Lecture 12: Alpha-Beta and state evaluation

Artificial Intelligence
CS-GY-6613
Julian Togelius
julian.togelius@nyu.edu

Properties of Minimax

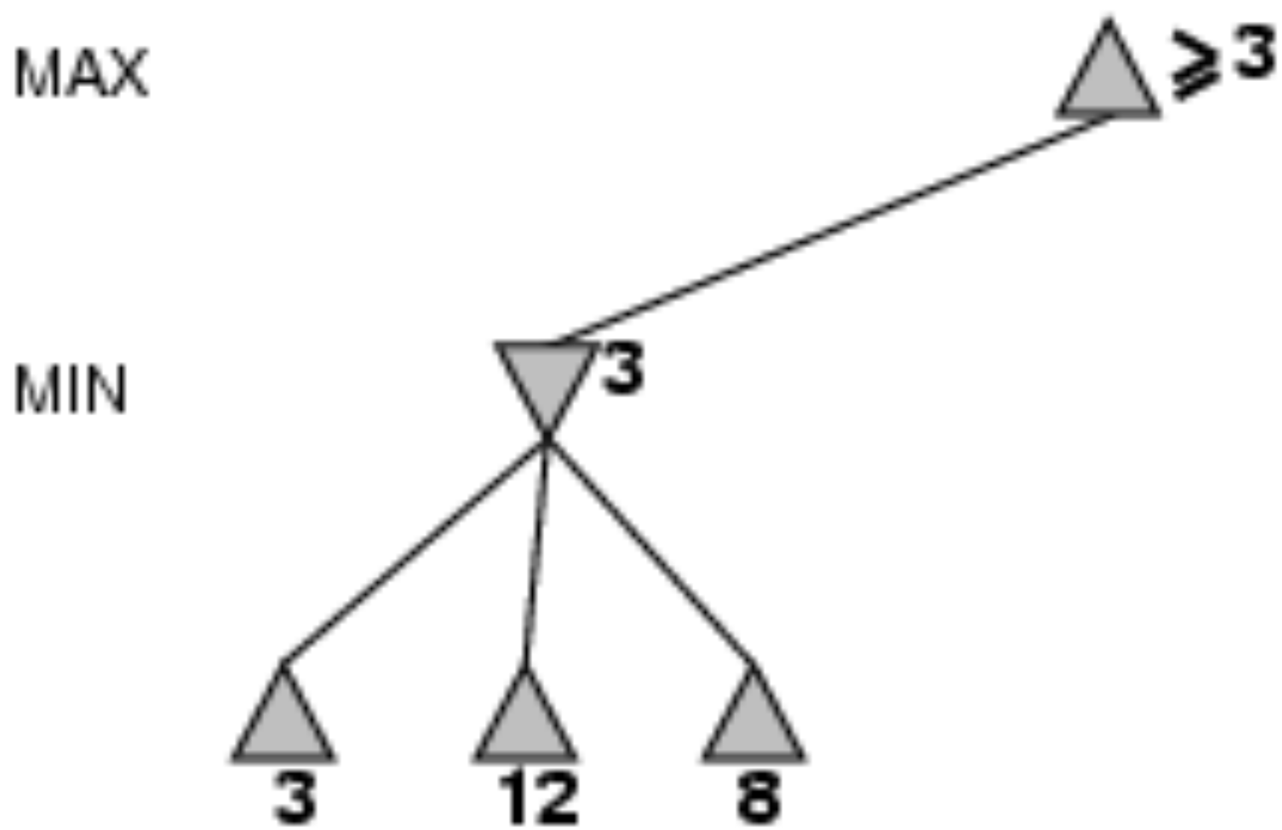
- *Complete?* Yes (if tree is finite)
- *Optimal?* Yes (against an optimal opponent)
- *Time complexity?* $O(b^m)$
- *Space complexity?* $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
–> exact solution completely infeasible

