Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG
CAIDAS
WüNLP

ALGORITHMS IN AI & DATA SCIENCE 1 (AKIDS 1)

Adversarial Search (aka Game Playing)

Prof. Dr. Goran Glavaš

11.1.2024

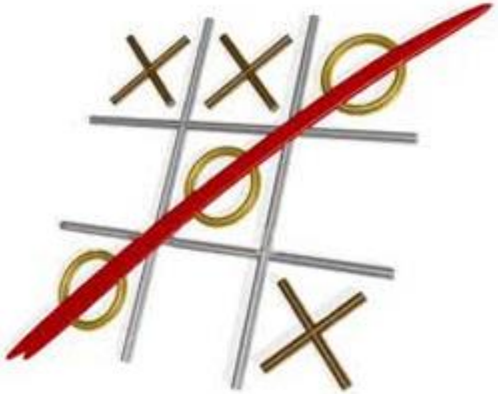
Content

- Introduction
- Minimax
- Heuristic Minimax
- Alpha-Beta Pruning

Based on the materials from Prof. Dr. Jan Šnajder:
https://www.fer.unizg.hr/_download/repository/AI-4-GamePlaying.pdf

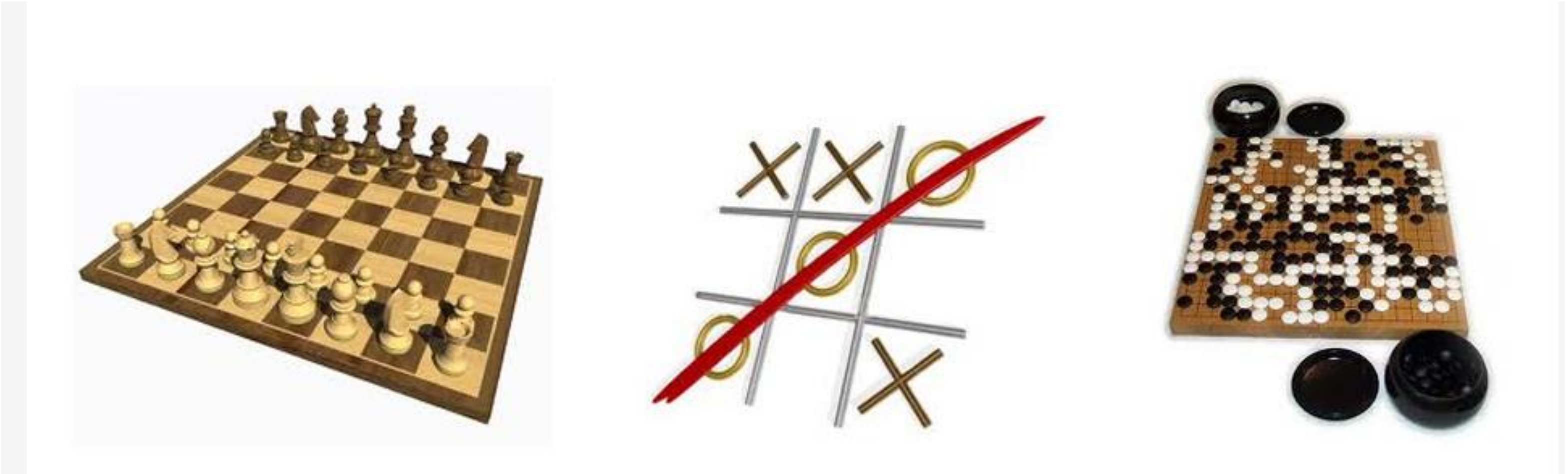
Game Playing: Adversarial Search

- Games also represent a **state space search** problem, but with the difference that there is an **adversary**
- In each game state one must make an **optimal decision** about which move to make next, i.e., one must find an **optimal strategy**


Game Playing: Adversarial Search

- We will focus on games that are
  - **Two-player**: that is, one adversary
  - **Deterministic**: no stochasticity/randomness like dice rolling
  - **Complete information**: both players have complete knowledge of the game (e.g., know the rules, see the full board, etc.)
  - **Zero sum**: one player winning (losing) means the other player losing (winning)



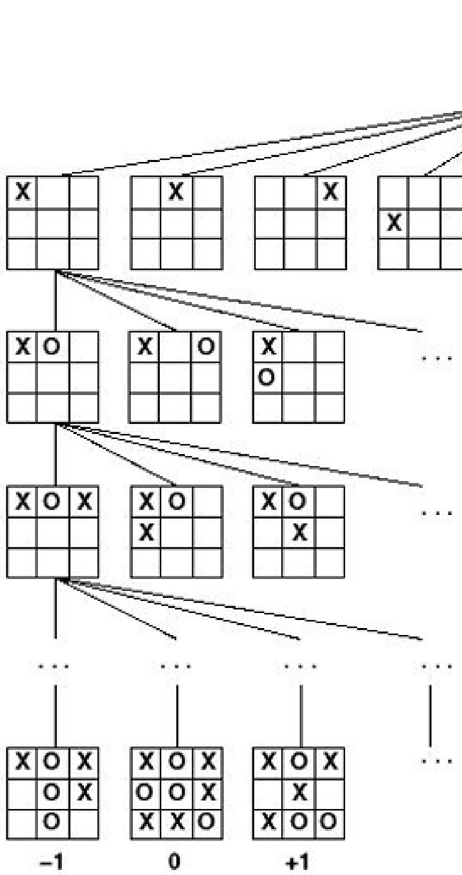
Adversarial Search (Game): Formalization

Game (Adversarial Search Problem)

Game is a state space search problem quadruple (s_0, *succ*, *terminal*, *utility*) where s_0 is the **initial game state**, *succ* : **S** → ℘(**S**) defines legal game moves (i.e., **transitions** between states), *terminal:* **S** → {True, False} is a predicate that tests if a state is a **game end state** and *utility*: S → ℝ is a **payoff function** that assigns numeric values to terminal (end) states of the game for the player(s).

