Algoritmi per Giochi

Corso di Sistemi Intelligenti

Capitolo 6, Russel & Norvig

Outline

- ♢ Gioco perfetto
- ♢ Limiti alle risorse
- ♢ α–β pruning (potatura)
- ♢ Giochi nondeterministici (chance)
- ♢ Giochi ad informazione parziale

Giochi vs. problemi di ricerca

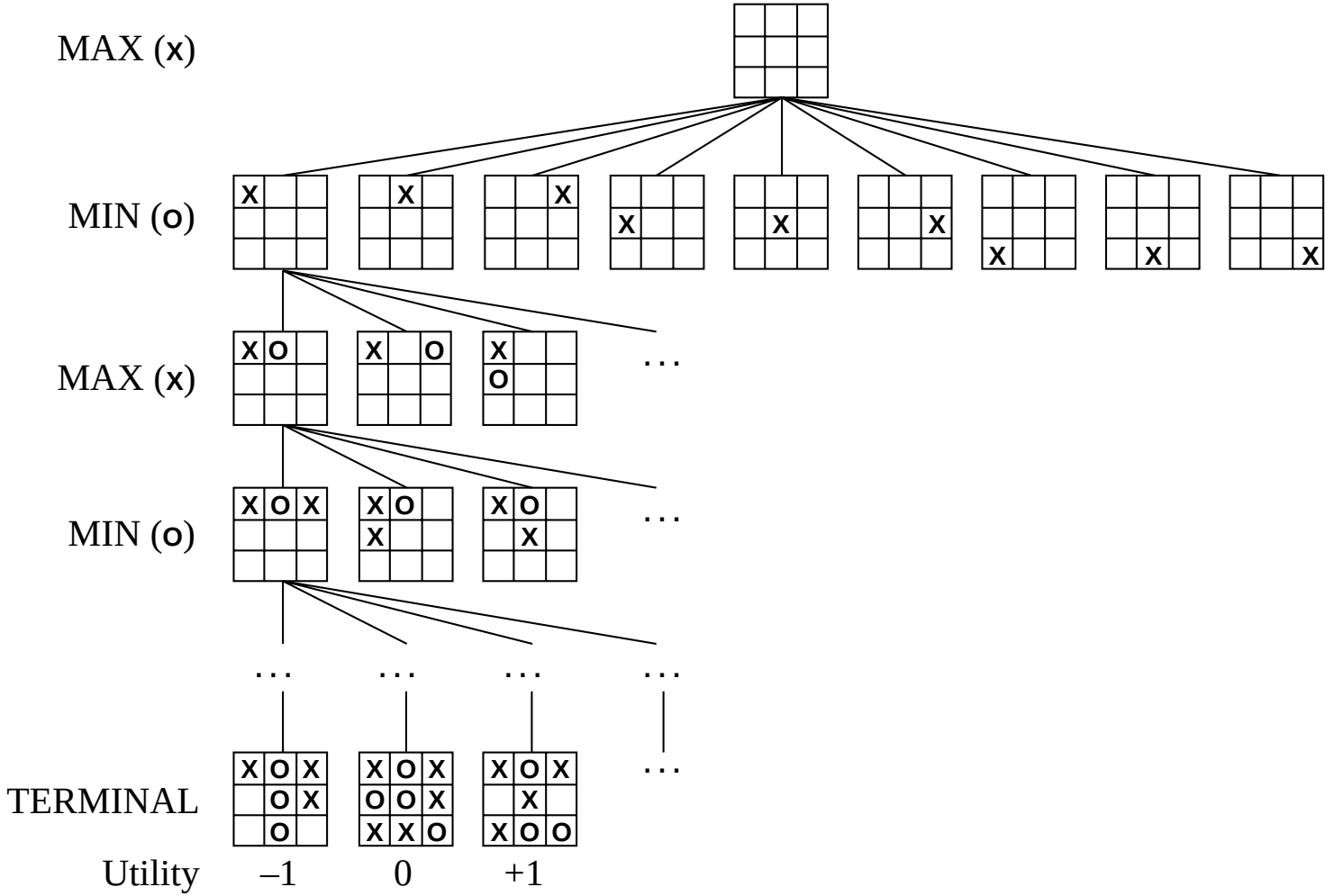
Avversario "imprevedibile" $\Rightarrow$ la soluzione è una strategia
che specifica una mossa per ogni possibile risposta dell'avversario

Limiti di tempo $\Rightarrow$ poco probabile il raggiungimento del goal, necessità di approssimare

Tipi di Giochi

	deterministico	nondeterministico
informazione completa	scacchi, dama, go, othello	backgammon monopoli
informazione parziale		bridge, poker, scrabble guerra nucleare

Albero di gioco (2-giocatori, deterministico, turni)

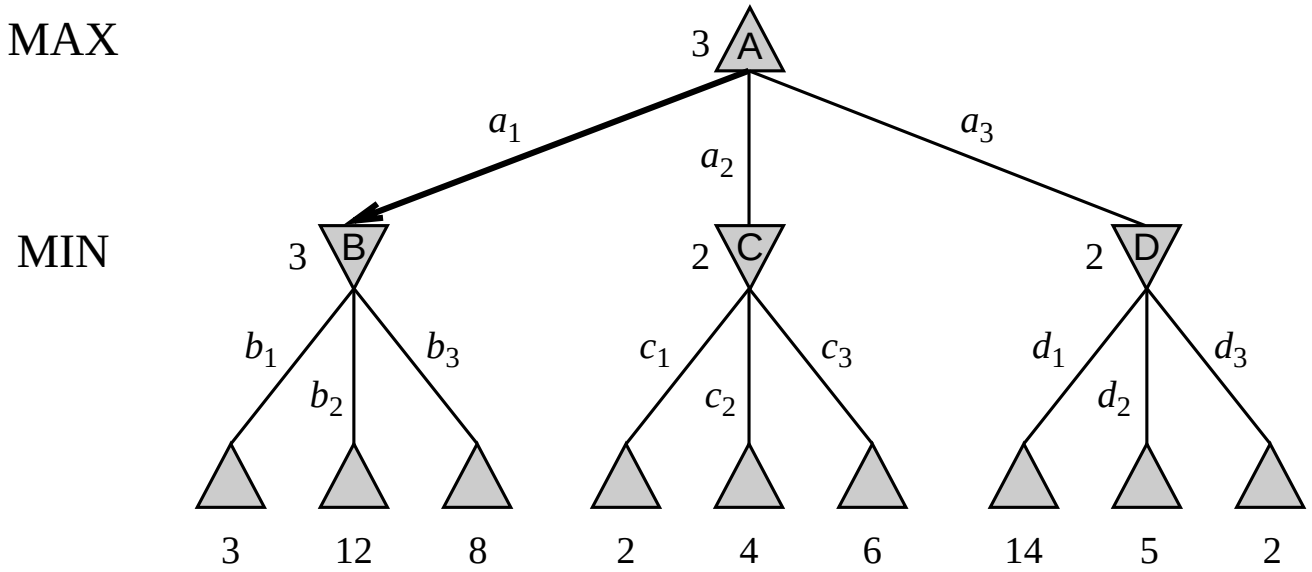


Minimax

Gioco perfetto per giochi deterministici e ad informazione perfetta

Idea: scegliere la mossa che conduce alla posizione con *valore minimax* più alto = migliore vantaggio raggiungibile contro un avversario che gioca in modo ottimo

E.g., gioco a 2-turni (2-ply game):



Algoritmo Minimax

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Proprietà di minimax

Completezza??

Proprietà di minimax

Completezza?? Solo se l'albero è finito (il gioco degli scacchi ha regole specifiche per garantire il termine del gioco).

NB una strategia finita può esistere anche se l'albero è infinito!

Ottimalità??

Propertà di minimax

Completezza?? Si, solo se l'albero è finito (il gioco degli scacchi ha regole specifiche per garantire il termine del gioco).

Ottimalità?? Si, contro un avversario "ottimo". Altrimenti??

Complessità in tempo??

Proprietà di minimax

Completezza?? Si, solo se l'albero è finito (il gioco degli scacchi ha regole specifiche per garantire il termine del gioco).

Ottimalità?? Si, contro un avversario "ottimo". Altrimenti??

Complessità in tempo?? $O(b^m)$

Complessità in spazio??

Proprietà di minimax

Completezza?? Si, solo se l'albero è finito (il gioco degli scacchi ha regole specifiche per garantire il termine del gioco).

Ottimalità?? Si, contro un avversario "ottimo". Altrimenti??

Complessità in tempo?? $O(b^m)$

Complessità in spazio?? $O(bm)$ (esplorazione depth-first)

Per il gioco degli scacchi, $b \approx 35$, $m \approx 100$ per giochi "ragionevoli"
$\Rightarrow$ una soluzione esatta è sicuramente non fattibile

Limiti alle risorse

Supponiamo di avere a disposizione 100 secondi per mossa, e di poter esplorare 10^4 nodi al secondo

$\Rightarrow$ si riescono ad esplorare 10^6 nodi per mossa

Approccio standard:

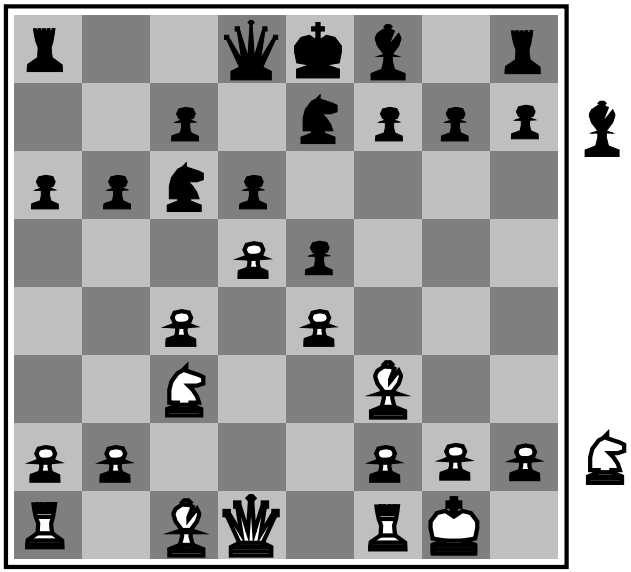
- *test di taglio (cutoff)*

  e.g., limite alla profondità (eventualmente aggiungendo *ricerca di quiescenza*)

- *funzione di valutazione*

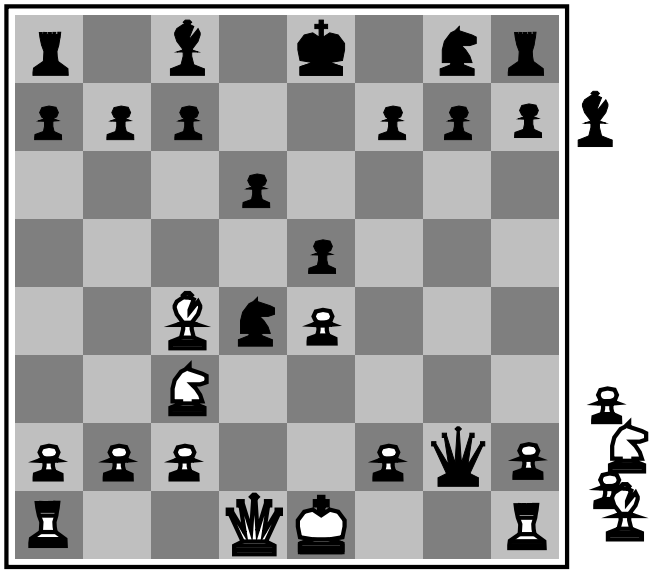
  = stima della desiderabilità della posizione

Funzioni di valutazione



Mossa al Nero

Bianco in leggero vantaggio



Mossa al Bianco

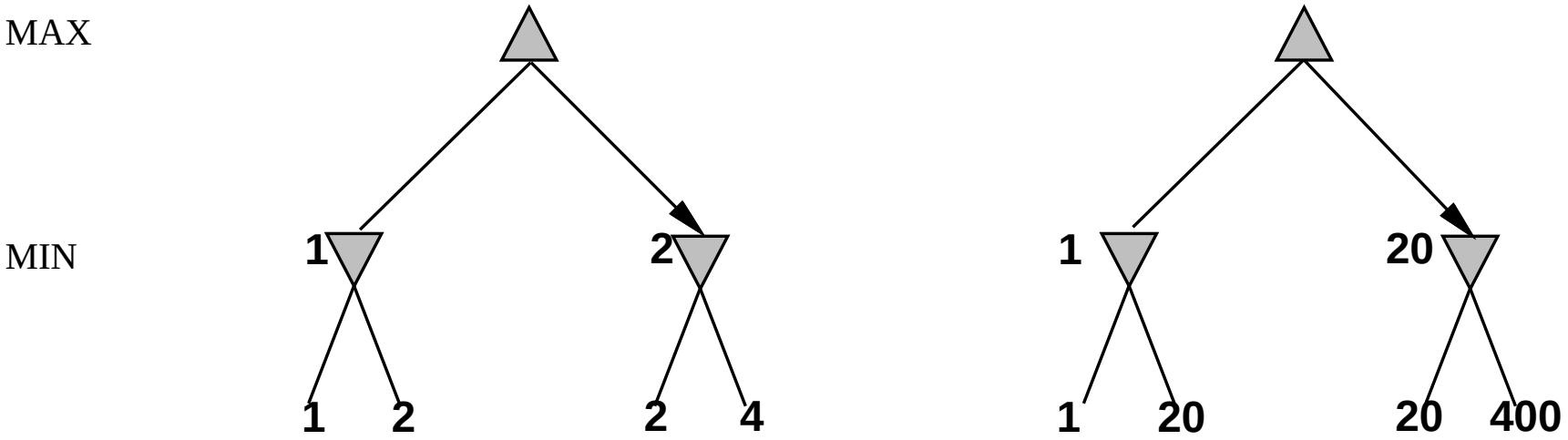
Vince il Nero

Per il gioco degli scacchi, tipicamente una somma pesata *lineare* di features

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

e.g., $w_1 = 9$ con
$f_1(s) =$ (numero di regine bianche) – (numero di regine nere), etc.

