Artificial Intelligence

4. Game Playing

Prof. Bojana Dalbelo Bašić
Assoc. Prof. Jan Šnajder

University of Zagreb
Faculty of Electrical Engineering and Computing

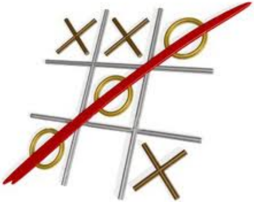
Academic Year 2019/2020

v2.12

Outline

Games

- Also a state space search problem, but the difference is 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**
- We focus on **deterministic** games with **two players**, **complete information** and **zero-sums**


Problem formalization

A state space search problem comprised of the following:

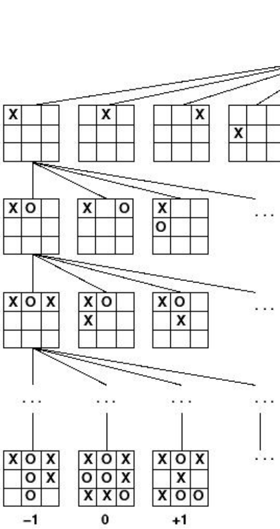
Game

- **Initial game state** s_0
- **Successor function** $\mathrm{succ} : S \to \wp(S)$, which defines the legal game moves (transitions between states)
- **Terminal state test** $\mathrm{terminal} : S \to \{\top, \bot\}$
- **Payoff function** $\mathrm{utility} : S \to \mathbb{R}$, which assigns numeric values awarded to a player in a terminal game state

  E.g., in chess: $\mathrm{utility}(s) \in \{+1, 0, -1\}$

The initial state and the successor function implicitly define the **game tree**

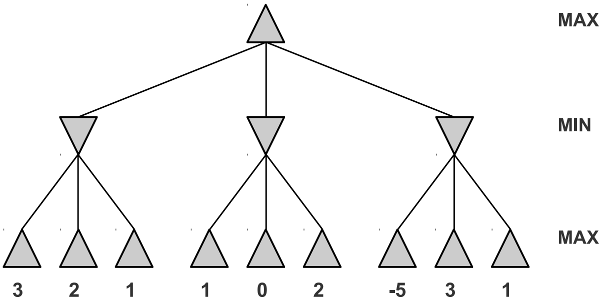
Game tree



Outline

Minimax method

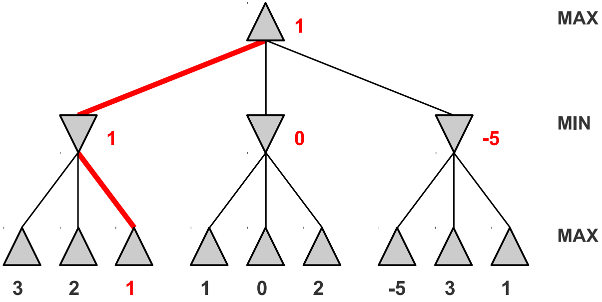
- Let's call the players MAX (computer) and MIN (opponent)
- MAX player tries to **maximize** his win, whereas MIN player tries to **minimize** MAX's win
- Players take turn: node at even depths are MAX nodes, nodes at odd depths are MIN nodes



- **Q**: What is the optimal strategy of MAX player in this case?

Optimal strategy

- MAX player's **optimal strategy** is the one that ensures the highest win, assuming that MIN player uses the same strategy
- Each player chooses a strategy so as to **minimize the maximum loss**

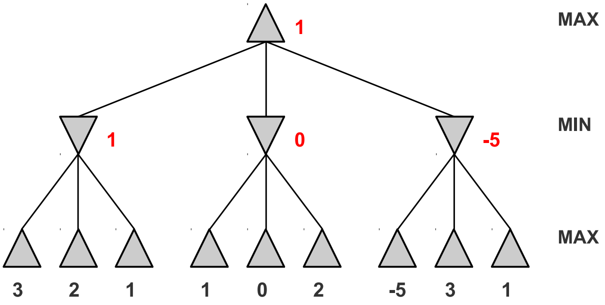


- To determine the **optimal strategy** of a player whose turn is next, we compute the **minimax value** of the root note

Minimax value

- The minimax value of node s is defined recursively:

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



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

Minimax algorithm

function $\mathrm{maxValue}(s)$
  if $\mathrm{terminal}(s)$ **then return** $\mathrm{utility}(s)$
  $m \leftarrow -\infty$
  for $t \in \mathrm{succ}(s)$ **do**
    $m \leftarrow \max(m, \mathrm{minValue}(t))$
  return m

function $\mathrm{minValue}(s)$
  if $\mathrm{terminal}(s)$ **then return** $\mathrm{utility}(s)$
  $m \leftarrow +\infty$
  for $t \in \mathrm{succ}(s)$ **do**
    $m \leftarrow \min(m, \mathrm{maxValue}(t))$
  return m

NB: This is a **depth-first search** implemented via two mutually recursive functions (which alternate between MAX and MIN states)

Minimax algorithm – remarks

- In practice, the opponent's strategy is **unknown** (most probably different from that of MAX player) and therefore the opponent's moves cannot be predicted perfectly (otherwise the game would be boring anyway)
- Therefore, 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**, thus space complexity is $\mathcal{O}(m)$, where m is the depth of the game-tree
- However, time complexity is $\mathcal{O}(b^m)$, where b is the game branching factor. This is very unfortunate!

Outline

Imperfect decisions

- In reality, we don't have the time to search through the complete game tree all the way down to the terminal nodes
- We must make **time-bounded** and **imperfect decisions**
- We need to **cut off** the search at a certain level d and make an **estimate** of the pay-off function using a **heuristic function**
- Value of h(s) is an estimate of the expected utility of state s for player MAX
- E.g., for chess: the sum of material values of player's chess pieces
- Heuristic function is commonly defined as a **weighted linear combination** of various features:

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

- **NB:** Players typically use different heuristic functions (this is why they appear to be unpredictible)

Minimax algorithm (2)

Minimax with a cut-off

function $\mathrm{maxValue}(s, d)$
  if $\mathrm{terminal}(s)$ **then return** $\mathrm{utility}(s)$
  if $d = 0$ **then return** $\mathsf{h}(s)$
  $m \leftarrow -\infty$
  for $t \in \mathrm{succ}(s)$ **do**
    $m \leftarrow \max(m, \mathrm{minValue}(t, d-1))$
  return m

function $\mathrm{minValue}(s, d)$
  if $\mathrm{terminal}(s)$ **then return** $\mathrm{utility}(s)$
  if $d = 0$ **then return** $\mathsf{h}(s)$
  $m \leftarrow +\infty$
  for $t \in \mathrm{succ}(s)$ **do**
    $m \leftarrow \min(m, \mathrm{maxValue}(t, d-1))$
  return m

Practice time: Minimax value

Let a game tree be defined by transitions $succ(A) = \{B, C, D\}$, $succ(B) = \{E, F\}$, $succ(C) = \{G, H\}$, $succ(D) = \{I, J\}$. The heuristic values of the leaves are $h(E) = -1$, $h(F) = 3$, $h(G) = 2$, $h(H) = 4$, $h(I) = 5$, $h(J) = 1$. What is the minimax value of the node A, if this is a MIN node?