- For example, in **chess** there are three possible outcomes
  - Win, loss, draw
  - utility(s) ∈ {+1, 0, −1}
- Initial state s_0 and successor function *succ* implicitly define the **game tree**

Game Tree

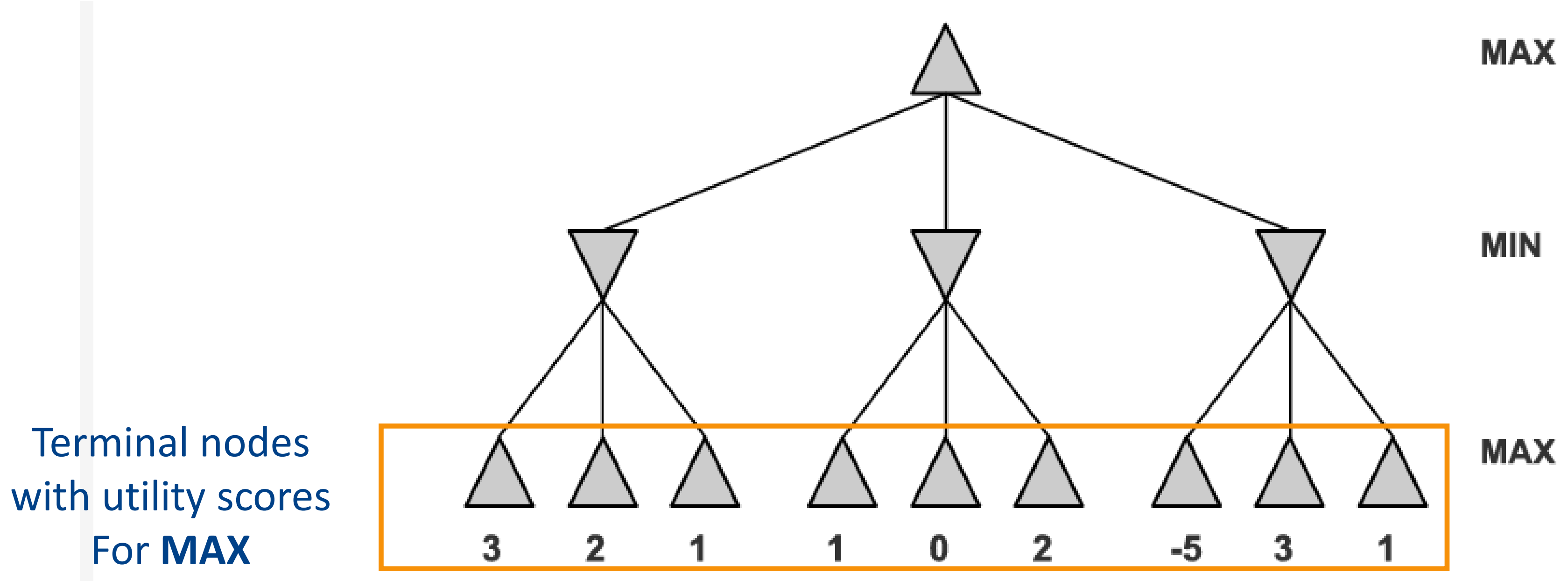


Content

- Introduction
- Minimax
- Heuristic Minimax
- Alpha-Beta Pruning

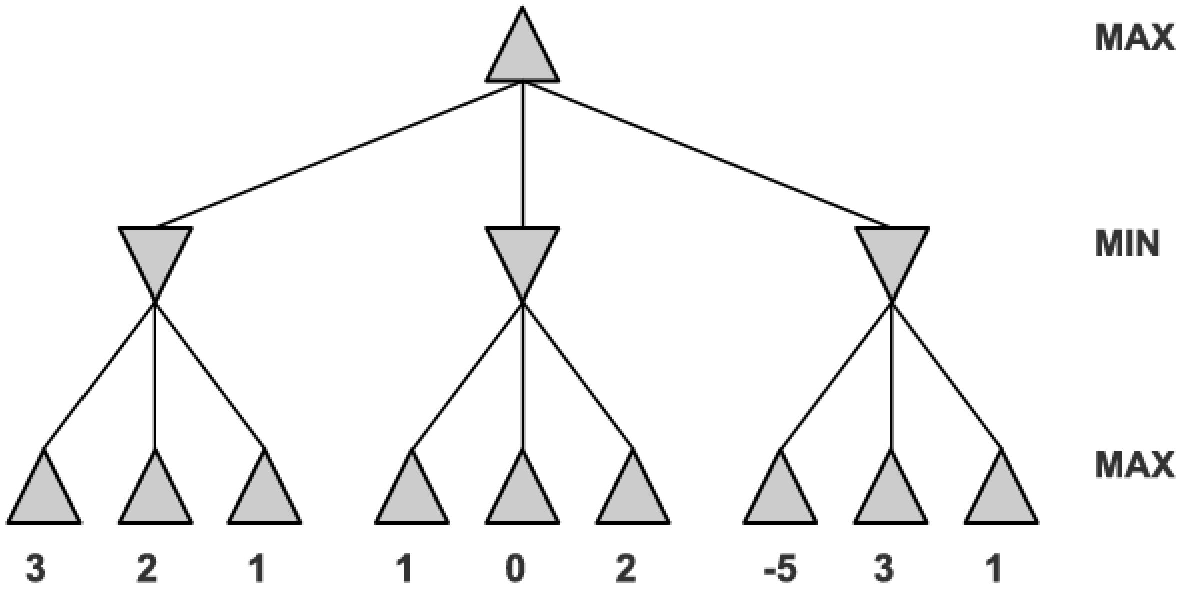
Minimax method

- Let's call the players **MAX** („*us*", the computer) and **MIN** (the opponent)
- **MAX** tries to **maximize** its win, **MIN** tries to **minimize** MAX's win
- Players take turn: nodes at **even depths** – **MAX**; **odd depth** – **MIN**



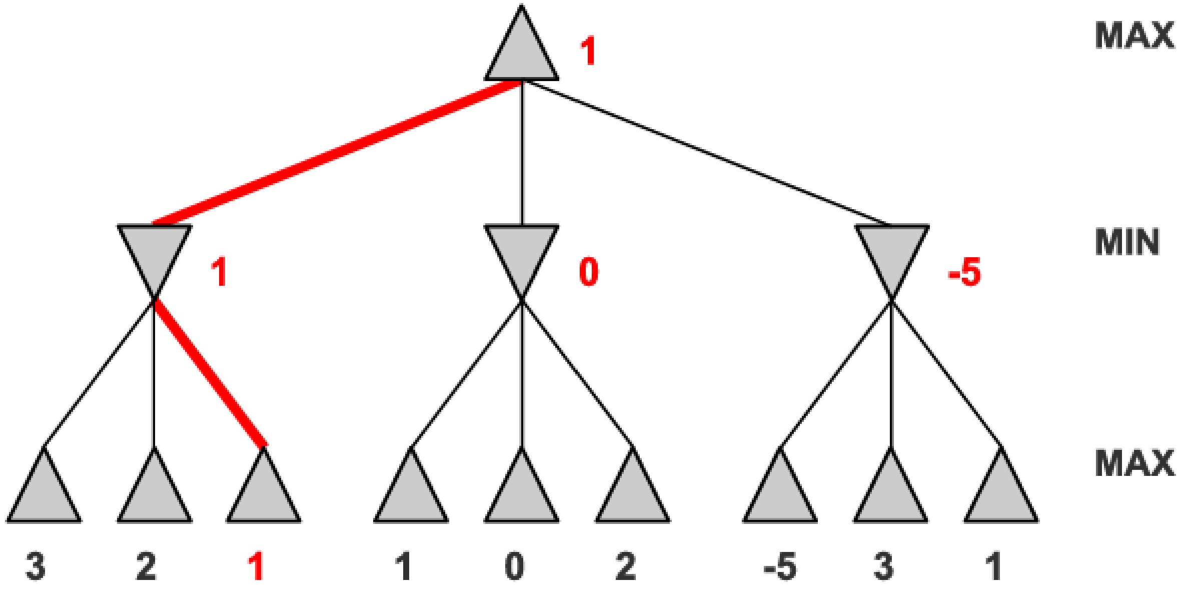