Digressione: non importa calcolare valori esatti



Il comportamento corretto è preservato per ogni trasformazione *monotona* di EVAL

Quello che conta è solo l'ordine:

il guadagno in giochi deterministici agisce come una funzione di *utilità ordinale*

Ricerca con taglio

MinimaxCutoff è identica a MinimaxValue eccetto che

1. Terminal? è rimpiazzata da Cutoff?
2. Utility è rimpiazzata da Eval

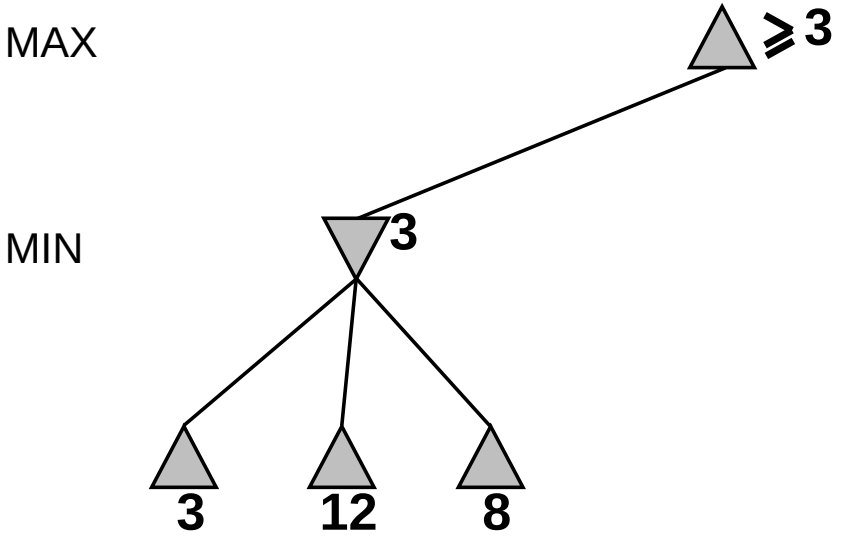
Funziona in pratica?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

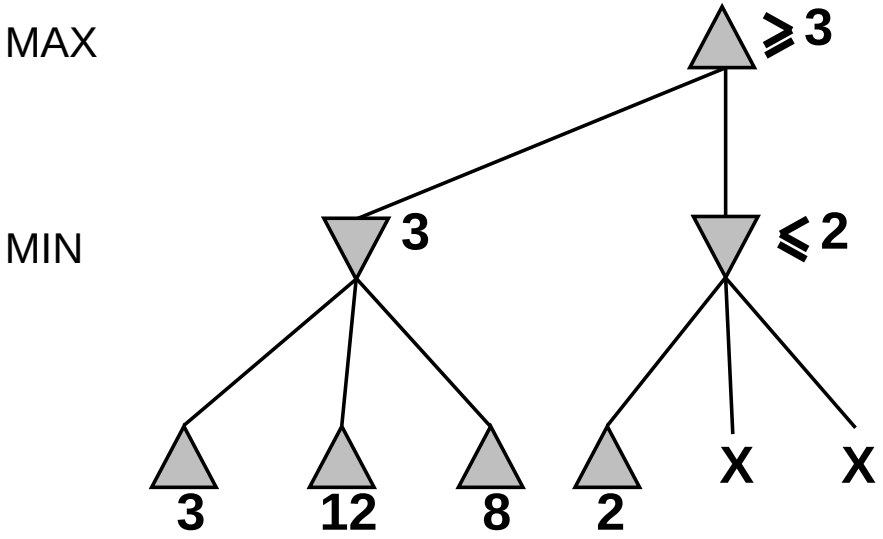
4-ply lookahead corrisponde ad un giocatore di scacchi pessimo!

4-ply $\approx$ umano a livello di novizio
8-ply $\approx$ prestazione di un PC, umano a livello di maestro
12-ply $\approx$ Deep Blue, Kasparov

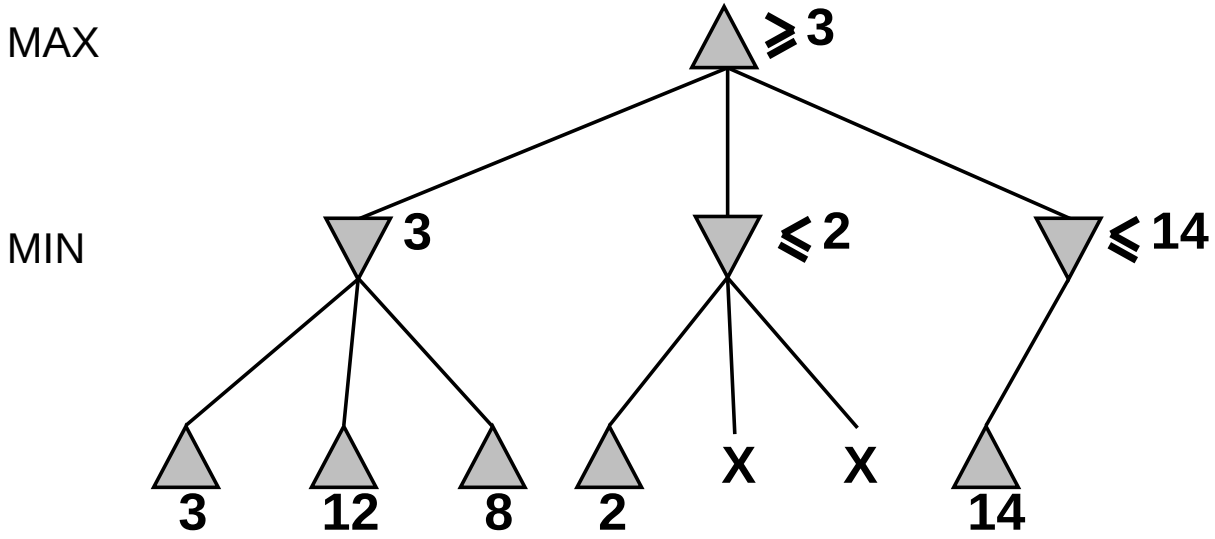
α–β pruning: esempio



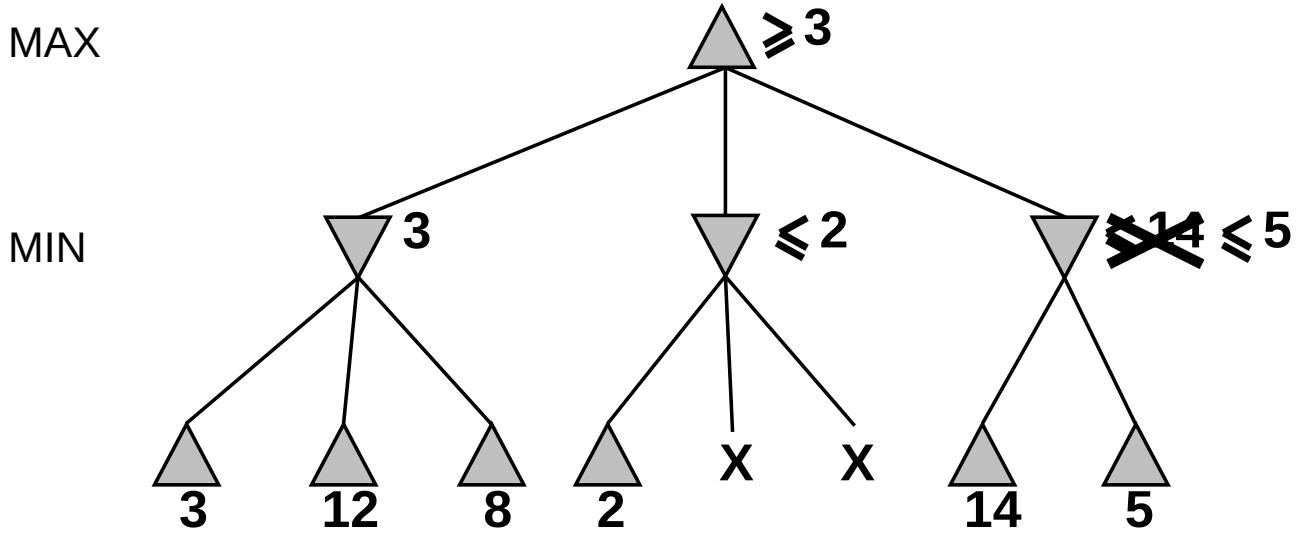
α–β pruning: esempio



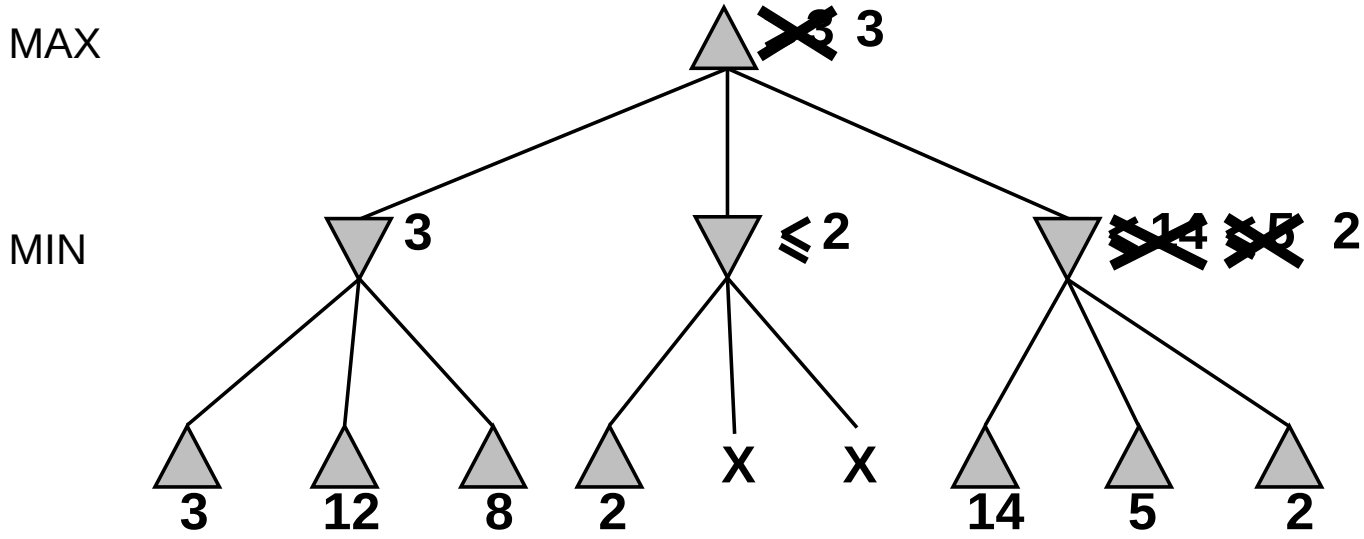
α–β pruning: esempio



α–β pruning: esempio



α–β pruning: esempio



Proprietà di α–β pruning

Il pruning *non* modifica il risultato finale

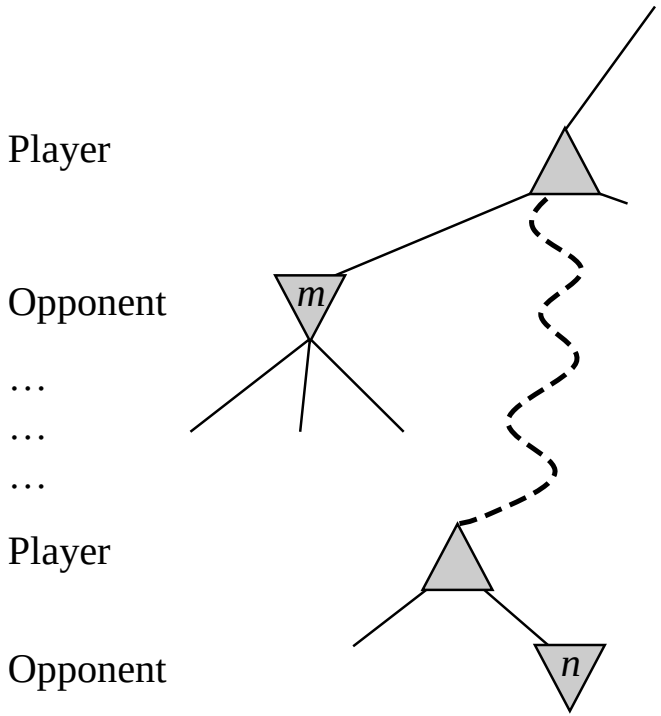
Un buon ordinamento delle mosse migliora l'efficacia del pruning

