Two-player Games

ZUI 2016/2017

Branislav Bošanský

bosansky@fel.cvut.cz

Two Player Games

- Important test environment for AI algorithms
- Benchmark of AI
  - Chinook (1994/96) – world champion in checkers
  - Deep Blue (1997) – beats G. Kasparov in chess (3.5 – 2.5)
  - …
  - Alpha Go (2016) – beats Lee Sedol in Go (4 – 1)
  - DeepStack (2016/2017) – beats Poker Pros (https://www.deepstack.ai/)
  - …

Game-tree Search / Adversarial Search

- until now – only the searching player acts in the environment
- there could be others:
  - Nature – stochastic environment (MDP, POMDP, …)
  - other agents – rational opponents

Game-tree Search / Adversarial Search

- until now – only the searching player acts in the environment
- there could be others:
  - Nature – stochastic environment (MDP, POMDP, …)
  - other agents – rational opponents

- Game Theory
  - mathematical framework that describes optimal behavior of rational self-interested agents
  - Mag. OI → B4M36MAS (Multi-agent Systems)

Game-tree Search / Adversarial Search

- What are the basic games categories?
  - perfect / imperfect information
  - deterministic / stochastic
  - zero-sum / general-sum
  - finite / infinite
  - two-player / n-player
  - …

Game-tree Search / Adversarial Search

- What are the basic games categories?
  - **perfect** / imperfect information
  - **deterministic** / stochastic
  - **zero-sum** / general-sum
  - **finite** / infinite
  - **two-player** / n-player
  - …

Game-tree Search / Adversarial Search

- What are the basic games categories?
  - **perfect** / imperfect information
  - **deterministic** / stochastic
  - **zero-sum** / general-sum
  - **finite** / infinite
  - **two-player** / n-player
  - …
- What is the goal?