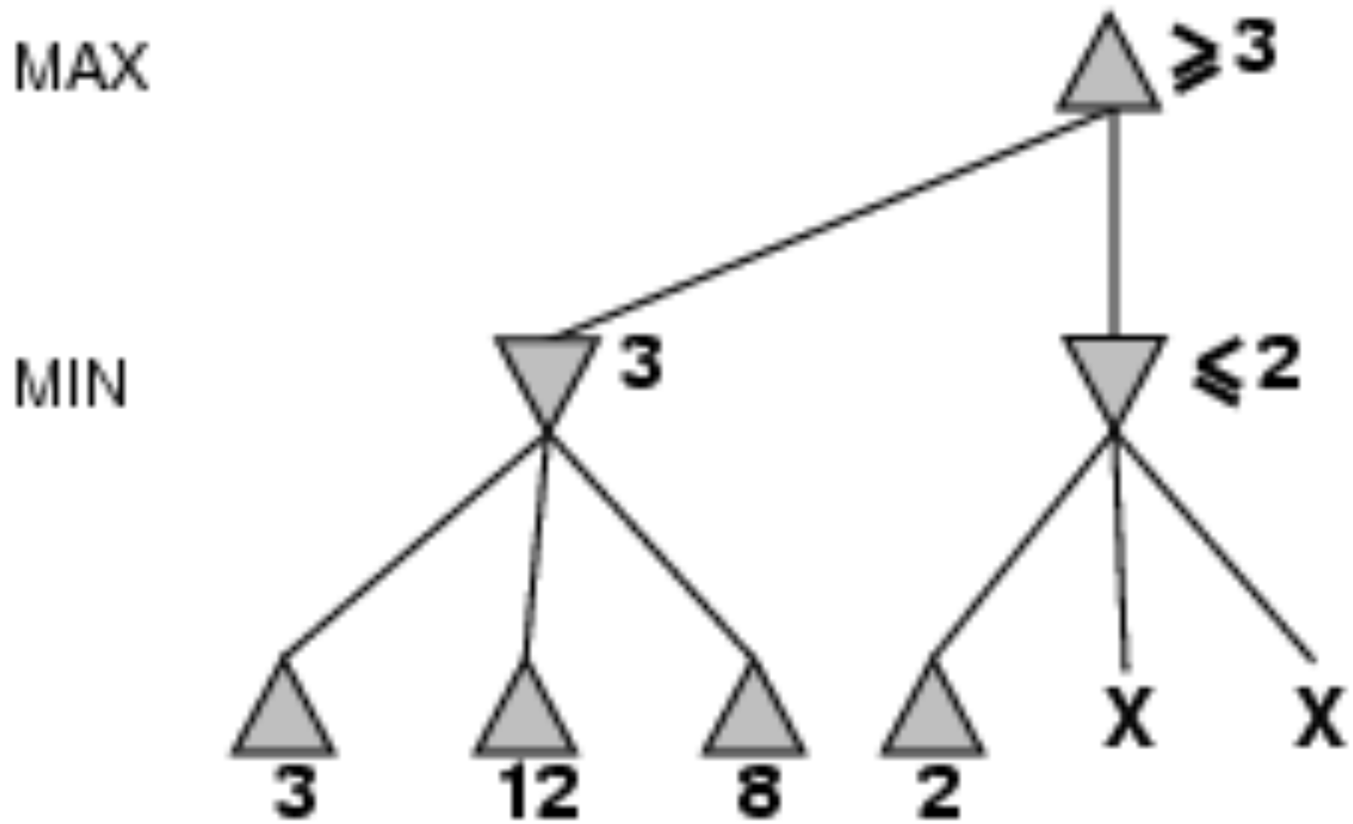
α-β pruning

- Can we improve this?
- Idea: don't consider branches of the tree that cannot lead to a better outcome than those that we have already explored