Con "ordine perfetto", complessità in tempo $= O(b^{m/2})$

$\Rightarrow$ *raddoppia* la profondità di ricerca

$\Rightarrow$ può facilmente raggiungere profondità 8 e giocare bene a scacchi

Perché si chiama α–β?



α è il miglior valore (per MAX) trovato fino a quel momento al di fuori del cammino corrente

Se V è peggiore di α, MAX lo eviterà $\Rightarrow$ pota il ramo corrispondente

Similmente si può definire β per MIN

The α–β algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

α–β pruning: strategie Max e Min (best case)

Idea di massima

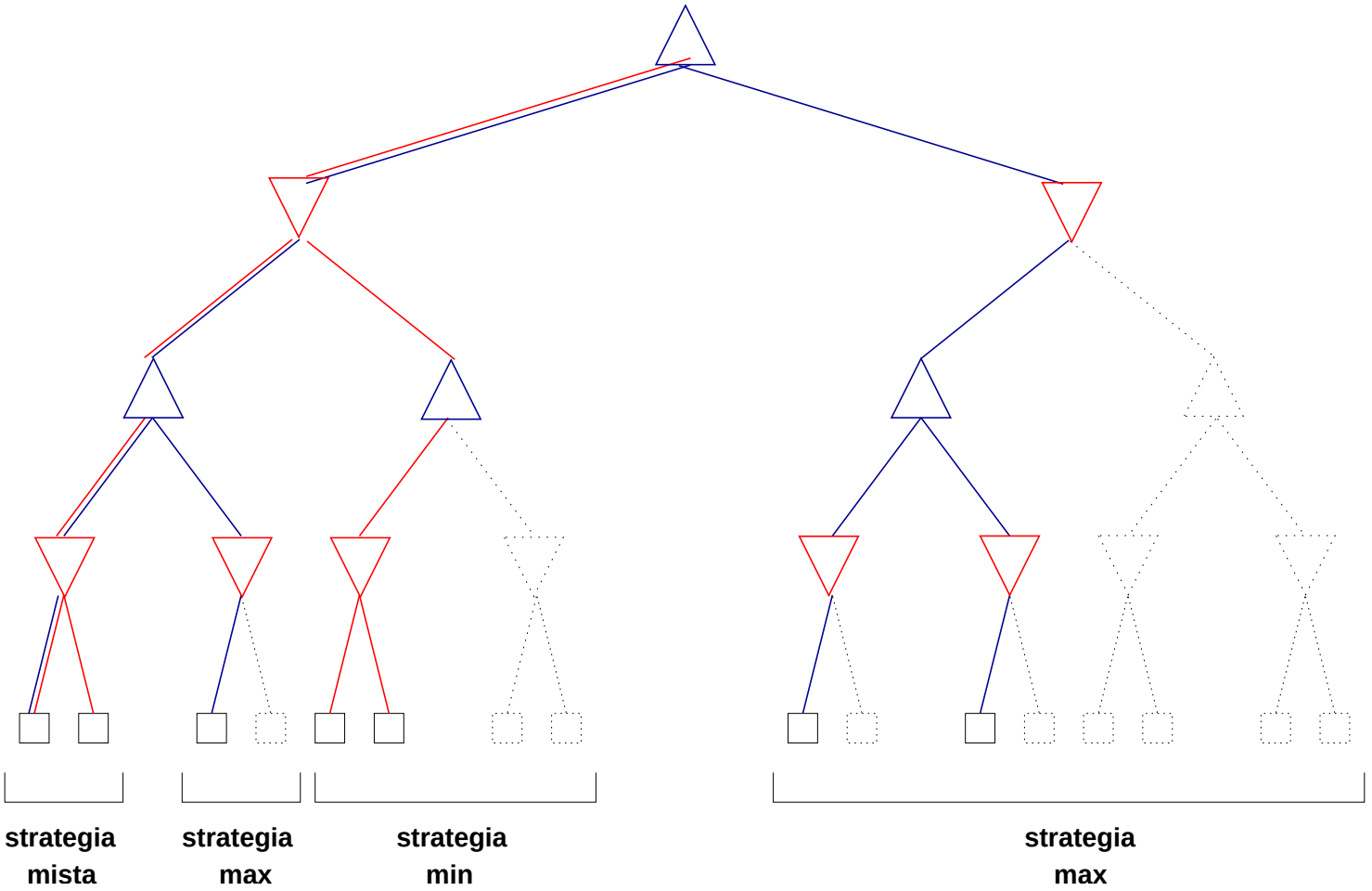
Strategia per Max (non importa cosa fa Min)

- esplora solo un figlio (il primo) per ogni modo Max
- esplora tutti i figli per ogni nodo Min

Strategia per Min (non importa cosa fa Max)

- esplora solo un figlio (il primo) per ogni modo Min
- esplora tutti i figli per ogni nodo Max

α–β pruning: visualizzazione strategie (best case)



α–β pruning: analisi più formale (best case)

Per avere il valore esatto di uno stato occorre

- conoscere il valore esatto di utilità per uno stato figlio, e
- conoscere un bound sulla utilità di tutti gli stati figli rimanenti

Per derivare un bound sulla utilità di uno stato occorre

- conoscere il valore esatto di utitlità per uno stato figlio

α–β pruning: analisi più formale (best case)

poniamo

- $E(d)$ essere il numero minimo di stati da considerare per conoscere il valore esatto di utilità di uno stato a distanza d ply dagli stati terminali
- $B(d)$ essere il numero minimo di stati da considerare per conoscere un bound sul valore di utilità di uno stato a distanza d ply dagli stati terminali

abbiamo

$$E(d+1) = E(d) + (b-1)B(d)$$
$$B(d+1) = E(d)$$

espandendo per $E(d+2)$ abbiamo:

$$\begin{aligned} E(d+2) &= E(d+1) + (b-1)B(d+1) \\ &= (E(d) + (b-1)B(d)) + (b-1)E(d) \\ &= bE(d) + (b-1)B(d) \\ &= bE(d) + (b-1)E(d-1) \end{aligned}$$

α–β pruning: analisi più formale (best case)

sapendo che

$$E(0) = B(0) = 1$$
$$E(d-1) < E(d)$$

abbiamo

$$\begin{aligned} E(d+2) &= bE(d) + (b-1)E(d-1) \\ &< (2b-1)E(d) \\ &< 2bE(d) \end{aligned}$$

quindi

$$E(m) \leq (\sqrt{2b})^m = (\sqrt{2})^m b^{m/2}$$

Giochi deterministici in pratica

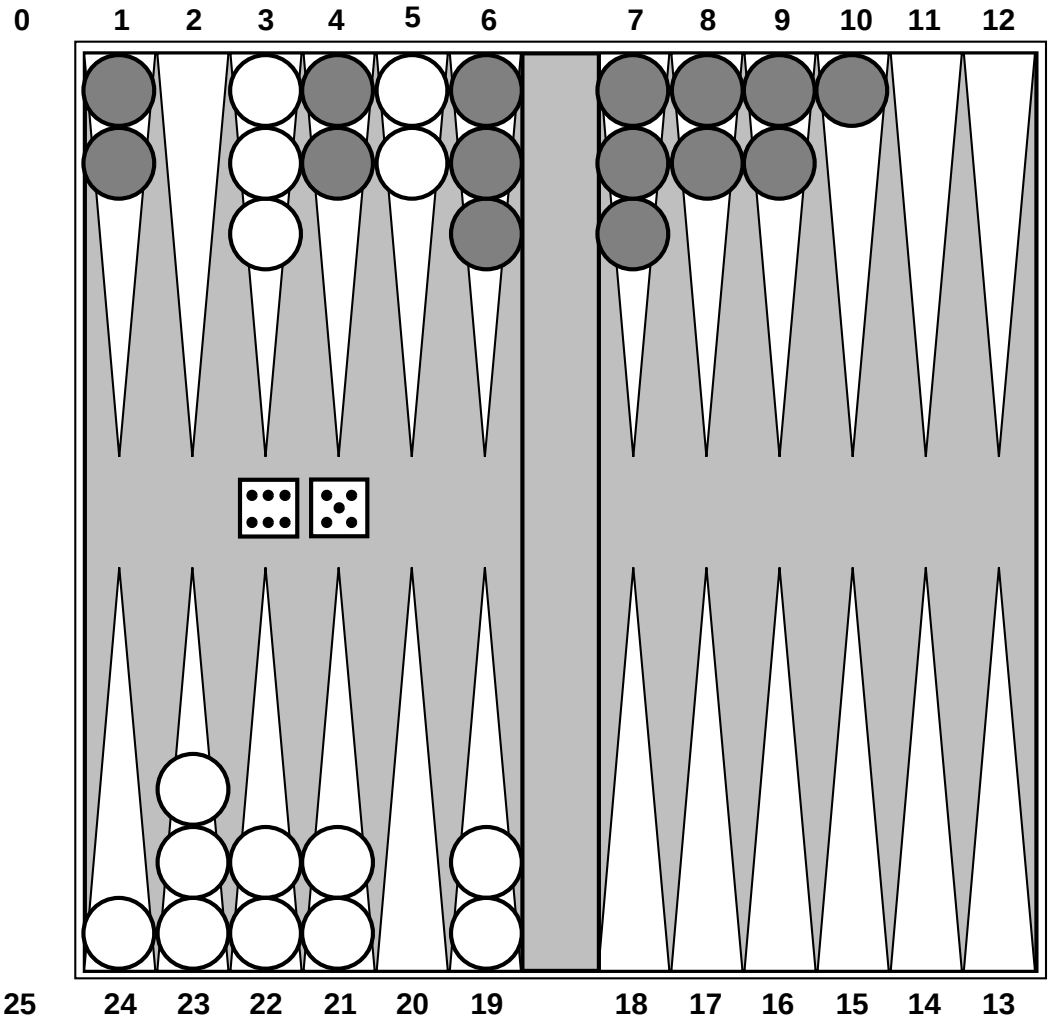
Dama: Chinook ha terminato il dominio durato 40 anni del campione mondiale (umano) Marion Tinsley nel 1994. Ha utilizzato un database che definiva tutte le mosse ottime a partire da tutte le posizioni della scacchiera con 8 o meno pezzi: un totale di 443.748.401.247 posizioni.

Scacchi: Deep Blue ha battuto il campione mondiale Gary Kasparov nel 1977 in un incontro a sei partite. Deep Blue è in grado di esplorare 200 milioni di posizioni al secondo, usa una funzione di valutazione molto sofisticata, e altri metodi segreti per estendere la ricerca fino a 40 ply.

Othello: campioni umani si rifiutano di giocare contro computer, perché questi ultimi sono troppo bravi.

Go: campioni umani si rifiutano di giocare contro computer, perché questi ultimi sono dei veri novizi. In go, $b > 300$, e quindi la maggior parte dei programmi usa basi di conoscenza su possibili situazioni per decidere le mosse da fare.