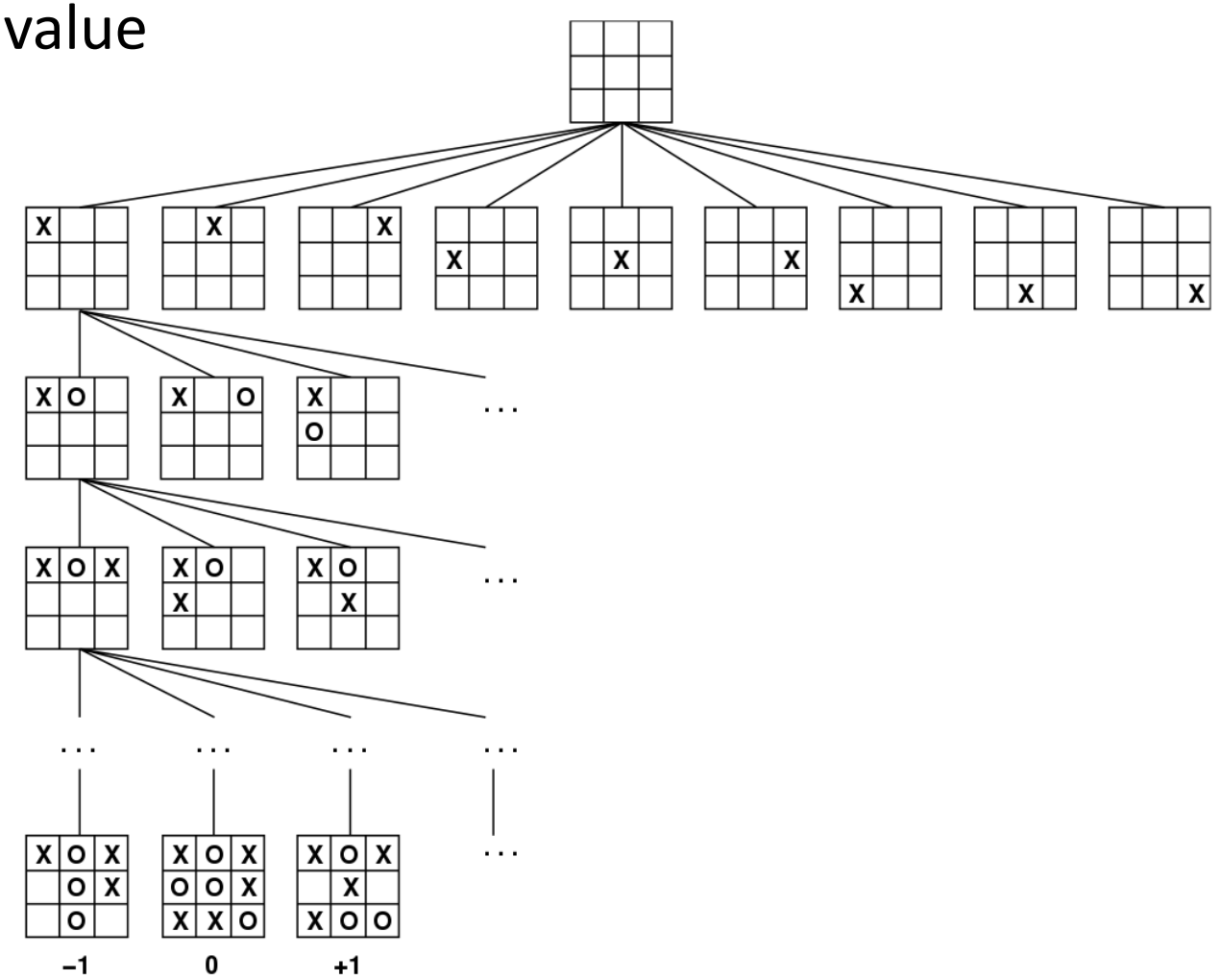
Game-tree Search / Adversarial Search

- What are the basic games categories?
  - **perfect** / imperfect information
  - **deterministic** / stochastic
  - **zero-sum** / general-sum
  - **finite** / infinite
  - **two-player** / n-player
  - …

- What is the goal?
  - Finding an optimal **strategy** (i.e., what to play in which situation)

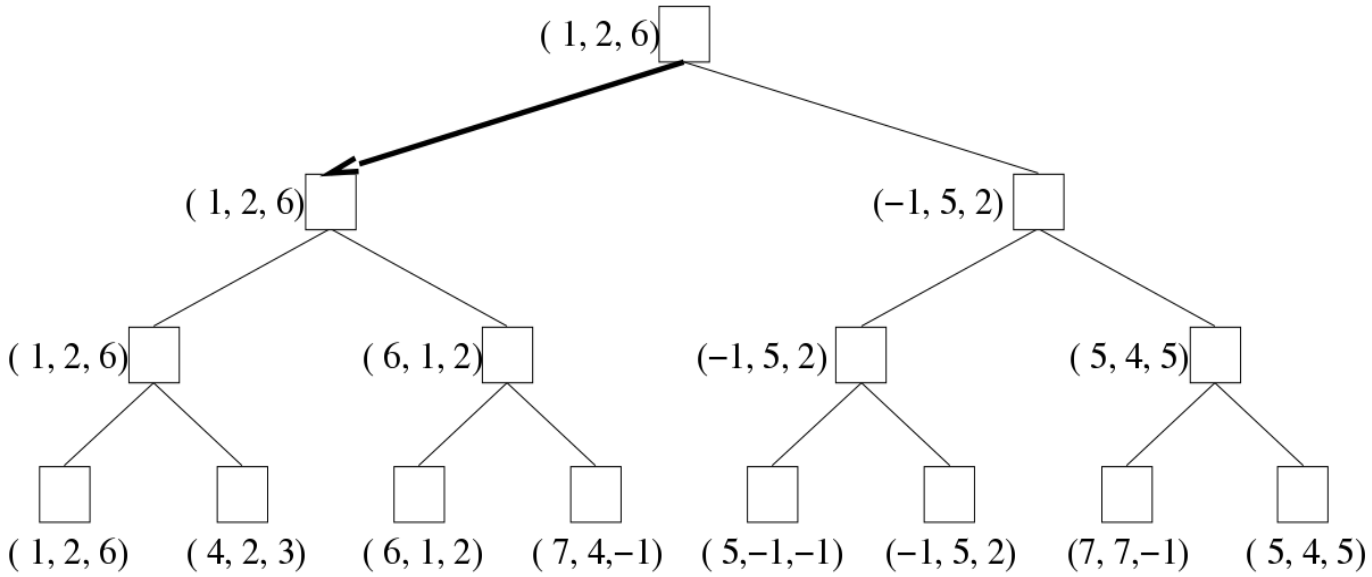
Game-tree Search / Adversarial Search

- Players are rational – each player wants to maximize her/his utility value



Game-tree Search / Adversarial Search

- Players are rational – each player wants to maximize her/his utility value



Minimax

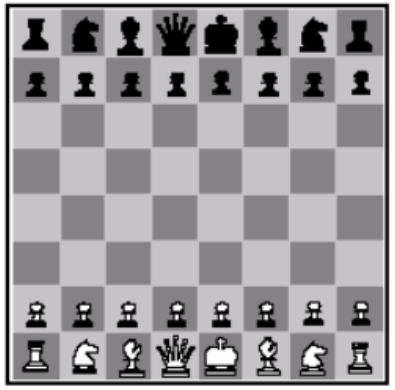
- **function** minimax(node, Player)
- **if** (node is a terminal node) **return** utility value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- α := max(α, minimax(child, switch(Player)))
-
- **return** α
- **else**
- **for each** child of node
- β := min(β, minimax(child, switch(Player)))
-
- **return** β

Minimax in Real Games

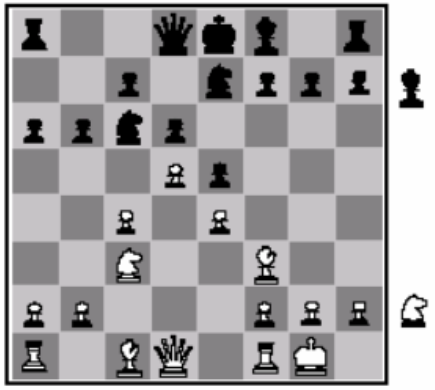
- search space in games is typically **extremely large**
  - exponential in branching factor b^d
    - e.g., 35 in chess, up to 360 in Go, up to 45000 in Arimaa
    - No-limit heads-up poker has 10^{160} decision points
- we have to limit the depth of the search
- we need an evaluation function

