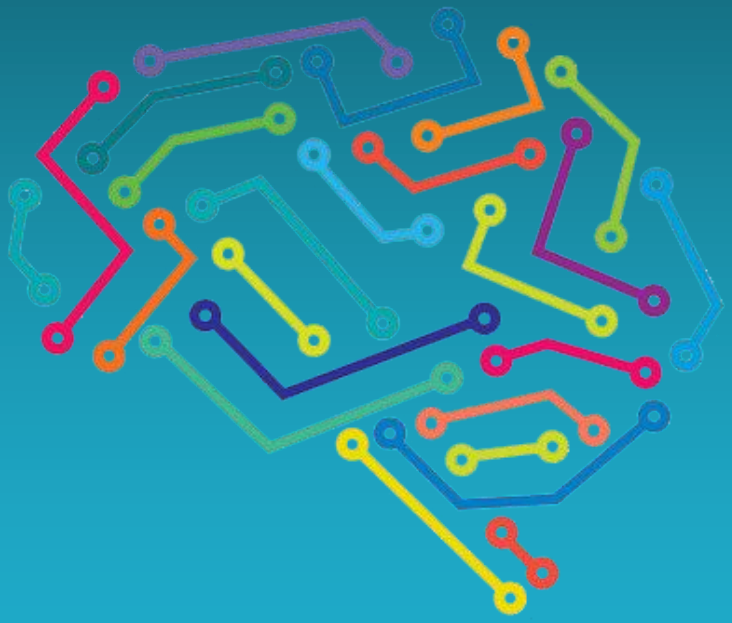
دانشگاه صنعتی شاهرود

جستجوهای خصمانه



درس هوش مصنوعی

نیم‌سال اول تحصیلی 98-1397

بازی‌ها

- **محیط‌های رقابتی:**
  - **اهداف عامل‌ها در تضاد با یکدیگر است.**
- **منجر به بروز روش‌های جستجوی خصمانه (adversarial) یا gameها شده‌اند:**
  - **دوعامله**
  - **قطعي**
  - **نوبتي**
  - **مجموع‌صفر (zero-sum):**
    - **مقادیر ممکن تابع هدف در انتهای بازی همیشه برابر و عکس هم هستند.**
    - **نام بهتر می‌توانست مجموع‌ثابت (Costant-sum) باشد.**
  - **کاملاً قابل مشاهده**
  - **مثلا: شطرنج**

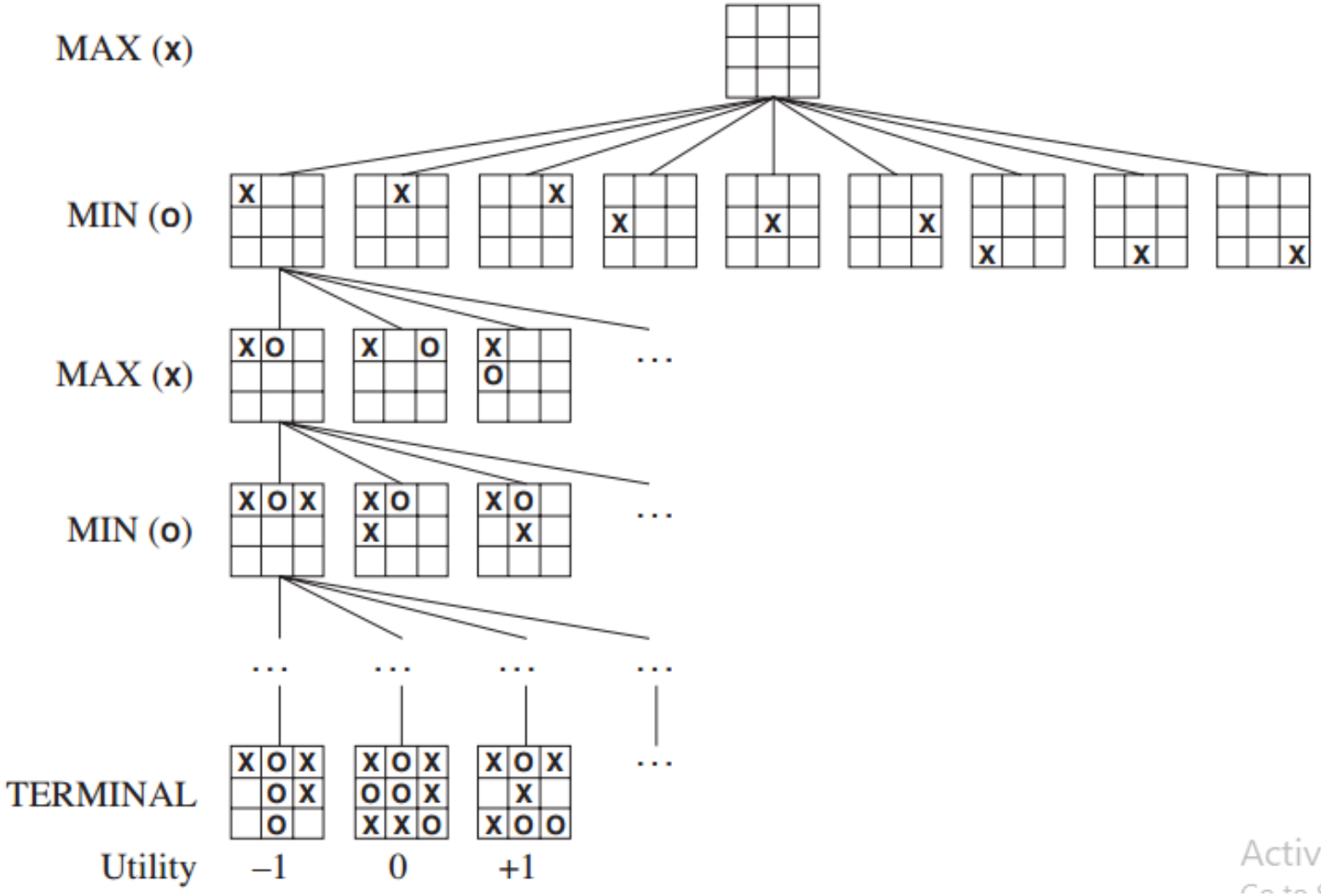
بازی‌ها

- در یک بازی، مثلا شطرنج، برد یک بازیکن قطعا به معنی باخت دیگری است و همین نکته است که مسئله را خصمانه (adversary) می‌کند.

