UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Intelligenza Artificiale

Ricerca con avversari

a.a. 2022/2023

Giochi

- Ambienti multiagente: dobbiamo considerare le azioni degli altri agenti e i loro effetti
- Ambienti competitivi: gli obiettivi degli agenti sono in conflitto
- Ricerca con avversari = giochi
- Teoria matematica dei giochi: ambiente multi-agente è un gioco (interazione competitiva o cooperativa)
- Giochi più comuni in AI: giochi a somma zero con informazione perfetta, a turni e a due giocatori
  - Ambienti deterministici, completamente osservabili
  - Valori di utilità uguali ma di segno opposto alla fine

I giochi con avversario

- Regole semplici e formalizzabili
- ambiente multi-agente: la presenza dell'avversario rende il problema *contingente* $\Rightarrow$ più difficile rispetto ai problemi di ricerca visti fino ad 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' più simili ai problemi reali

Tipi di Giochi

Outline

- Decisioni ottime
  - come si sceglie la mossa migliore in un gioco con uno spazio di ricerca limitato
- Tecniche di ottimizzazione della ricerca
  - Potatura dell'albero di ricerca
  - Euristiche per approssimare l'utilità di uno stato
- Estensione a giochi più complessi
  - Stocastici

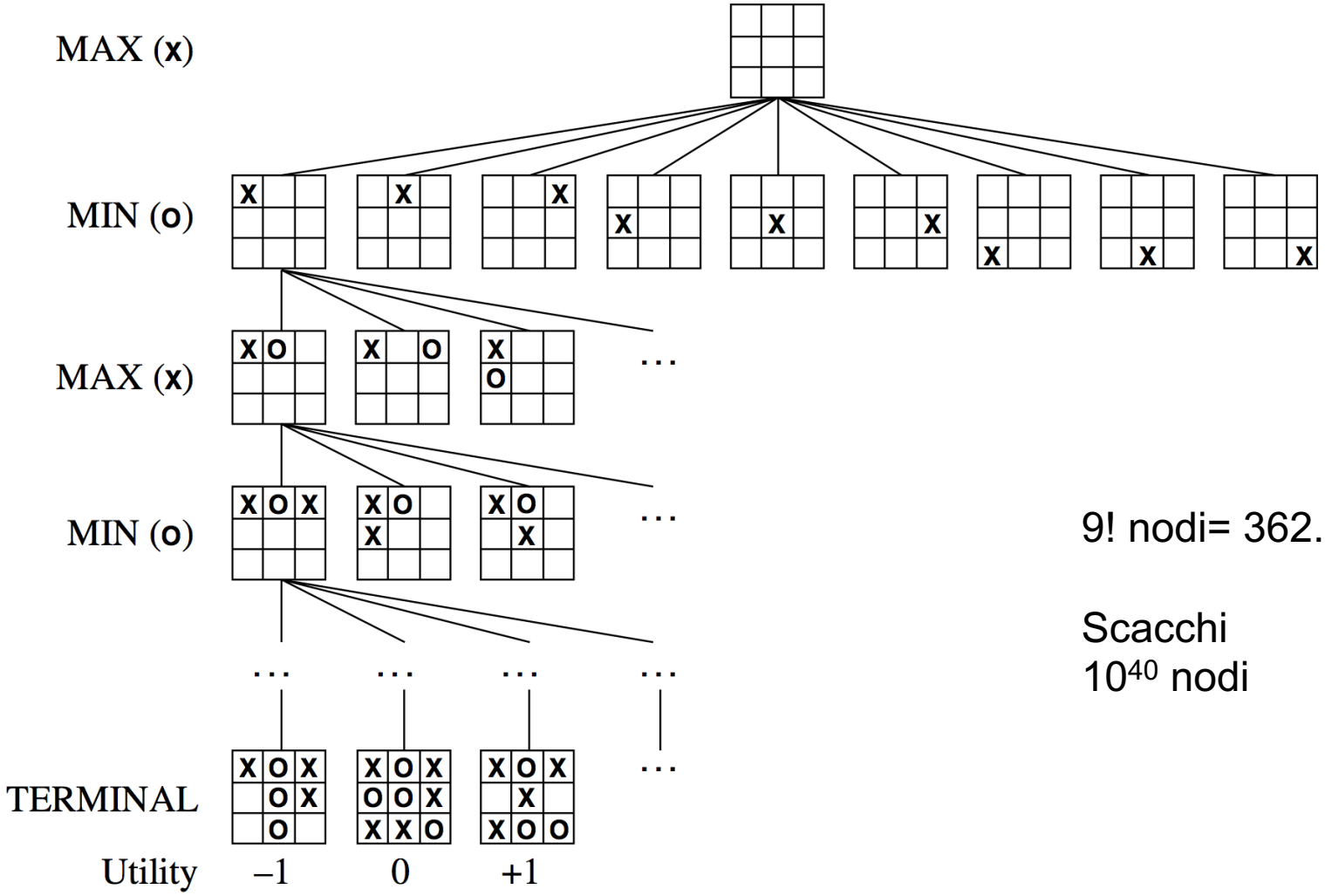
Giochi come problemi di ricerca

- Per ora due giocatori: **MIN** e **MAX**
- **MAX** muove per prima, poi un turno per uno
- ***Stato iniziale s***: configurazione iniziale del gioco
- **GIOCATORE(s)**: il giocatore a cui tocca muovere nello stato *s*
- **AZIONI(s)**: insieme delle mosse lecite nello stato *s*
- **RISULTATO(s,a)**: risultato di una mossa
- **TEST-TERMINAZIONE(s)**: vero se la partita è finita (stato terminale)
- **UTILITA(s,p)**: funzione di utilità (o obiettivo), valore per giocatore **p** in stato terminale **s**. Es. scacchi: vittoria (+1), sconfitta (0), o pareggio (1/2). Backgammon da 0 a 192.
- Gioco a somma zero: payoff totale è lo stesso per ogni istanza del gioco. Es. scacchi: 0+1, o 1+0, o 1/2 + 1/2
- **Albero di gioco:** stato iniziale, azioni e risultato (nodi=stati, archi=mosse)

Il gioco del Tris

- Due giocatori: MAX e MIN
- Nello stato iniziale, MAX ha nove possibili mosse
- MAX mette una X e MIN mette una O
- Finché non si raggiunge un nodo foglia (stato terminale): un giocatore ha fatto tris o tutte le caselle sono riempite
- Per le foglie, valore di utilità (per MAX): buono per MAX e cattivo per MIN