Minimax in Real Games

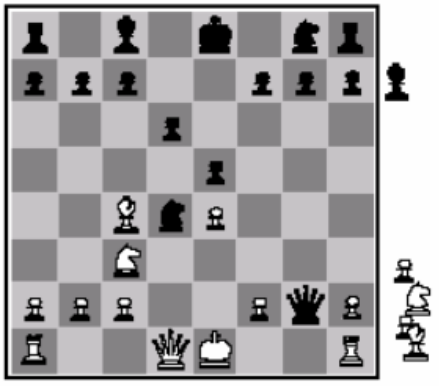
- search space in games is typically **very large**
  - exponential in branching factor b^d
    - e.g., 35 in chess, up to 360 in Go, up to 45000 in Arimaa
    - No-limit heads-up poker has 10^{160} decision points
- we have to limit the depth of the search
- we need an evaluation function



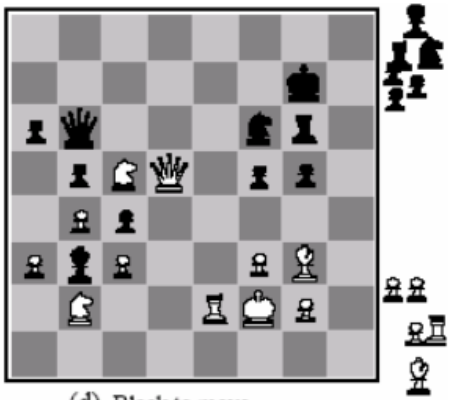
(a) White to move
Fairly even



(b) Black to move
White slightly better



(c) White to move
Black winning



(d) Black to move
White about to lose

Minimax

- **function** minimax(node, depth, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- α := max(α, minimax(child, depth-1, switch(Player)))
-
- **return** α
- **else**
- **for each** child of node
- β := min(β, minimax(child, depth-1, switch(Player)))
-
- **return** β

Minimax in Real Games - Problems

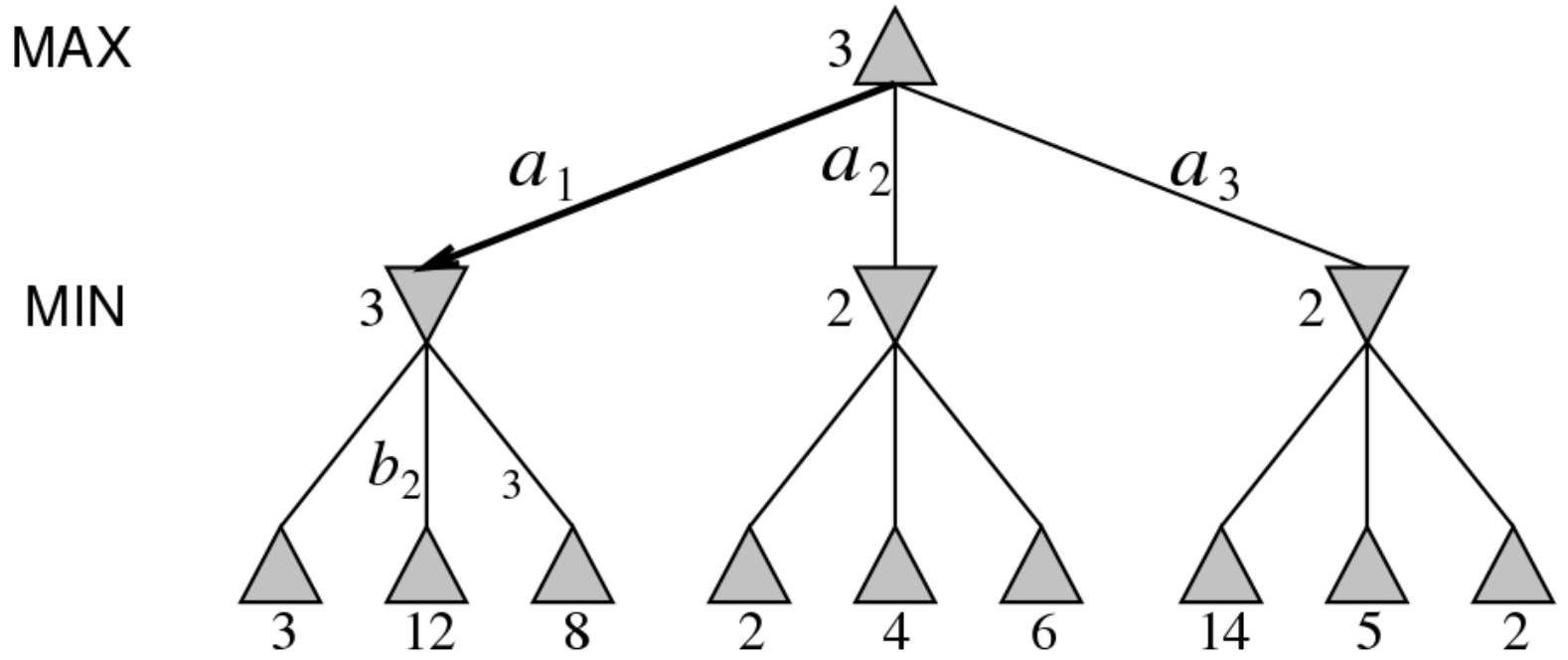
- good evaluation function
- depth?
  - horizon problem
  - iterative deepening
  - searching deeper does not always improve the results