Minimax method: Optimal Strategy

- **Q:** What is the optimal game-playing strategy for **MAX**?
  - **Q:** Is it to take a step in the direction of its **own maximal utility**?



Minimax method: Optimal Strategy

- **MAX**'s **optimal strategy** is the one that ensures the highest win for MAX, but assuming that **MIN** uses the same strategy
- Each player chooses a strategy so as to **minimize the maximum loss**
- To determine the optimal strategy of a player who's turn is next, we compute the **minimax value** of the root node



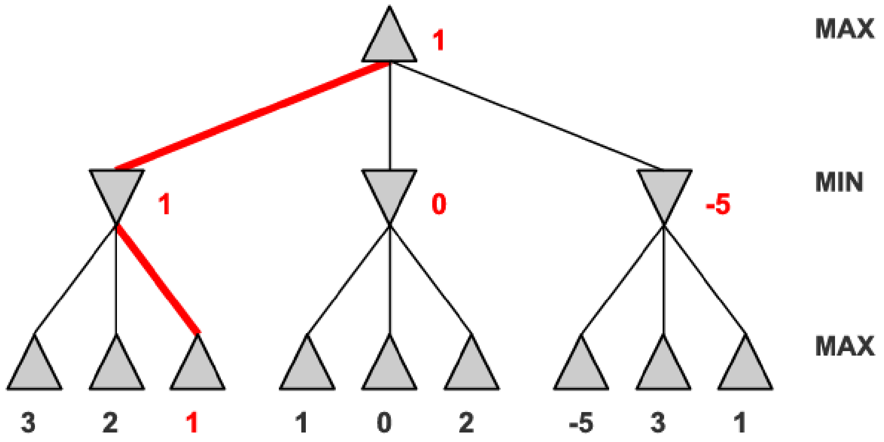
Minimax value

Minimax value

The **minimax value** of a state s is defined recursively:

$$m(s) = \begin{cases} utility(s) \textbf{ if } terminal(s) \\ \max_{t \in succ(s)} m(t) \textbf{ if } s \text{ is a } \textbf{MAX} \text{ node} \\ \min_{t \in succ(s)} m(t) \textbf{ if } s \text{ is a } \textbf{MIN} \text{ node} \end{cases}$$

$$m(s_0) = \mathbf{max}(\mathbf{min}(3, 2, 1), \mathbf{min}(1, 0, 2), \mathbf{min}(-5, 3, 1))$$

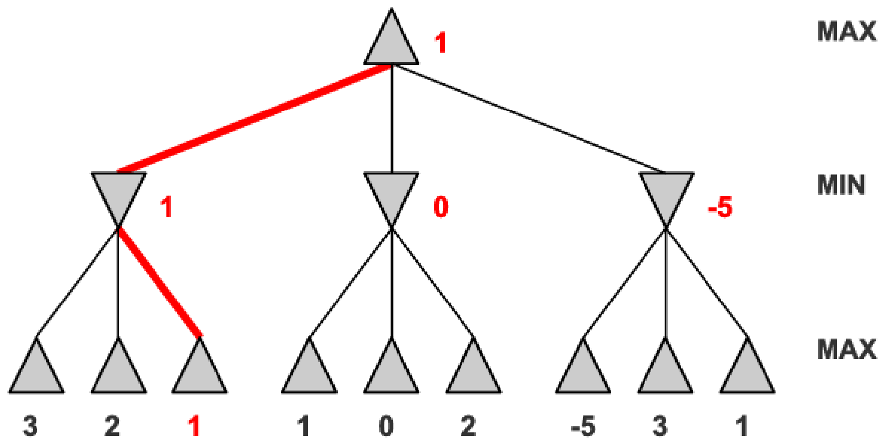


Minimax value: algorithm

```
max_val(s)
  if terminal(s)
    return utility(s)
  m = -inf
  for t in succ(s)
    m = max(m, min_val(t))
  return m
```

```
min_val(s)
  if terminal(s)
    return utility(s)
  m = +inf
  for t in succ(s)
    m = min(m, max_val(t))
  return m
```

$$m(s) = \begin{cases} \mathit{utility}(s) & \text{if } \mathit{terminal}(s) \\ \max_{t \in succ(s)} m(t) & \text{if } s \text{ is a MAX node} \\ \min_{t \in succ(s)} m(t) & \text{if } s \text{ is a MIN node} \end{cases}$$



- Note that this is essentially a **DFS** implemented via two mutually recursive functions

Minimax: remarks

- In practice, the opponent's strategy is **unknown**
  - Most probably different from that of **MAX** player
  - The opponent's moves therefore cannot be predicted perfectly (otherwise the game would be boring anyways)

- Because of this – in order to make the **optimal move**, in each turn the players need to **re-compute** their **optimal strategy**, starting from the current position as the root of the game tree

- Minimax is a **depth-first search**
  - **Q:** Time and space complexity?

Content

- Introduction
- Minimax
- Heuristic Minimax
- Alpha-Beta Pruning

Imperfect decisions

- Minimax is a **DFS –** that makes its time complexity **$O(b^m)$**
  - b: the branching factor of the game
  - m: total number of moves needed for the game to end (reach terminal state)
- For most games b and m are **big enough** that we don't have the time to search through the complete game tree all the way to terminals
- We need to cut the search off at a certain depth d (we can afford at most runtime of **$O(b^d)$**)
  - For states at depth d, we need to **estimate** the utility with a **heuristic**
  - E.g., in chess, a simple heuristic can be the sum of „worth" of remaining pieces (vs. the value of remaining pieces of the opponent)

Heuristic minimax

```
max_val(s, d, h)
  if terminal(s)
    return utility(s)
  if d = 0
    return h(s)
  m = -inf
  for t in succ(s)
    m = max(m, min_val(t, d-1, h))
  return m


min_val(s, d, h)
  if terminal(s)
    return utility(s)
  if d = 0
    return h(s)
  m = +inf
  for t in succ(s)
    m = max(m, max_val(t, d-1, h))
  return m
```