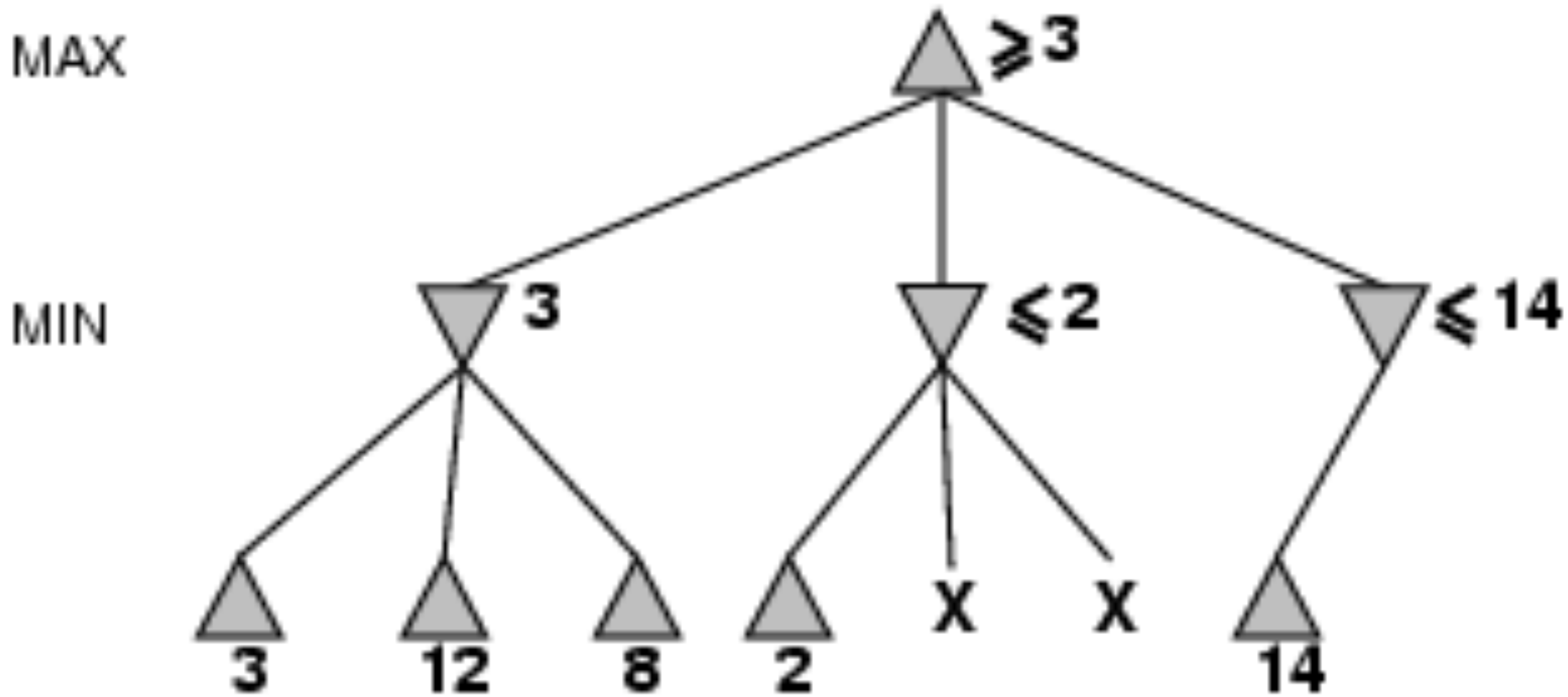
α-β pruning



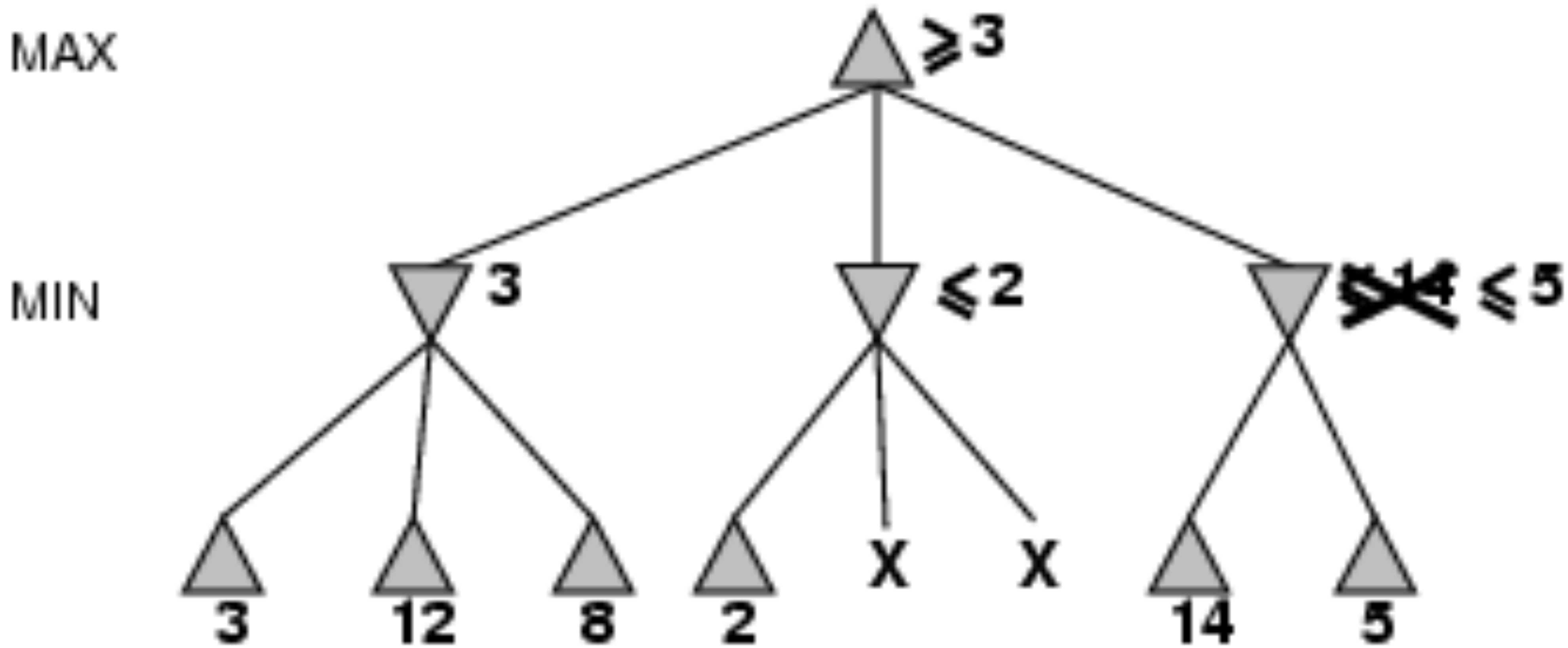
α-β pruning



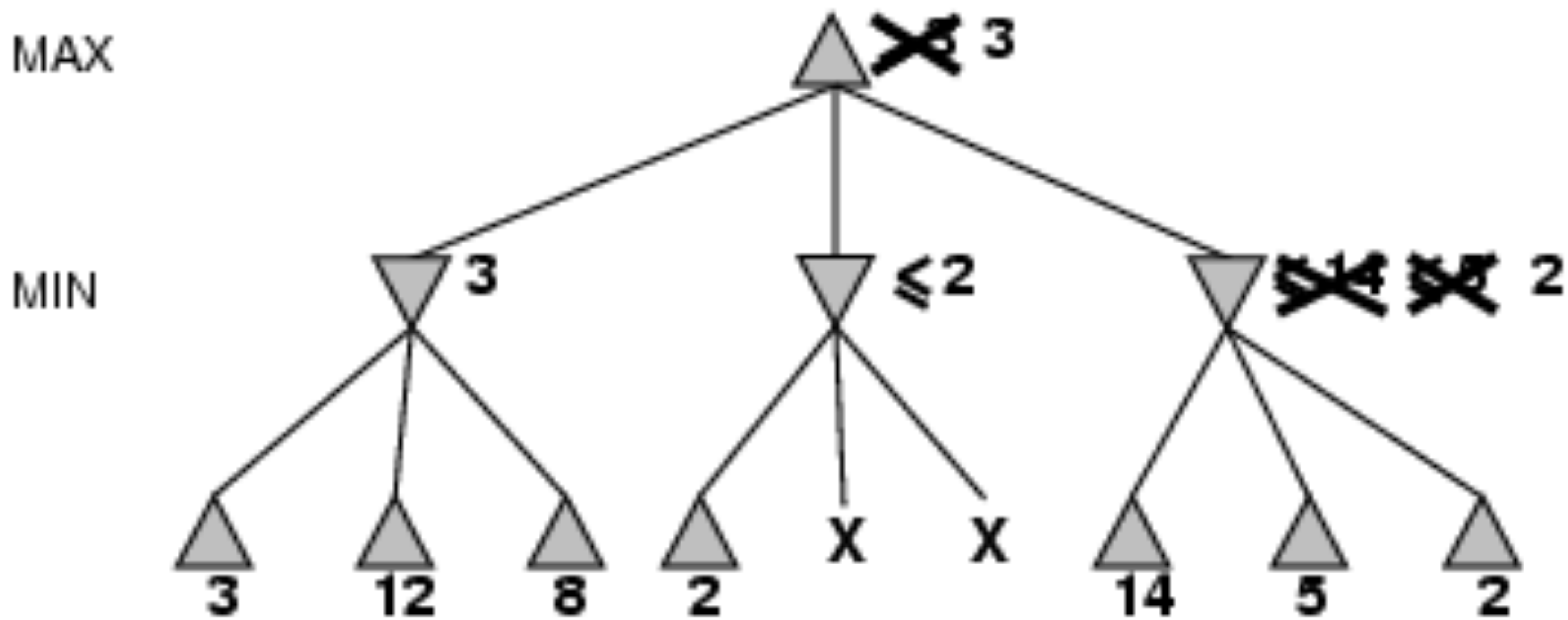
α-β pruning



α-β pruning



α-β pruning



Properties of α-β

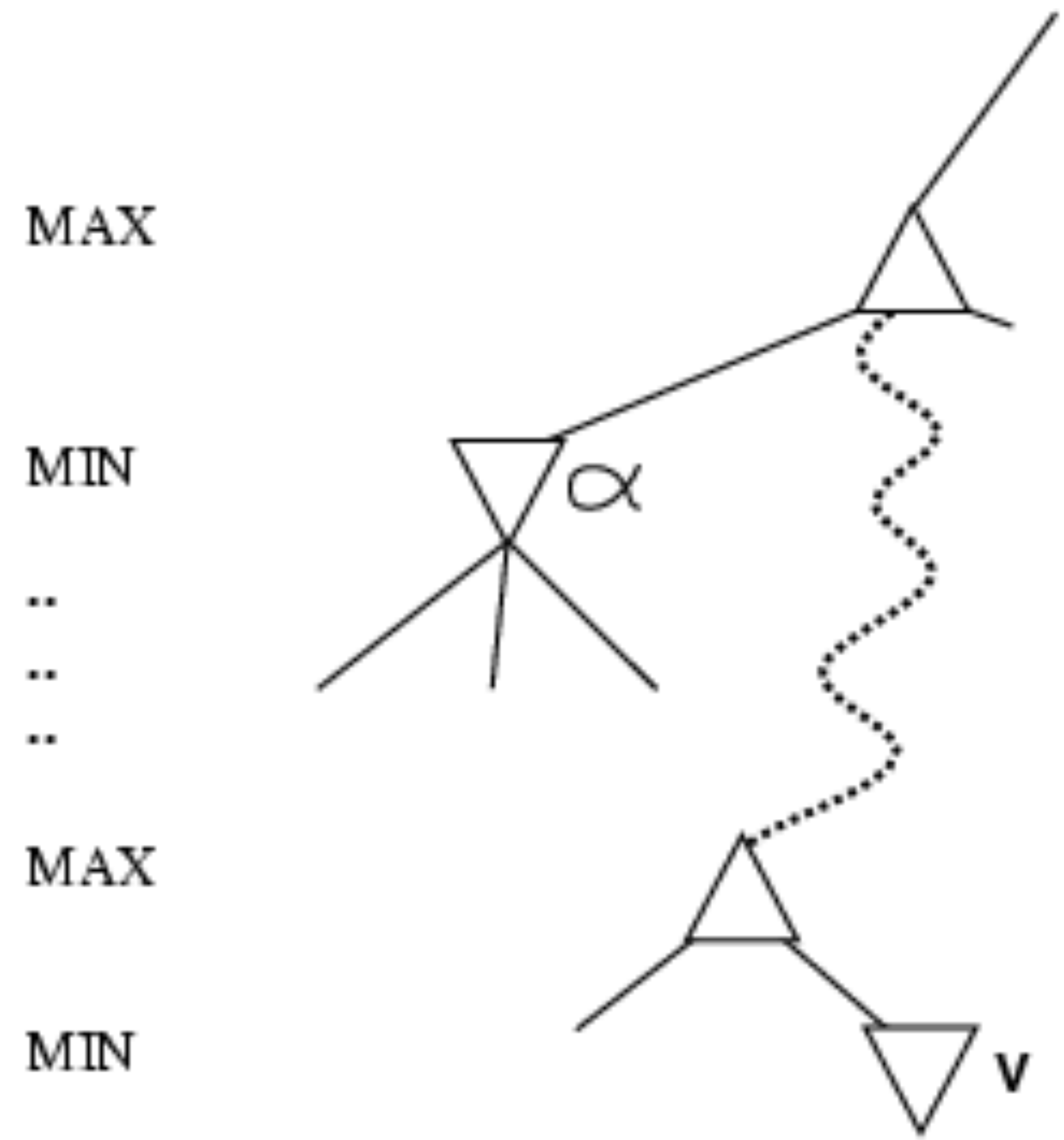
- Pruning does **not** affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering”, time complexity = $O(b^{m/2})$
  - doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (metareasoning)

Why is it called α-β?

- *α* is the value of the best (i.e. highest-value) choice found so far at any choice point along the path for *max*

