità

- Estensione Minimax
- 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***
- Per nodi di causalita: considerare utilità media di tutti i figli, secondo la loro probabilità

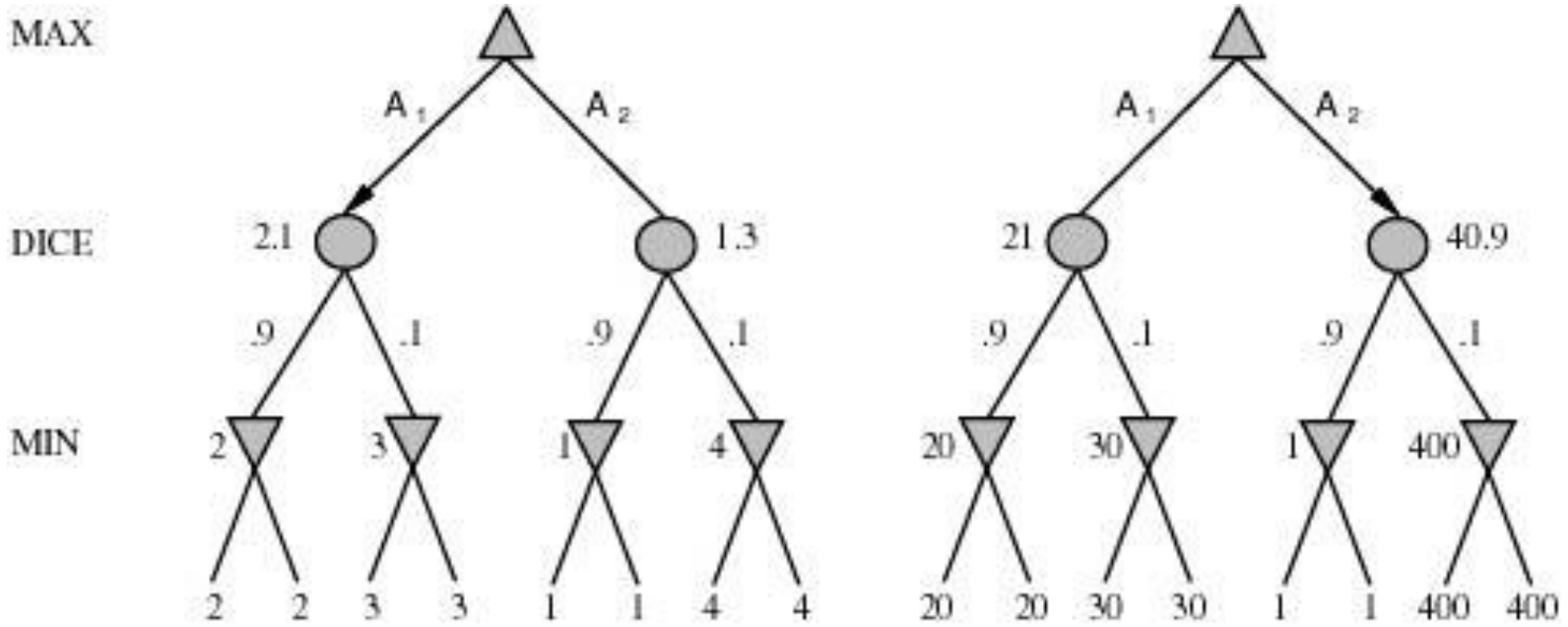
Stop

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 x 2 + 0.1 x 3 = 2.1

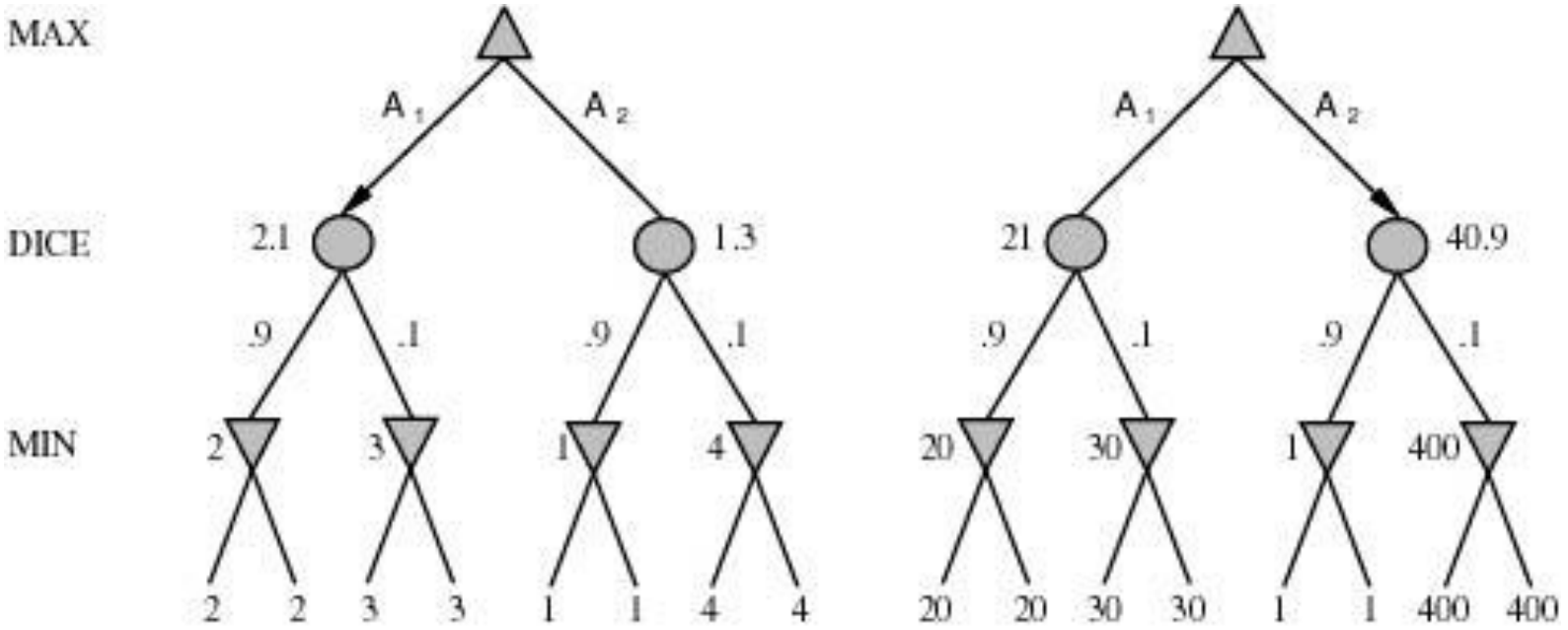
0.9 x 1 + 0.1 x 4 = 1.3

La mossa migliore è la prima.

Expectiminmax: la regola

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

Min-max con nodi *chance*: nota



- Non è sufficiente l'ordinamento relativo dei successori; nel secondo caso la mossa scelta è diversa.
- Is 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 gli effetti delle mosse dell'avversario
  - Es. Scacchi Kriegspiel (si vedono solo i propri pezzi)
- Giochi parzialmente osservabili *stocastici*
  - Le carte distribuite a caso in molti giochi di carte
  - Es. Bridge, whist, peppa, briscola …

Lo stato dell'arte (fino al 2010)

- 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*

Go & Machine Learning

Nature 529, 445–446 (28 January 2016)
The news: ***Deep-learning software defeats human professional for the first time.***

Nature 529, 484–489 (il metodo)
Mastering the game of Go with deep neural networks and tree search

E applicazioni al "mondo reale"…

Abstract (Nature 529, 484–489)

- The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its **enormous search space** and the **difficulty of evaluating board positions and moves.** Here we introduce a new approach to computer Go that **uses 'value networks' to evaluate board positions and 'policy networks' to select moves.** These **deep neural networks** are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play.
- Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play.
- We also introduce a **new search algorithm** that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0.
- This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

Per informazioni

Alessio Micheli

micheli@di.unipi.it

Dipartimento di Informatica
Università di Pisa - Italy

Computational Intelligence &
Machine Learning Group