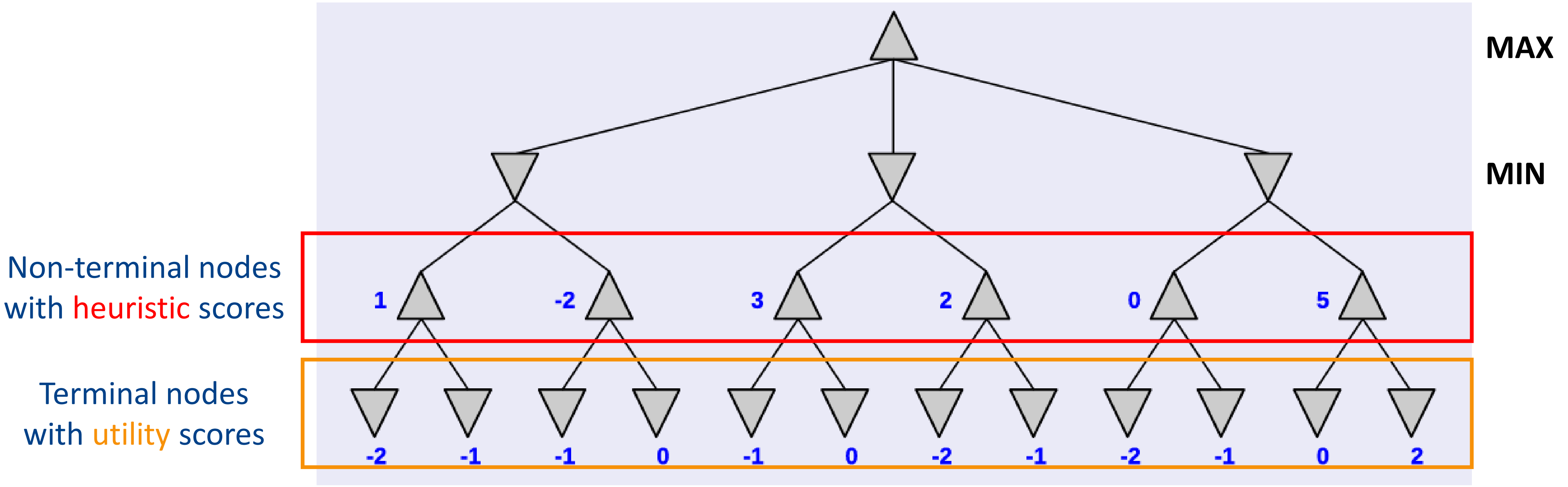
- Game heuristics typically combinations of different features

  $h(s) = w_1 * x_1(s) + w_2 * x_2(s) + \ldots + w_n * x_n(s)$

- Example features for chess:
  - x_1 = value of all my remaining pieces
  - x_2 = value of all opponent's remaining pieces
  - x_3 = do I have a queen left
  - …
- Weights $w_1, w_2, \ldots, w_n$ indicate how important each feature is

Example

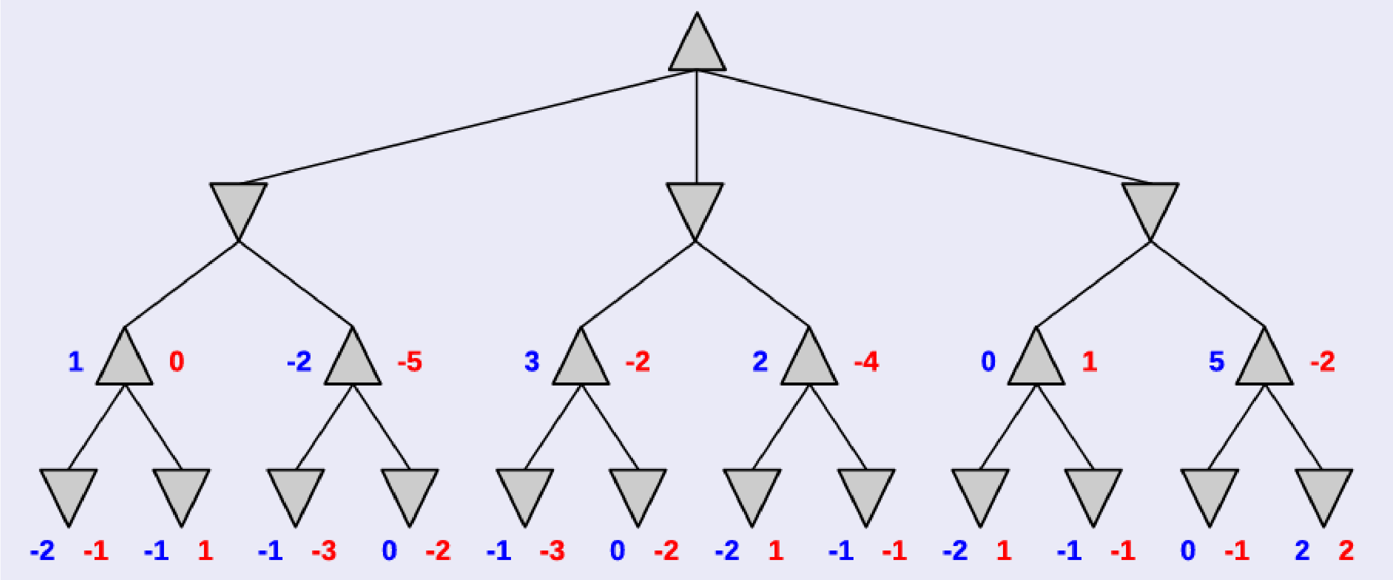
- **Q:** What is the outcome of the game if both players can explore all nodes (until game terminal nodels)?
- **Q:** What is the outcome if each player can search only two levels deep?



Example

- The two players are **unlikely** to use the same heuristic
- **Q:** what is the end state (outcome) if each player searches two levels deep but use different heuristics (**MAX** blue, **MIN** red)?
    - **Important**: both heuristics are estimates of utility for **MAX**!