تعریف بازی

- یک مسئله جستجو با عناصر زیر:
  - ✓ حالت اولیه (s_0)
  - ✓ PLAYER(s): بازیکني که در حالت s باید حرکت کند
  - ✓ ACTION(s): مجموعه حرکات مجاز
  - ✓ RESULT(s, a): نتیجه حاصل از یك حرکت (مدل انتقالي)
  - ✓ TERMINAL(s): ارزیابي این که آیا حالت sیك حالت پایاني بازي است
  - ✓ UTILITY(s, p): میزان مطلوبیت (امتیاز) حالت s براي بازیکن
- درخت بازی، درختی است که در آن نودها نماینده حالت‌های بازی و یال‌ها نماینده actionها هستند و یک بازی بین max و min را نشان می‌دهد.
- حالت‌های هدف در برگ‌ها اتفاق می‌افتند و برد یا باخت از دید max هستند.

درخت بازی دوز (tic-tac-toe)



- در مسائل معمولی، راه‌حل بهینه دنباله‌ای از حرکت‌ها (actions) است که منجر به حالت هدف می‌شوند.
- در جستجوی خصمانه، حرکات min هم در این روال دخالت دارد.
- بنابراین max باید یک استراتژی مشروط بیابد شامل:
  - ✓ حرکت Maxدر حالت اولیه
  - ✓ حرکات max در هر یک از حالات منتج از حرکات جواب ممکن برای min (بعد از هر حرکت max، نوبت min است)
  - ✓ سپس، حرکات max در هر یک از حالات منتج از حرکات جواب ممکن برای min به حرکات قبلی
  - ✓ و به همین ترتیب

تصمیمات بهینه در بازی‌ها

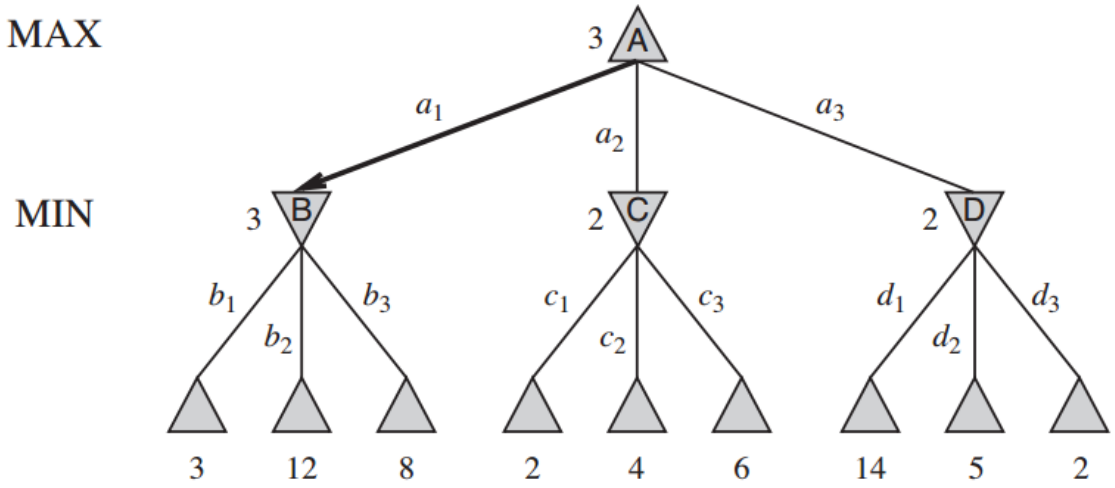
- مقادیر بالای تابع هدف برای MAX خوب هستند.
- درحالی که مقادیر پایین تابع هدف برای MIN خوب هستند.
- با توجه به تعریف ارائه‌شده برای درخت بازی، یك مقدار MINIMAX(s) براي هر يك از حالتهاي بازي محاسبه مي‌شود که بر اساس آن، استراتژی بهینه انتخاب می‌شود.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

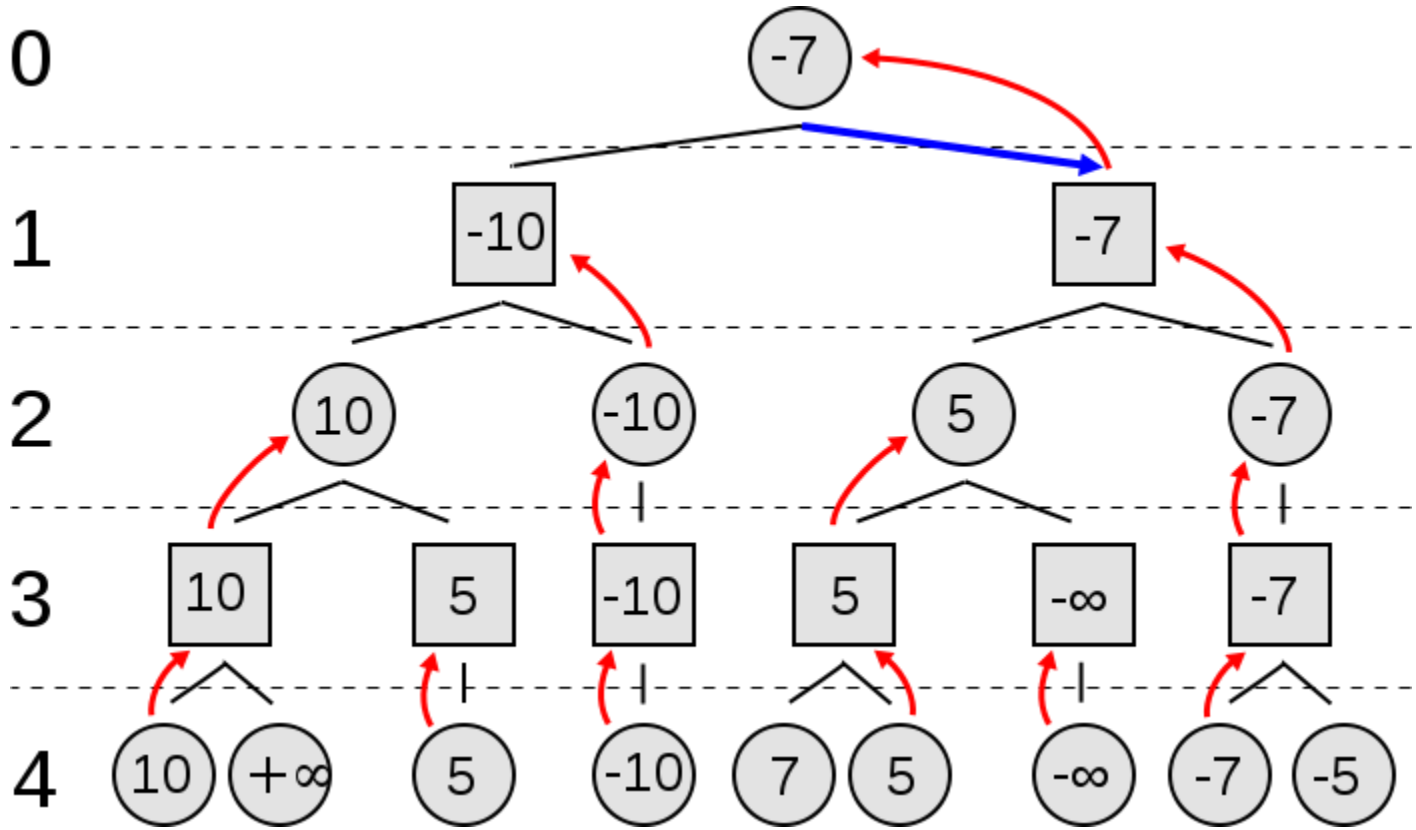
تصمیمات بهینه در بازی‌ها

الگوریتم MINIMAX:

- ✓ محاسبه مقادیر MINIMAX برای همه گره‌هاي درخت بازي با استفاده از يك جستجوي اول عمق كامل در درخت جستجو.
- ✓ داراي پيچيدگي مشابه الگوريتم جستجوي اول عمق است.