    - If *v* is worse than *α*, *max* will avoid it
      > prune that branch

- Define *β* similarly for *min*



```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Still, there is no time…

- Suppose we have 100 secs, and can explore 10^4 nodes/sec
  –> 10^6 nodes per move
  (a far cry from 35^{100}…)

- Standard approach: use an *evaluation function* (heuristic)

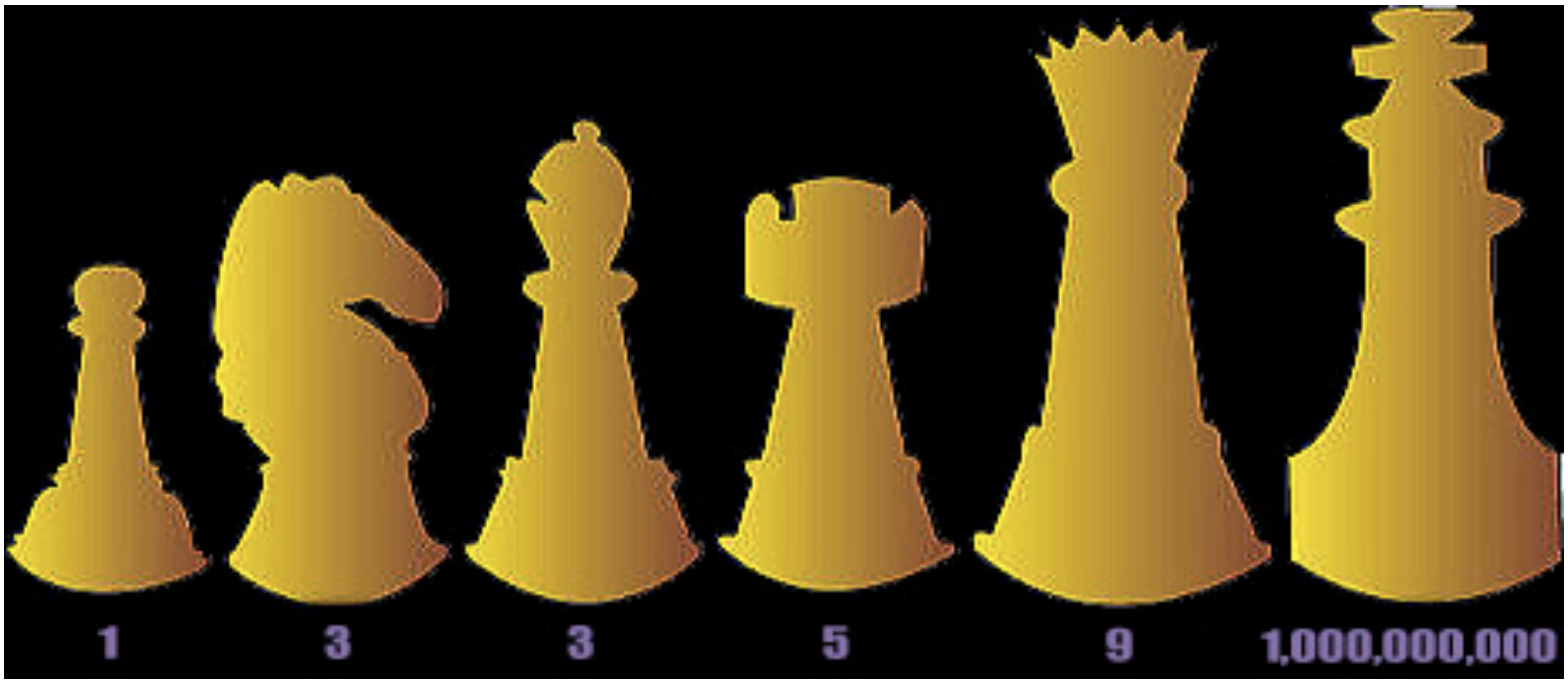
    - Cut off search and treat states as end states

- Either at the same depth for all branches, or use a *cutoff test* such as in quiescence search

Evaluation functions

- Simplest: number of white pieces - number of black pieces
- More complex: assign values to piece types
- Even more complex: count number of threats, try to recognize known positions
- Generally: Linear weighted sum of features
- Neural network
- Read more: *Blondie24* by David Fogel

Piece weights?



- MinimaxCutoff is identical to MinimaxValue except
  - Terminal? is replaced by Cutoff?
  - Utility is replaced by Eval
- Does it work in practice?
  - $b^m = 10^6$, b=35 means m=4
- 4-ply lookahead is (in general) a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ old-school Chess program, human master
  - 12-ply ≈ Deep Blue, Kasparov

Some deterministic two-player games

- Chess: Kasparov vs Deep Blue 1997
- Checkers: Chinook defeated Tinsley (grand champion) 1994
  - Solved 2007
- Othello: computers vastly better than humans
- Go: AlphaGo defeated Lee Sedol 2016


Google DeepMind
Challenge Match
8 - 15 March 2016
AlphaGo
ALPHAGO
01:59:54
LEE SEDOL
01:59:02
Google
AlphaGo
Lee Sedol

AlphaGo

- Go has enormous branching factor
- Very hard to come up with an accurate state evaluation function
- AlphaGo is a combination of:
  - Monte Carlo Tree Search (next lecture)
  - Supervised learning in neural networks (October)
  - Reinforcement learning through self-play (November)