Foundations of Artificial Intelligence

43. Board Games: Alpha-Beta Search

Thomas Keller

Universität Basel

May 27, 2016

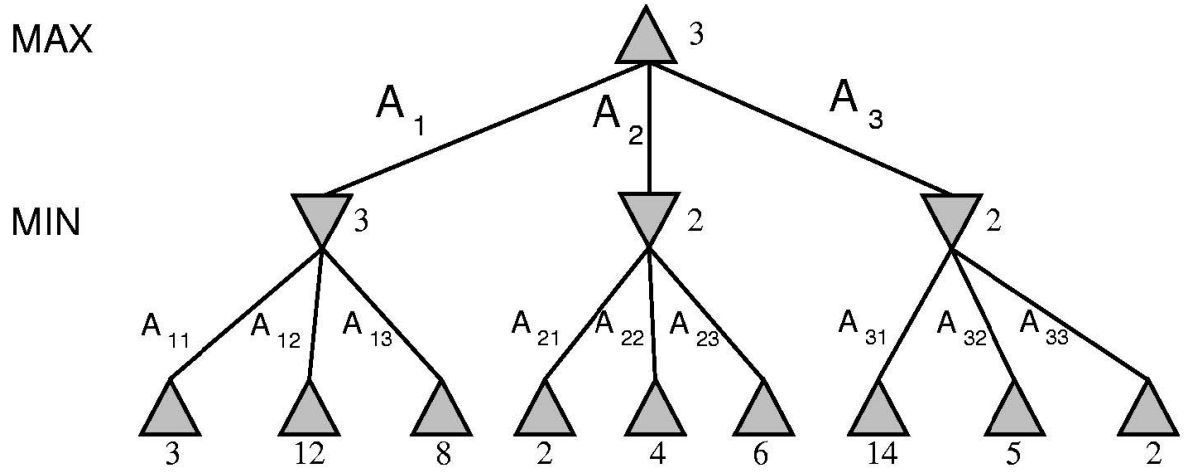
Board Games: Overview

chapter overview:

- 41. Introduction and State of the Art
- 42. Minimax Search and Evaluation Functions
- 43. Alpha-Beta Search
- 44. Monte-Carlo Tree Search: Introduction
- 45. Monte-Carlo Tree Search: Advanced Topics
- 46. AlphaGo and Outlook