(A) -1

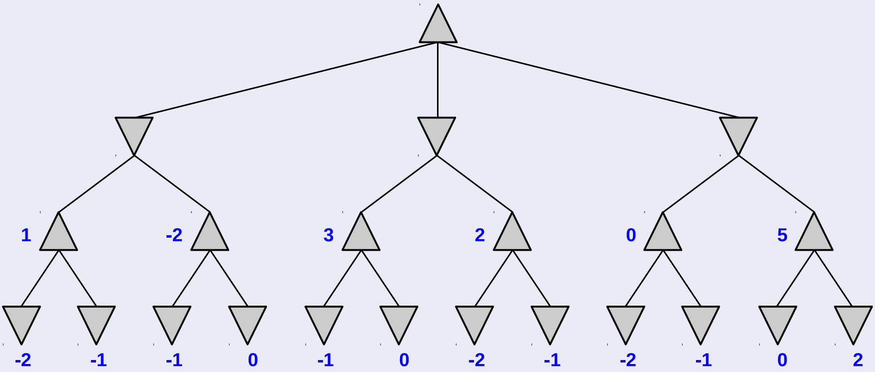
(B) 3

(C) 2

(D) 4

Practice time: Minimax with a heuristics

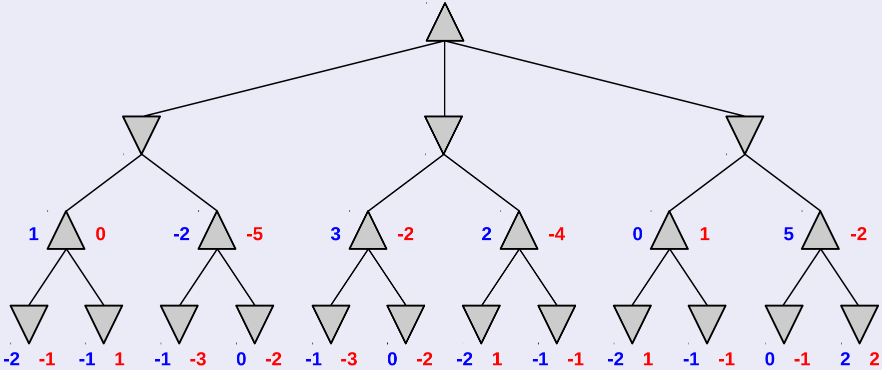
What is the game's end state, if each of the players search two levels deep?



Q: What if they search three levels deep?

Practice time: Minimax with two heuristics

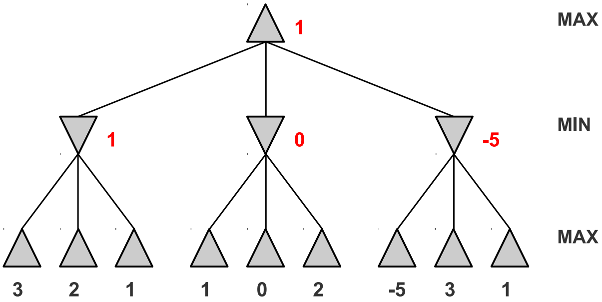
What is the game's end state, if each of the players search two levels deep, but use different heuristics, h_1 (blue) and $-h_2$ (red):



Outline

Alpha-beta pruning

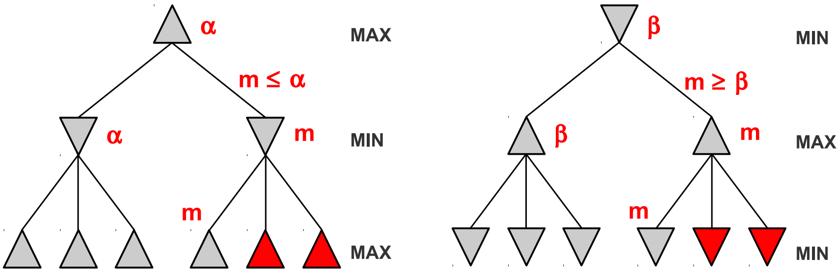
- Number of states increases exponentially with the number of turns
- We can effectively cut this number in half using **alpha-beta pruning**
- **Q:** Can we compute the minimax value without traversing the whole game tree? **A:** Yes!



$$m(s_0) = \max\big(\min(3,2,1), \min(1, X, X), \min(-5, X, X)\big) = 1$$

Alpha-beta pruning

- We prune every time we're certain that the unexplored moves **can under no circumstances be better** than the best move found so far
- If pruning below the MIN node: **alpha pruning**
- If pruning below the MAX node: **beta pruning**



α – the largest MAX value found

β – the smallest MIN value found

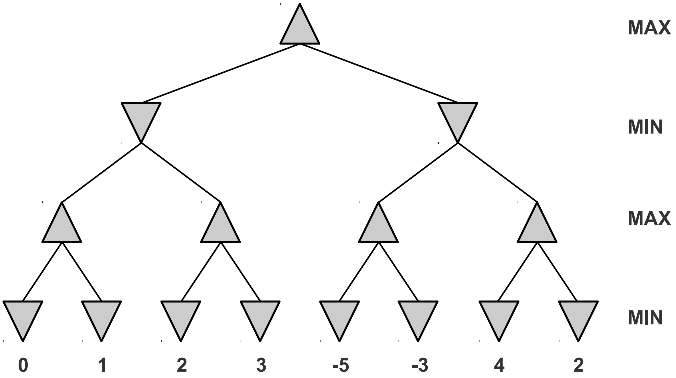
Minimax algorithm (3)