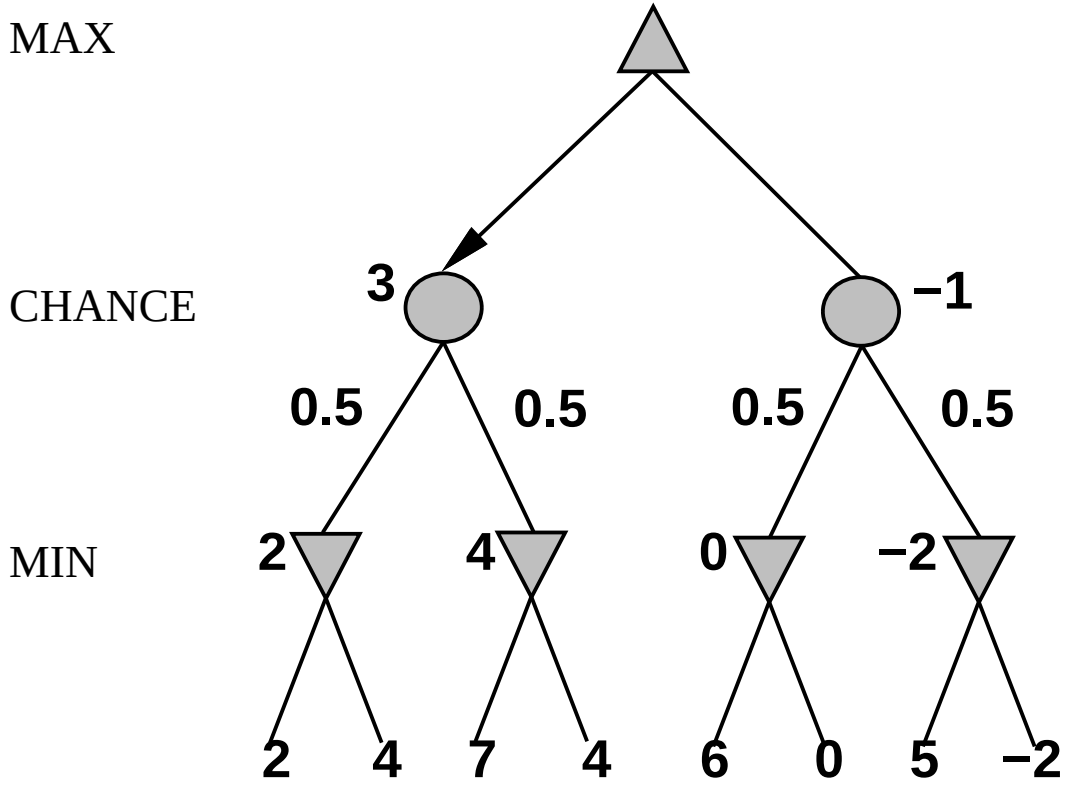
Giochi nondeterministici: backgammon



Giochi nondeterministici in generale

In giochi nondeterministici, eventi casuali sono introdotti da dadi, o mescolanza di carte a caso

Esempio semplificato con lancio di moneta:



Algoritmo per giochi nondeterministici

EXPECTIMINIMAX restituisce "gioco perfetto"

Come MINIMAX, con l'eccezione che si devono trattare anche gli eventi casuali

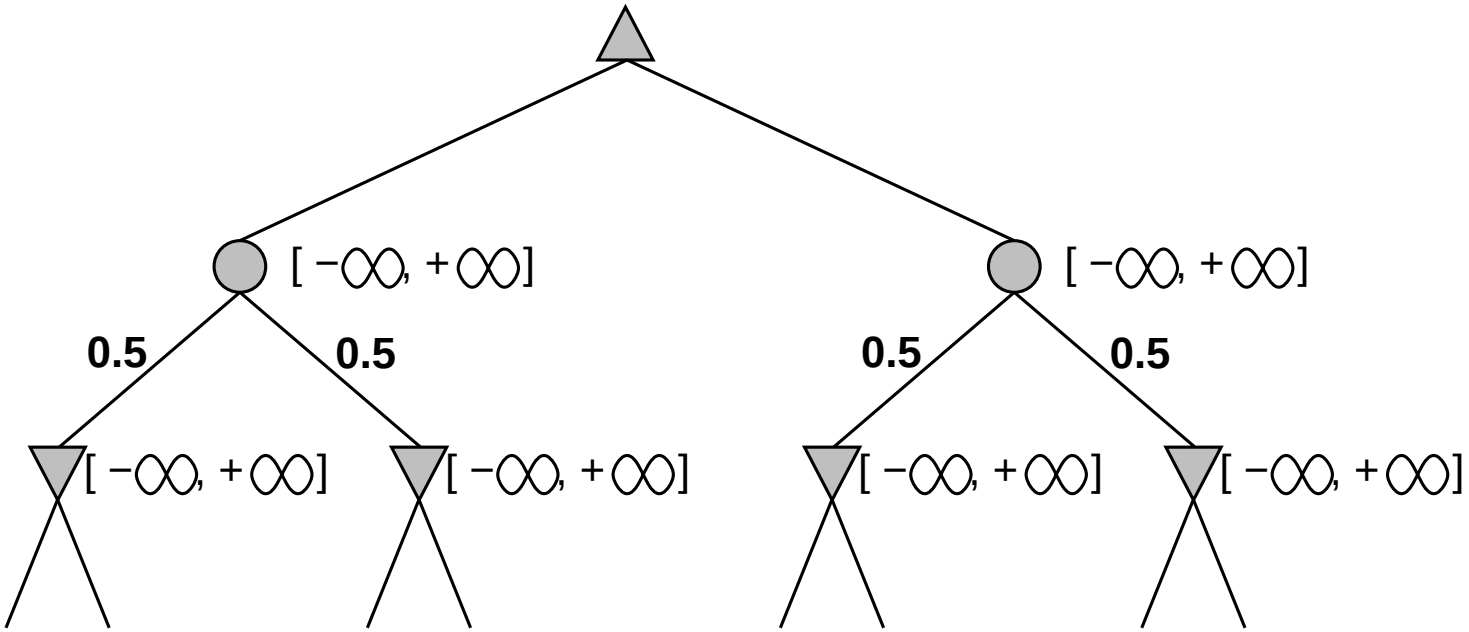
...
if *state* is a MAX node **then**
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a MIN node **then**
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
...

Più precisamente, per nodi chance:

ExpectiMinimax-Value($state$) = $\Sigma_{s \in Successors(state)} P(s)$ExpectiMinimax-Value($s$)

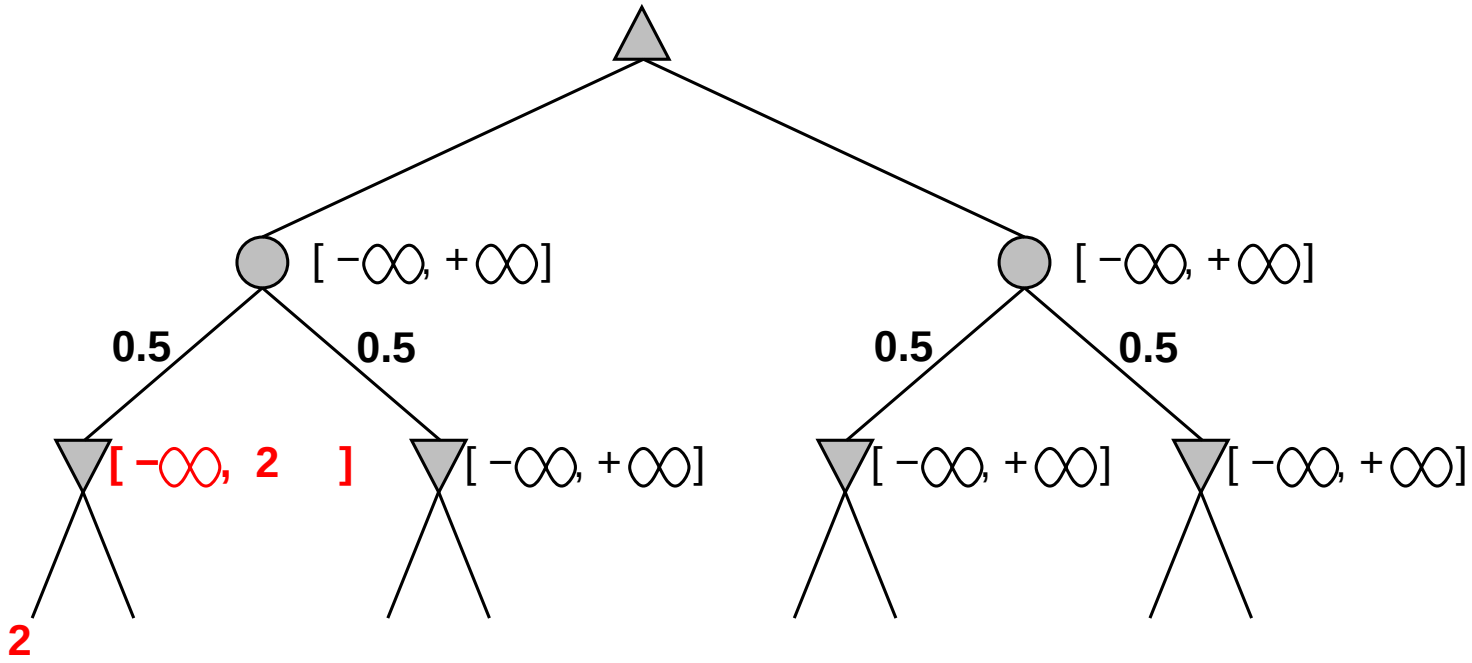
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



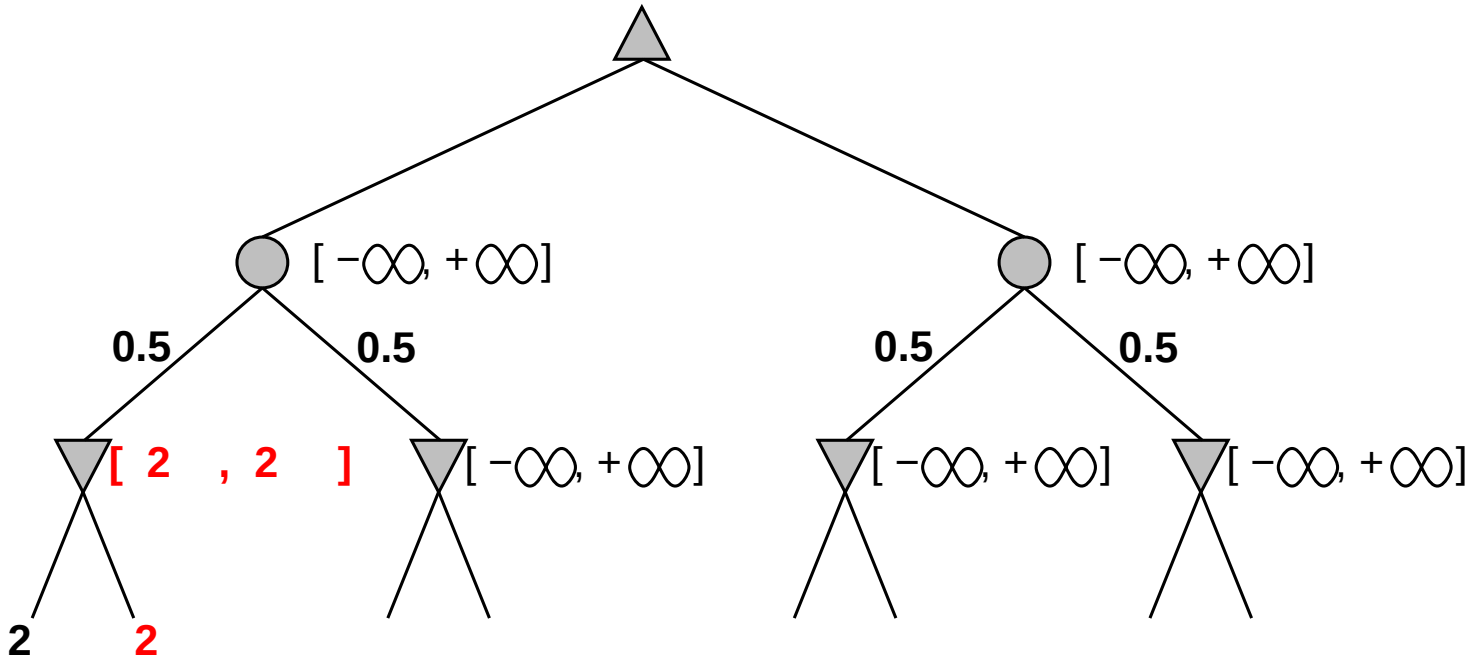
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