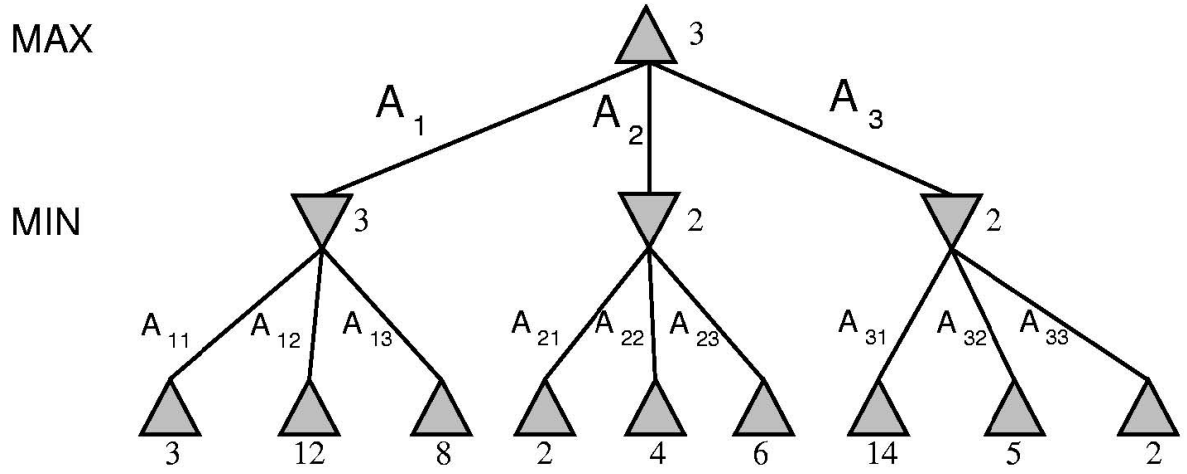
Alpha-Beta Search

Alpha-Beta Search



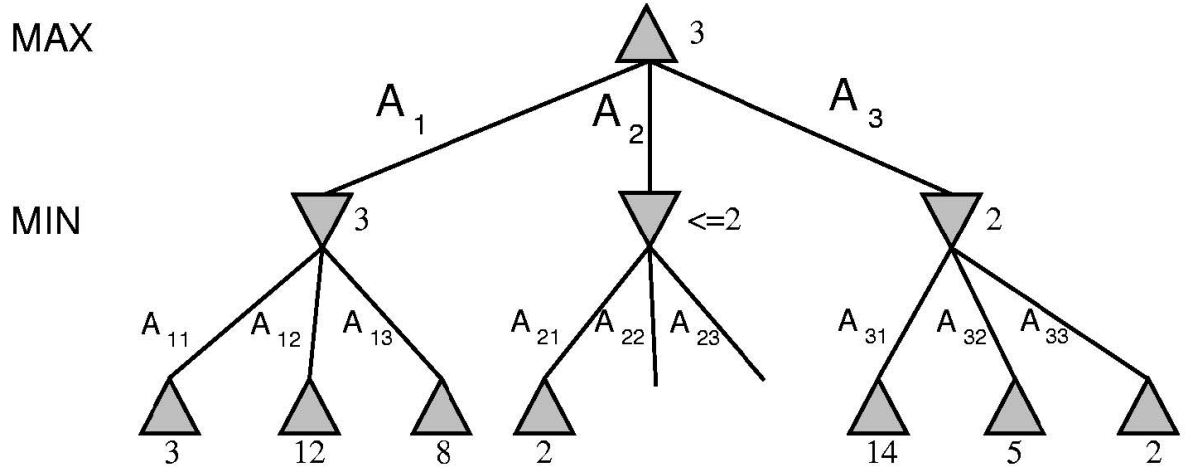
Can we save search effort?

Alpha-Beta Search

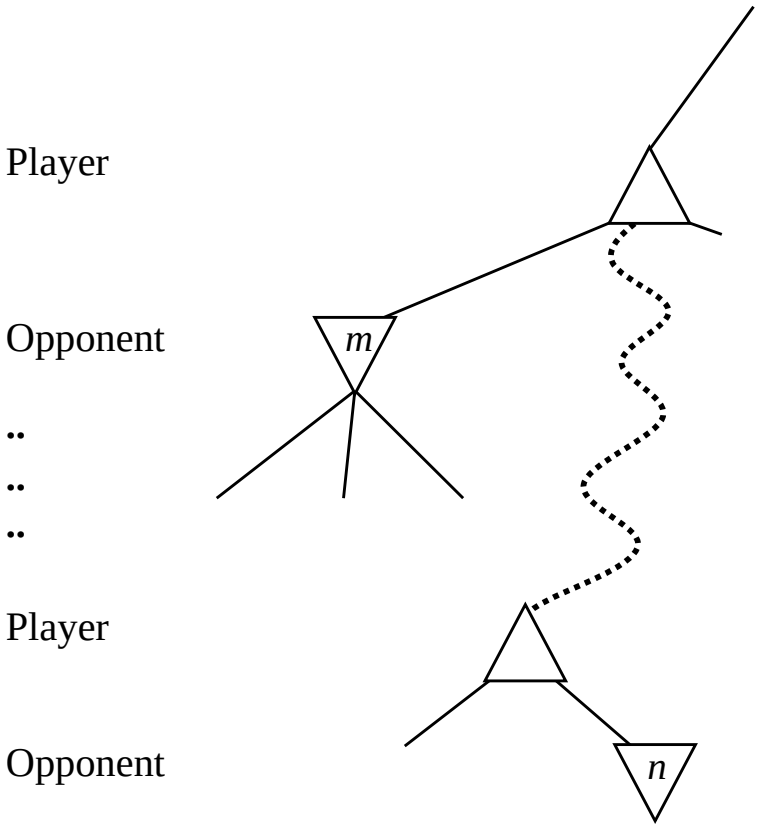


Can we save search effort?

We do not need to consider all the nodes!



Alpha-Beta Search: Generally



If $m > n$, then node with utility n will never be reached when playing perfectly!

