Foundations of Artificial Intelligence

42. Board Games: Alpha-Beta Search

Malte Helmert

University of Basel

May 17, 2021

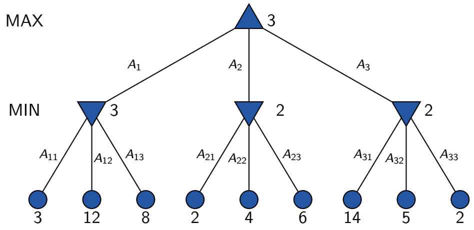
Board Games: Overview

chapter overview:

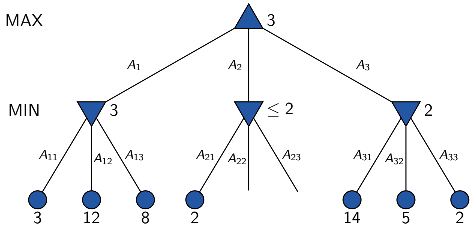
- 40. Introduction and State of the Art
- 41. Minimax Search and Evaluation Functions
- 42. Alpha-Beta Search
- 43. Monte-Carlo Tree Search: Introduction
- 44. Monte-Carlo Tree Search: Advanced Topics
- 45. AlphaGo and Outlook

42.1 Alpha-Beta Search

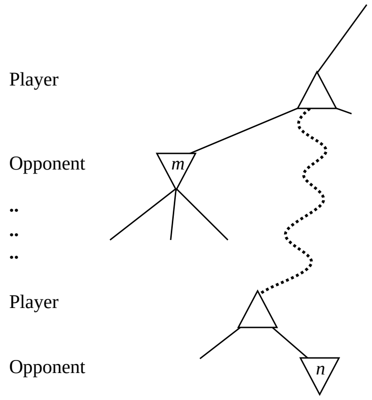
Alpha-Beta Search



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:

- 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

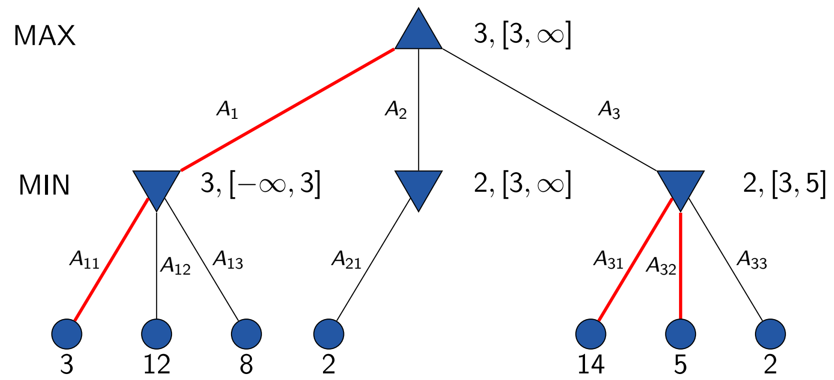
function alpha-beta-main(p)
$\langle v, move\rangle :=$ alpha-beta($p, -\infty, +\infty$)
return *move*

Alpha-Beta Search: Pseudo-Code

function alpha-beta(p, α, β)

```
if p is terminal position:
    return ⟨u(p), none⟩
initialize v and best_move                          [as in minimax]
for each ⟨move, p′⟩ ∈ succ(p):
    ⟨v′, best_move′⟩ := alpha-beta(p′, α, β)
    update v and best_move                          [as in minimax]
    if player(p) = MAX:
        if v ≥ β:
            return ⟨v, none⟩
        α := max{α, v}
    if player(p) = MIN:
        if v ≤ α:
            return ⟨v, none⟩
        β := min{β, v}
return ⟨v, best_move⟩
```

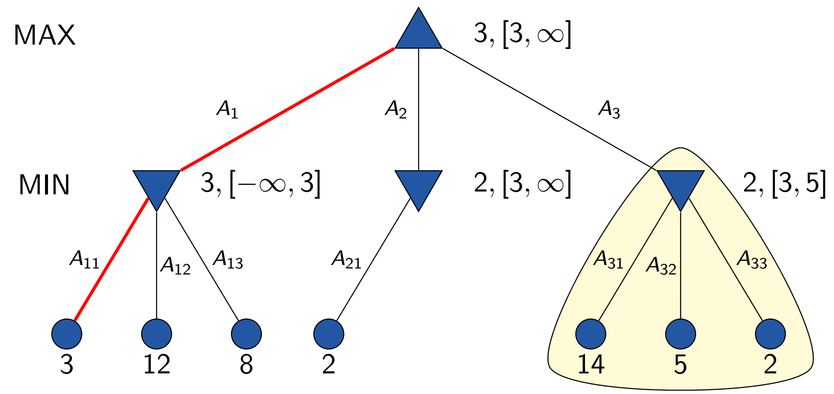

Alpha-Beta Search: Example



Cf. screen slides for detailed steps.

42.2 Move Ordering

Alpha-Beta Search: Example



If the last successor had been first, the rest of the subtree would have been pruned.

Move Ordering

idea: first consider the successors that are likely to be best

- Domain-specific ordering function
  e.g. chess: captures $<$ threats $<$ forward moves $<$ backward moves
- Dynamic move-ordering
  - first try moves that have been good in the past
  - e.g., in iterative deepening search:
    best moves from previous iteration

How Much Do We Gain with Alpha-Beta Search?

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

- perfect move ordering
  - best move at every position is considered first
    (this cannot be done in practice – Why?)
  - maximizing move for MAX, minimizing move for MIN
  - effort reduced from $O(b^d)$ (minimax) to $O(b^{d/2})$
  - doubles the search depth that can be achieved in same time
- random move ordering
  - effort still reduced to $O(b^{3d/4})$ (for moderate b)

In practice, it is often possible to get close to the optimum.

42.3 Summary

Summary

alpha-beta search

- stores which utility both players can force
  somewhere else in the game tree
- exploits this information to avoid unnecessary computations
- can have significantly lower search effort than minimax
- best case: search twice as deep in the same time