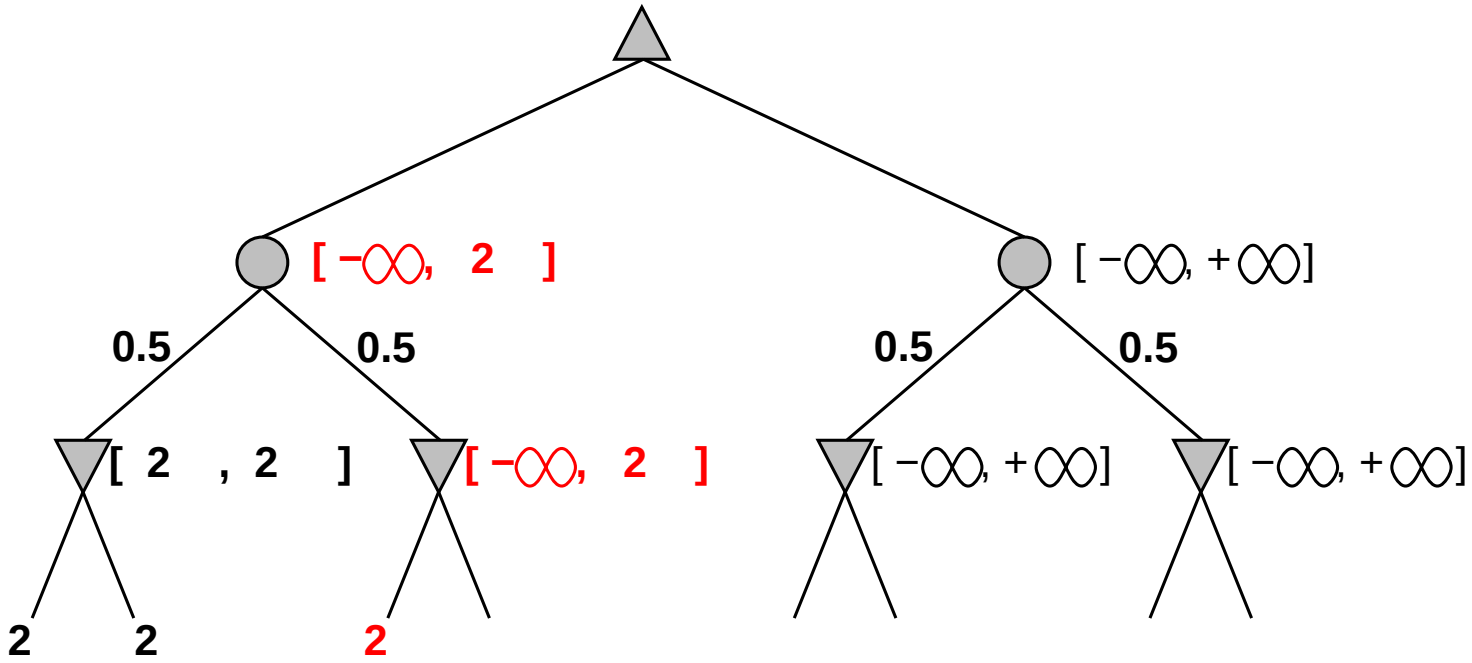
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



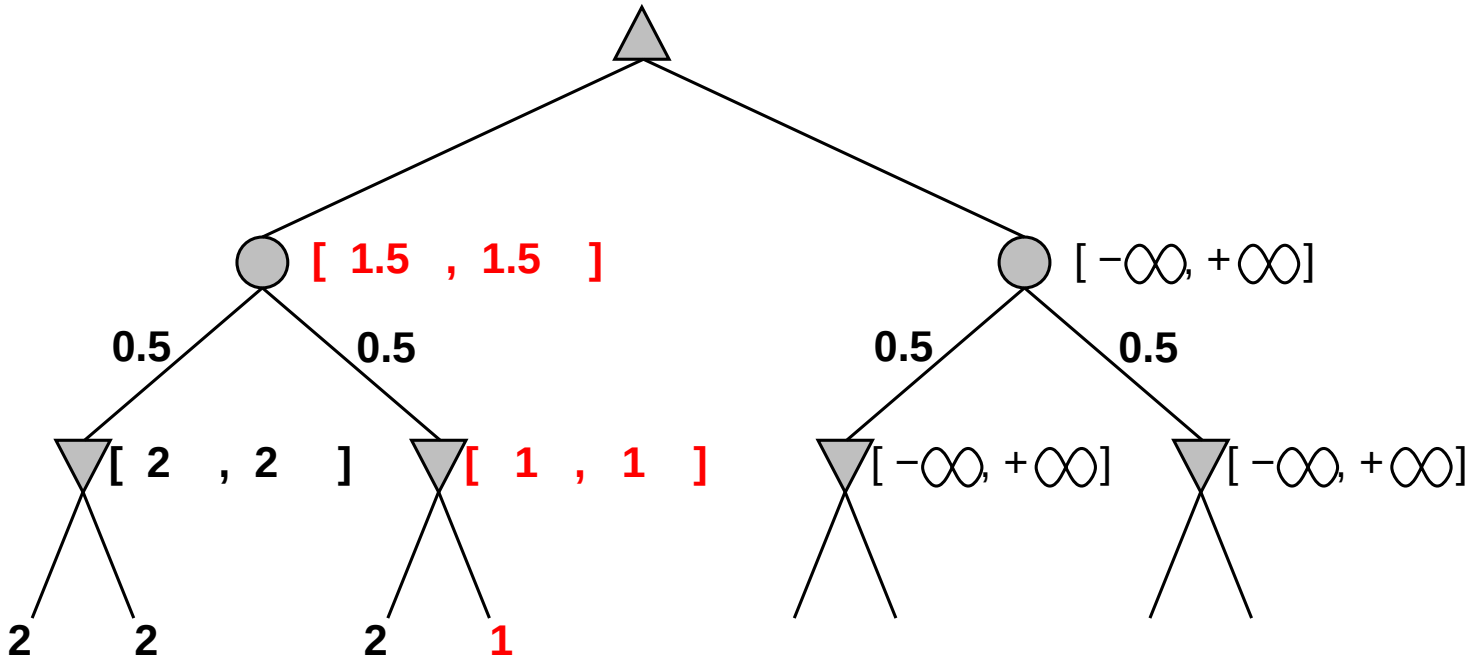
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



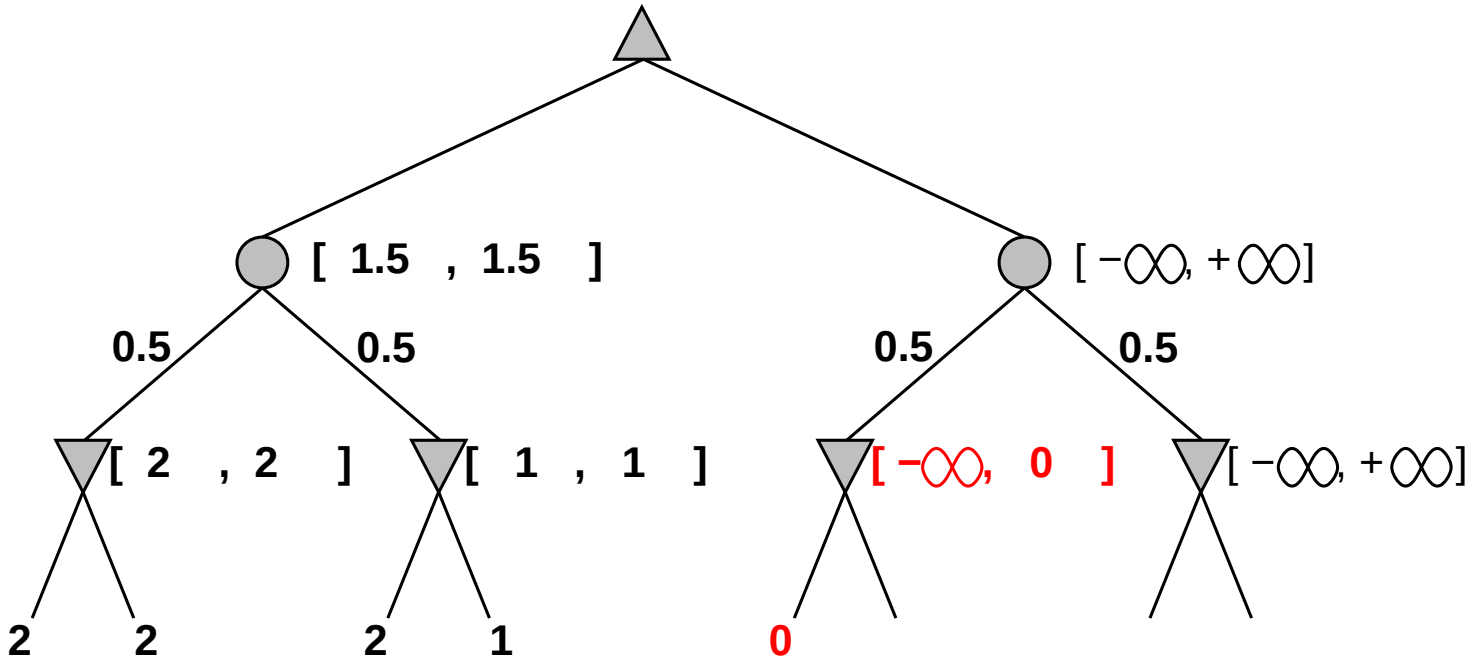
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



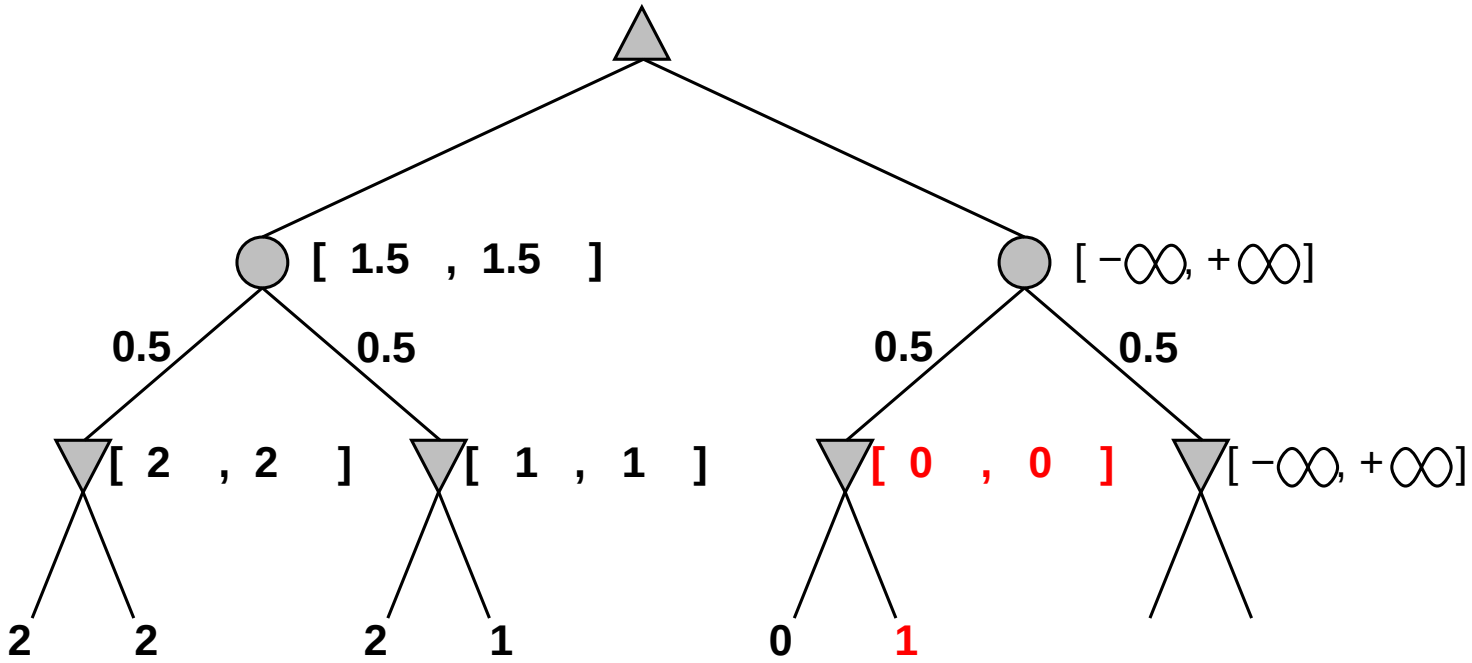
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