Alpha-Beta Search: Idea

idea: Use two values α and β during minimax depth-first search, such that the following holds for every recursive call:

Alpha-Beta Search: Idea

idea: Use two values α and β during minimax depth-first search, such that the following holds for every recursive call:

- If the utility value in the current subtree is $\leq \alpha$,
  then the subtree is not interesting
  because MAX will never enter it when playing perfectly.
- If the utility value in the current subtree is $\geq \beta$,
  then the subtree is not interesting
  because MIN will never enter it when playing perfectly.

Alpha-Beta Search: Idea

idea: Use two values α and β during minimax depth-first search, such that the following holds for every recursive call:

- If the utility value in the current subtree is $\leq \alpha$,
  then the subtree is not interesting
  because MAX will never enter it when playing perfectly.
- If the utility value in the current subtree is $\geq \beta$,
  then the subtree is not interesting
  because MIN will never enter it when playing perfectly.

If $\alpha \geq \beta$ in the subtree, then the subtree is not interesting and does not have to be searched further (α-β pruning).

Alpha-Beta Search: Idea

idea: Use two values α and β during minimax depth-first search, such that the following holds for every recursive call:

- If the utility value in the current subtree is $\leq \alpha$,
  then the subtree is not interesting
  because MAX will never enter it when playing perfectly.
- If the utility value in the current subtree is $\geq \beta$,
  then the subtree is not interesting
  because MIN will never enter it when playing perfectly.

If $\alpha \geq \beta$ in the subtree, then the subtree is not interesting and does not have to be searched further (α-β pruning).

Starting with $\alpha = -\infty$ and $\beta = +\infty$, alpha-beta search produces the identical result as minimax, with lower seach effort.

Alpha-Beta Search: Pseudo Code

- algorithm skeleton the same as minimax
- function signature extended by two variables α and β

function alpha-beta-main(p)

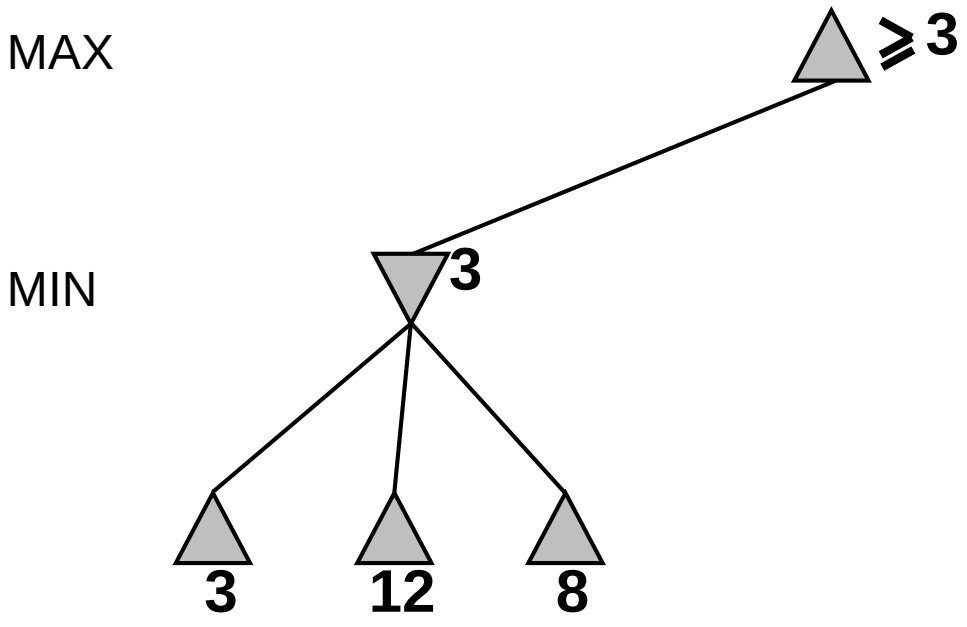
$\langle v, move \rangle := \text{alpha-beta}(p, -\infty, +\infty)$
return *move*