Minimax with alpha-beta pruning

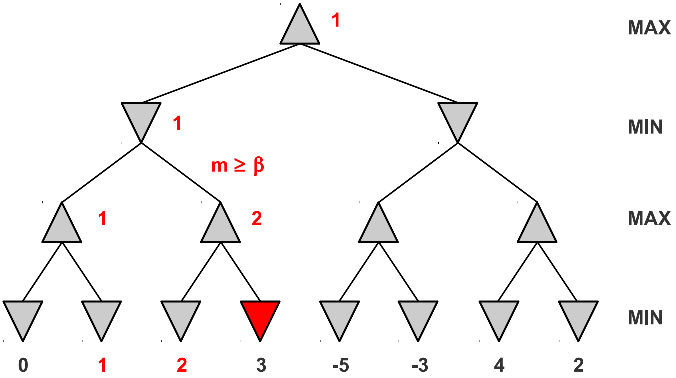
function $\mathrm{maxValue}(s, \alpha, \beta)$ -- *initially:* $\mathrm{maxValue}(\mathrm{s}_0, -\infty, +\infty)$
  if $\mathrm{terminal}(s)$ **then return** $\mathrm{utility}(s)$
  $m \leftarrow \alpha$
  for $t \in \mathrm{succ}(s)$ **do**
    $m \leftarrow \max(m, \mathrm{minValue}(t, m, \beta))$
    if $m \geq \beta$ **then return** β -- *beta pruning*
  return m

function $\mathrm{minValue}(s, \alpha, \beta)$
  if $\mathrm{terminal}(s)$ **then return** $\mathrm{utility}(s)$
  $m \leftarrow \beta$
  for $t \in \mathrm{succ}(s)$ **do**
    $m \leftarrow \min(m, \mathrm{maxValue}(t, \alpha, m))$
    if $m \leq \alpha$ **then return** α -- *alpha pruning*
  return m

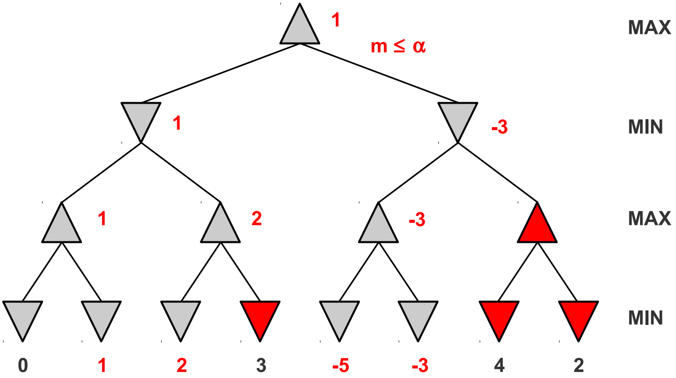
Alpha-beta pruning – example (1)



Alpha-beta pruning – example (2)



Alpha-beta pruning – example (3)



Lab assignment: Matches game

Write a program that plays the Matches game using **minimax algorithm**.

In this game, there are two players and n piles of matches. Each pile may contain a different number of matches. The two players play in turn. In each turn, one can choose a pile and take away at least one and at most k matches. The game is over when all matches have been removed. The player whose turn was last completed looses the game.

Implement a minimalistic user interface that shows the current game state and enables the user to play against the computer. After the user makes a move, the program should print out whether her move is minimax-optimal. The input to the program are the number n (the number of piles), k (the maximum number of matches that can be removed in one move), and the initial number of matches in each of the n piles.

Lab assignment: Checkers

Write a program that plays Checkers using the **minimax algorithms** with a search cut-off and **alpha-beta pruning**.

Define at least two different heuristic functions for estimating the value of a game state. When designing heuristic functions, pay special care to situations when the pieces become crowned. You should put a time limit on the search; the limit may be based on the number of explored states, search depths, or wall-clock time.

Implement a minimalistic user interface that enables the user to play against the computer and shows the current game state. After the user makes a move, the program should print out whether her move is minimax-optimal. Implement the program so that it can play against itself, whereby the players may use different heuristic functions.

Wrap-up

- Game playing is a search problem in which opposing players take turns
- **Minimax algorithm** finds an optimal strategy that **minimizes the maximum expected loss** that an opponent can inflict
- In reality it is impossible to search through the complete game tree, thus we cut off the search 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 uknown
- **Alpha-beta pruning** reduces the number of nodes to traverse
- Things we didn't talk about: multiplayer games, games that include an element of chance, Monte Carlo Tree Search (MCTS)

Next topic: Knowledge representation