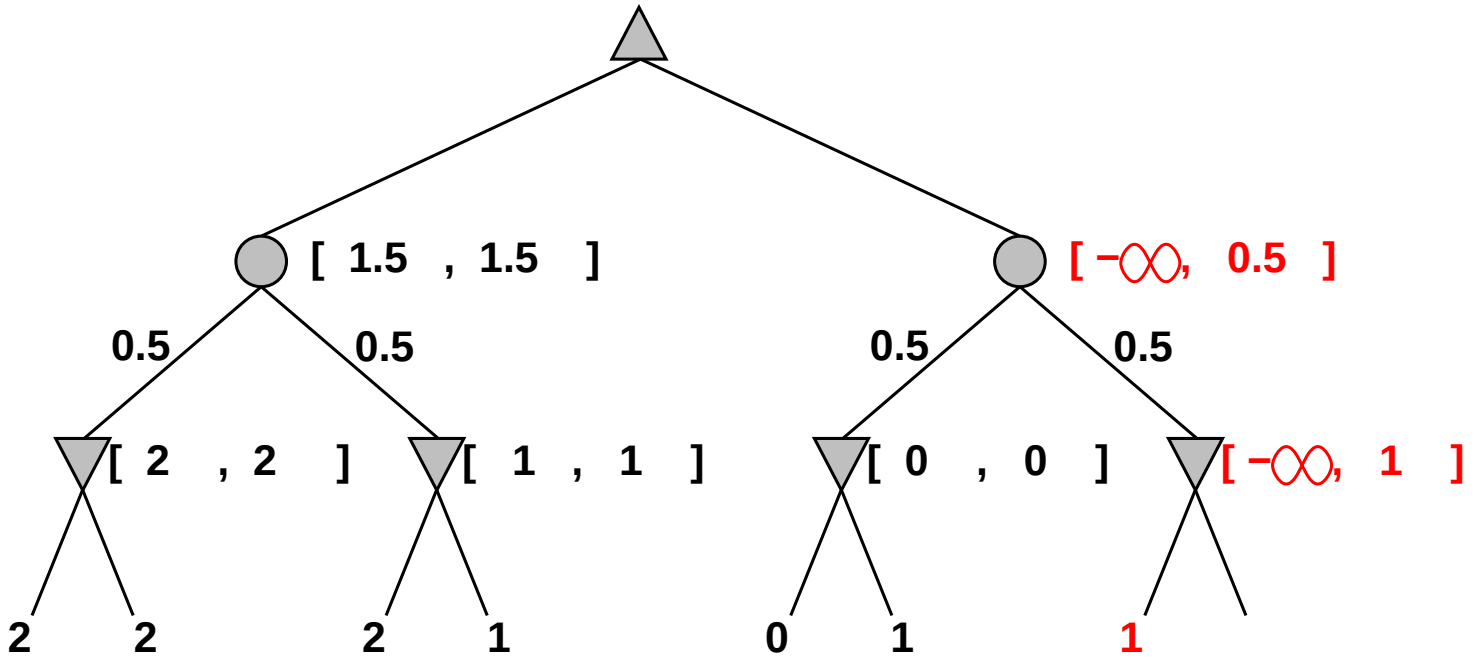
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



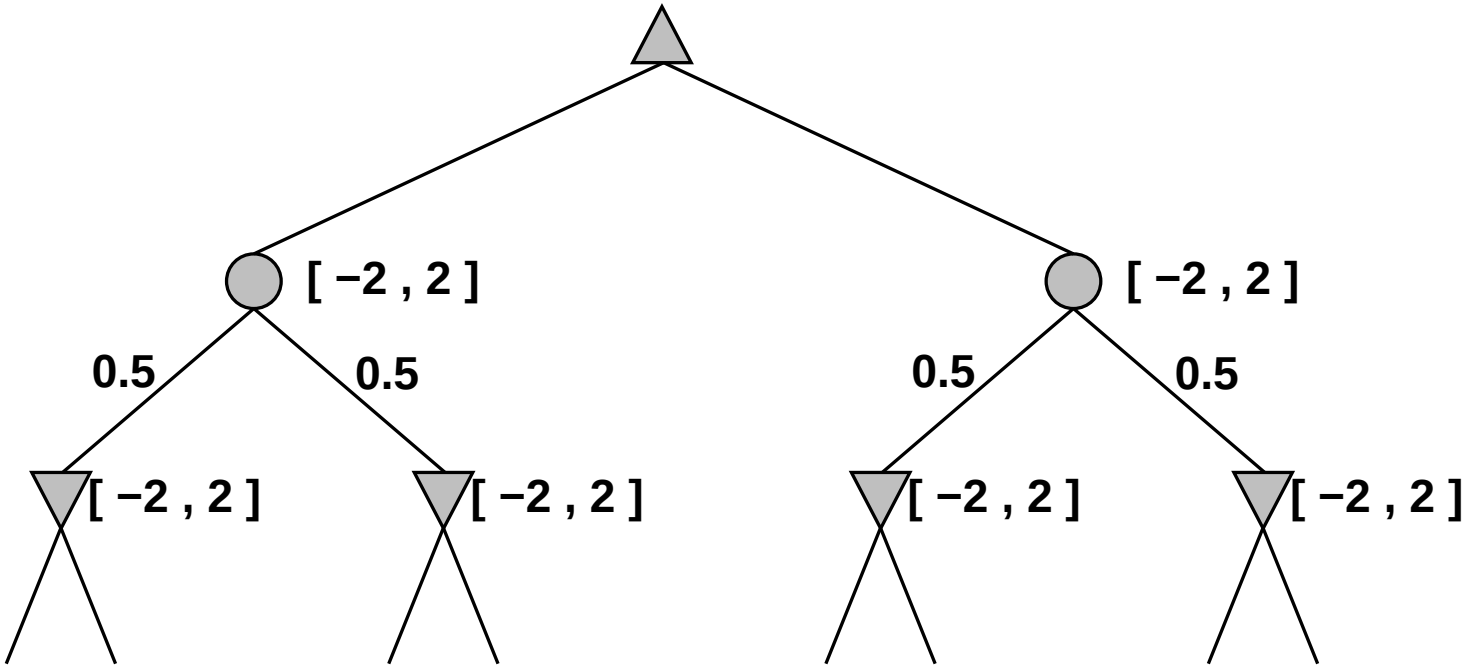
Potatura in alberi da gioco nondeterministici

Una versione di potatura in stile α-β è possible:



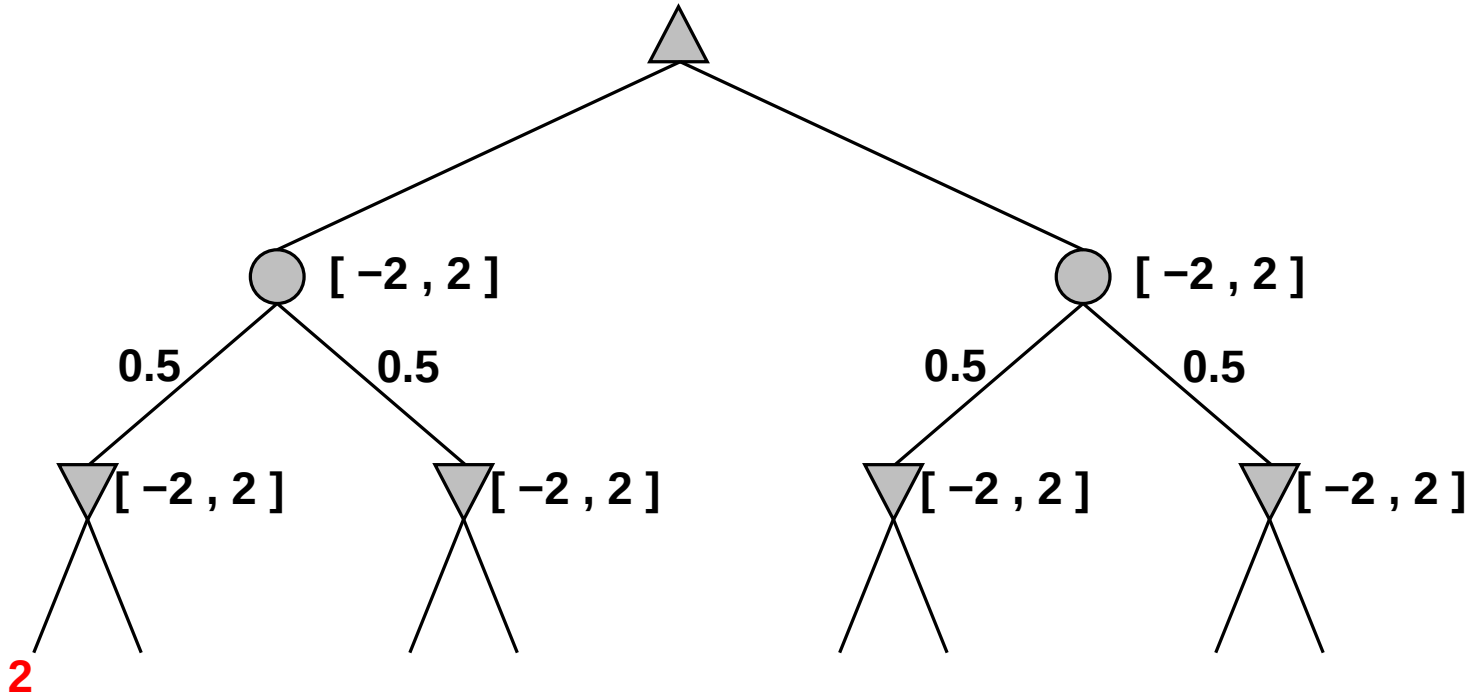
Potatura cont.

Una potatura maggiore può essere effettuata se possiamo limitare i valori delle foglie



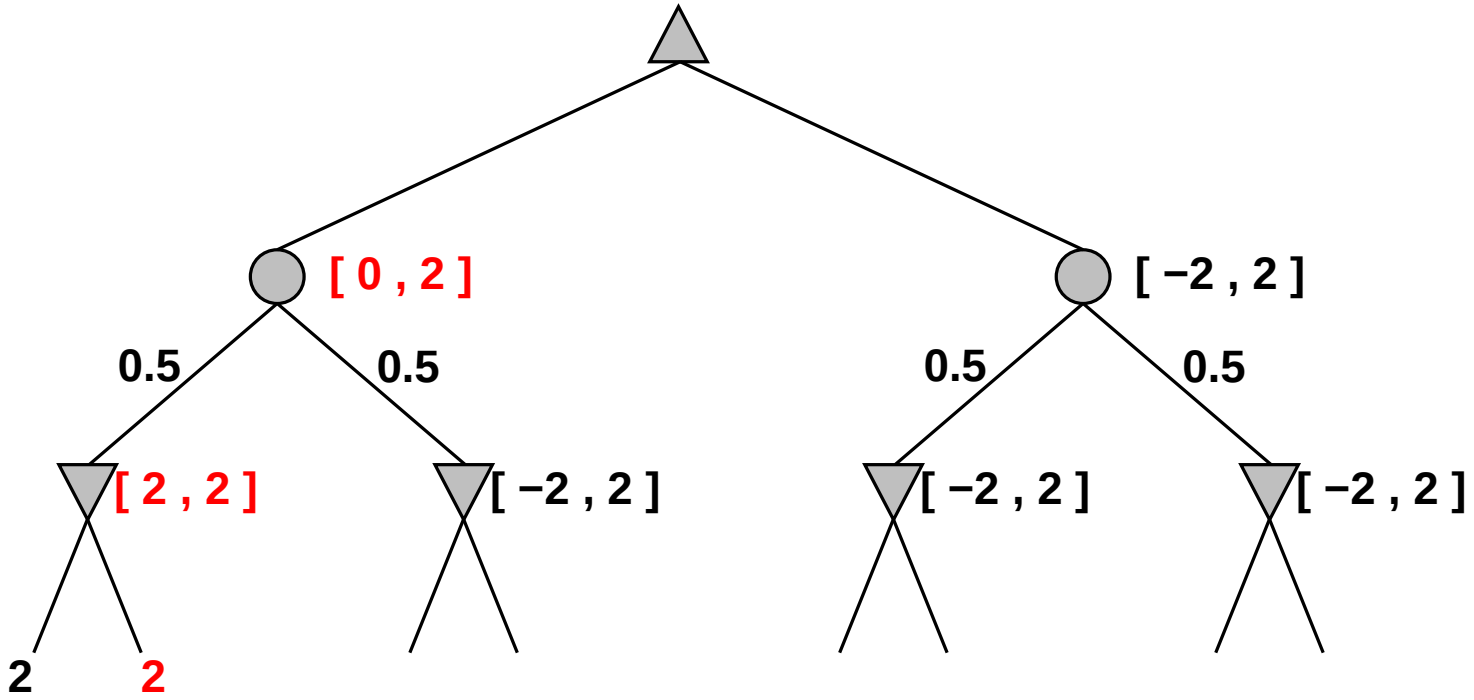
Potatura cont.