Alpha-Beta Search: Pseudo-Code

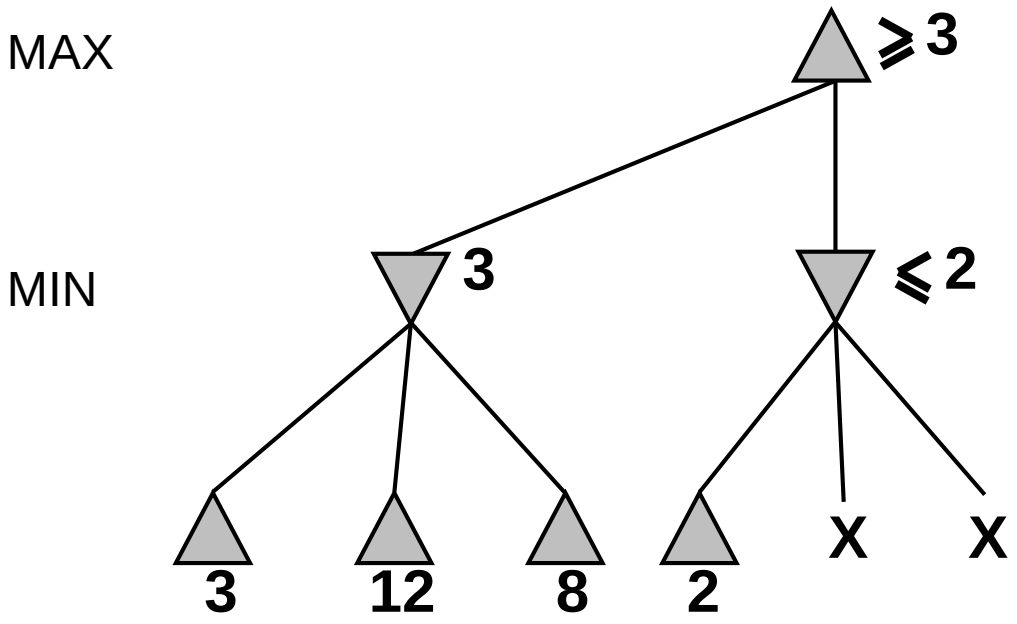
function alpha-beta(p, α, β)

if p is final position:
    return $\langle u(p), \textbf{none} \rangle$
initialize v and *best_move* [as in minimax]
for each $\langle move, p' \rangle \in \text{succ}(p)$:
    $\langle v', best\_move' \rangle := alpha\text{-}beta(p', \alpha, \beta)$
    update v and *best_move* [as in minimax]
    if $player(p) = \text{MAX}$:
        if $v \geq \beta$:
            return $\langle v, \textbf{none} \rangle$
        $\alpha := \max\{\alpha, v\}$
    if $player(p) = \text{MIN}$:
        if $v \leq \alpha$:
            return $\langle v, \textbf{none} \rangle$
        $\beta := \min\{\beta, v\}$
return $\langle v, best\_move \rangle$