Minimax in Real Games - Problems

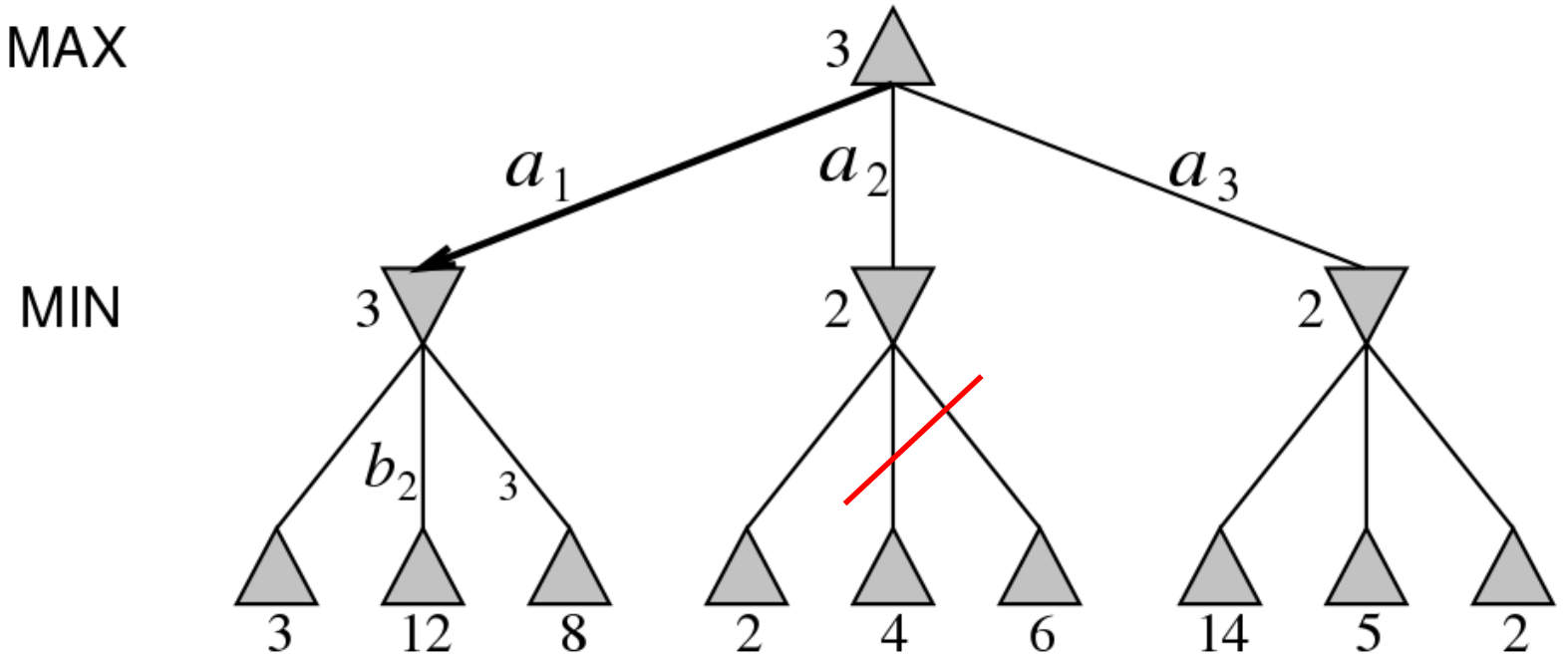
- good evaluation function
- depth?
  - horizon problem
  - iterative deepening
  - searching deeper does not always improve the results

- online vs. offline decision making
  - game playing vs. equilibrium computation
  - appear also in robotics / planning / decision making

Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning

- **function** alphabeta(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- α := max(α, alphabeta(child, depth-1, α, β, switch(Player)))
- **if** (β≤α) break
- **return** α
- **else**
- **for each** child of node
- β := min(β, alphabeta(child, depth-1, α, β, switch(Player)))
- **if** (β≤α) break
- **return** β

Negamax

- **function** negamax(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** the heuristic value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- α := max(α, -negamax(child, depth-1, -β, -α, switch(Player)))
- **if** (β≤α) break
- **return** α
- **else**
- **for each** child of node
- β := min(β, alphabeta(child, depth-1, α, β, not(Player)))
- **if** (β≤α) break
- **return** β

Aspiration Search

- [α, β] interval – window
- alphabeta initialization [-∞, +∞]