Una potatura maggiore può essere effettuata se possiamo limitare i valori delle foglie



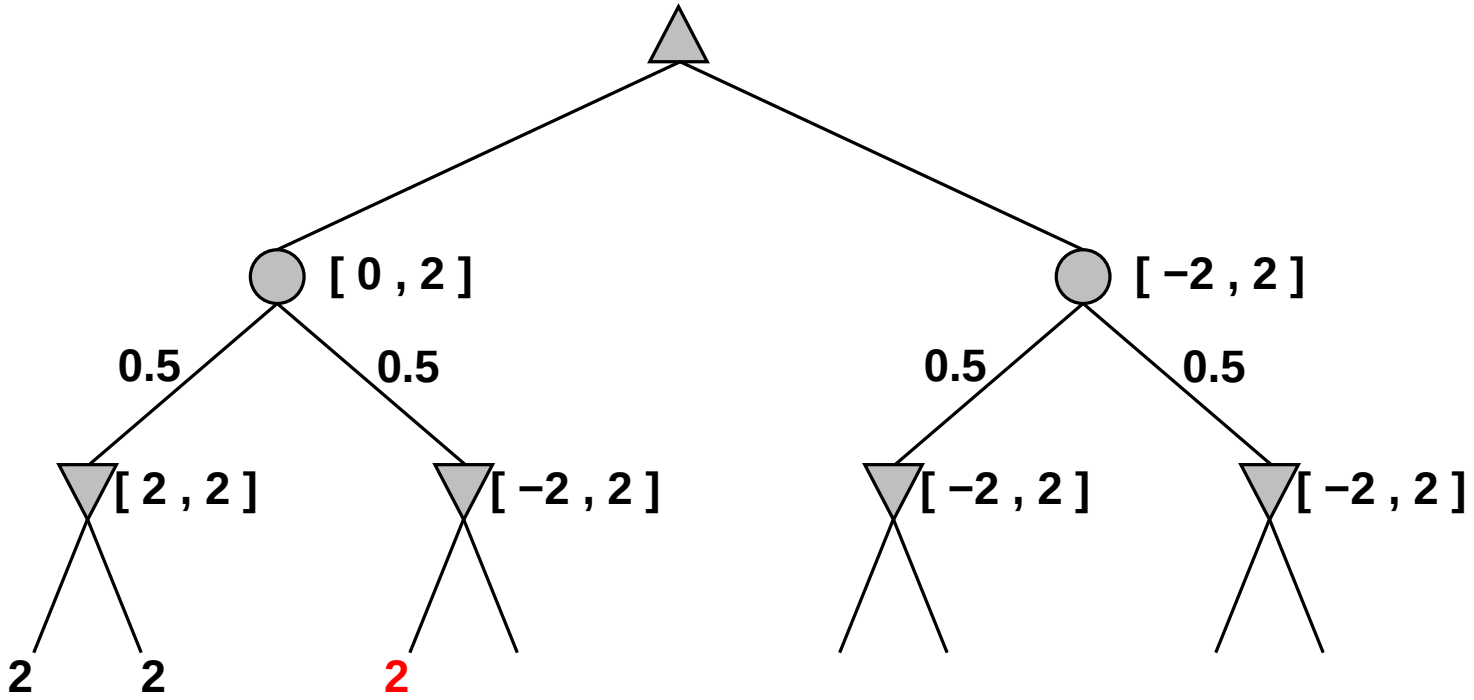
Potatura cont.

Una potatura maggiore può essere effettuata se possiamo limitare i valori delle foglie



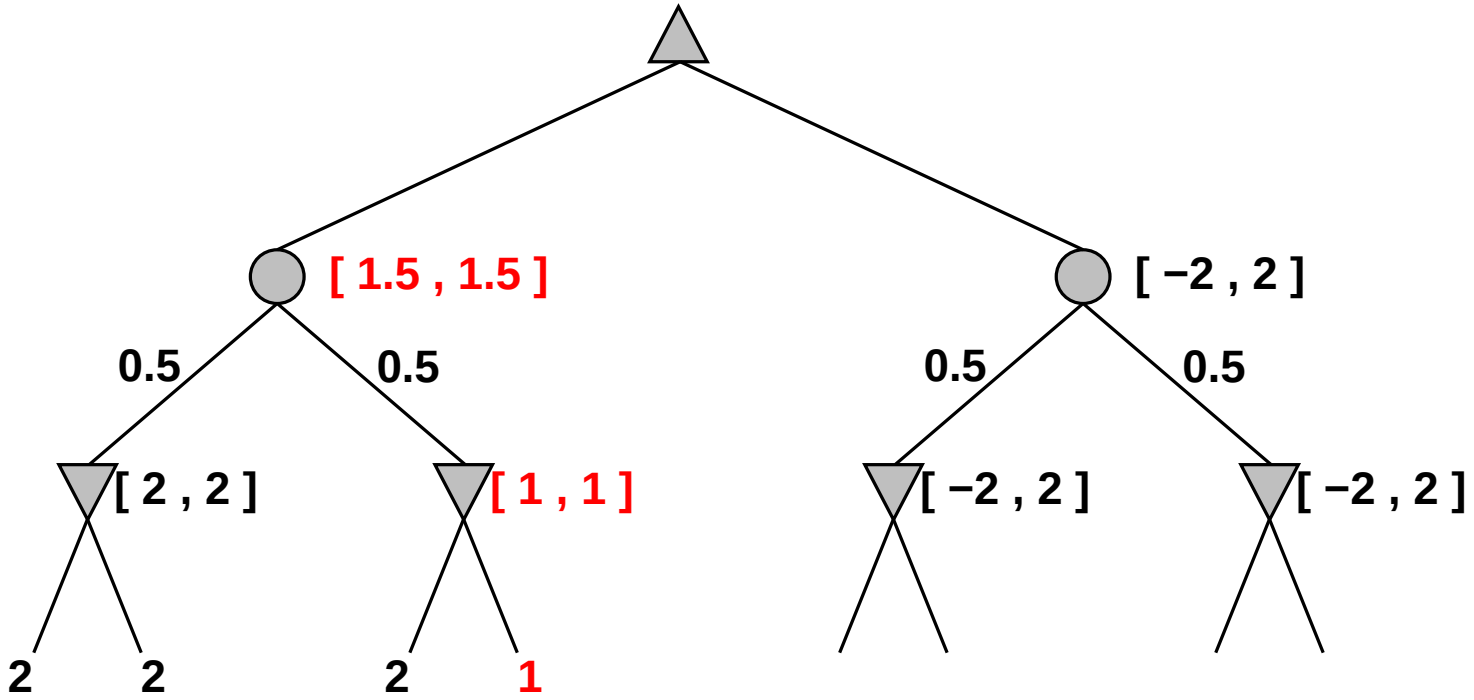
Potatura cont.

Una potatura maggiore può essere effettuata se possiamo limitare i valori delle foglie



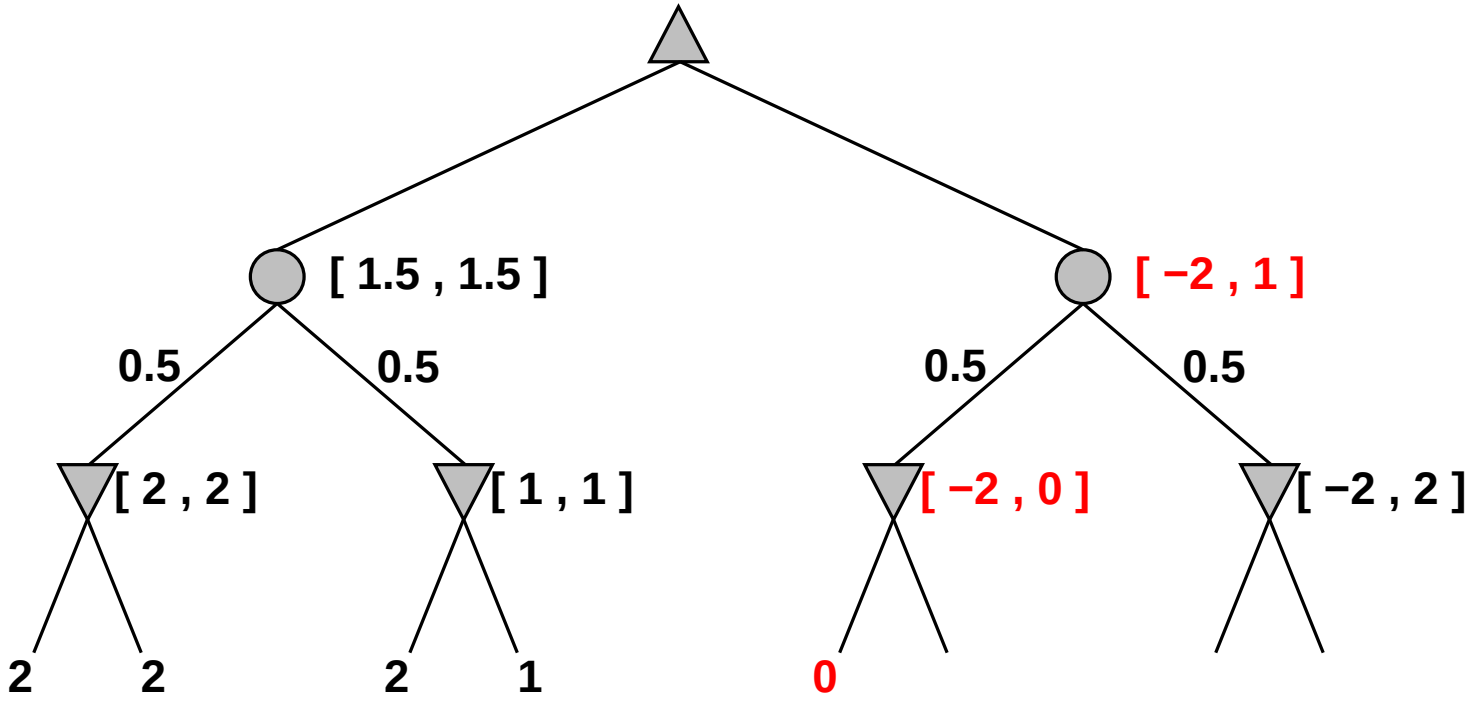
Potatura cont.

Una potatura maggiore può essere effettuata se possiamo limitare i valori delle foglie



Potatura cont.