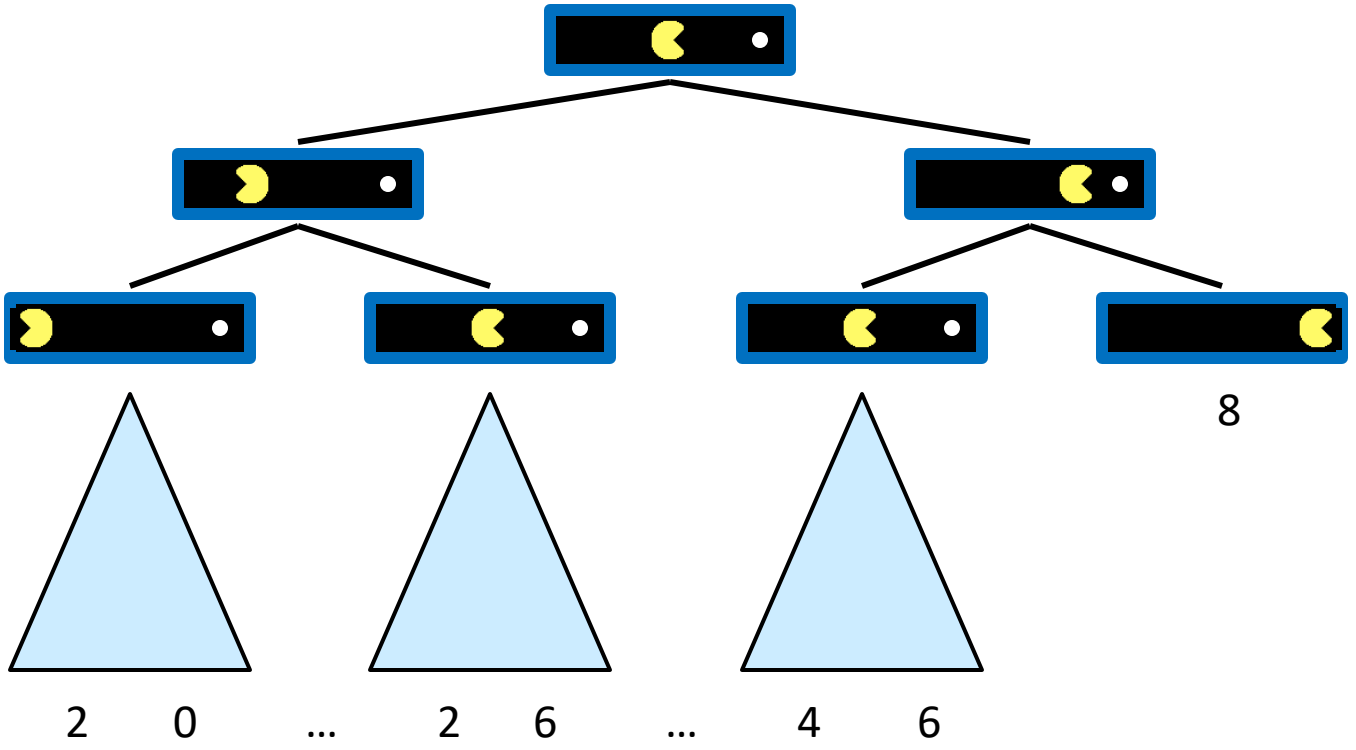
Albero di gioco



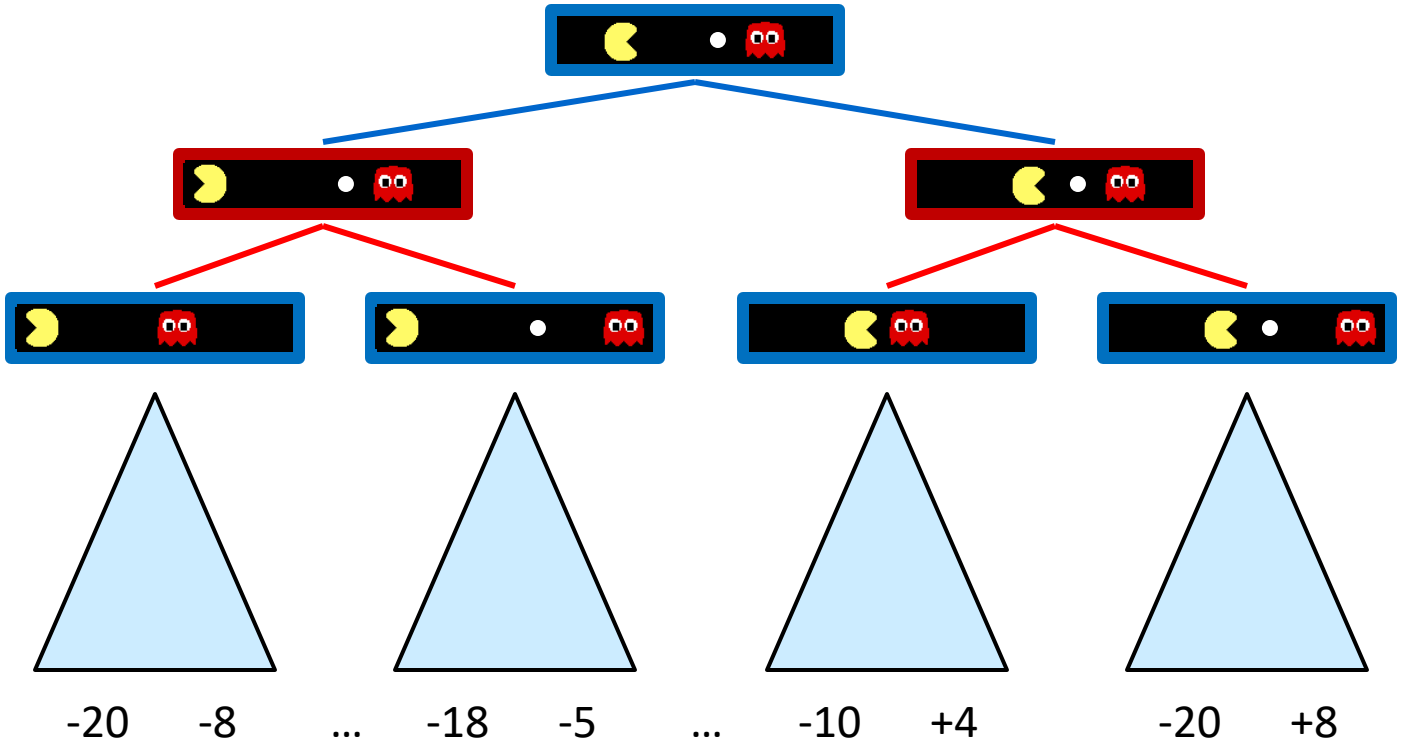
9! nodi= 362.880

Scacchi
10^{40} nodi

Single-Agent Pacman



Pacman con avversario



Minimax

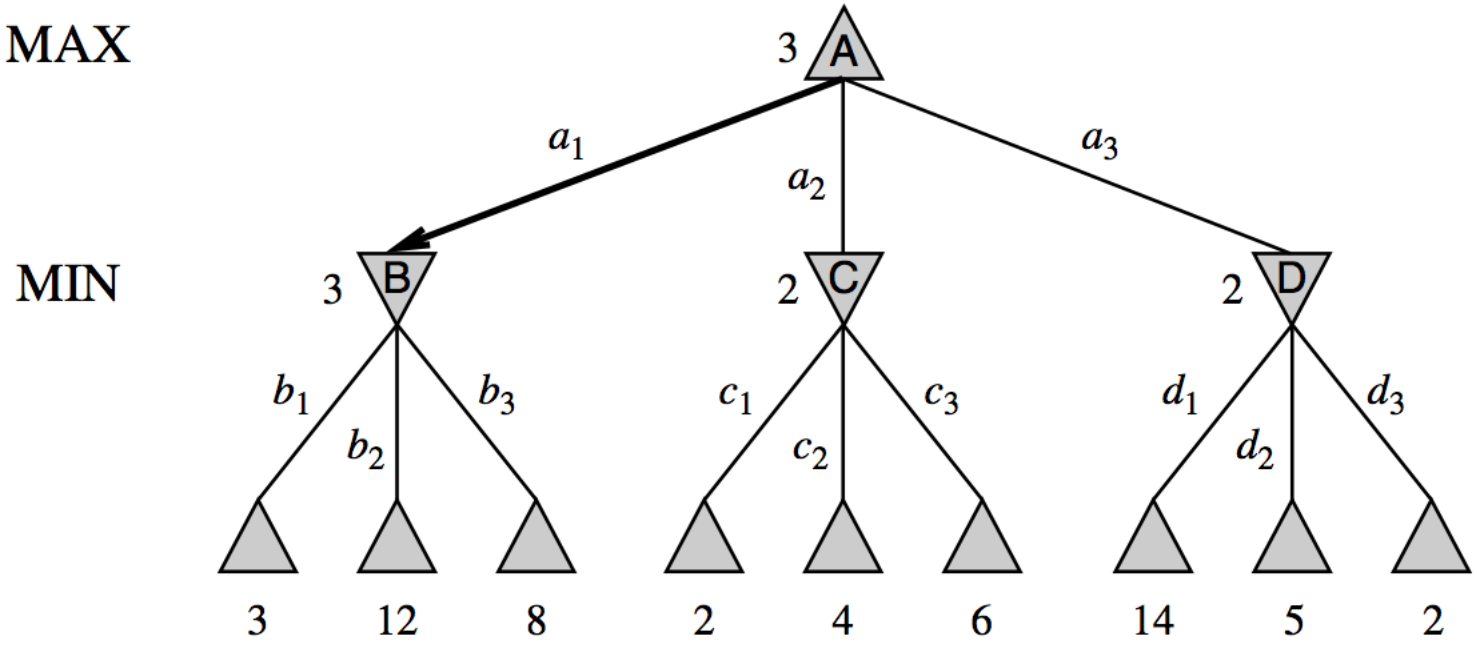
- Albero di gioco vs. albero di ricerca (numero di nodi sufficienti per decidere la mossa)
- Non dobbiamo trovare la strategia ottima per arrivare all'obiettivo
- MAX deve prendere MIN in considerazione
- Algoritmo Minimax per scegliere la mossa ottima, assumendo avversario infallibile
- Va bene per 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

Minimax

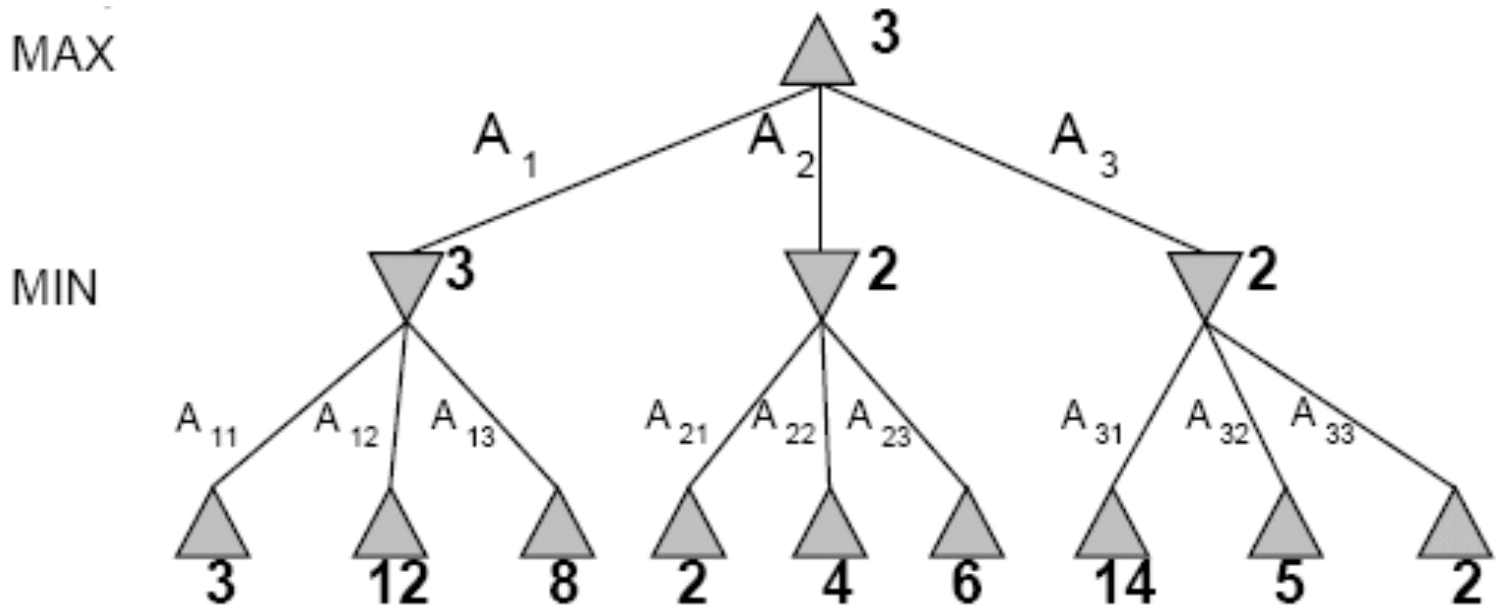
- MINIMAX(n): utilità per MAX in nodo n, assumendo che i due agenti giochino in modo ottimo da li alla fine (nodo terminale: utilità)
- MAX va verso valori alti, MIN verso valori bassi
- MINIMAX(n):
  - UTILITA(s, MAX) se TEST-TERMINALE(s)
  - Per MAX: $max_{a \in azioni(s)}$VALORE-MINIMAX-RISULTATO(s, a)
  - Per MIN: $min_{a \in azioni(s)}$VALORE-MINIMAX-RISULTATO(s, a)
- Massimizza il risultato di MAX nel caso pessimo (e minimizza il risultato di MIN nel caso pessimo)
- Esplorazione completa in profondità dell'albero di gioco