Aspiration Search

- $[\alpha, \beta]$ interval – window
- alphabeta initialization $[-\infty, +\infty]$
- what if we use $[\alpha_0, \beta_0]$
  - x = alphabeta(node, depth, α_0, β_0,player)
  - $\alpha_0 \leq x \leq \beta_0$ - we found a solution
  - $x \leq \alpha_0$ - failing low (run again with $[-\infty, x]$)
  - $x \geq \beta_0$ - failing high (run again with $[x, +\infty]$)

NegaScout – Main Idea

- enhancement of alpha-beta algorithm
- assume some heuristic that determines move ordering
  - the algorithm assumes that the first action is the best one
  - after evaluating the first action, the algorithm checks whether the remaining actions are worse
  - the “test” is performed via null-window search

Scout –Test

- what we really need at that moment is a bound (not the precise value)

Scout –Test

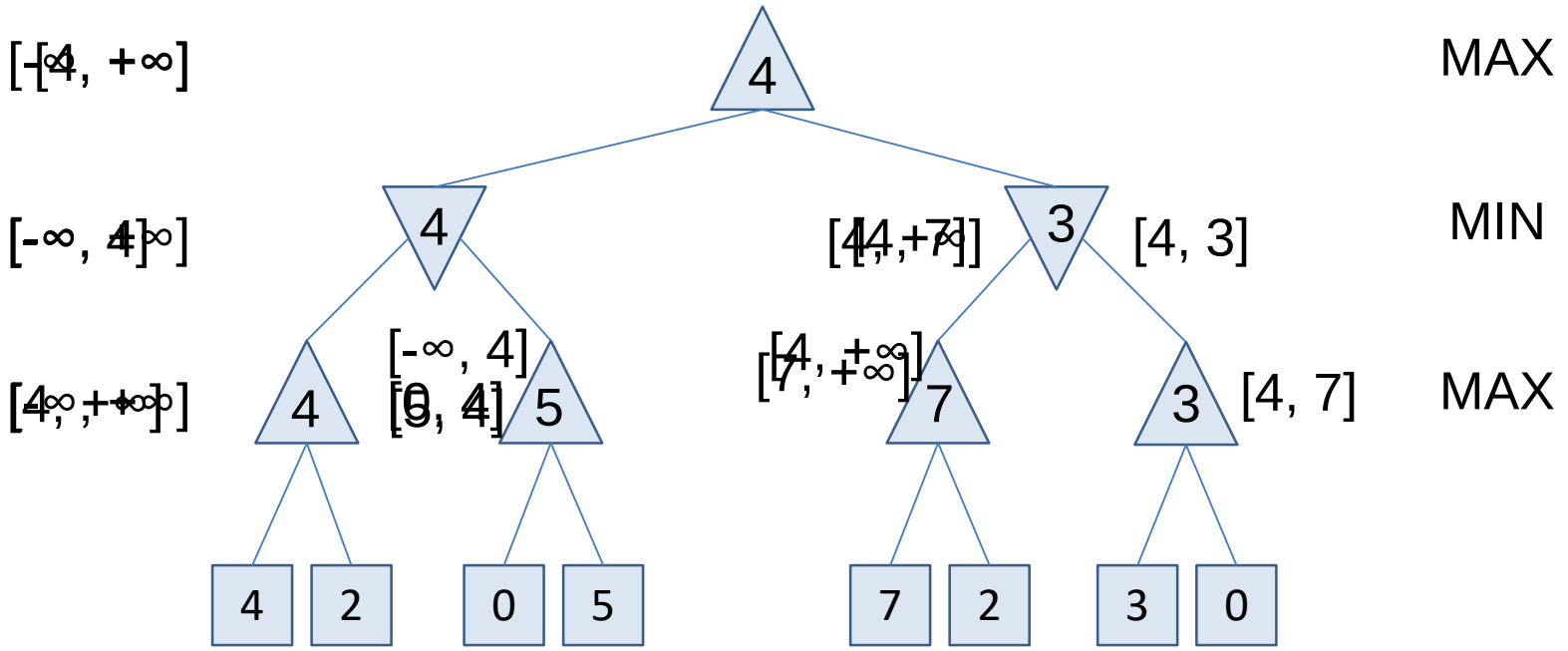
- what we really need at that moment is a bound (not the precise value)
- Remember Aspiration Search?
  - $x \leq \alpha_0$ - failing low (we know, that solution is ≤ x)
  - $x \geq \beta_0$ - failing high (we know, that solution is ≥ x)
- What if we use a null-window [α, α+1] (or [α,α])?
  - we obtain a bound …