Una potatura maggiore può essere effettuata se possiamo limitare i valori delle foglie



Giochi nondeterministici in pratica

L'uso dei dadi aumenta il valore di b: 21 possibili risultati con 2 dadi
Backgammon $\approx$ 20 mosse legali (fino a 6,000 con risultato 1-1)

$$\text{profondità } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

Con l'aumentare della profondità, la probabilità di raggiungere un dato nodo si riduce

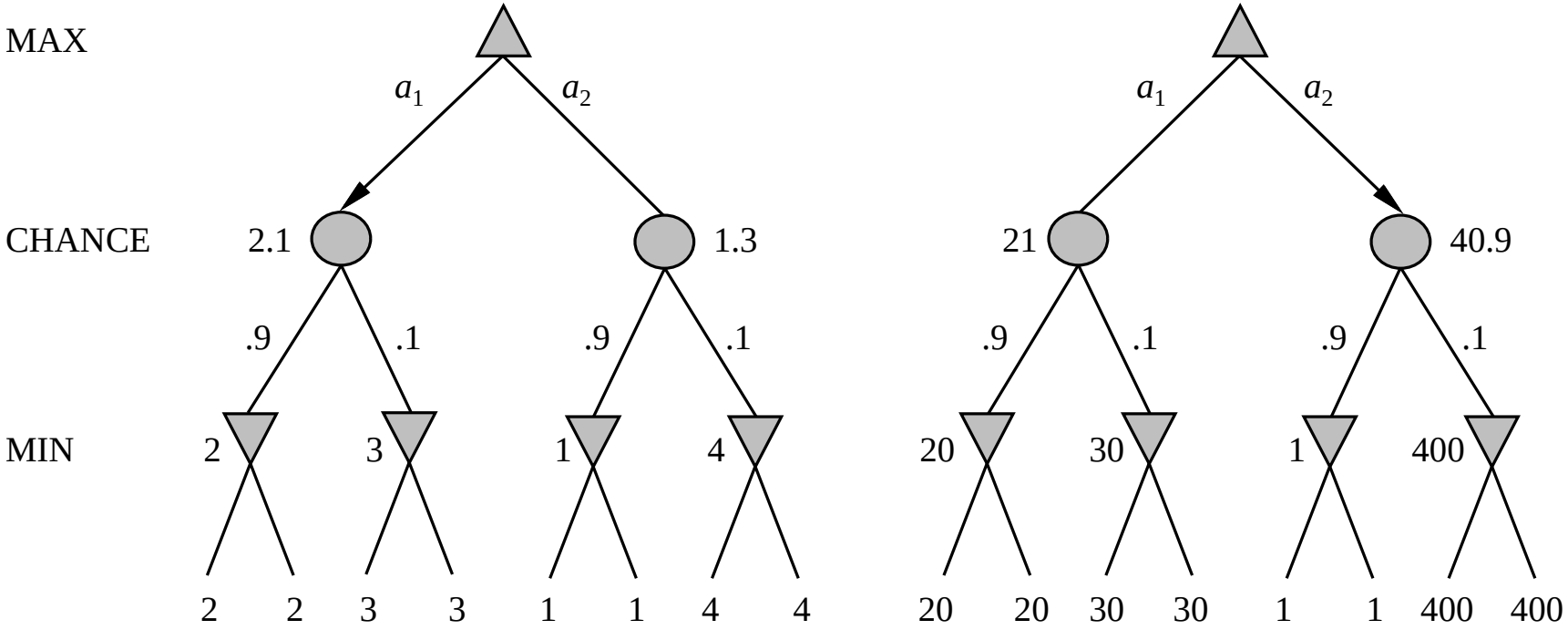
$\Rightarrow$ il valore di lookahead si riduce

potatura α–β è molto meno efficace

TDGammon usa una ricerca di profondità 2 + una funzione Eval molto buona

$\approx$ livello di campione mondiale

Sono IMPORTANTI i valori ESATTI



Il comportamento è preservato solo da trasformazioni *lineari e positive* di EVAL

Quindi EVAL dovrebbe essere proporzionale al valore aspettato di guadagno

Giochi ad informazione parziale

E.g., giochi di carte, dove le carte iniziali dell'avversario sono sconosciute

Tipicamente si può calcolare una probabilità per ogni possibile "smazzata"

Sembrerebbe proprio come se si avesse un dado con tantissime facce all'inizio del gioco*

Idea: calcolare il valore minimax di ogni azione in ogni smazzata,
quindi scegliere l'azione con il valore aspettato più alto su tutte le smazzate*

Caso speciale: se una azione è ottima per ogni partita, essa è ottima*

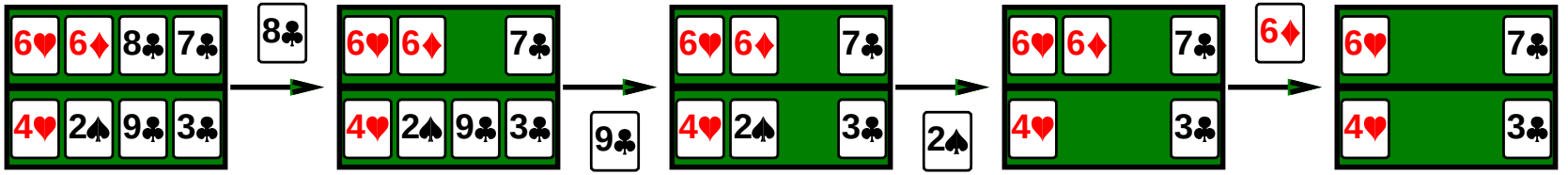
GIB, il miglior programma che gioca a bridge, approssima questa idea nel seguente modo:

1) genera 100 partite consistenti con informazione sulle dichiarazioni di presa

2) sceglie l'azione che vince più mani in media

Esempio

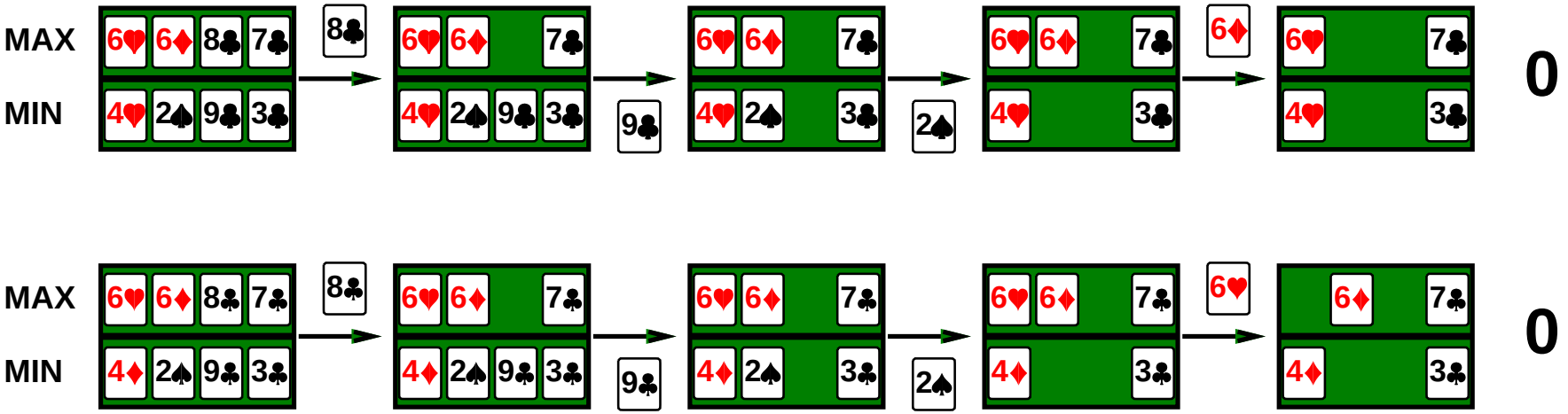
Bridge a 4 carte, MAX gioca per primo



0

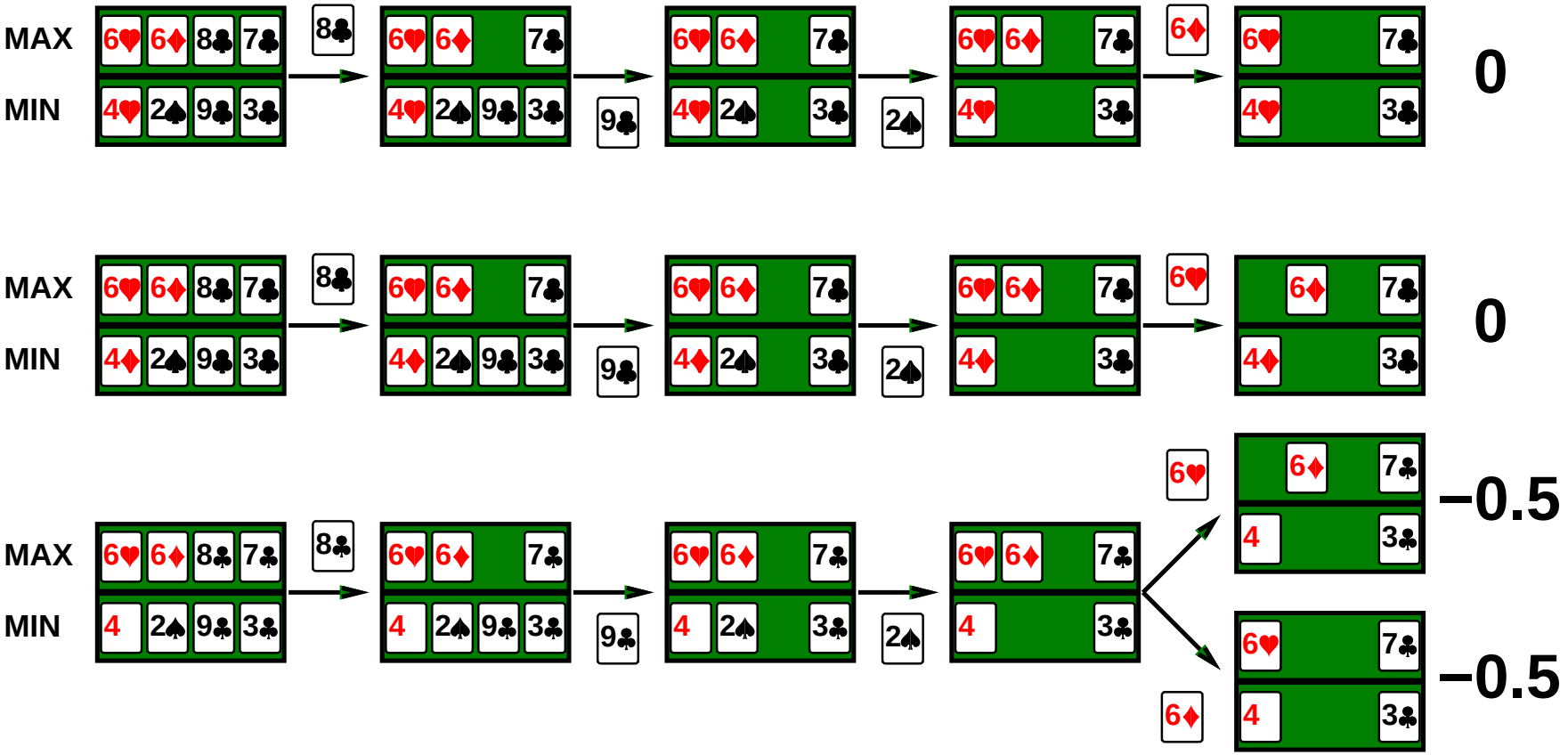
Esempio

Bridge a 4 carte, MAX gioca per primo



Esempio

Bridge a 4 carte, MAX gioca per primo



Esempio di senso comune

La strada A conduce ad un piccolo mucchio di pezzi d'oro
La strada B conduce ad un incrocio:
    se si prende la sinistra si trova una montagnola di gioielli;
    se si prende la destra si viene investiti da un autobus.

Esempio di senso comune

La strada A conduce ad un piccolo mucchio di pezzi d'oro
La strada B conduce ad un incrocio:
    se si prende la sinistra si trova una montagnola di gioielli;
    se si prende la destra si viene investiti da un autobus.

La strada A conduce ad un piccolo mucchio di pezzi d'oro
La strada B conduce ad un incrocio:
    se si prende la sinistra si viene investiti da un autobus;
    se si prende la destra si trova una montagnola di gioielli.

Esempio di senso comune

La strada A conduce ad un piccolo mucchio di pezzi d'oro
La strada B conduce ad un incrocio:
    se si prende la sinistra si trova una montagnola di gioielli;
    se si prende la destra si viene investiti da un autobus.

La strada A conduce ad un piccolo mucchio di pezzi d'oro
La strada B conduce ad un incrocio:
    se si prende la sinistra si viene investiti da un autobus;
    se si prende la destra si trova una montagnola di gioielli.

La strada A conduce ad un piccolo mucchio di pezzi d'oro
La strada B conduce ad un incrocio:
    se si indovina la strada giusta si trova una montagnola di gioielli;
    se non si indovina la strada giusta si viene investiti da un autobus.

Analisi corretta

* L'intuizione che il valore di ogni azione è la media dei suoi valori in tutti i possibili stati è SBAGLIATA

Nel caso di informazione non completa, il valore di una azione dipende dallo stato di informazione o stato di credenza (belief state) in cui si trova l'agente

E' possibile generare e ricercare all'interno di un albero di stati di credenza

Conduce a comportamenti razionali quali

- ♢ agire con lo scopo di ottenere informazione
- ♢ trasmettere informazione al proprio compagno di gioco
- ♢ agire in modo casuale per minimizzare la perdita di informazione (rivelare informazione agli avversari)

Riassunto

I giochi illustrano molti aspetti importanti dell'Intelligenza Artificiale

- ♢ non si può ottenere la perfezione ⇒ si deve approssimare
- ♢ l'incertezza vincola l'assegnamento di valori a stati