- Depends on whose turn it is ☺



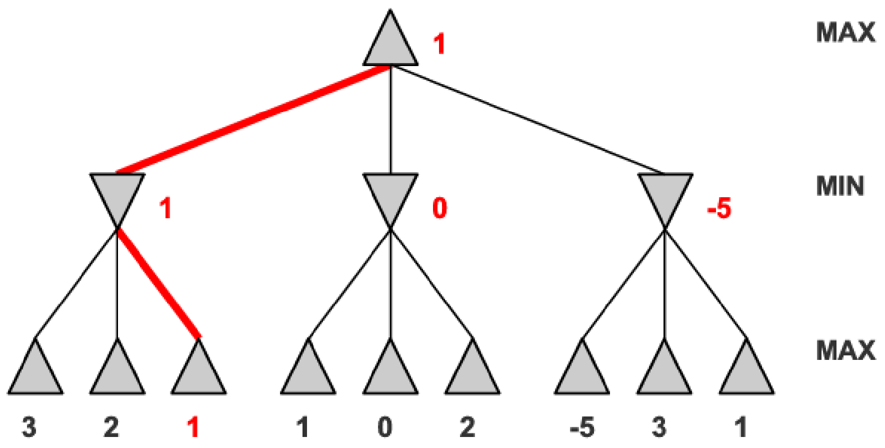
Content

- Introduction
- Minimax
- Heuristic Minimax
- Alpha-Beta Pruning

Alpha-beta pruning

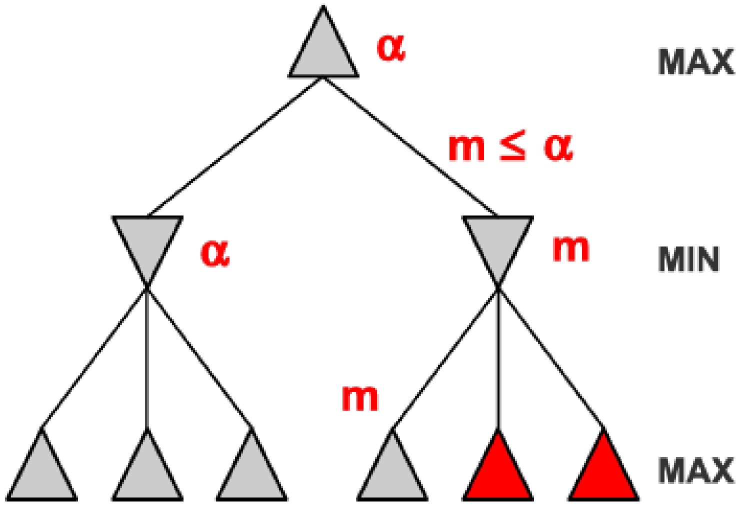
- Number of states increases exponentially with the number of turns
- We can effectively cut this number **in half** with **alpha-beta pruning**
- **Q:** Can we compute the minimax value of a node without traversing the whole game (sub)tree?
- **Yes!**

$$m(s) = \max(\min(3, 2, 1), \min(1, X, X), \min(-5, X, X)) = 1$$

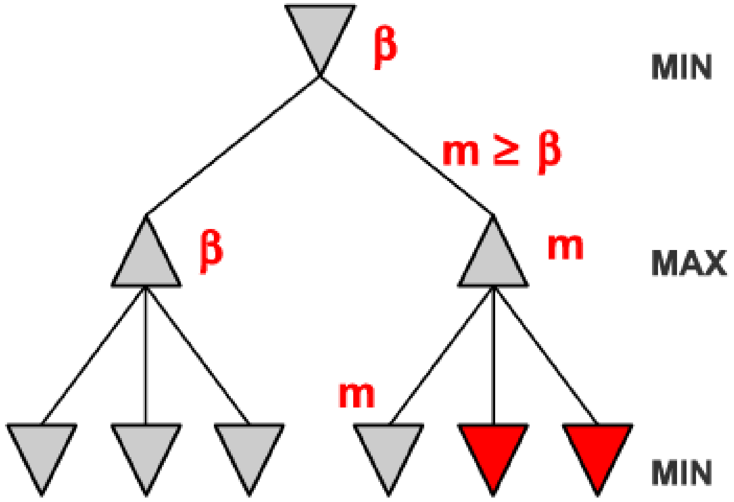


Alpha-beta pruning

- We prune whenever we're certain that unexplored moves cannot change the result of the max/min operation
  - Cannot lead to a **move that is better** than the best found so far
- If pruning below the **MIN** node (but when root is **MAX** node): **alpha pruning**
- If pruning below the **MAX** node (but when root is **MIN** node): **beta pruning**