تصمیمات بهینه در بازی‌ها



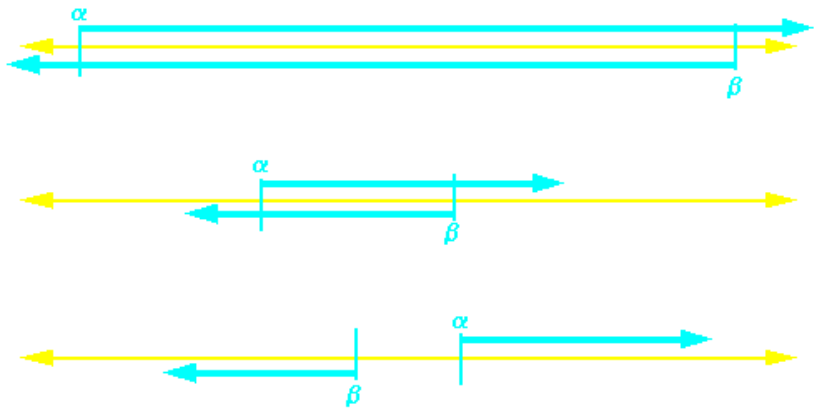
هرس آلفا-بتا

- **هرس کردن:**
  - ✓ **حذف شاخه هايي از درخت جستجو**
- **هرس آلفا-بتا:**
  - ✓ **حذف شاخه هاي اضافي در الگوريتم MINIMAX**
  - ✓ **تغييري در جواب يافته شده ايجاد نمي‌كند.**
  - ✓ **آلفا: بهترين امتياز يافته شده براي بازيكن MAX**
    - حد پایین یافته شده برای مقدار حداکثر
  - ✓ **بتا: بهترين امتياز يافته شده براي بازيكن MIN**
    - حد بالای یافته شده برای مقدار حداقل

هرس آلفا-بتا

- محدوده ممکن تابع هدف برای هر گره، بین آلفا و بتا است:

$$\alpha \leq MinMax(n) \leq \beta$$

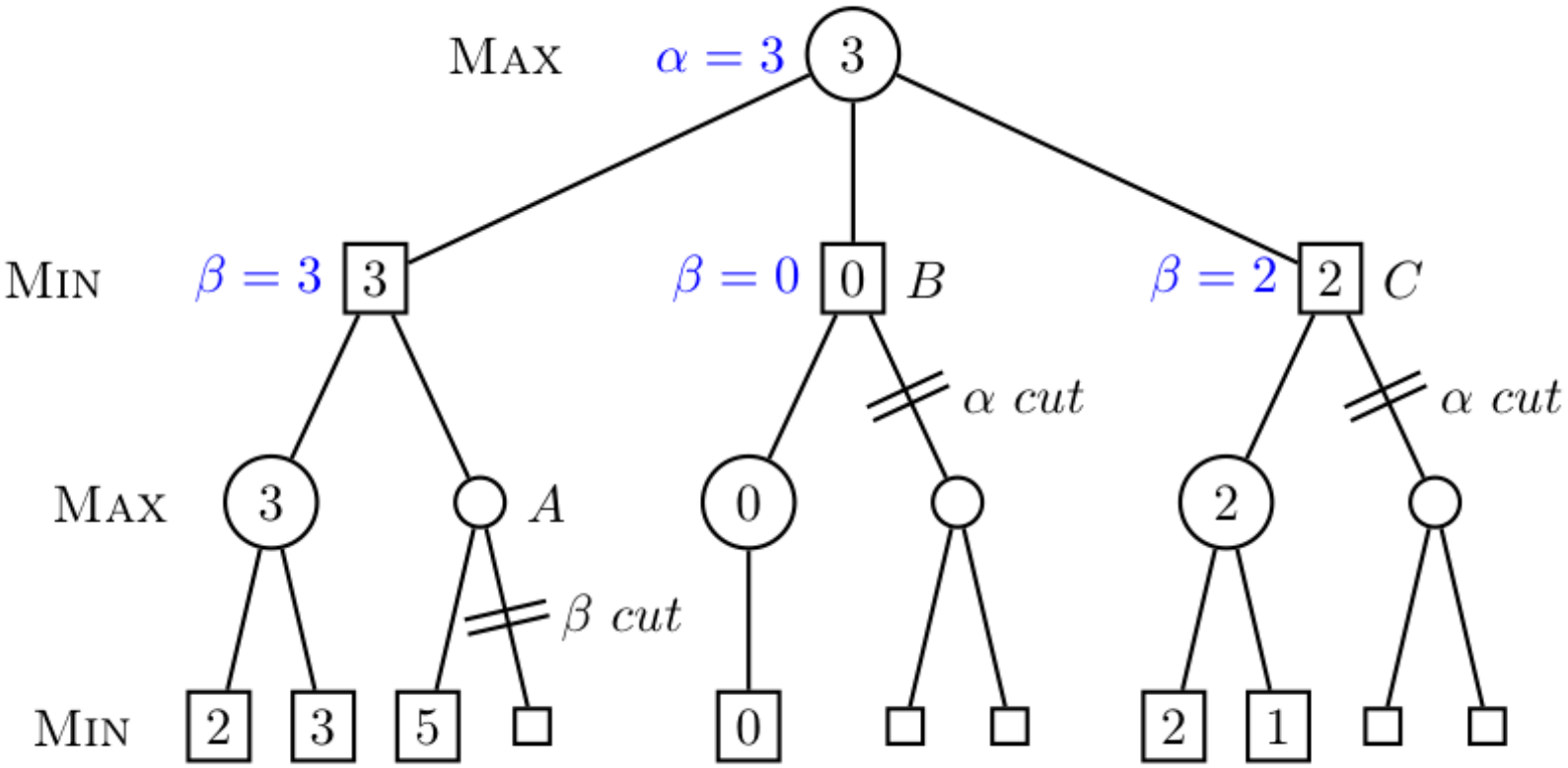


- پس اگر شرط زیر برقرار نباشد، می‌توان زیرشاخه‌ها را هرس کرد.

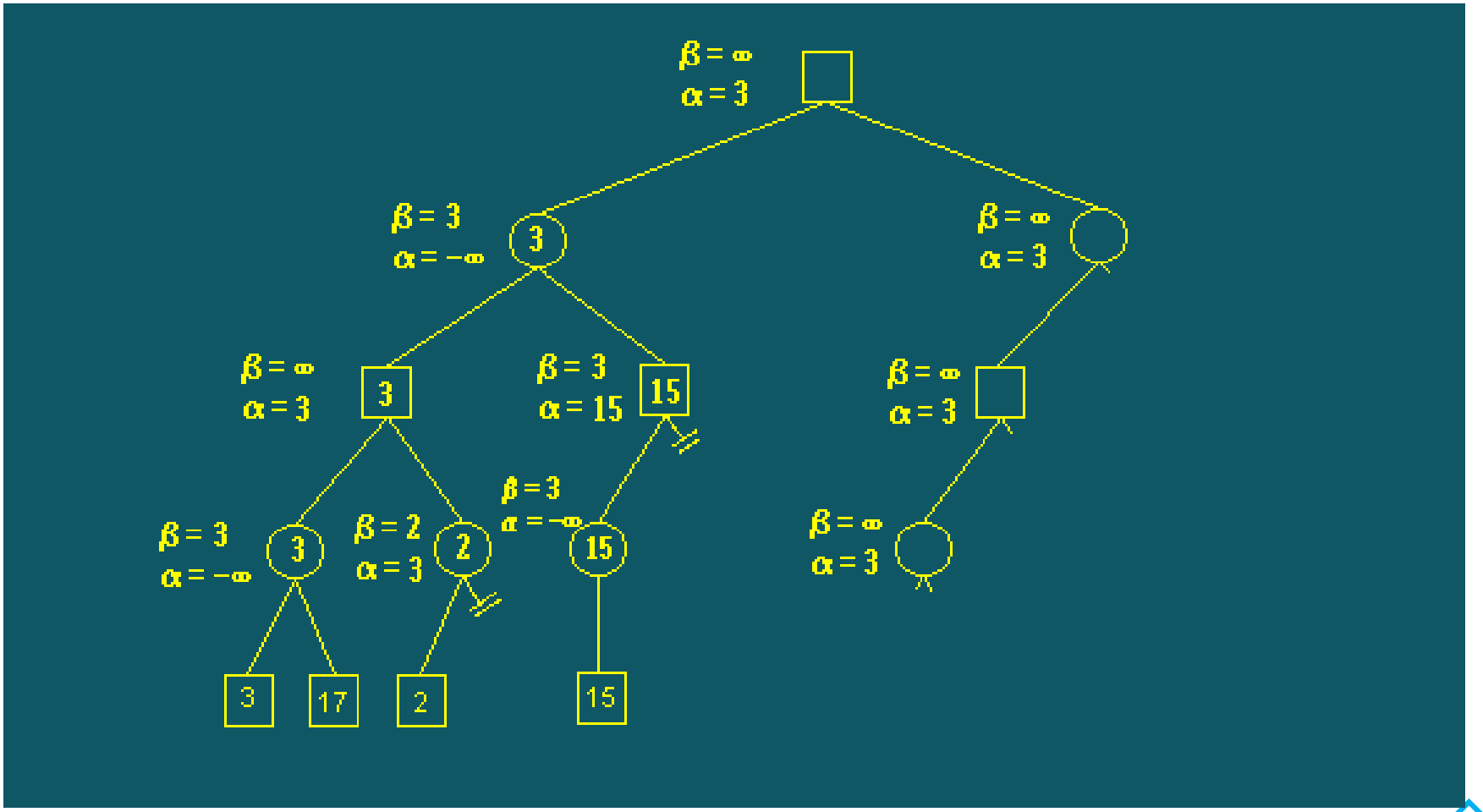
$$\alpha \leq \beta$$

هرس آلفا-بتا

- Alpha-cutoff و Beta-cutoff:



هرس آلفا-بتا



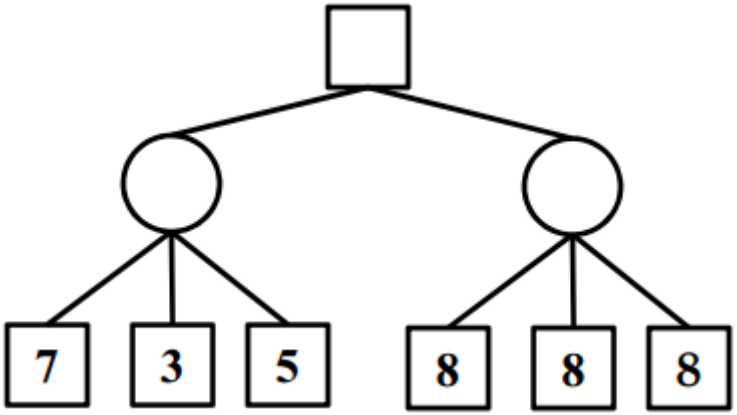
هرس آلفا-بتا

- آدرس سایت برای مشاهده مثال:

http://inst.eecs.berkeley.edu/~cs61b/fa/ecitcarp_eert_ba/sppa/slairetam-at/14

هرس آلفا-بتا

- **تأثير حدود امتيازها**
  - ✓ **در صورتي كه حداقل و حداكثر مقدار MINIMAXگره‌ها از قبل مشخص باشد مي‌توان از آن در هرس آلفا-بتا استفاده كرد.**
- **مثال: محدوده امتيازات در درخت زير [3,8] است:**

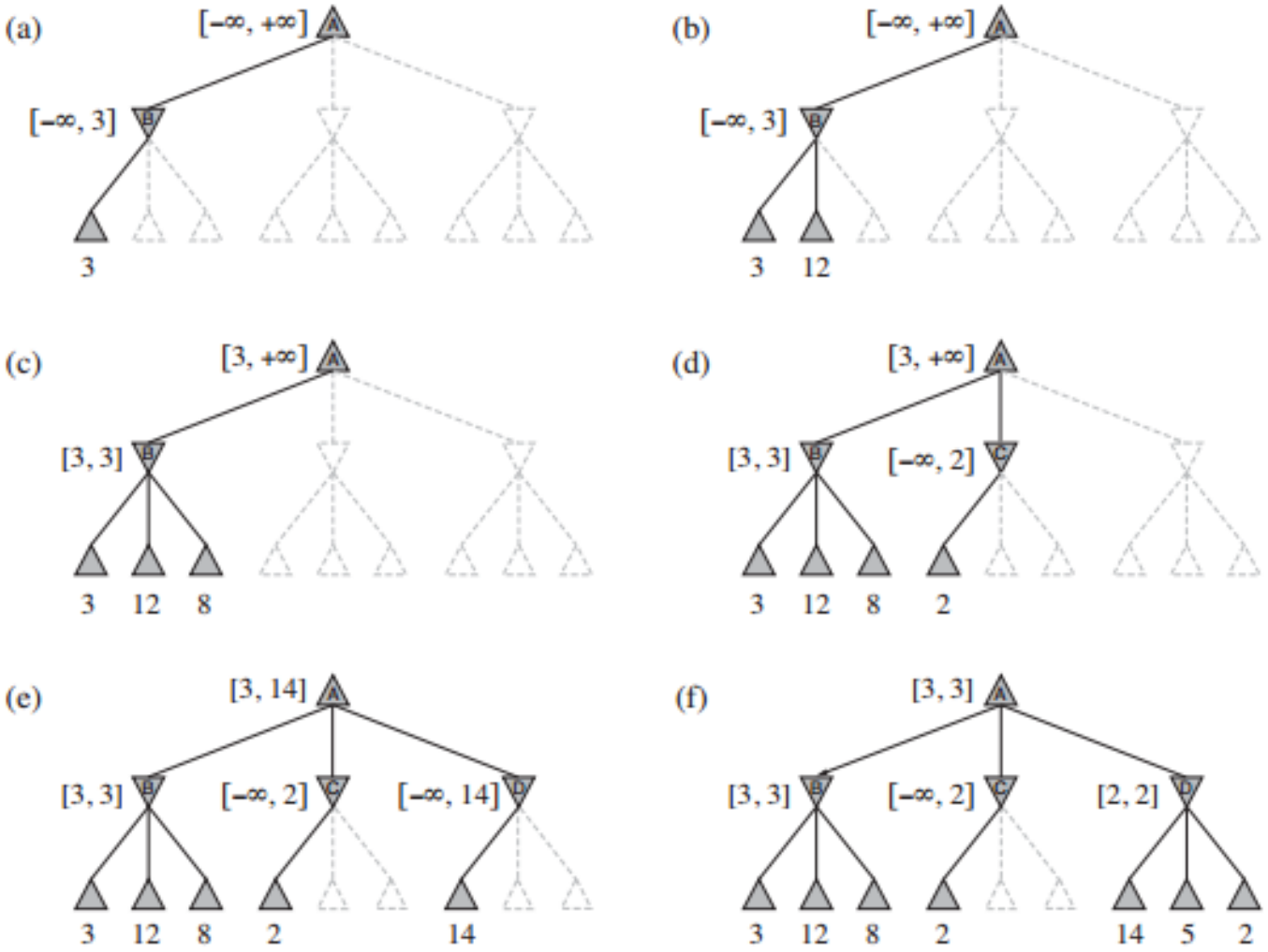


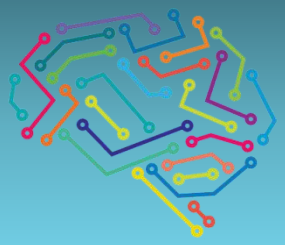
هرس آلفا-بتا

ترتیب انتخاب عمل‌ها

- ترتیب بررسي حرکت هاي‌ممکن در هر حالت بر کارآيي هرس آلفا بتا- تأثيردارد.
- هر چه حرکت هاي بهتر زودتر بررسي شوند، شاخه‌هاي بيشتري از درخت هرس می‌خواهند.
- میتوان بر اساس تجربیات گذشته بهترین حرکات را شناخت.
- در منطق جستجو، یک راه برای استفاده از تجربیات گذشته، روش عمیق شدن تکراری (iterative deepening) است.
- در هر عمق بهترین حرکت را یافته و در مرحله بعد، اولین حرکت، بهترین حرکت مرحله قبل خواهد بود.

ترتیب حرکت‌ها





Move order

- Even with alpha-beta pruning, if we always start with the worst move, we still get $O(b^*b^*..^*b) = O(b^d)$
- If we always start with the best move (also recursive) it can be shown that complexity is $O(b^*1^*b^*1^*b^*1...) = O(b^{d/2})$
- We can **double** the **search depth** without using more resources
- Conclusion: It is very important to try to **start** with the **strongest moves first**

Killer move Heuristic

- Killer-move heuristics is based on the assumption that a **strong move** which gave a **large pruning** of a sub tree, might also be a strong move in **other nodes** in the search tree
- Therefore we start with the killer moves in order to maximize search tree pruning
- In practical implementation, game playing programs frequently keep track of two killer moves for each depth of the game tree (greater than depth of 1) and see if either of these moves, if legal, produces a cutoff before the program generates and considers the rest of the possible moves. If a non-killer move produces a cutoff, it replaces one of the two killer moves at its depth. This idea can be generalized into a set of transposition tables.
- In computer chess and other computer games, **transposition tables** are used to speed up the search of the game tree. Transposition tables are primarily useful in perfect-information games (where the entire state of the game is known to all players at all times). The usage of transposition tables is essentially memoization applied to the tree search and is a form of dynamic programming.

تصمیمات غیربهینه بهنگام

- اجراي الگوريتم MINIMAX مستلزم گسترش درخت بازي تا حالات انتهايي بازي است.
- در عمل، به دليل تعداد حالات بسيار زياد بازي و محدوديت زماني بازيكنان براي انتخاب حركت، امكان استفاده از الگوريتم MINIMAX و يافتن راه‌حل بهينه وجود ندارد.
- راه‌حل: استفاده از راه‌حل‌هاي نسبتاً بهينه با محاسبه يك مقدار MINIMAX شهودي براي گره‌های میانی

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

تصمیمات غیربهینه بهنگام

- **توابع ارزيابي:**
  - ✓ جستجوي غيركامل درخت بازي مستلزم ارائه تابعي براي انتساب يك مقدار minimax به حالات غيرپاياني بازي است.
- **معيار قطع جستجو**
  - ✓ مشخص مي‌كند در چه مرحله‌اي بايد عمليات جستجو متوقف شود.
  - ✓ مي‌تواند بر اساس يك عمق ثابت (d) مشخص تصميم‌گيري شود.
  - ✓ بهتر است جستجو در حالت‌هاي ساكن (quiescent)كه به نظر نمي‌رسد در حركات بعدي دچار تغيير شديدي شوند متوقف شود.
  - ✓ با مشكل horizon effect روبرو است كه در آن رخداد يك پديده نامطلوب به تأخير انداخته مي‌شود بدون آن كه از وقوع آن جلوگيري شود.

Evaluation function

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

- نمونه ساده‌ای از تابع ارزیابی (تابع وزن‌دار خطی)

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

- بسیاری اوقات توابع ارزیابی غیرخطی هستند.

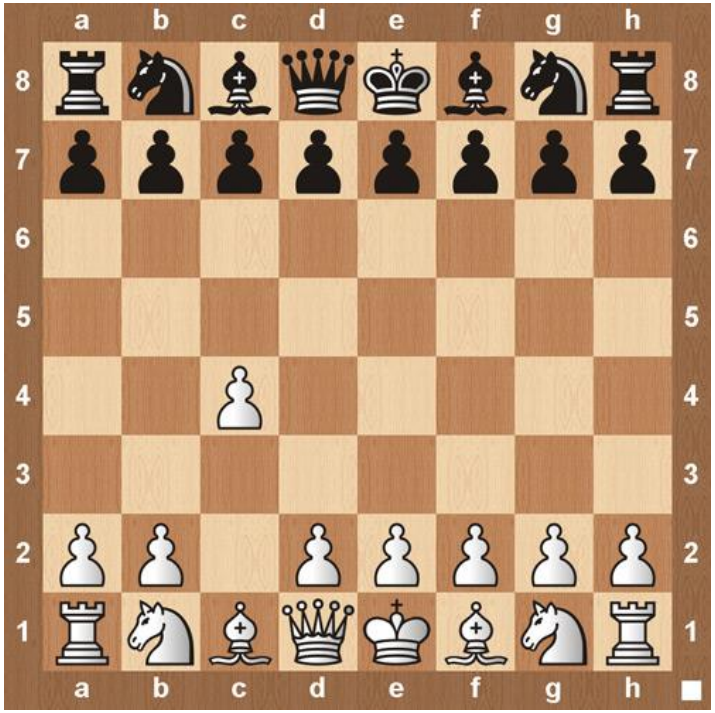
تصمیمات غیربهینه بهنگام

- **هرس پیشرو(forward pruning):**
  - ✓ **حذف برخي شاخه‌هاي درخت كه احتمالاً (و نه قطعاً) تأثيري در حركت انتخابي بازيكن ندارند.**
  - ✓ **جستجوي پرتو (beam search): در هر مرحله تنها n بهترين حركت را در نظر مي‌گيرد.**
  - ✓ **مشكل: تضميني نيست كه بهترين عمل هرس نشود.**
  - ✓ **استفاده از الگوهاي از پيش ذخيره شده**
  - ✓ **در ابتدا و انتهاي بازي به جاي جستجو مي‌توان از الگوهاي از پيش ذخيره شده براي انتخاب حركت استفاده كرد.**

Chess

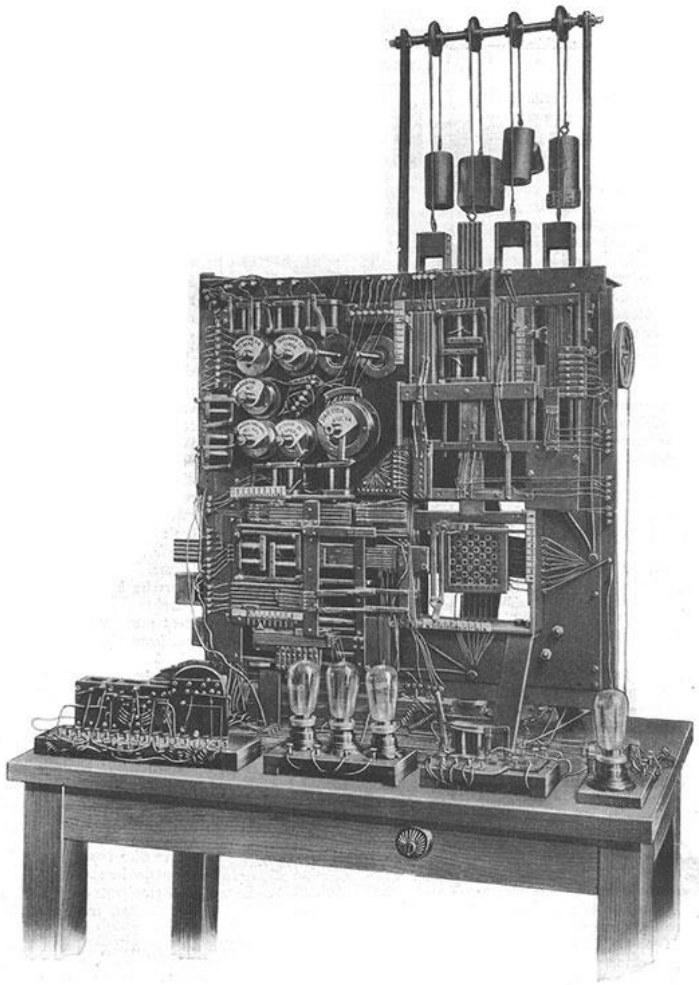
- **20** possible start moves (16 with the pawns and 4 with the knights), **20** possible replies, etc.
- So **400** possible positions (games) of **2** ply (half moves)
- **197 281** positions of **4** ply
- 7^{13} positions after 10 ply (5 White moves and 5 Black moves)
- **Exponential explosion**!
- Approximately 30 to **40 legal moves** in a typical position
- Typical game lasts~40 moves There exists about 10^{120} possible chess games

(If you say that at each move you have a choice between 30 moves (this is an estimate) and if you say that an average chess game lasts 40 moves (80 plies), you end up with the number of possible games equal to: 30^80=(30^2)^40 which is approximately equal to 1000^40=10^120)



History of computer Chess

- As old as computers
- The first chess-automaton of Torres (front view) is shown in the figure
- The machine was designed for the end game of King and Rook against King.

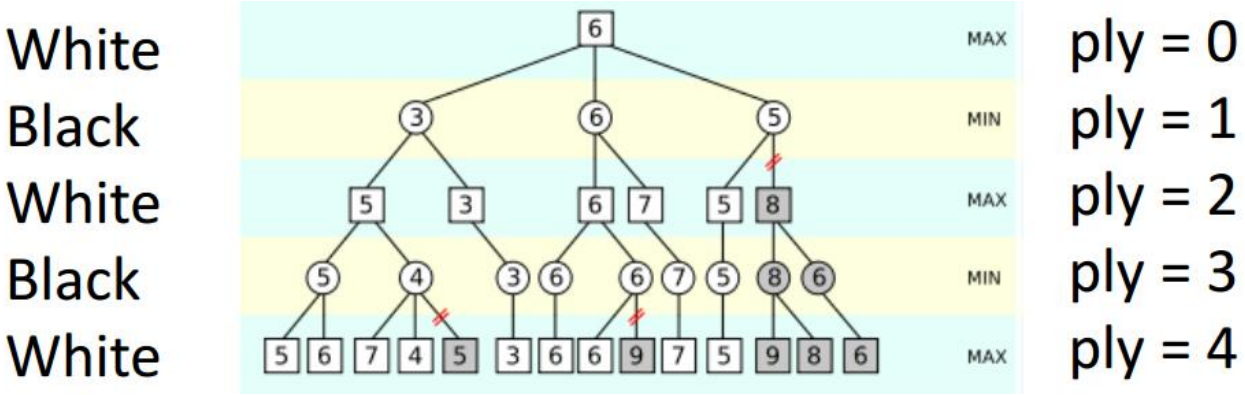


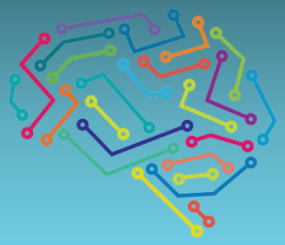
History of computer Chess

- Chess is a good fit for computers:
  - ✓ **Clearly defined rules**
  - ✓ Game of **complete information**
  - ✓ **Easy to evaluate (judge) positions**
  - ✓ **Search tree is not too small or infinite**
- 1950: Programming a Computer for Playing Chess (Claude Shannon)
- 1951: First chess playing program (on paper) (Alan Turing)
- 1958: First computer program that can play a complete chess game
- 1981: Cray Blitz wins a tournament in Mississippi and achieves master rating
- 1989: Deep Thought loses 0-2 against World Champion Garry Kasparov
- 1996: Deep Blue wins a game against Kasparov, but loses match 2-4
- 1997: Upgraded Deep Blue wins 3.5-2.5 against Kasparov
- 2005: Hydra destroys GM Michael Adams 5.5-0.5
- 2006: World Champion Vladimir Kramnik looses 2-4 against Deep Fritz (PC Chess engine)

Chess-minmax

- Minimax: Assume that both White and Black plays the best moves. We maximize White's score
- Perform a **depth-first search** and **evaluate** the **leaf nodes**
- Choose child node with **highest value** if it is **White** to move
- Choose child node with **lowest value** if it is **Black** to move
- **Branching factor** is **40** in a typical chess position



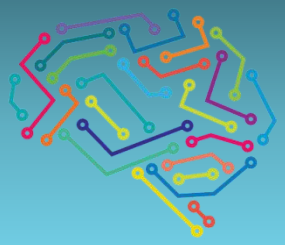


Chess

- The complexity of searching *d* ply ahead is $O(b^*b^*\ldots^*b) = O(b^d)$
- With a branching factor (*b*) of 40 it is crucial to be able to prune the search tree

alpha-beta pruning

- +



Transposition tables

- **Same position** will commonly occur from **different move orders**
- All chess engines therefore has a **transposition table** (position cache)
- Implemented using a **hash table** with chess position as key
- Doesn't have to evaluate large sub trees over and over again
- Chess engines typically uses half of available memory to hash table – proves how important it is

Evaluation function

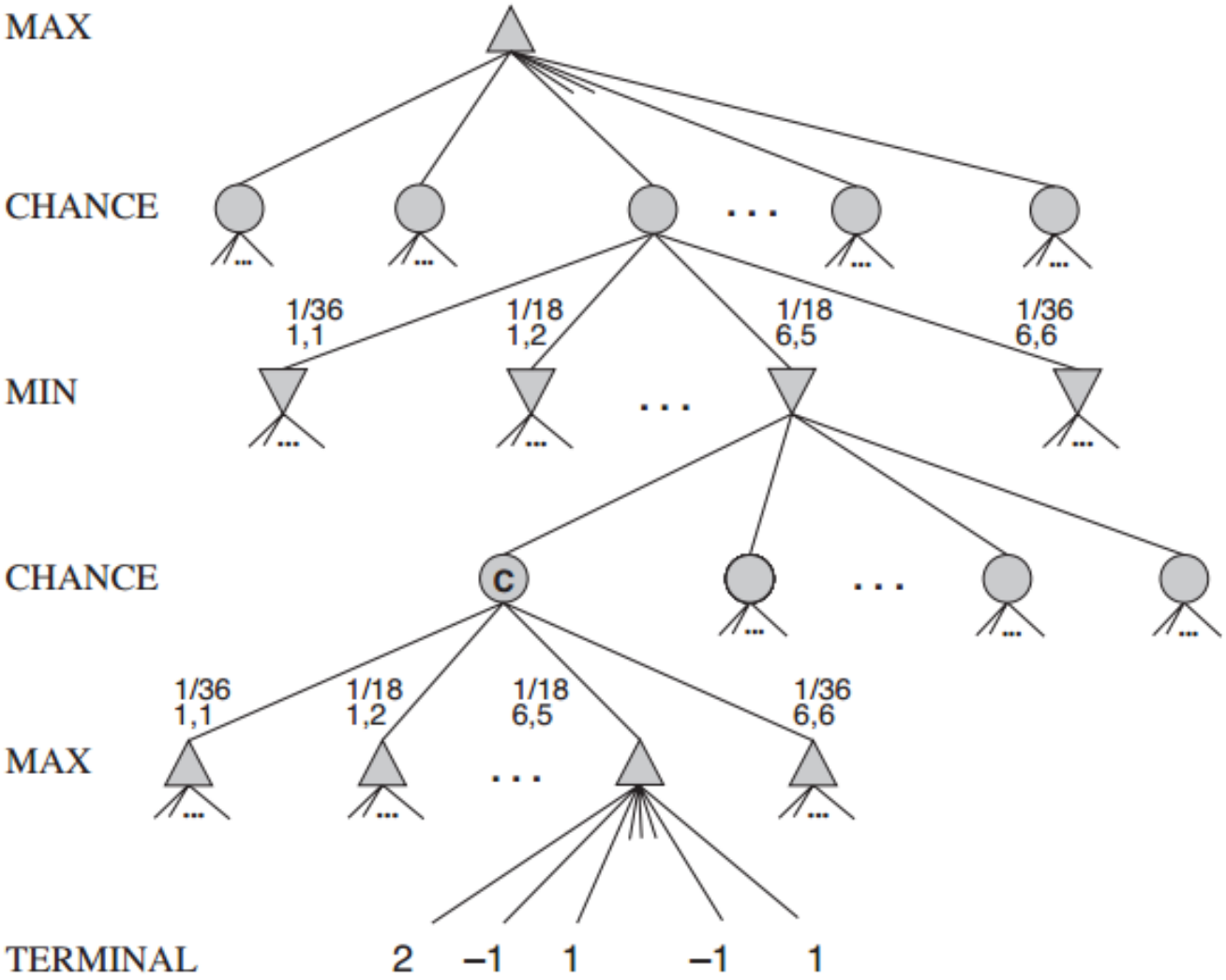
$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