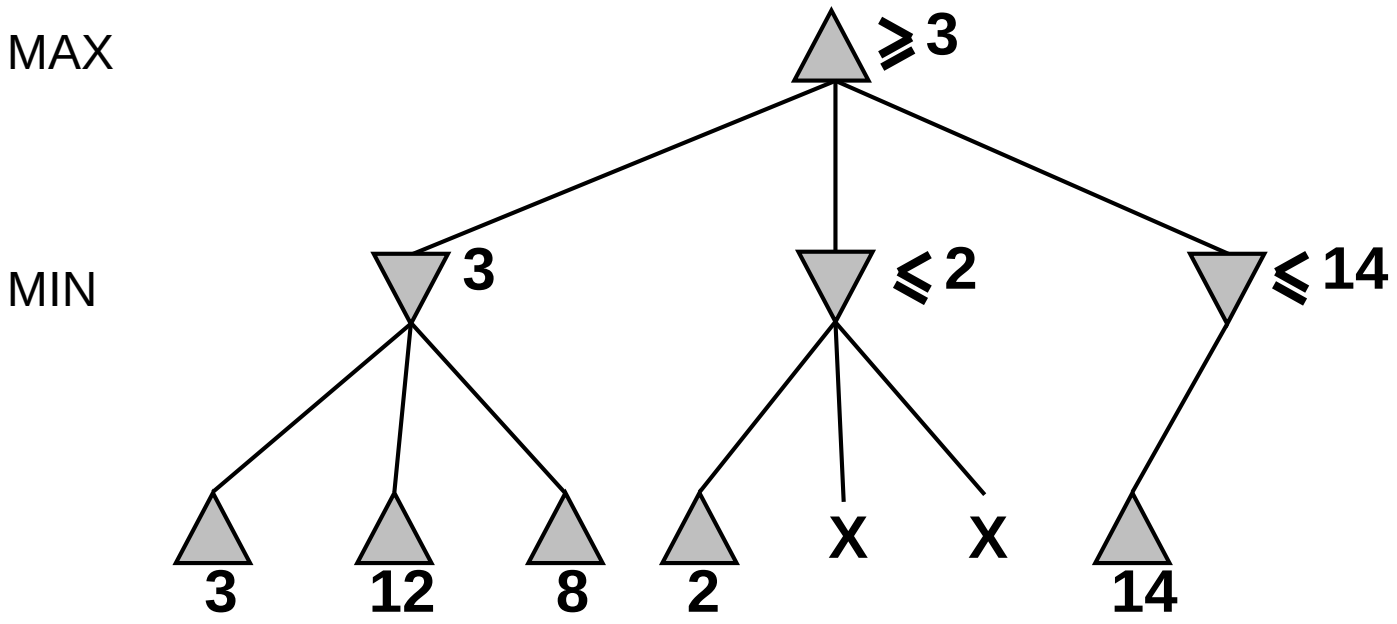
Alpha-Beta Search: Example



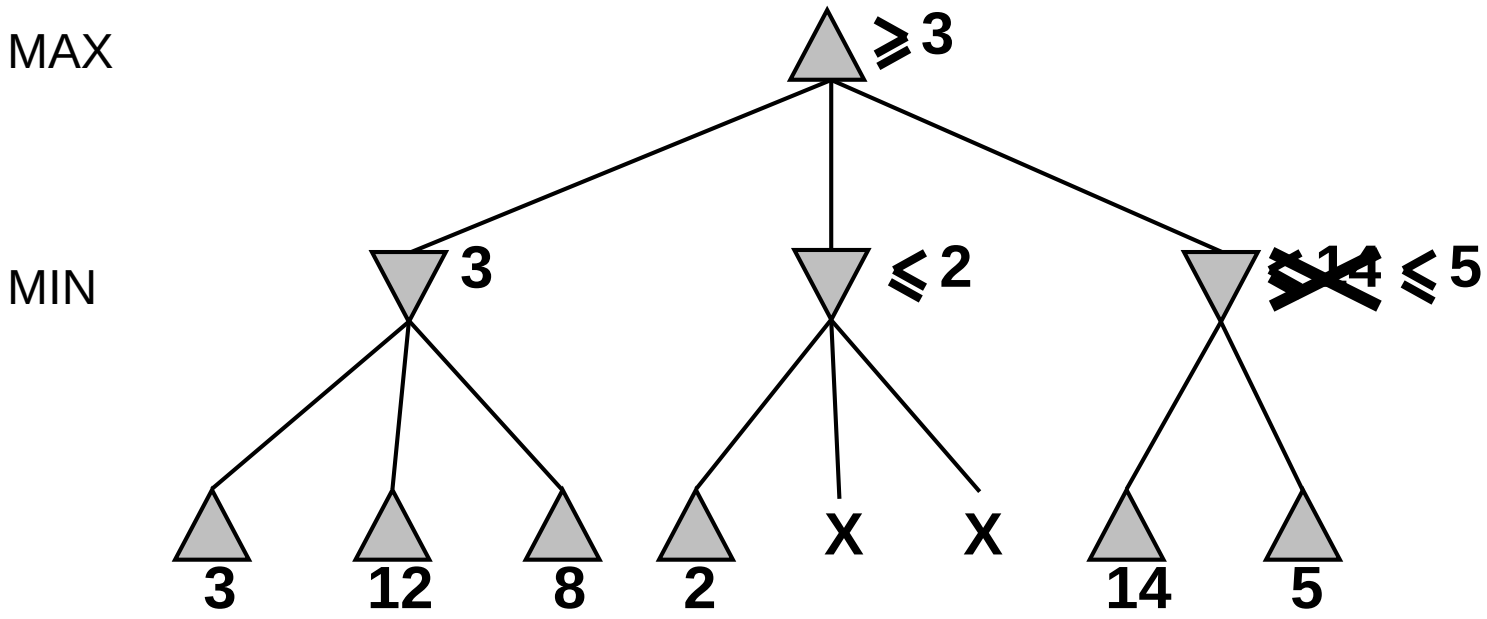
Alpha-Beta Search: Example



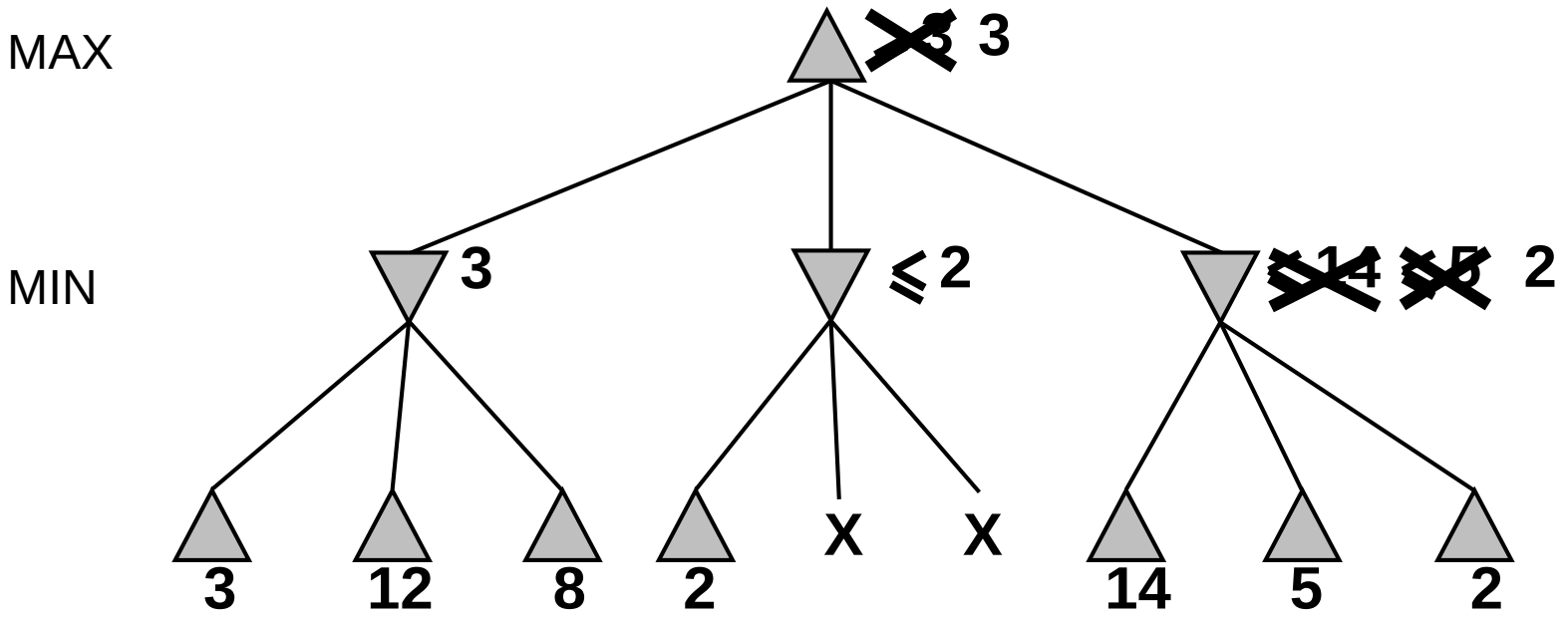
Alpha-Beta Search: Example



Alpha-Beta Search: Example



Alpha-Beta Search: Example



How Much Do We Get with Alpha-Beta Search?

assumption: uniform game tree, depth d, branching factor $b \geq 2$; MAX and MIN positions alternating

- alpha-beta search most successful if best move (for the player whose turn it is) is considered first
  - maximizing move for MAX, minimizing move for MIN

$\rightsquigarrow$ best case: effort reduced from $O(b^d)$ (minimax) to $O(b^{d/2})$

$\rightsquigarrow$ doubles the search depth that can be achieved in same time

- in practice, it is often possible to get close to the optimum

Summary

Summary

- alpha-beta search stores which utility both players can force somewhere else in game tree and avoids many unnecessary computations
- in best case, search effort $O(b^{d/2})$ for uniform trees

$\rightsquigarrow$ search depth can be doubled compared to minimax