α – the largest MAX value found



β – the smallest MIN value found

Minimax with alpha-beta pruning

beta pruning

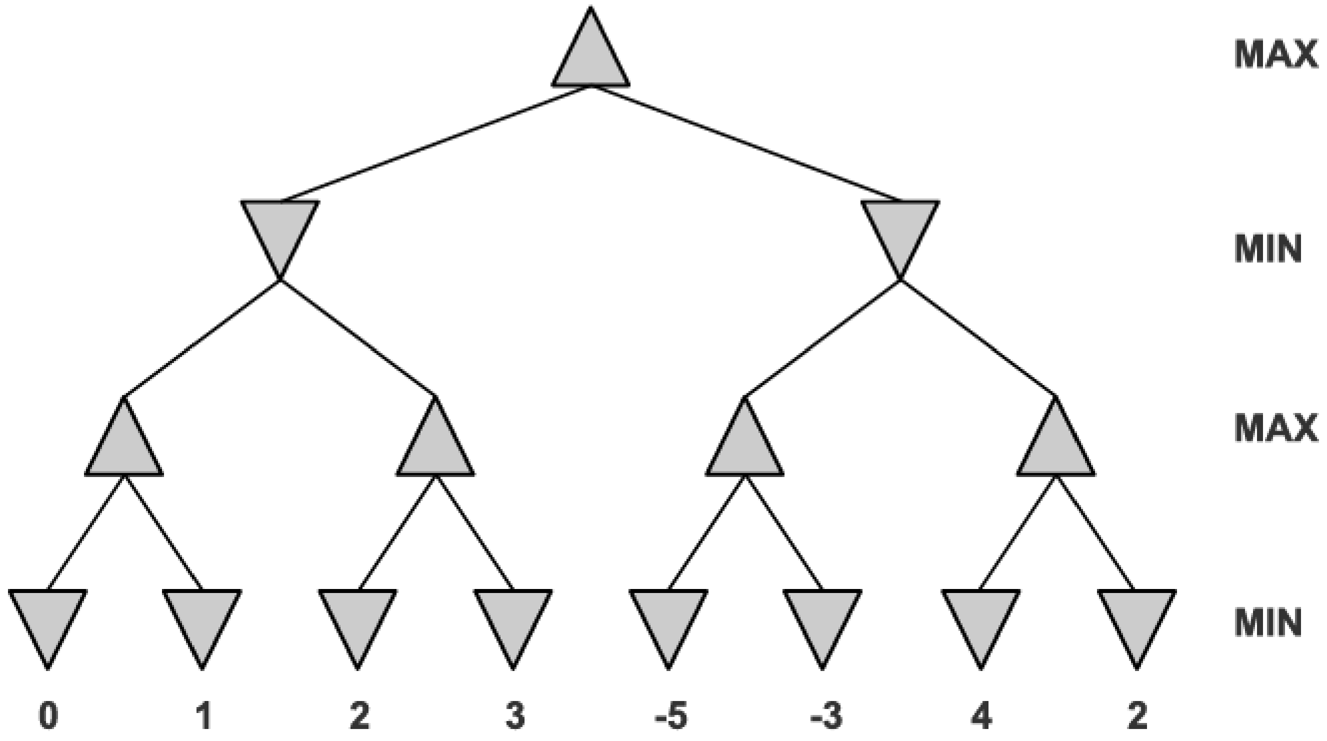
```
max_val(s,α,β)
  if terminal(s)
    return utility(s)
  m = α
  for t in succ(s)
    m = max(m, min_val(t, m, β))
    if m ≥ β
      return β
  return m
```

alpha pruning

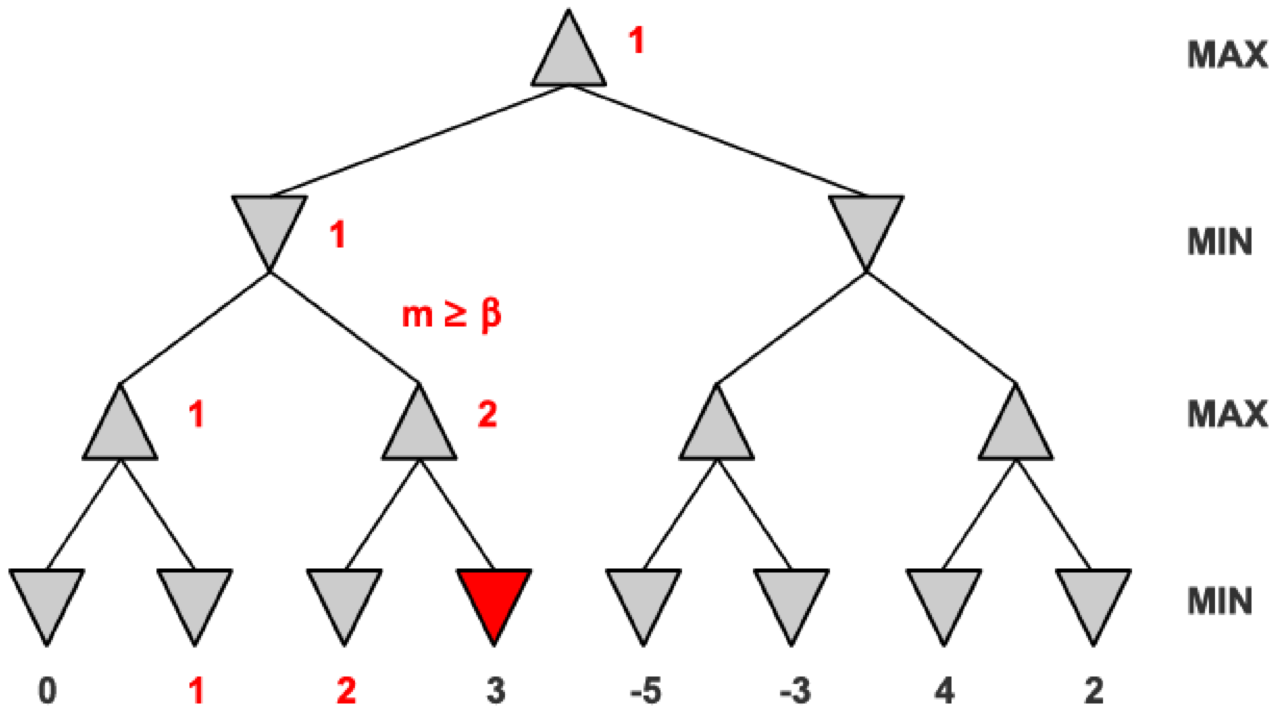
```
min_val(s,α,β)
  if terminal(s)
    return utility(s)
  m = β
  for t in succ(s)
    m = min(m, max_val(t, α, m))
    if m ≤ α
      return α
  return m
```