NegaScout

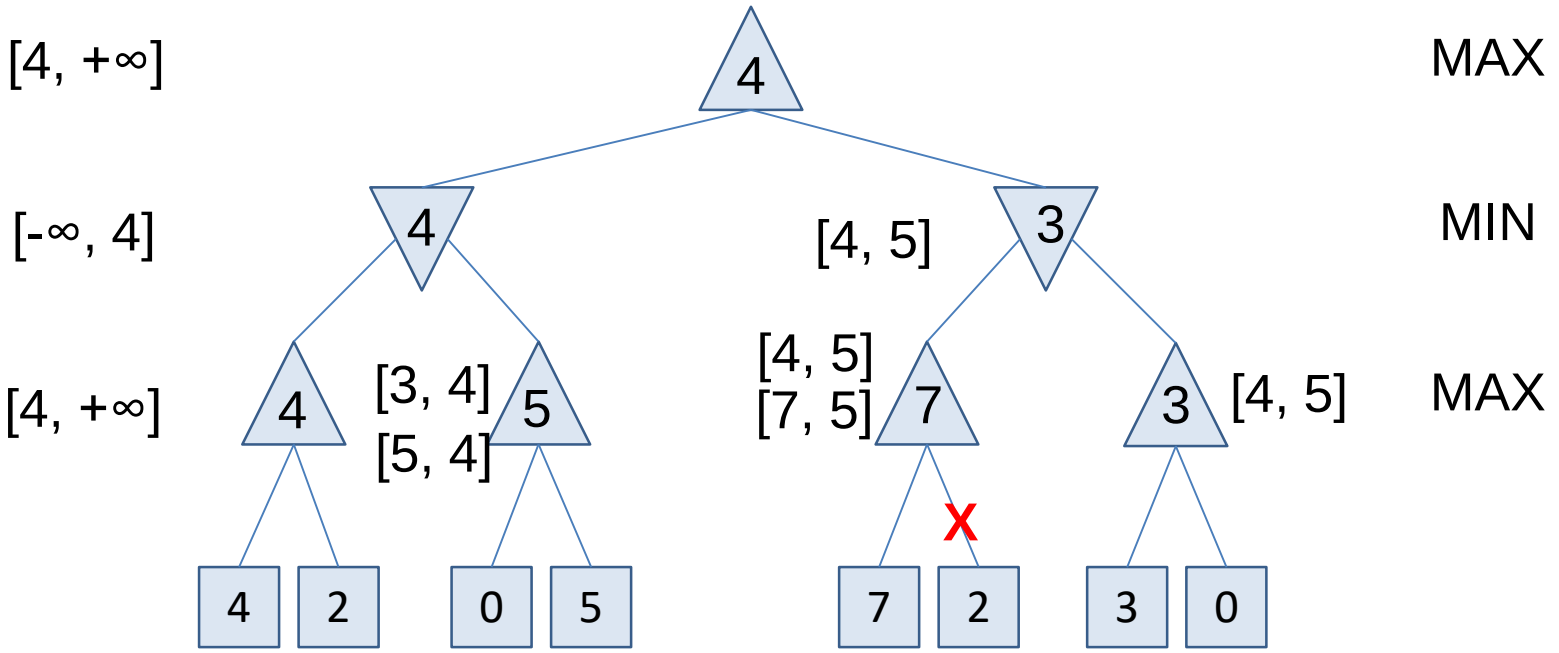
function negascout(node, depth, α, β, Player)

- **if** ((depth = 0) or (node is a terminal node)) **return** eval(node)
- b := β
- **for each** child of node
- v := -negascout(child, depth-1, -b, -α, switch(Player)))
- **if** ((α < v) and (child is not the first child))
- v := -negascout(child, depth-1, -β, -α, switch(Player)))
- α := max(α, v)
- **if** (β≤α) break
- b := α + 1
- **return** α