بازي‌هاي تصادفي

- در برخي بازي‌ها، يك عامل تصادفي نيز در انتخاب عمل بازيكنان اثر دارد.
  - ✓ مثال: پرتاب دو تاس در بازي تخته نرد
- در اين موارد گره‌هاي شانس (chance nodes) نيز به درخت بازي اضافه مي‌شوند.
- نحوه محاسبه ارزش مورد انتظار (Expectiminimax) هر حالت:

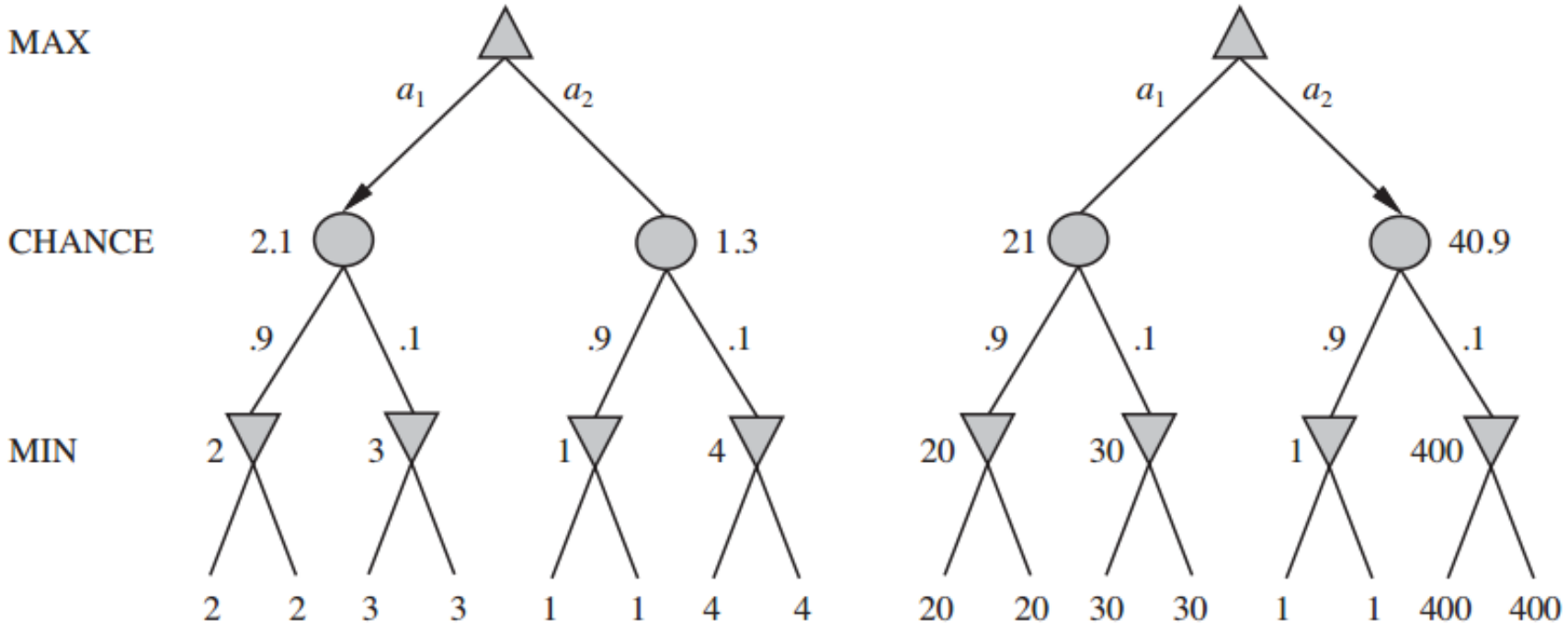
$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

بازی تخته‌نرد



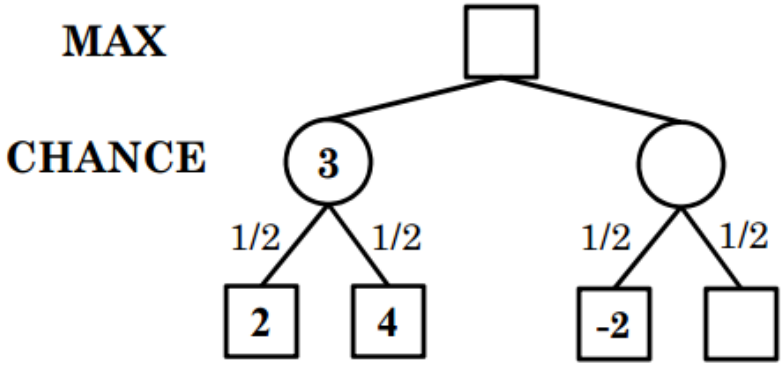
بازی تخته‌نرد

- تغییر نحوه امتیازدهي به حالات پاياني مي‌تواند باعث تغيير عمل انتخابي شود.



بازي‌هاي تصادفي

- برخلاف بازيهاي قطعي، بخش تصادفي بازي‌هاي تصادفي باعث مي‌شود نتوان يك دنباله از حركتها را به عنوان استراتژي بهينه بازي از قبل مشخص كرد.
- هرس آلفا-بتا بازي هاي تصادفي در ضمن محاسبه : EXPECTIMINIMAX
  - ✓ تحليل گره هاي MAXو MINمانند بازي هاي قطعي است.
  - ✓ در گره هاي شانس با معلوم بودن حدود حداقل و حداكثر امتياز مي توان بدون محاسبه مقدار برخي از شاخه‌ها، حد بالا يا پايين EXPECTIMINIMAX يك گره را محاسبه كرد.
- مثال: محدوده امتياز [4 ,4-]:



بازي‌هاي نيمه‌قابل‌مشاهده

- برخي از اطلاعات بازي در اختيار بازيكن قرار ندارد.
  - ✓ مثال: بازیهای کارتی
- در اين بازي‌ها هر بازيكن، يك مجموعه حالات باور را نگهداري و بر اساس مشاهدات جزئي خود آن را به‌روز مي‌كند.
- يك استراتژي براي چنين بازي‌هايي بايد براي هر دنباله دريافت ممكن يك دنباله حركت مشخص كند.
- يك استراتژي قطعي برد، استراتژي‌ای است كه به ازاء كليه حركات ممكن بازيكن حريف، منجر به برد شود.

بازي‌هاي نيمه‌قابل‌مشاهده

$$\underset{a}{\operatorname{argmax}} \sum_{s} P(s)\, \textsc{Minimax}(\textsc{Result}(s, a))$$

- معمولا تعداد حالات ممکن خیلی زیاد است، پس از تخمین مونت کارلو استفاده می‌کنیم:

$$\underset{a}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \textsc{Minimax}(\textsc{Result}(s_i, a))$$