Initially: max_val(s_0,-inf, +inf)

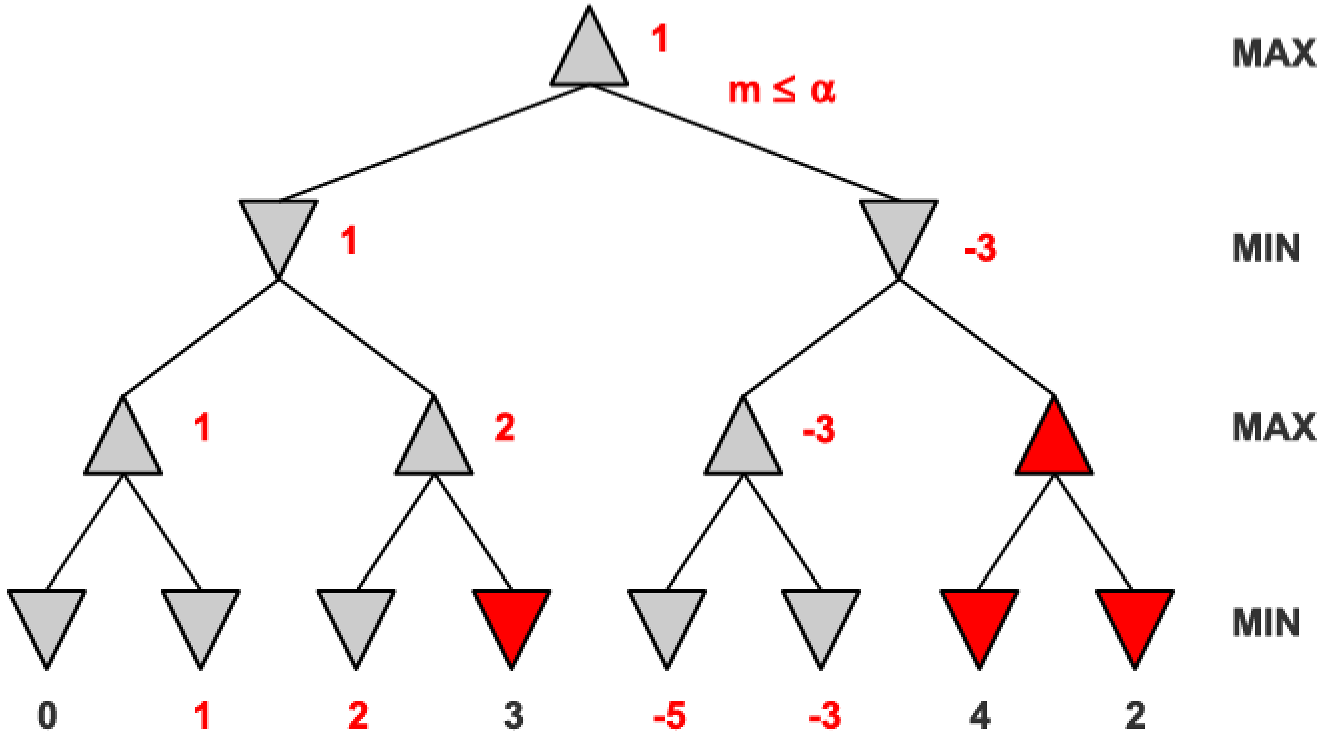
Alpha-beta pruning example



Alpha-beta pruning example



Alpha-beta pruning example



Adversarial search: **takeaways**

- **Game playing** is a search problem in which opposing players take turns (i.e., we have an adversary)
- **Minimax algorithm** finds an optimal strategy that minimizes the maximum expected loss that an opponent can inflict
- In real games it is impossible to search through the complete game tree, thus we **cut the search off** at a certain depth and use a **heuristic** function to estimate the values of game states
  - Different players use different heuristic functions
  - The opponent's heuristic is in principle unknown
- **Alpha-beta pruning** reduces the number of nodes to traverse

Questions?


ਸਵਾਲ?
Küsimusi?
Turite klausimų?
Awọn ibeere?
Pitanja?
Sorusu olan?
Dúvidas?
質問は？
有问题吗？
Fragen?
¿Preguntas?
Frågor?
Questions?
Pytania?
Domande?
Питання?
Vragen?
Ερωτήσεις;
Porandukuéra?
คำถาม?
أسئلة؟