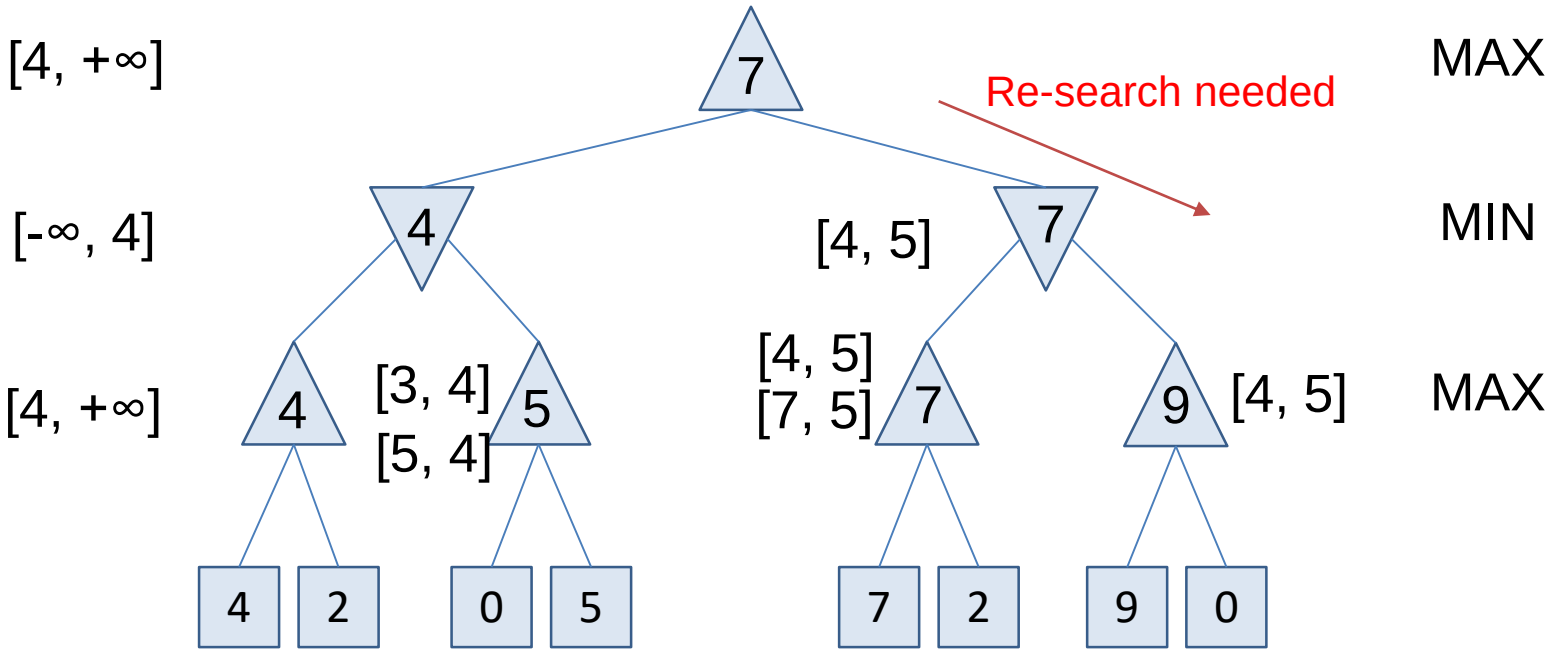
Alpha-Beta vs. Negascout



Alpha-Beta vs. Negascout



Alpha-Beta vs. Negascout

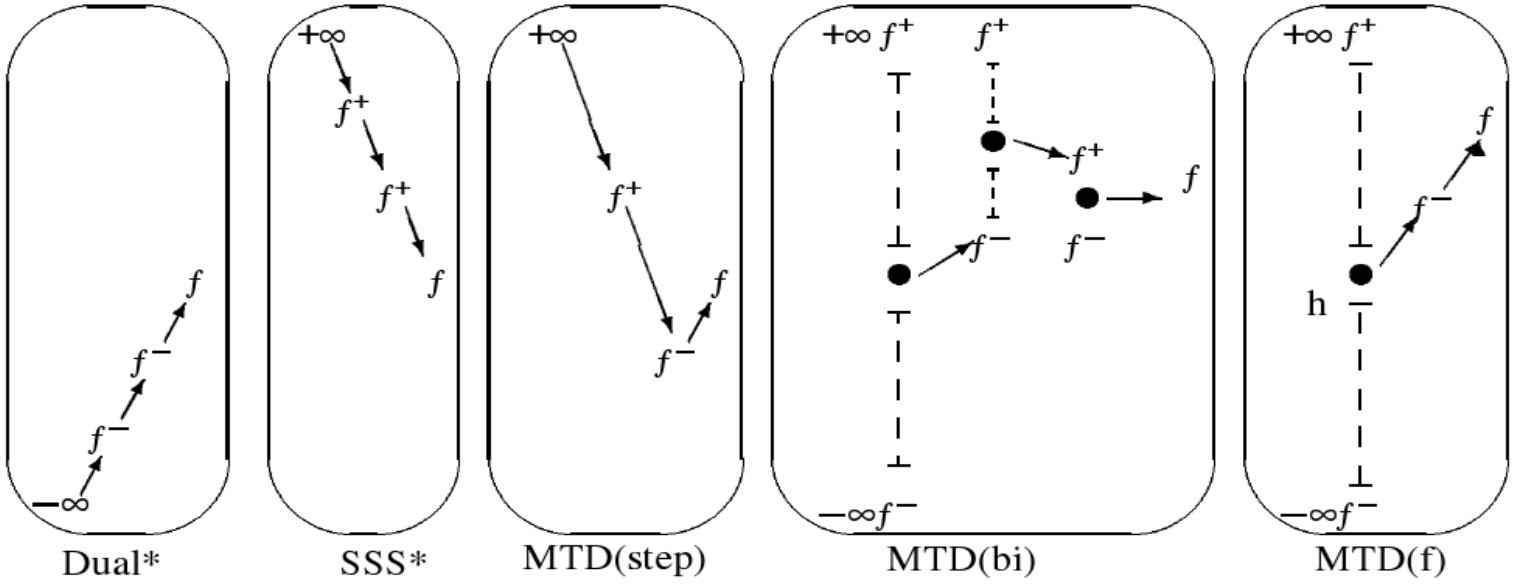


NegaScout

- also termed Principal Variation Search (PVS)
- dominates alpha-beta
  - never evaluates more different nodes than alpha-beta
  - can evaluate some nodes more than once
- depends on the move ordering
- can benefit from transposition tables
- generally 10-20% faster compared to alpha-beta