Minimax per gioco a due strati

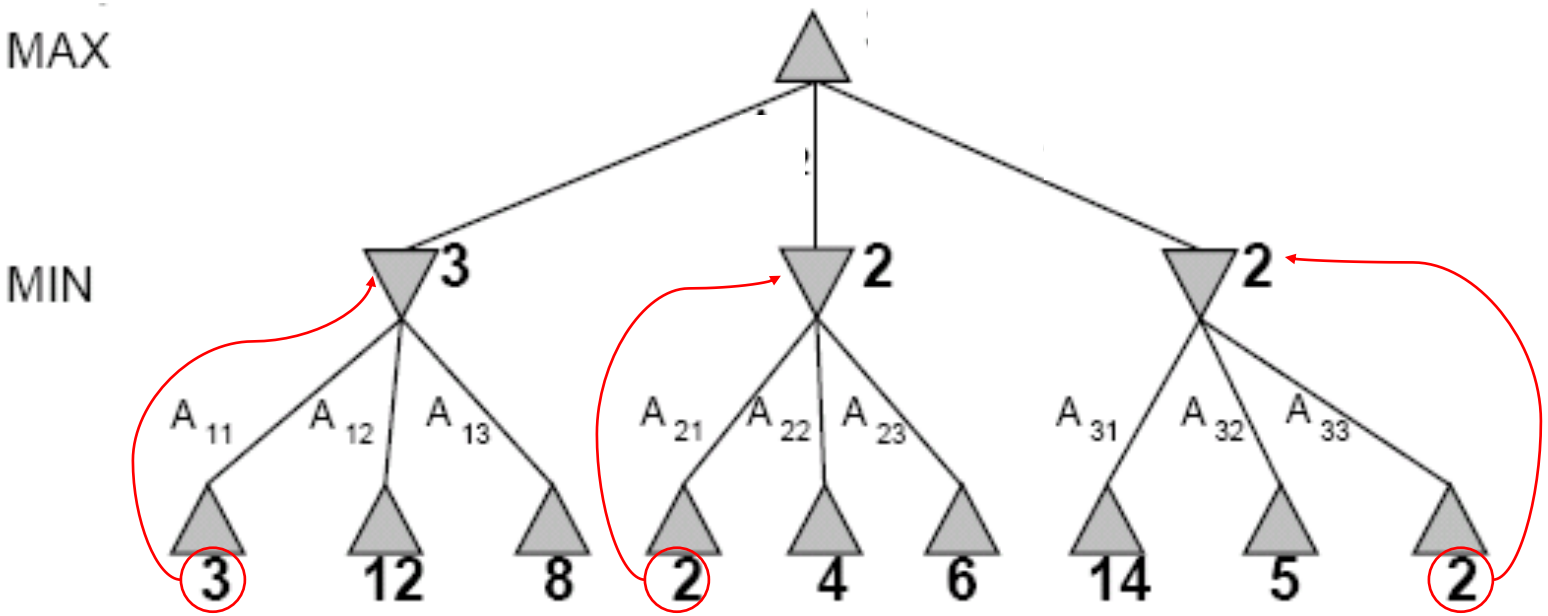
- Tre mosse per MIN e tre per MAX
- Finisce dopo una mossa di MAX e una di MIN



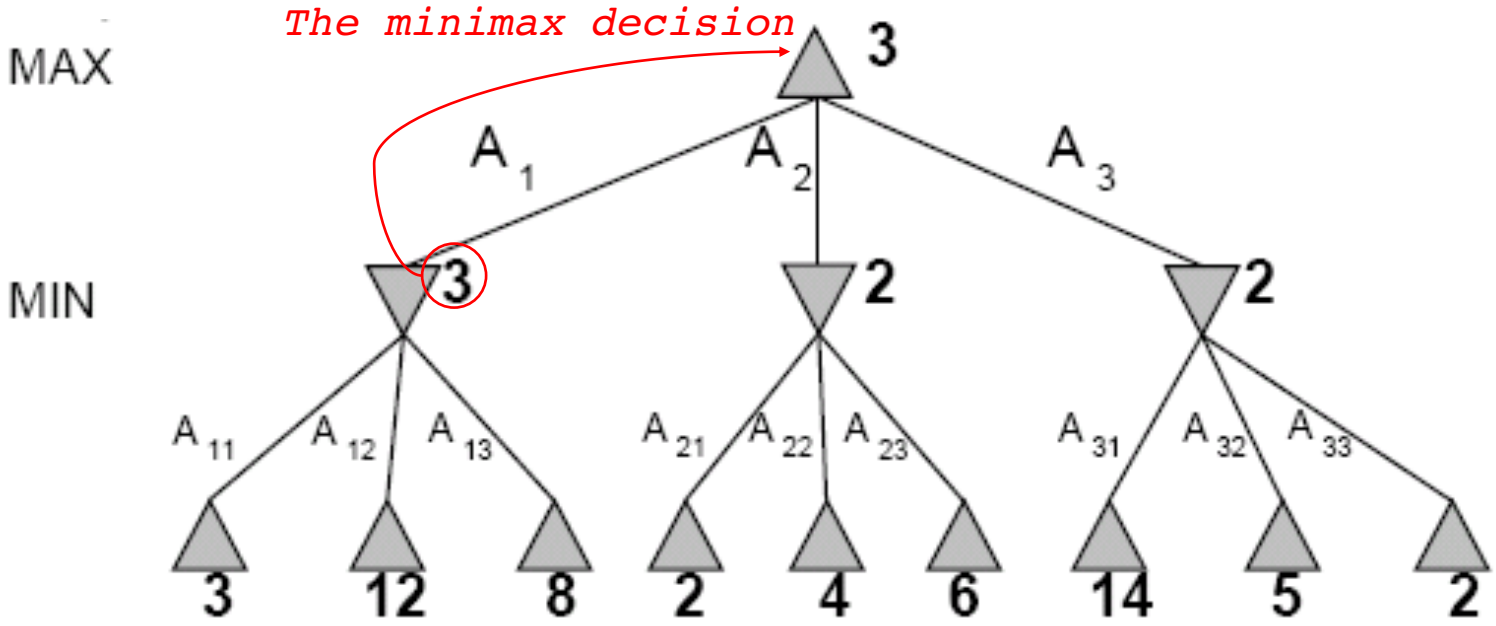
Minimax per gioco a due strati



Minimax per gioco a due strati



Minimax per gioco a due strati



Proprietà di Minimax

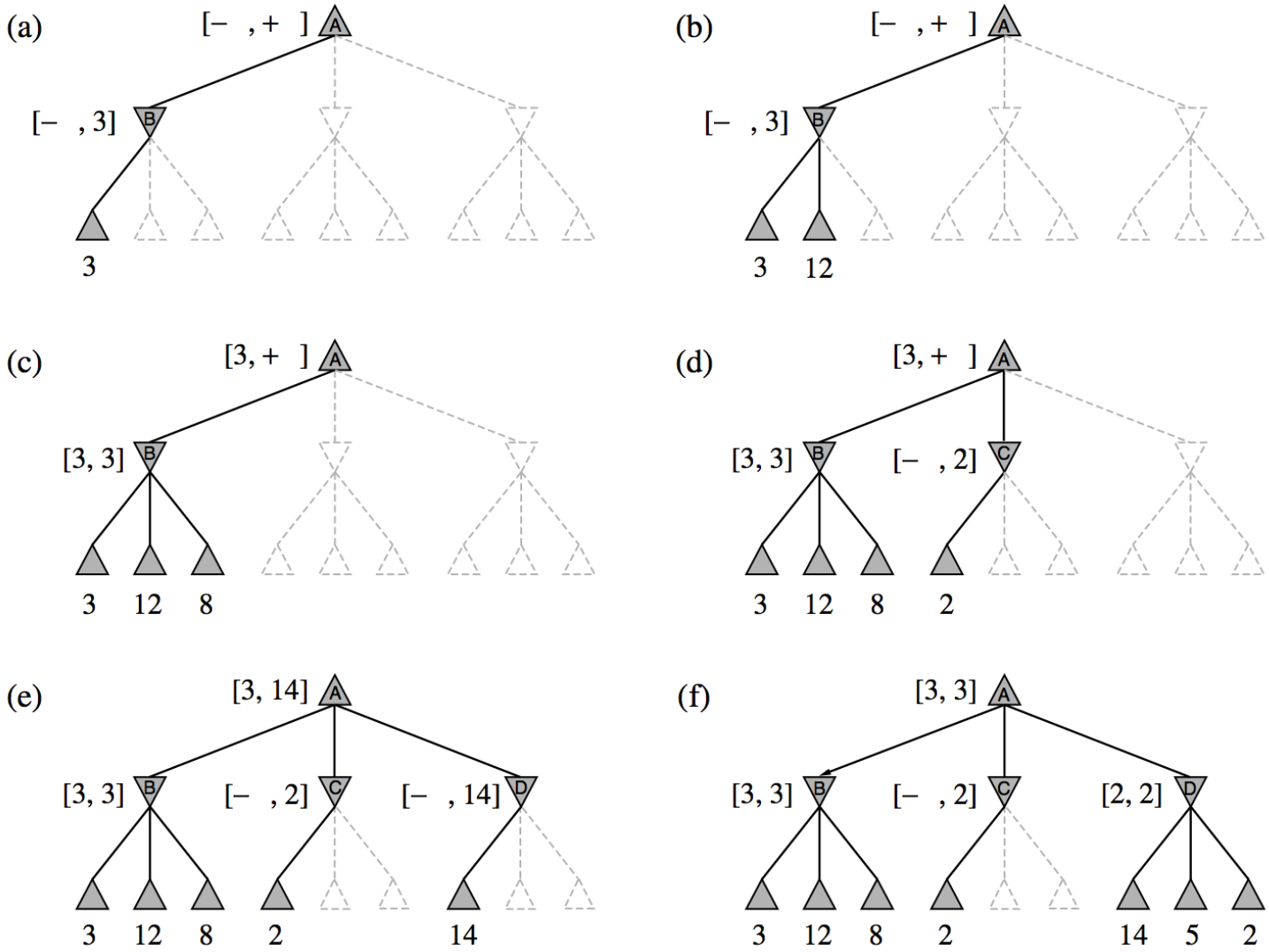
- Completezza?? Si, solo se l'albero è finito (il gioco degli scacchi ha regole specifiche per garantire il termine del gioco)
- Ottimalità?? Si, contro un avversario ottimo. Altrimenti??
- Complessità di tempo?? $O(b^m)$
- Complessità di spazio?? O(bm) (esplorazione depth-first)

- Per il gioco degli scacchi, $b \approx 35$, $m \approx 80$ => $\approx 10^{123}$
- per giochi "ragionevoli" ⇒ una soluzione esatta è sicuramente non fattibile

Potatura alfa-beta

- Numero di nodi esaminati da MINIMAX: cresce esponenzialmente con la profondità dell'albero
- Possiamo dimezzare l'esponente
- Potiamo alcuni rami dell'albero, che non ci servono per calcolare i valori minimax
- α: valore della scelta migliore per MAX che abbiamo trovato fino a qui in un qualunque punto di scelta lungo il cammino
- β: lo stesso per MIN
- Aggiorniamo α e β potando i rami restanti che escono da un nodo non appena ci accorgiamo che il valore del nodo è peggio di α (per MAX) o di β (per MIN)

Potatura alfa-beta: esempio



Potatura alfa-beta

- La potatura non modifica il risultato finale
- Un buon ordinamento delle mosse migliora l'efficacia della potatura
- Con "ordine perfetto" (guardo prima i figli più promettenti), complessità in tempo = $O(b^{m/2})$
  - => raddoppia la profondità di ricerca
  - => può facilmente raggiungere profondità 8 e giocare bene a scacchi

Limiti alle risorse

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

  => si riescono ad esplorare 10^6 nodi per mossa

- Anche la potatura può non essere sufficiente

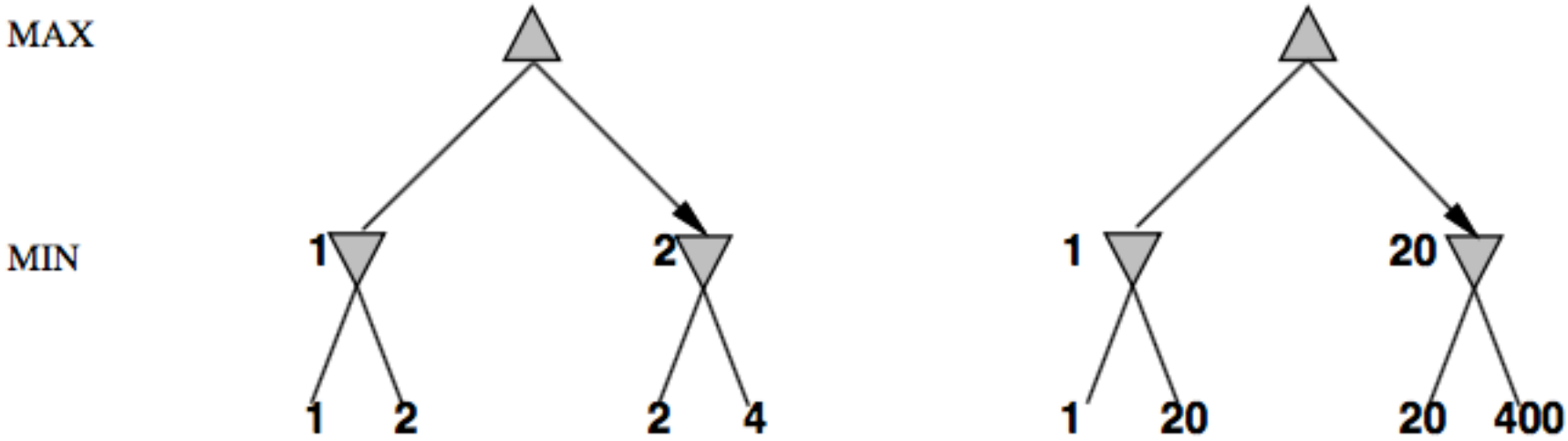
- Approccio standard:
  - Test di taglio (cutoff)
    - Cioè limite alla profondità
  - Funzione di valutazione (Eval)
    - stima del guadagno atteso in una certa posizione

Funzioni di valutazione

- Proprietà
  1. Deve ordinare gli stati terminali come la vera funzione di utilità
  2. Non deve richiedere troppo tempo
  3. Per stati non terminali, forte correlazione con la probabilità di vincere (incertezza perché ricerca interrotta prima di guardare tutto l'albero)
- Spesso basate sulle caratteristiche di uno stato (es: numero di pedoni bianchi, regine bianche, alfieri bianchi, etc) => classi di equivalenza di stati
- Ogni classe di equivalenza può contenere stati che portano alla vittoria, al pareggio e alla sconfitta => la funzione di valutazione riflette la proporzione di stati che portano ad un certo risultato
- Esempio (scacchi): 72% porta a vittoria (+1), 20% a sconfitta (-1), 8% a pareggio => valore atteso (0.72 x 1) + (0.20 x 0) + (0.8 x 1/2) = 0.76
- Anche se non calcola esattamente il valore atteso, almeno non deve alterare l'ordinamento tra stati
- Spesso, troppe classi di equivalenza => un valore per ogni caratteristica, e poi combinazione

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

Non importa calcolare i 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

Minimax con decisione imperfetta

- *Strategia*: guardare avanti *k* mosse
  - Si espande l'albero di ricerca un certo numero di livelli k (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:

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN.} \end{cases}$$

La funzione di valutazione

- La funzione di valutazione *EVAL* 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 …

- Funziona in pratica?
- Consideriamo $b^m=10^6$, b = 35 => m = 4
- 4-strati lookhead corrisponde ad un giocatore di scacchi pessimo
- 4-strati ≈ umano a livello di novizio
- 8-strati ≈ prestazione di un PC, umano a livello di maestro
- 12-strati ≈ Deep Blue, Kasparov

Altri miglioramenti

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

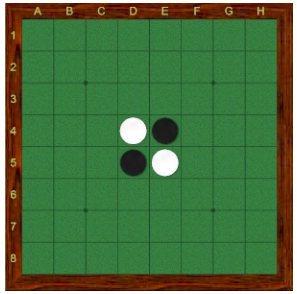
Giochi deterministici in pratica

- **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.

  Dal 2007 gioca in modo perfetto utilizzando la ricerca alpha-beta e un DB contenente 39mila miliardi di posizioni finali

- **Scacchi**: Deep Blue ha battuto il campione mondiale Cary Kasparov nel 1997 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 strati

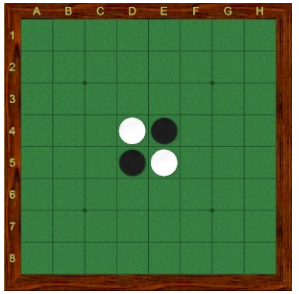
Giochi deterministici in pratica



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

- **Go**:

Giochi deterministici in pratica



- **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.*" **RN 2° edizione - 2002**
- *"oggi i migliori programmi giocano la maggior parte delle mosse a livello dei maestri; l'unico problema è che nel corso di una partita solitamente commettono almeno uno sbaglio grossolano che consente a un avversario forte di vincere"* **RN 3° edizione-2009**
- *AlphaGo di Google batte 4-1 campione mondiale Go* - 2016

Ricerca ad albero di Monte Carlo

- La ricerca alfa-beta non è utilizzabile per il gioco del Go
  - Fattore di ramificazione >361 (visita limitata a 4 o5 strati)
  - Difficile definire una buona funzione di valutazione
- Strategia di ricerca ad albero di Monte Carlo
  - Stima il valore di uno stato **s** come utilità media su un certo numero di **simulazioni** di partite complete che iniziano da **s**
  - Una simulazione sceglie prima le mosse per un giocatore poi per l'altro, fino ad arrivare ad uno stato terminale
  - Per i giochi con vittoria e sconfitta l'utilità media coincide con percentuale di vittoria

Ricerca ad albero di Monte Carlo

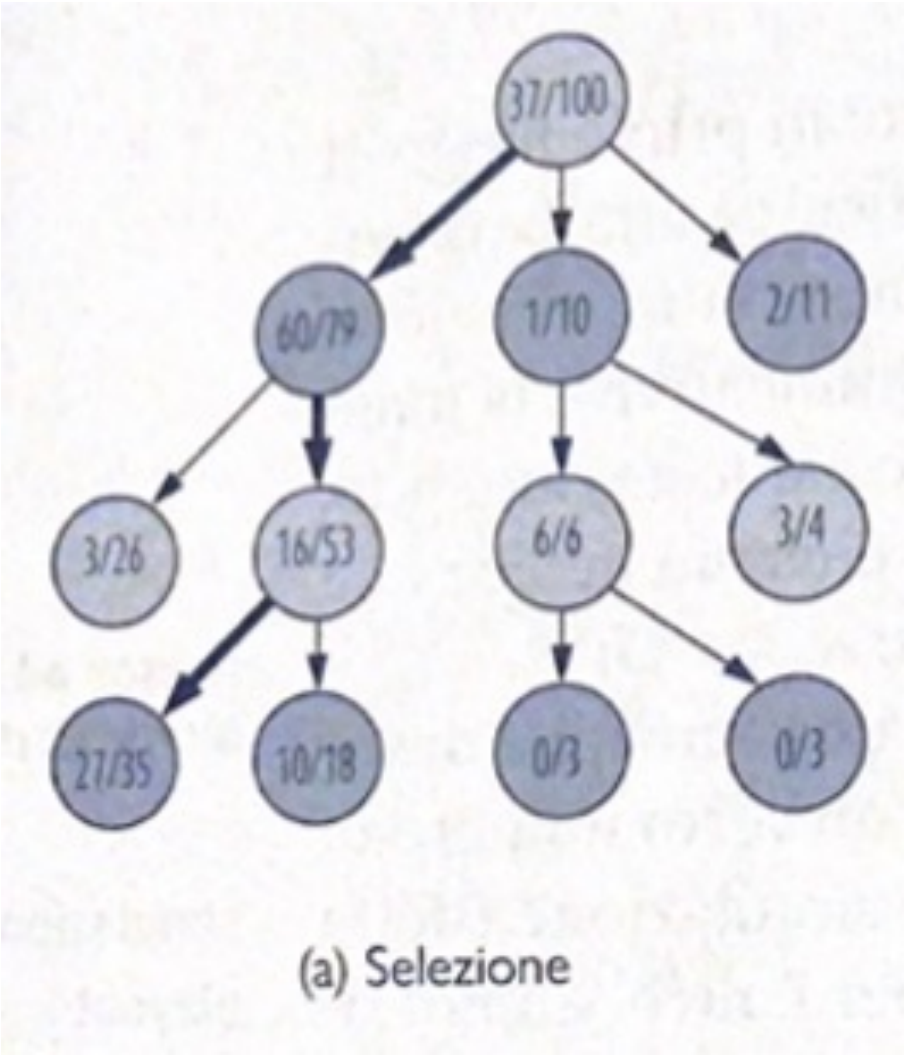
- Come si fanno a scegliere le mosse durante la simulazione?
  - Casuale? **NO**
  - Politica di simulazione che introduca una distorsione verso le mosse migliori
    - Le politiche di simulazione sono state apprese facendo giocare il programma contro se stesso e con l'uso di reti neurali
- Da quali posizioni iniziare le simulazioni?
- Quante esecuzioni effettuare per ogni posizione?
  - **Ricerca Monte Carlo pura**: N simulazioni dallo stato corrente e si determina quale delle mosse possibili nella posizione corrente ha la più alta percentuale di vittoria

Ricerca ad albero di Monte Carlo

- Convergenza al gioco ottimo al crescere di N
  - Non sufficiente per molti giochi
  - Necessaria una politica di selezione
- Bilanciamento di due fattori
  - **Esplorazione** di stati per cui sono state fatte poche simulazioni
  - **Sfruttamento** di stati in cui le precedenti simulazioni hanno prodotto buoni risultati (per avere stime più precise)
- L'albero di ricerca cresce ad ogni iterazioni con le operazioni:
  - Selezione
  - Espansione
  - Simulazione
  - Retropropagazione

Ricerca ad albero di Monte Carlo

1. Selezione
   - Sfruttamento avendo selezionato uno stato con 27/35 di vittorie
   - Esplorazione se si sceglie uno stato con poche simulazioni

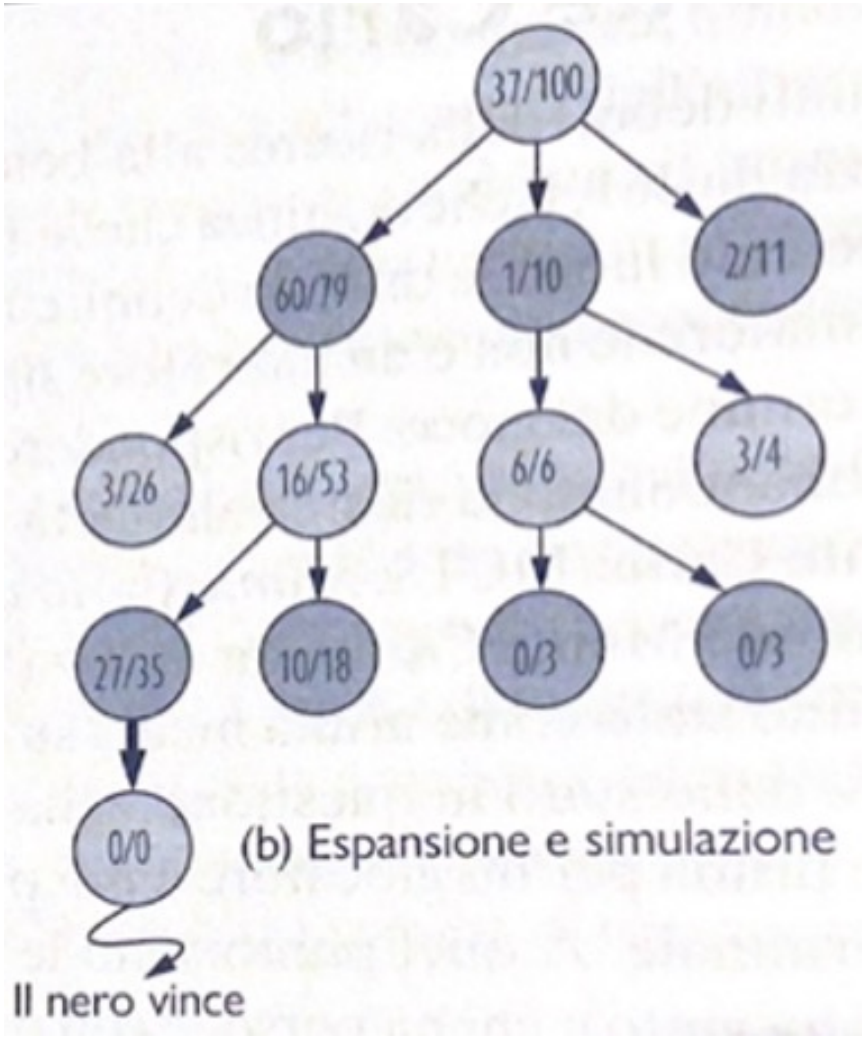


(a) Selezione

Ricerca ad albero di Monte Carlo

2. Espansione
   - Facciamo crescere l'albero generando un nuovo figlio del nodo selezionato

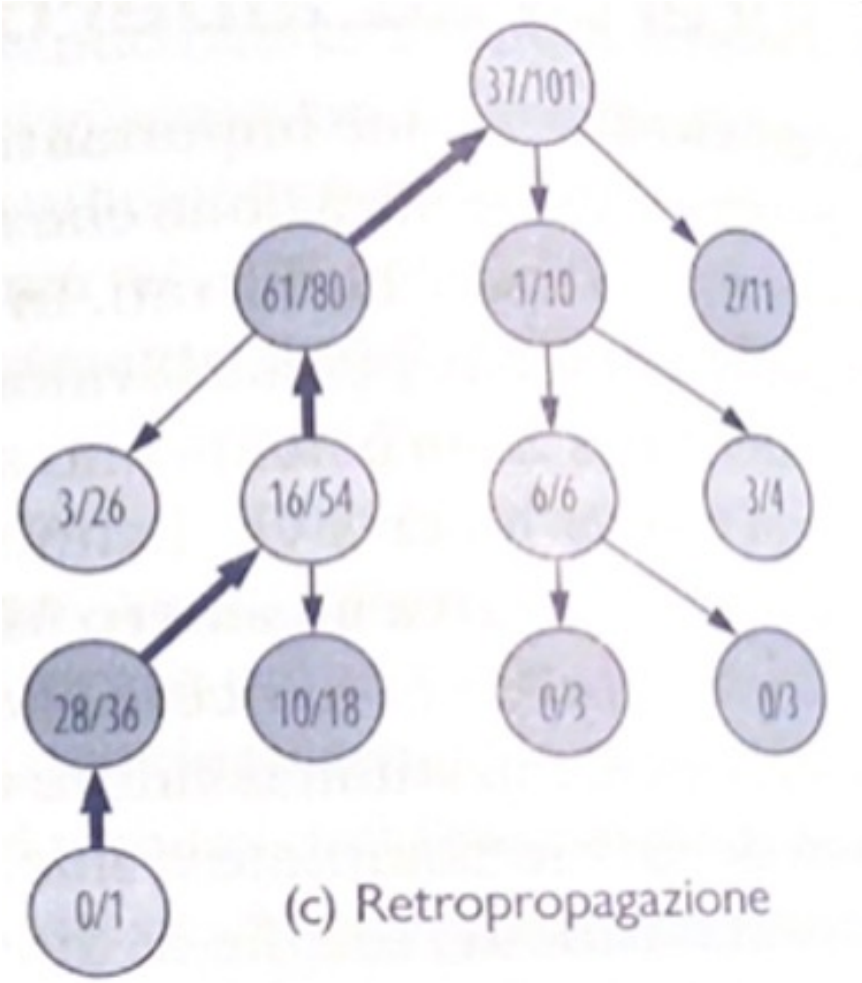
3. Simulazione
   - Eseguiamo una simulazione dal nodo figlio generato scegliendo le mosse dei giocatori secondo la politica di simulazione



Ricerca ad albero di Monte Carlo

4. Retropropagazione
   - Il risultato della simulazione viene usato per aggiornare tutti i nodi dell'albero di ricerca risalendo fino alla radice

- Il processo si ripete N volte oppure fino alla fine del tempo a disposizione
- Si restituisce la mossa con il più alto numero di simulazioni



(c) Retropropagazione

Confronto

- Il tempo per il calcolo di una simulazione è lineare nella profondità dell'albero di gioco
- Se un gioco ha b=32 e m=100, con un miliardo di stati
  - Minmax cerca fino a profondità 6
  - Alfa-beta con ordinamento delle mosse fino a 12
  - Montecarlo potrebbe fare 10 milioni di simulazioni
- Le performance dipendono dall'accuratezza della funzione euristica rispetto alle politiche di selezione e simulazione

Confronto

- Se la funzione di valutazione è imprecisa, la ricerca alfa-beta sarà imprecisa
  - L'errore in un nodo può portare ad una scelta errata
- La ricerca Monte Carlo
  - si basa sull'aggregato di simulazioni quindi non è vulnerabile al singolo errore
  - può essere usata anche per nuovi giochi in cui non si ha esperienza per definire una funzione di valutazione
  - Le politiche di selezione e simulazione possono essere apprese usando reti neurali addestrate facendo giocare il programma contro se stesso

Confronto

- La ricerca Monte Carlo ha il rischio che
  - Una linea di gioco importante non venga mai esplorata
  - in una posizione critica contro un giocatore esperto, può esserci un solo ramo che porta a una sconfitta. Poiché questo non è facilmente identificabile a caso, la ricerca potrebbe non "vederlo" e non ne terrà conto.
  - In sostanza, la ricerca tenta di tagliare le sequenze meno rilevanti.

AlphaGo (Nature 529, 484 - 489 2016)

https://www.youtube.com/watch?v=g-dKXOlsf98&ab_channel=naturevideo

- 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.

AlphaGo

alphago

Lee Sedol, il campione di Go si ritira per sempre: "L'intelligenza artificiale è imbattibile"

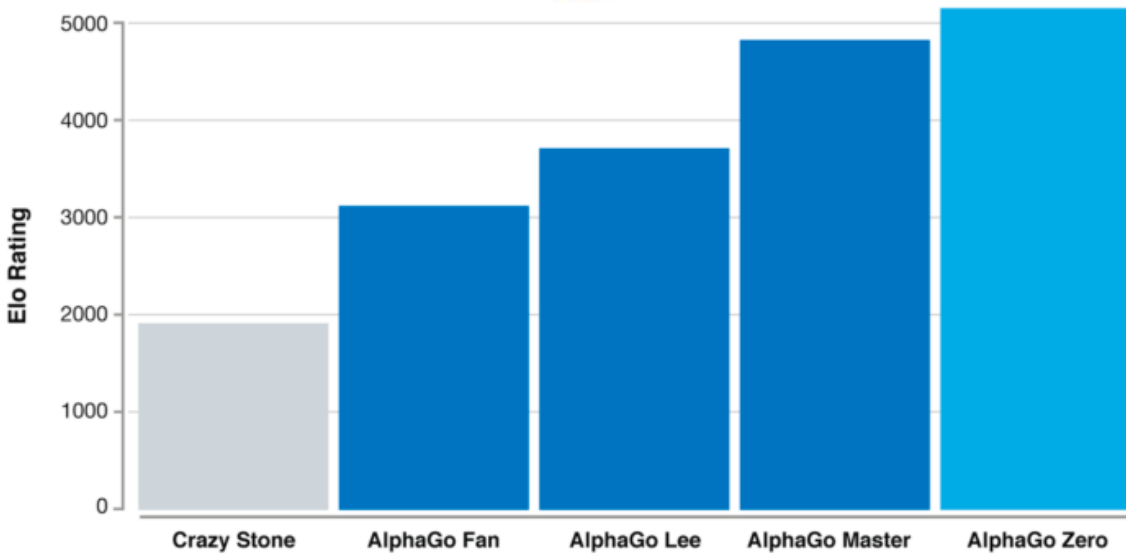
Il 36enne sudcoreano Lee Sedol, fenomeno del complicato gioco da tavola cinese, molla tutto: "Non mi sentirei mai il primo al mondo, ci sarebbe sempre un'altra entità in grado di battermi". Quell'entità è AlphaGo di Google. Ultima sfida contro un'altra AI

AlphaGo Zero

- AlphaGo Zero a version created without using data from human games, and stronger than any previous version. By playing games against itself, AlphaGo Zero surpassed the strength of AlphaGo Lee in three days by winning 100 games to 0, reached the level of AlphaGo Master in 21 days, and exceeded all the old versions in 40 days.

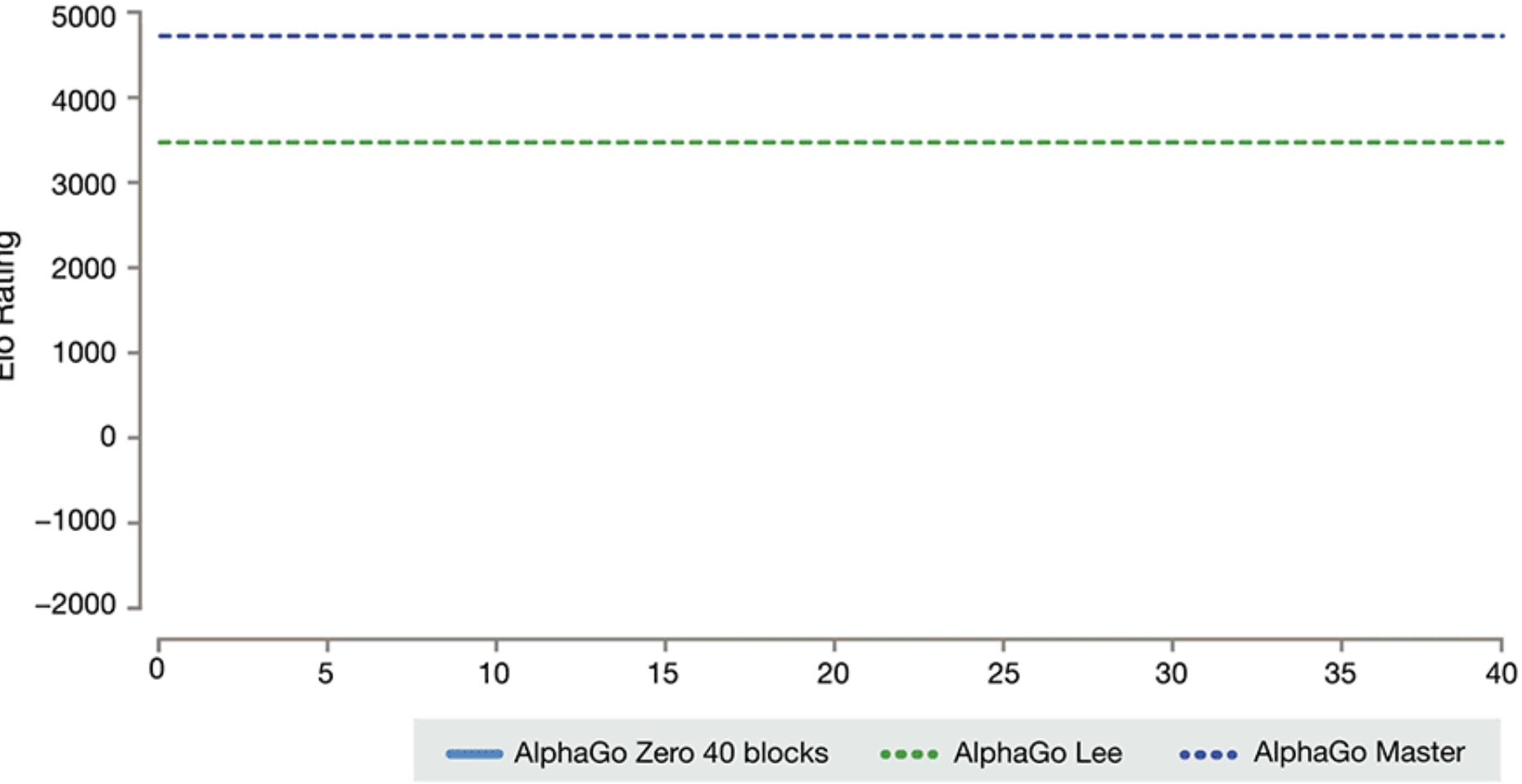
Comparison with predecessors [edit]

Versions
AlphaGo Fan
AlphaGo Lee
AlphaGo Master
AlphaGo Zero (40 days)
AlphaZero (34 hours)



AlphaGo Zero

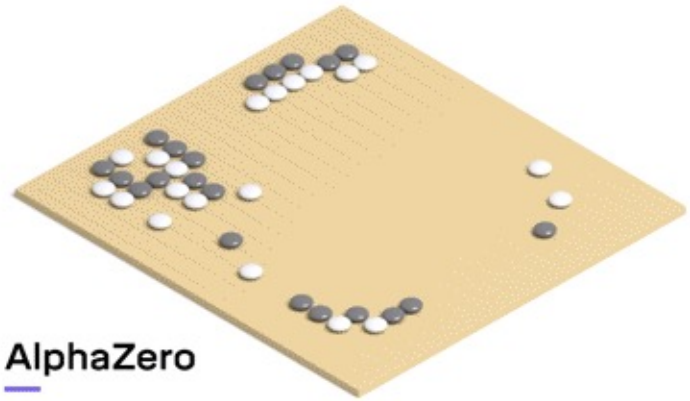
AlphaGo Zero

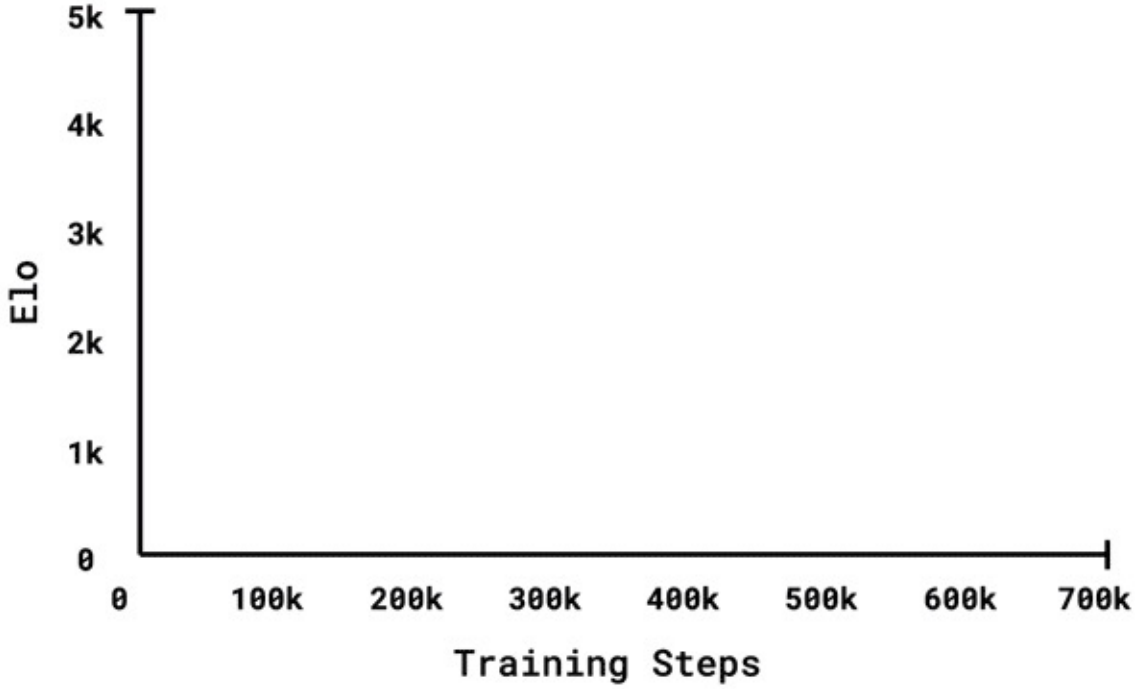


AlphaZero

- AlphaZero generalizes AlphaGo Zero's approach, which achieved within 24 hours a superhuman level of play in chess, shogi, and Go, defeating world-champion programs, Stockfish, Elmo, and 3-day version of AlphaGo Zero in each case.
- Differences between AZ and AGZ include:
  - AZ has hard-coded rules for setting search hyperparameters.
  - The neural network is now updated continually.
  - Go (unlike Chess) is symmetric under certain reflections and rotations; AGZ was programmed to take advantage of these symmetries. AZ is not.
  - Chess (unlike Go) can end in a tie; therefore AZ can take into account the possibility of a tie game.
- An open source program, Leela Zero, based on the ideas from the AlphaGo papers is available. It uses a GPU instead of the TPUs recent versions of AlphaGo rely on.

AlphaZero


AlphaZero

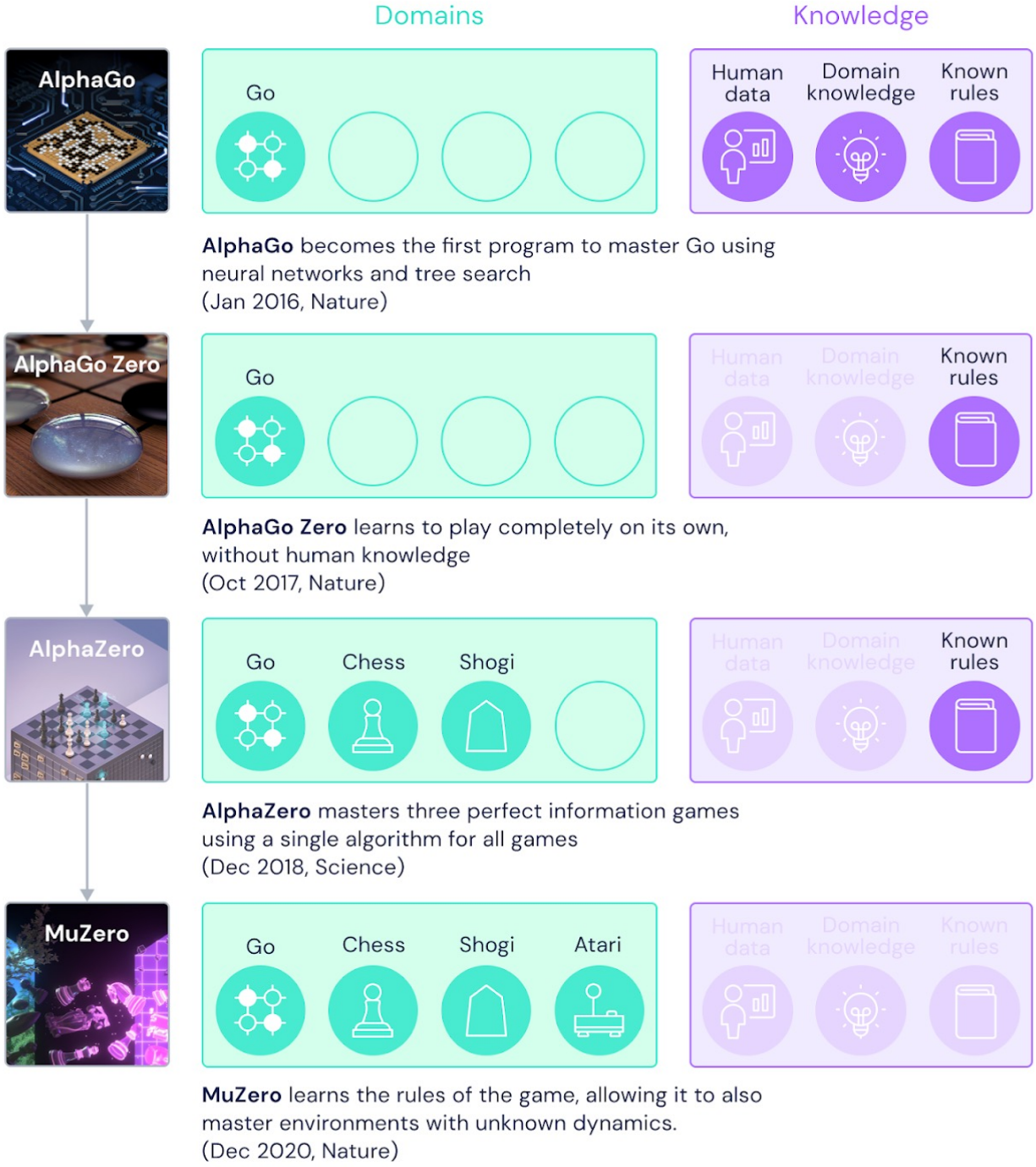

5k
4k
3k
2k
1k
0
Elo
0
100k
200k
300k
400k
500k
600k
700k
Training Steps

MuZero

- MuZero uses an approach similar to AlphaZero to master games without knowing their rules.
- It matched AlphaZero's performance in chess and shogi, improved on its performance in Go (setting a new world record), and improved on the state of the art in mastering a suite of 57 Atari games (the Arcade Learning Environment), a visually-complex domain.
- MuZero was trained via self-play, with no access to rules, opening books, or endgame tablebases. The trained algorithm used the same convolutional and residual algorithms as AlphaZero, but with 20% fewer computation steps per node in the search tree.

MuZero

- MuZero models three elements of the environment that are critical to planning:
  - The **value:** how good is the current position?
  - The **policy:** which action is the best to take?
  - The **reward:** how good was the last action?
- These are all learned using a deep neural network and are all that is needed for MuZero to understand what happens when it takes a certain action and to plan accordingly.


Domains
Knowledge
AlphaGo
Go
Human data
Domain knowledge
Known rules
AlphaGo becomes the first program to master Go using neural networks and tree search
(Jan 2016, Nature)
AlphaGo Zero
Go
Known rules
AlphaGo Zero learns to play completely on its own, without human knowledge
(Oct 2017, Nature)
AlphaZero
Go
Chess
Shogi
Known rules
AlphaZero masters three perfect information games using a single algorithm for all games
(Dec 2018, Science)
MuZero
Go
Chess
Shogi
Atari
MuZero learns the rules of the game, allowing it to also master environments with unknown dynamics.
(Dec 2020, Nature)

Tipi di Giochi

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Giochi con causalità (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

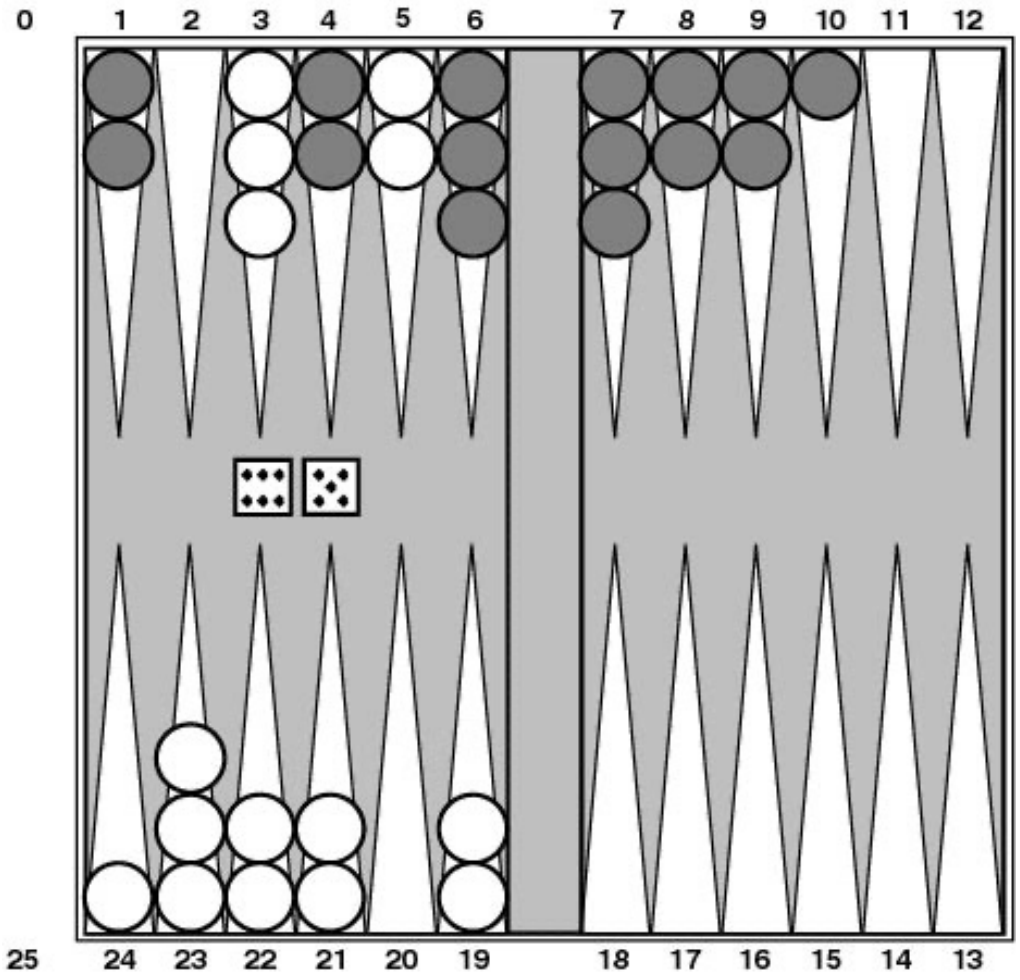
Lancio dadi 6-5,
4 mosse legali
per il bianco:

(5-10, 5-11)

(5-11,19-24)

(5-10, 10-16)

(5-11, 11-16)



Minimax con nodi possibilità (*chance*)

- Non si sa quali saranno le mosse legali dell'avversario: accanto ai nodi di scelta dobbiamo introdurre i nodi *possibilità*

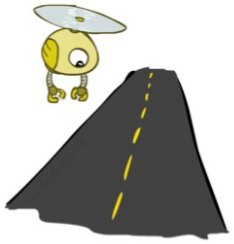
- Nel calcolare il valore dei nodi MAX e MIN adesso dobbiamo tenere conto delle probabilità dell'esperimento casuale

- Si devono calcolare il valore massimo e minimo *attesi*

Reminder: Probabilità

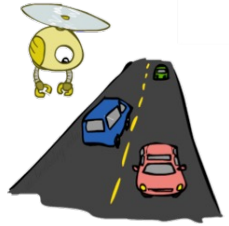
- Una variabile casuale rappresenta un evento il cui risultato è sconosciuto
- Una distribuzione di probabilità è un'assegnazione dei pesi agli esiti

- Esempio: Traffico su autostrada
  - Variabile casuale: T = se c'è traffico
  - Risultati: T in {nessuno, leggero, pesante}
  - Distribuzione: P(T= nessuno) = 0.25, P(T= leggero) = 0.50, P(T= pesante) = 0.25



0.25

- Alcune leggi sulla probabilità:
  - Le probabilità sono sempre non negative
  - La somma delle probabilità di tutti i possibili risultati è 1



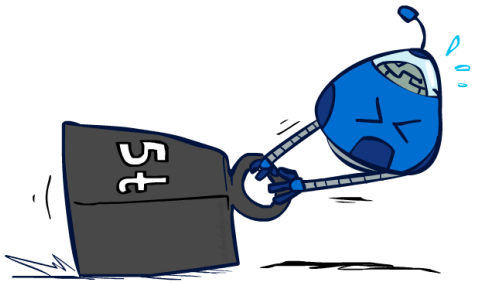
0.50

- Le probabilità possono cambiare:
  - P(T=pesante) = 0.25, P(T=pesante | ora=8am) = 0.60

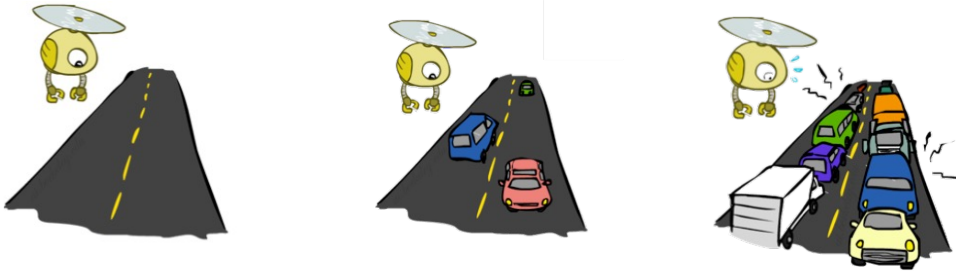
0.25

Reminder: Valori attesi

- Il valore atteso di una variabile casuale è la media ponderata dalla distribuzione di probabilità sui risultati
- Esempio: quanto tempo per arrivare all'aeroporto?

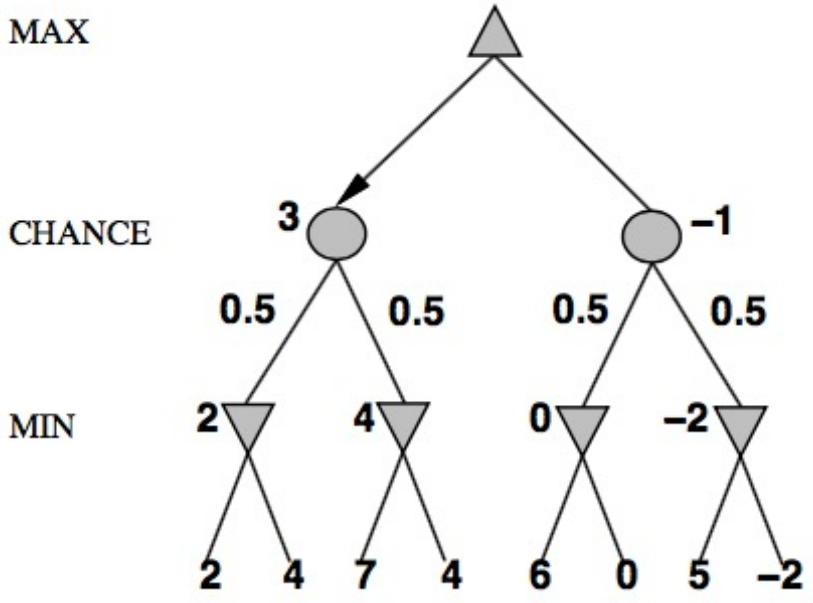


Tempo:	20 min	+	30 min	+	60 min	→	35 min
	x		x		x		
Probabilità:	0.25		0.50		0.25		



Minimax con nodi possibilità (*chance*)

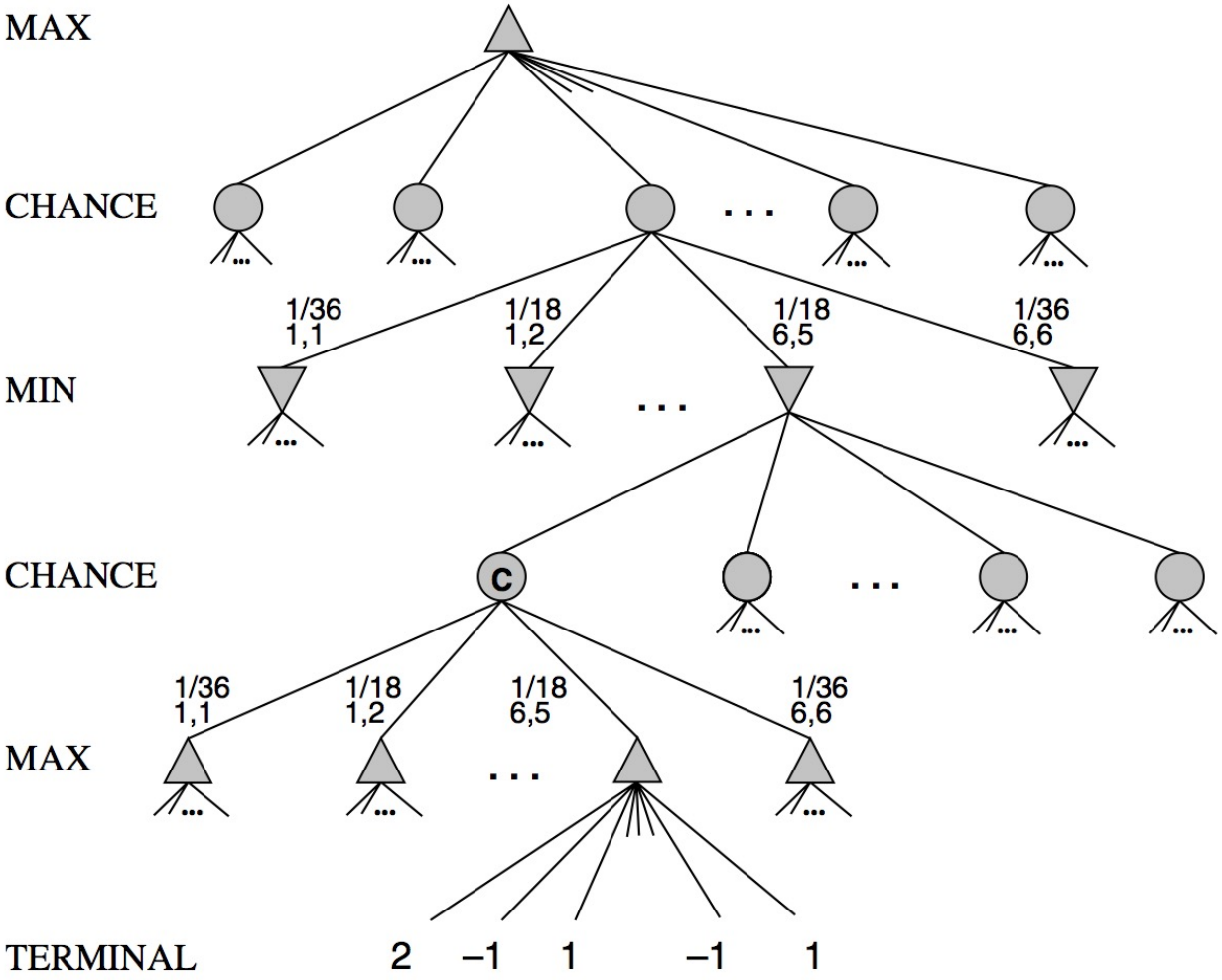
- Esempio con lancio moneta:



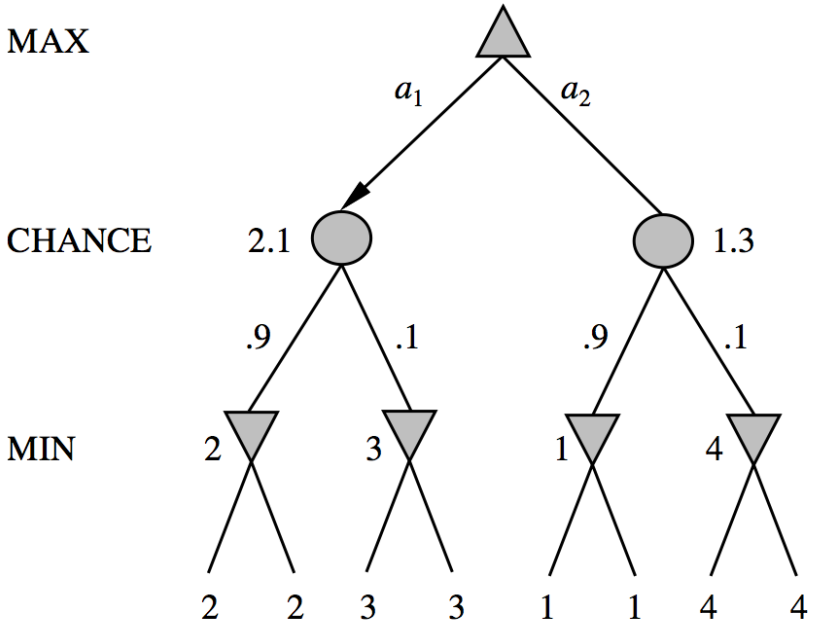
- Non abbiamo più valori Minimax, ma solo valori attesi (media su tutti i possibili risultati dei nodi di casualità)

Minimax per il Backgammon

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



Minimax 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 per MAX è la prima.

Expectiminmax: la regola

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$