MTD

- Memory-enhanced Test Driver

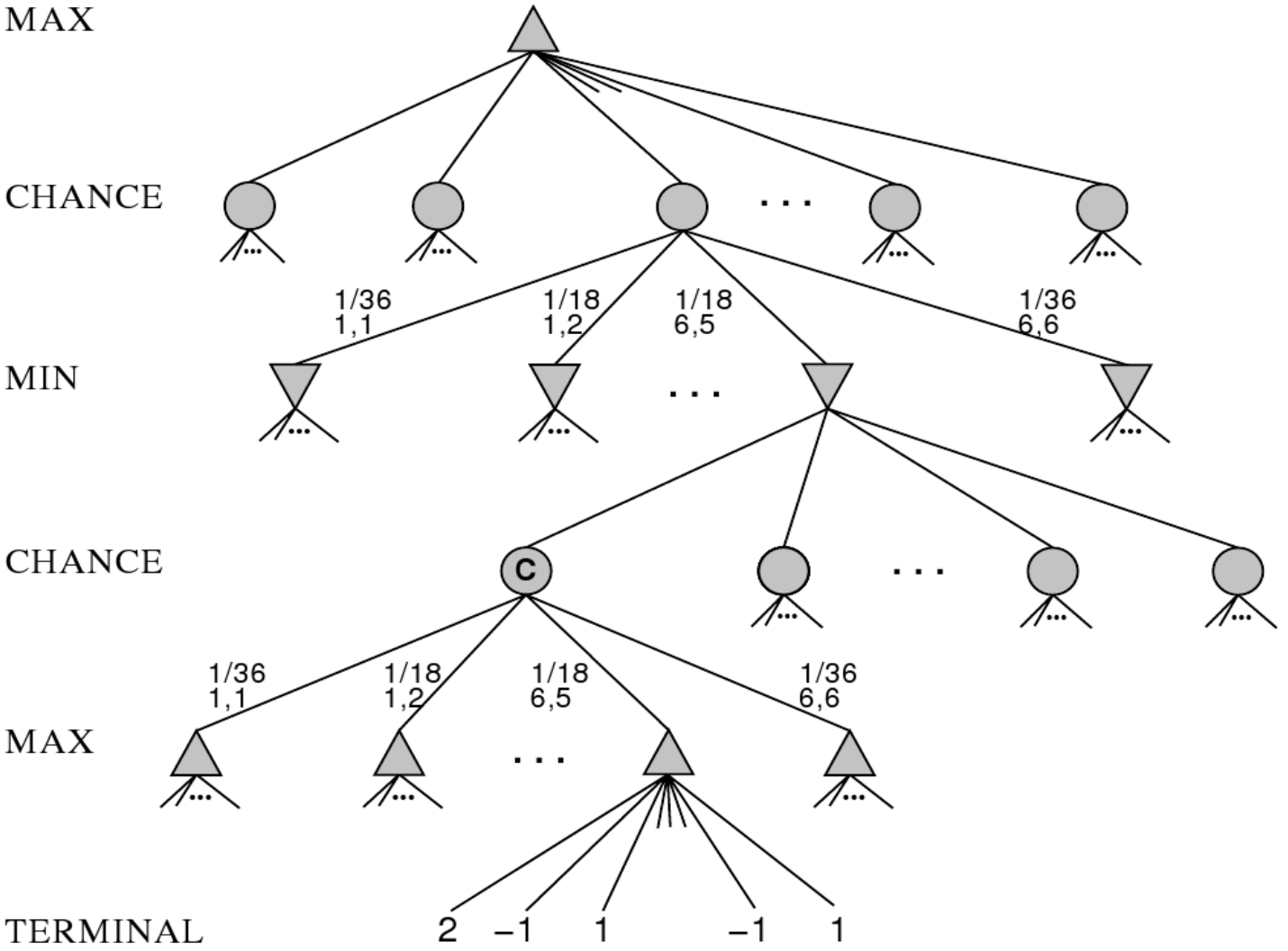


- Best-first fixed-depth minimax algorithms. Plaat et. al. , In *Artificial Intelligence,* Volume 87, Issues 1-2, November 1996, Pages 255-293

Further Topics

- more advanced algorithms
  - Monte-Carlo Tree Search (leading algorithm for many games)
  - further enhanced with many heuristics and techniques
- more complex games
  - games with uncertainty
    - chance (Nature player), calculating expected utilities
    - imperfect information (players cannot distinguish certain states)

Other Games - Chance nodes



Games and Game Theory in AIC

- more fundamental research
  - general algorithms for solving sequential games with imperfect information
  - implementation of domain independent algorithms

Invitation

Artificial Intelligence Goes All-In: Computers Playing Poker

Lecture by prof. Michael Bowling, head of Computer Poker Research Group at University of Alberta.

Prof. Michael Bowling

- World-famous expert on AI and reinforcement learning
- Led many outstanding computer poker results:
  - Polaris, beating pros in heads-up limit poker
  - Cepheus, playing optimally heads-up limit poker
  - **DeepStack, beating pros in heads-up no-limit**
  - Two publications on poker in prestigious Science
- Proposed Atari games as a benchmark for AI
- Won one of the first RoboCup challenges

March 30, 2017 at 16:00

Auditorium KN:E-107, FEL CTU,
Karlovo nám. 13, Prague 2

UNIVERZITA KARLOVA
Matematicko-fyzikální fakulta

COMPUTER SCIENCE

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE