Games 1:
Minimax, Evaluation function, Alpha-beta pruning

Hwanjo Yu

POSTECH

http://di.postech.ac.kr/hwanjoyu

Search problem

Markov decision processes

Reinforcement learning & games

Constraint satisfaction problems

Markov networks

Bayesian networks

Reflex **States** **Variables** **Logic**

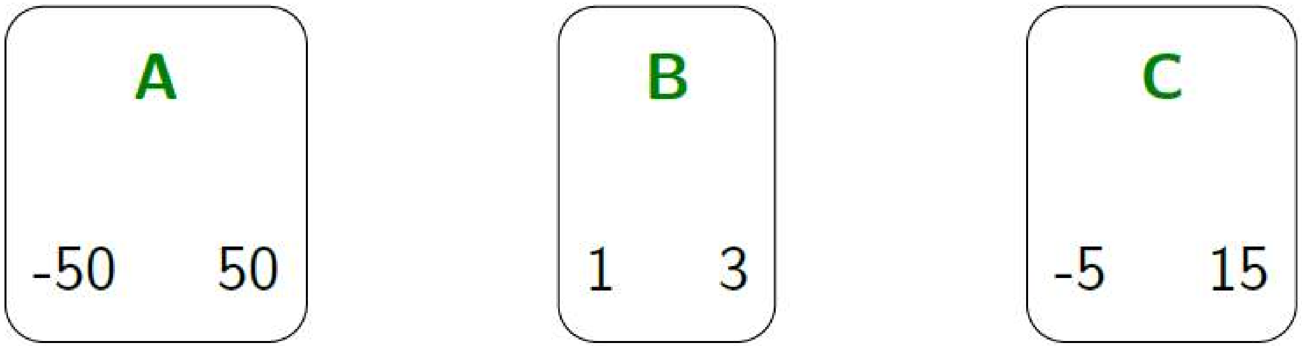
"Low-level intelligence" "High-level intelligence"

Machine learning

A simple game

Example: game 1

- You choose one of the three bins.
- I choose a number from that bin.
- Your goal is to maximize the chosen number.



- Your action depends on your mental model of me: me working with you, against you, or at random?

Roadmap

Games, minimax

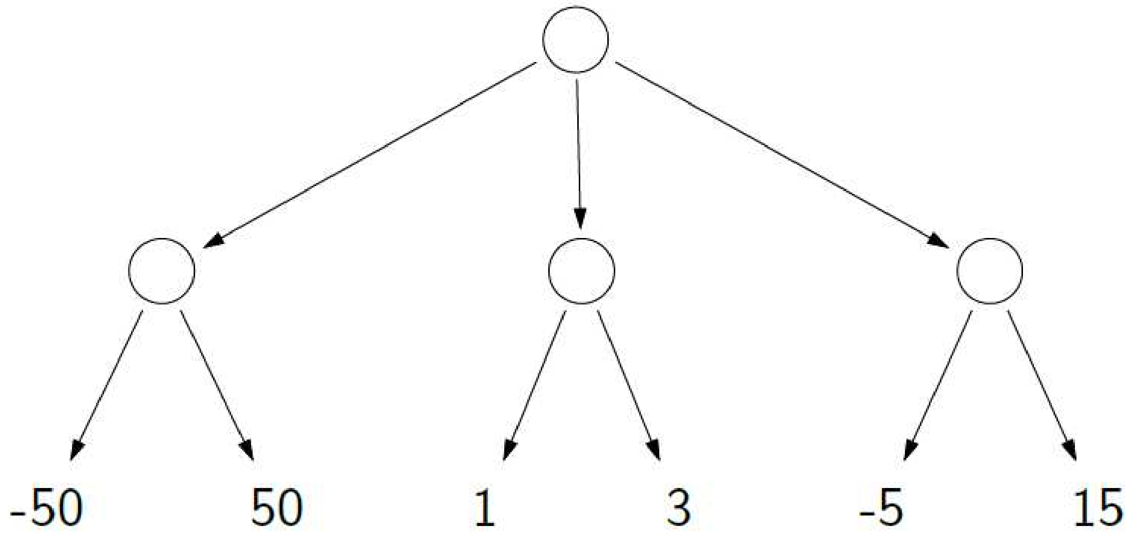
Evaluation functions

Alpha-beta pruning

Game tree

Key idea: **game tree**

- Each node is a decision point for a player.
- Each root-to-leaf path is a possible outcome of the game.



Two-player zero-sum games

Players = {**agent** (your program); **opp** (opponent)}

Definition: two-player zero-sum game (e.g., chess)

- $s_{\text{start}} \in \text{States}$: starting state
- $\text{Actions}(s)$: possible actions from state s
- $\text{Succ}(s, a)$: resulting state if choose action a in state s
- $\text{IsEnd}(s)$: end of game (game over)
- Utility(s) : agent's utility for end state s
- Player(s) ∈ Players: player who controls state s

- Transition()?
- Reward()?

Example: chess

- Players = {white, black}
- State s: (position of all pieces, whose turn it is)
- Actions(s): legal chess moves that Player(s) can make
- IsEnd(s): whether s is checkmate or draw
- Utility(s): $+\infty$ if white wins, 0 if draw, $-\infty$ if black wins

Characteristics of games

- All the utility is at the end state

- Different players in control at different states

Policies

Deterministic policies: $\pi_p(s) \in \mathrm{Actions}(s)$

- action that player p takes in state s

Stochastic policies: $\pi_p(s, a) \in [0,1]$

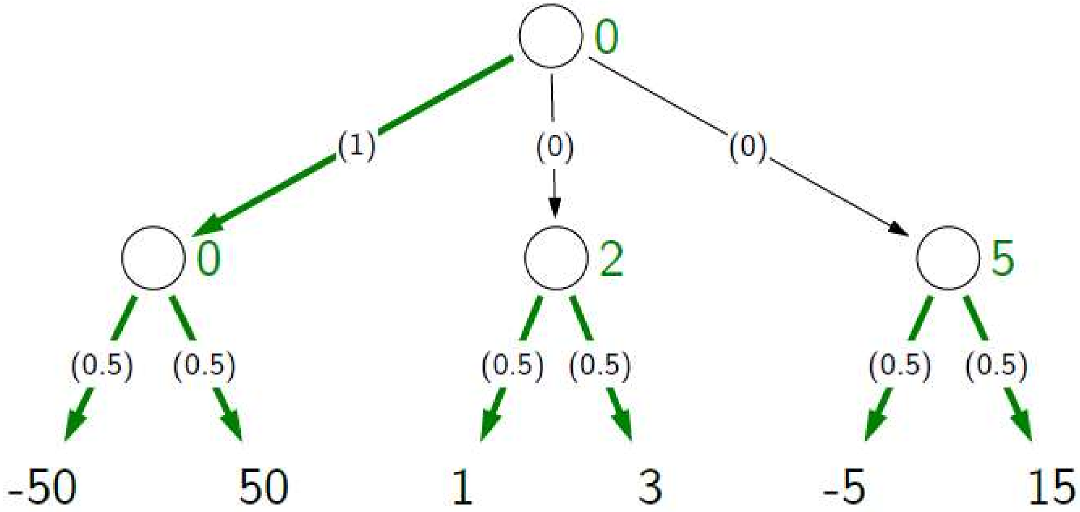
- probability of player p taking action a in state s

* We can think of MDP as a game between the agent and nature where the agent acts on state s according to π and the nature acts on the chance nodes according to $T(s, a, s')$.

Game evaluation example

Given two policies π_{agent} and π_{opp}, what is the (agent's) expected utility?

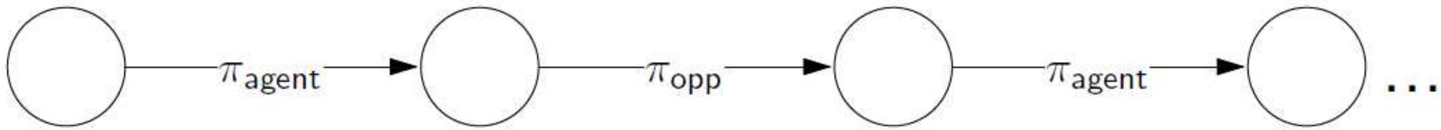
- $\pi_{\text{agent}}(s) = \mathrm{A}$
- $\pi_{\text{opp}}(s, a) = \frac{1}{2}$ for $a \in \mathrm{Actions}(s)$



$$V(s_{\text{start}}) = 0$$

Game evaluation recurrence

Analogy: recurrence for policy evaluation in MDPs



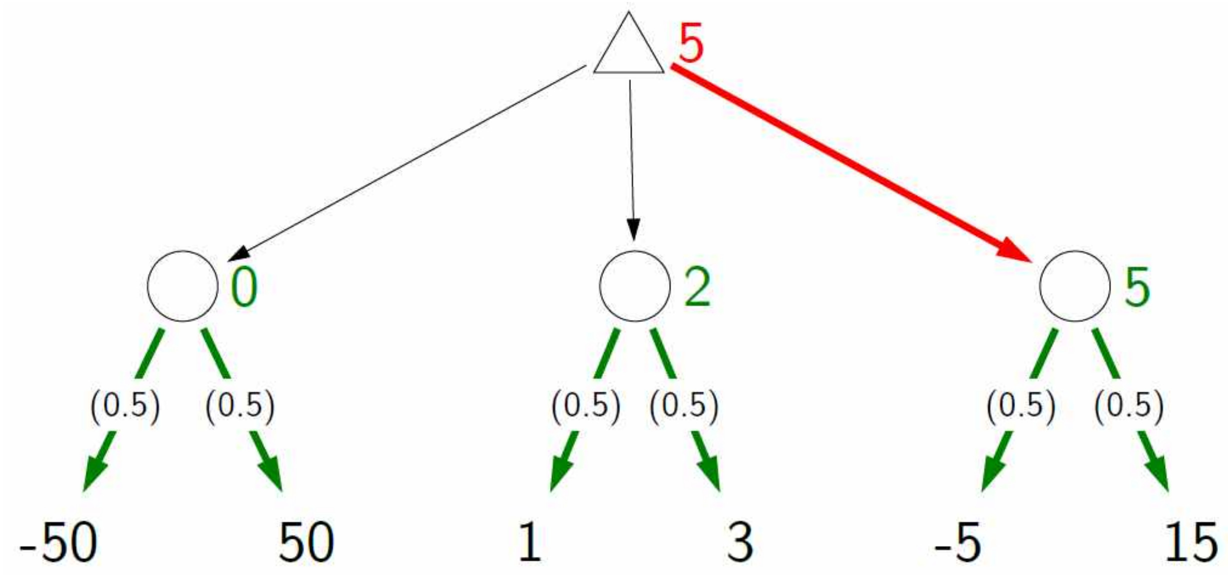
Value of the game:

$$V(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \displaystyle\sum_{a \in \text{Actions}(s)} \pi_{\text{agent}}(s,a) V(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \displaystyle\sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Expectimax example

Example: expectimax

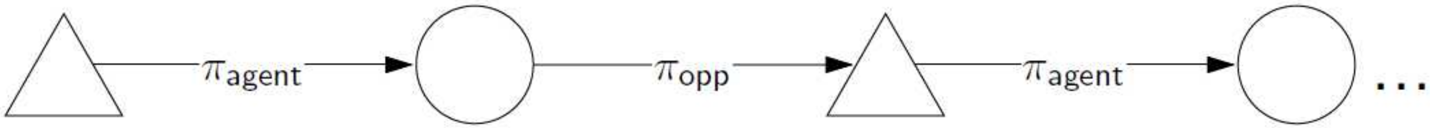
- $\pi_{\text{opp}}(s, a) = \frac{1}{2}$ for $a \in \text{Actions}(s)$



$$\boxed{V_{\max,\text{opp}}(s_{\text{start}}) = 5}$$

Expectimax recurrence

Analogy: recurrence for value iteration in MDPs



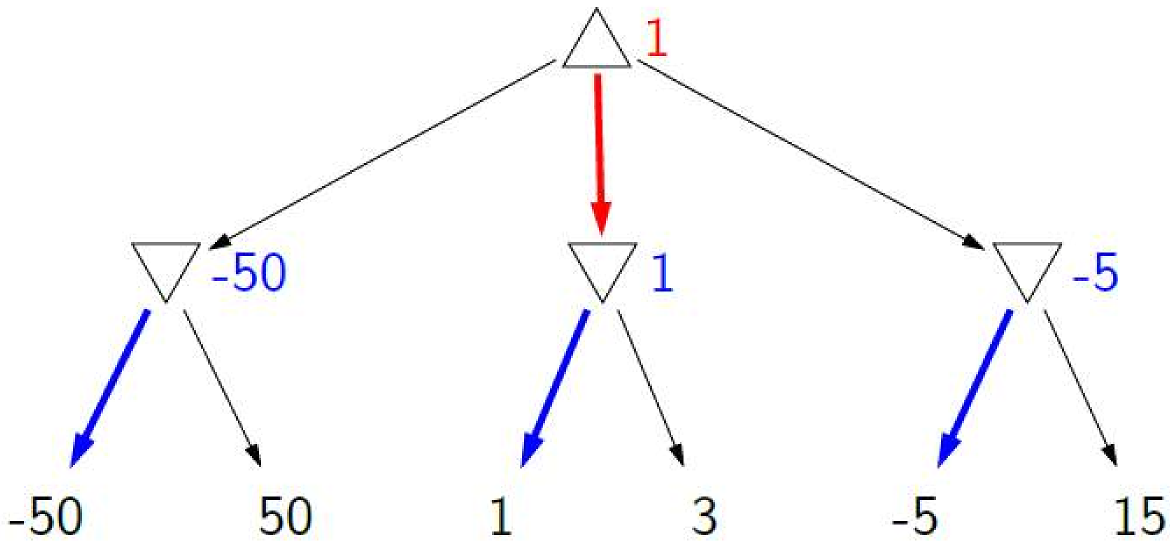
$$V(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Problem: don't know opponent's policy

Approach: assume the worst case

Minimax example

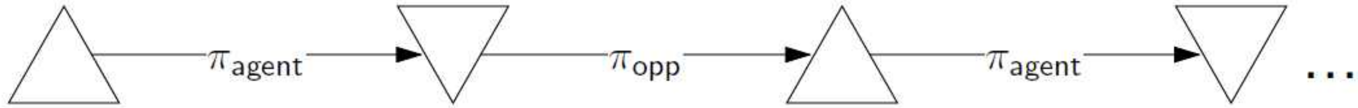
Example: minimax



$$\boxed{V(s_{\text{start}}) = 1}$$

Minimax recurrence

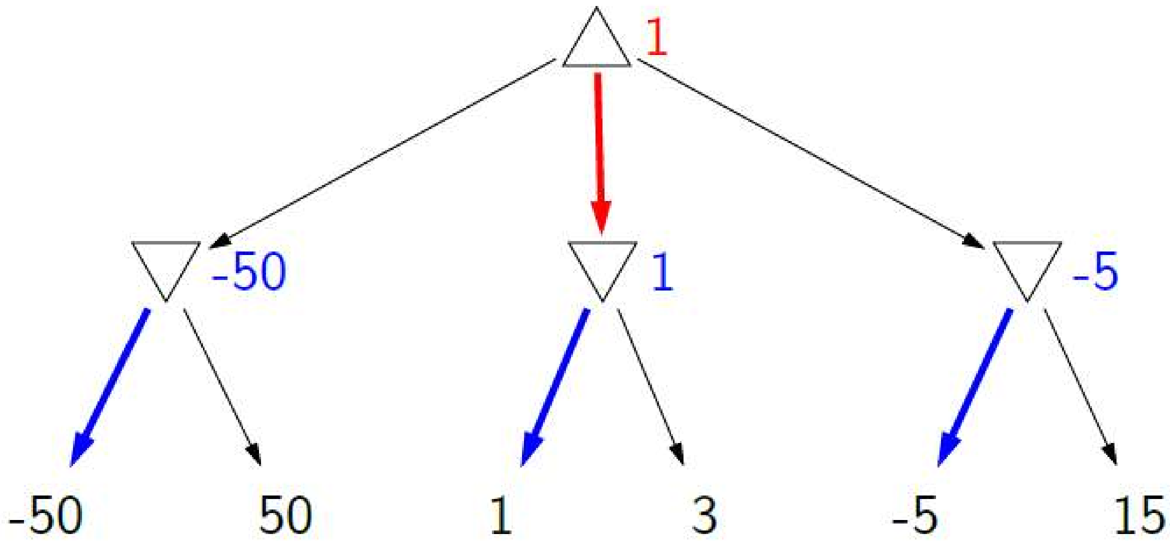
No analogy in MDPs



$$V(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V(\text{Succ}(s, a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V(\text{Succ}(s, a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Extracting minimax policies

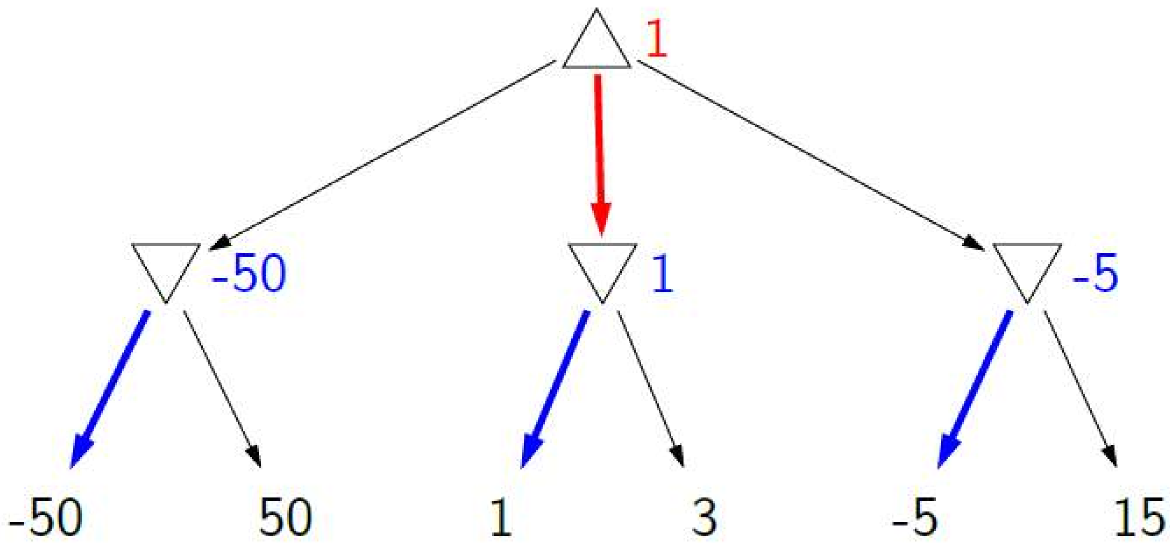
- $\pi_{\max}(s) = \arg\max_{a \in \mathrm{Actions}(s)} V(\mathrm{Succ}(s,a))$
- $\pi_{\min}(s) = \arg\min_{a \in \mathrm{Actions}(s)} V(\mathrm{Succ}(s,a))$



Minimax property 1

Proposition: best against minimax opponent

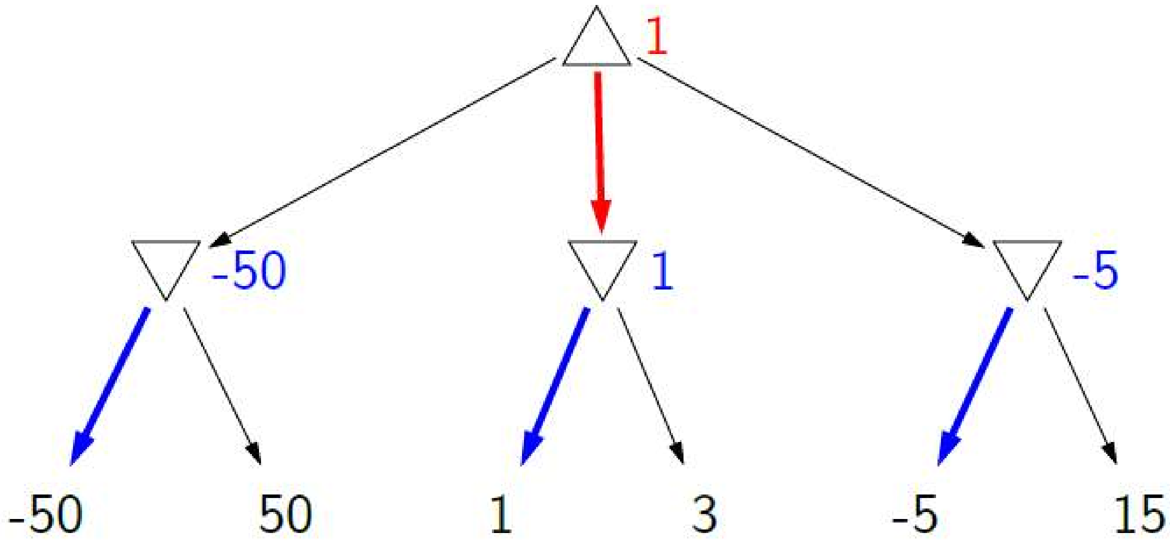
- $V_{\max,\min}(s_{\text{start}}) \geq V_{\text{agent},\min}(s_{\text{start}})$ for all π_{agent}
  - $V_{\max,\min}(s) = V(s)$ when $\pi_{\text{agent}} = \pi_{\max}$ and $\pi_{\text{opp}} = \pi_{\min}$
  - $V_{\text{agent},\min}(s) = V(s)$ when $\pi_{\text{agent}} \neq \pi_{\max}$ and $\pi_{\text{opp}} = \pi_{\min}$



Minimax property 2

Proposition: lower bound against any opponent

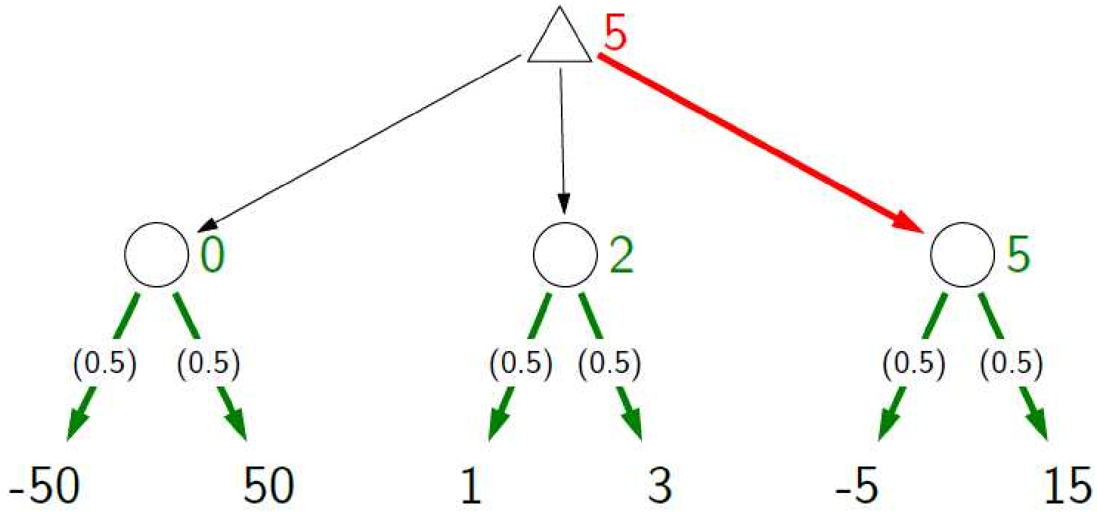
- $V_{\max,\min}(s_{\text{start}}) \leq V_{\max,\text{opp}}(s_{\text{start}})$ for all π_{opp}
  - $V_{\max,\min}(s) = V(s)$ when $\pi_{\text{agent}} = \pi_{\max}$ and $\pi_{\text{opp}} = \pi_{\min}$
  - $V_{\max,\text{opp}}(s) = V(s)$ when $\pi_{\text{agent}} = \pi_{\max}$ and $\pi_{\text{opp}} \neq \pi_{\min}$



Minimax non-property 3

Proposition: not optimal against all opponents

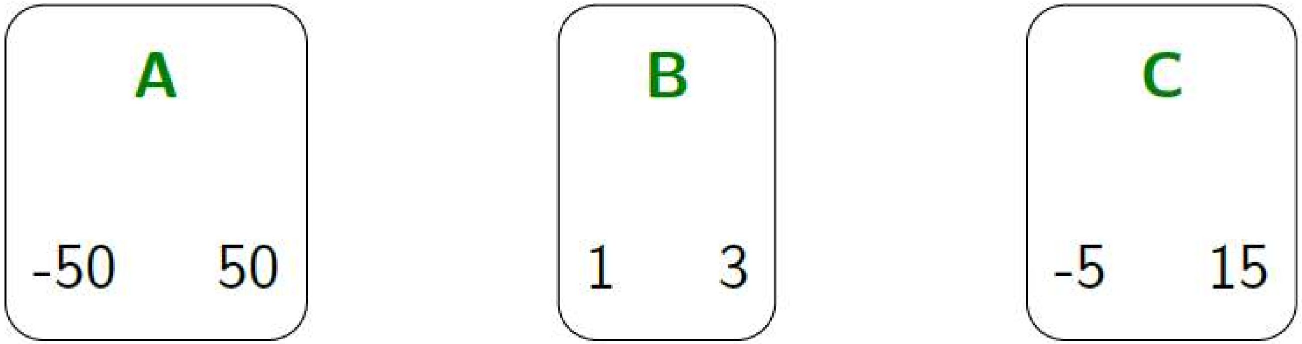
- $V_{\max,\text{opp}}(s_{\text{start}}) \ngeq V_{\text{agent},\text{opp}}(s_{\text{start}})$ for all π_{agent}
  - $V_{\max,\text{opp}}(s) = V(s)$ when $\pi_{\text{agent}} = \pi_{\max}$ and $\pi_{\text{opp}} \neq \pi_{\min}$
  - $V_{\text{agent},\text{opp}}(s) = V(s)$ when $\pi_{\text{agent}} \neq \pi_{\max}$ and $\pi_{\text{opp}} \neq \pi_{\min}$



A modified game

Example: game 2

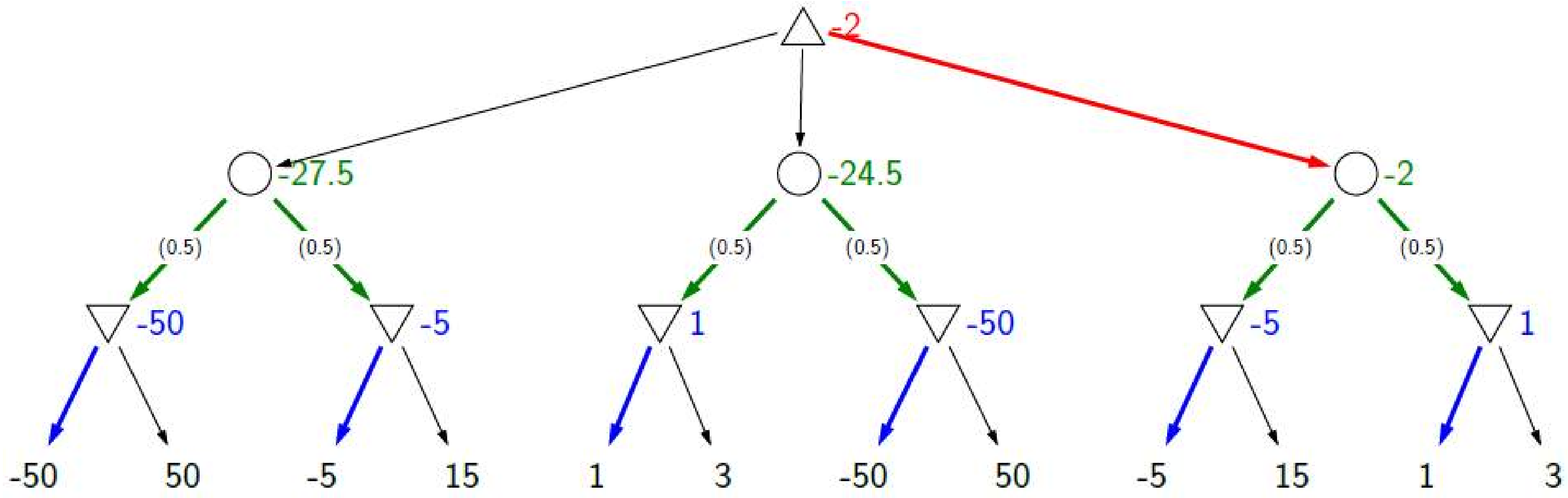
- You choose one of the three bins.
- Flip a coin; if heads, then move one bin to the right (with wrap around).
- I choose a number from that bin.
- Your goal is to maximize the chosen number.



Expectiminimax example

Example: expectiminimax

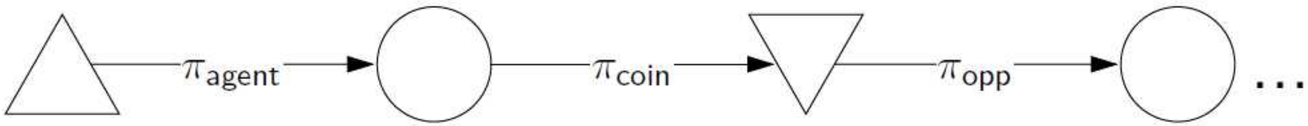
- $\pi_{\text{coin}}(s,a) = \frac{1}{2}$ for $a \in \text{Actions}(s)$



$$V(s_{\text{start}}) = -2$$

Expectiminimax recurrence

- Players = {agent, opp, coin}



$$V(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{coin}}(s,a) V(\text{Succ}(s,a)) & \text{Player}(s) = \text{coin} \end{cases}$$

Summary so far

Primitives: max nodes, chance nodes, min nodes

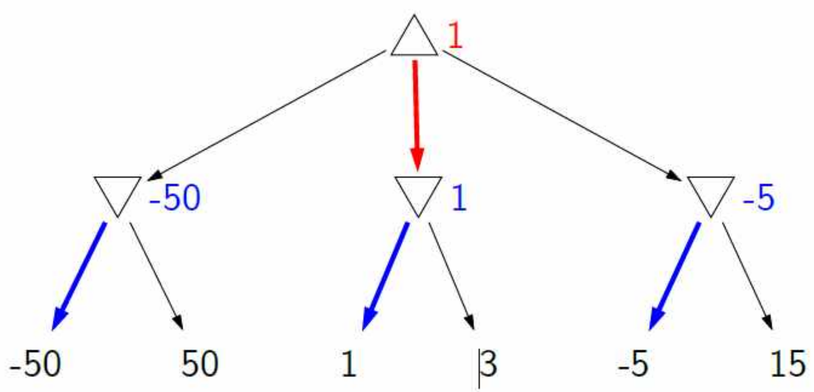
Composition: alternate nodes according to model of game

Value function $V(s)$: recurrence for expected utility

Scenarios to think about:

- What if you are playing against multiple opponents?
- What if you and your partner have to take turns (table tennis)?
- Some actions allow you to take an extra turn?

Computation



Approach: tree search

Complexity:

- Branching factor b, depth d ($2d$ plies)
- $O(d)$ space, $O(b^{2d})$ time

Chess: $b \cong 35, d \cong 50$

25515520672986852924121150151425587630190414488161019324176778440771467258239937365843732987043555789782336195637736653285543297897675074636936187744140625

Speeding up minimax

- Evaluation functions: use domain-specific knowledge, compute approximate answer
- Alpha-beta pruning: general-purpose, compute exact answer

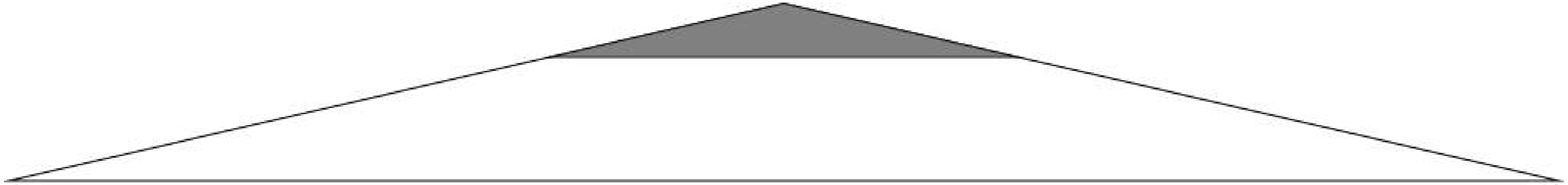
Roadmap

Games, minimax

Evaluation functions

Alpha-beta pruning

Depth-limited search



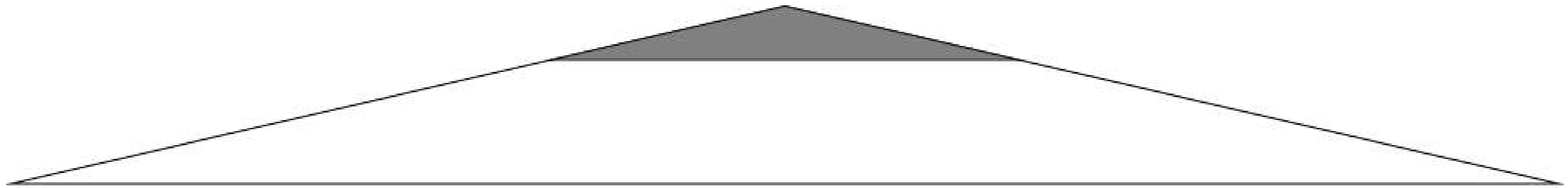
Limited depth tree search (stop at maximum depth d_{max}):

$$V(s,d)=\begin{cases}\mathrm{Utility}(s) & \mathrm{IsEnd}(s)\\ \mathrm{Eval}(s) & d=0\\ \max_{a\in\mathrm{Actions}(s)} V(\mathrm{Succ}(s,a),d) & \mathrm{Player}(s)=\mathrm{agent}\\ \min_{a\in\mathrm{Actions}(s)} V(\mathrm{Succ}(s,a),d-1) & \mathrm{Player}(s)=\mathrm{opp}\end{cases}$$

Use: at state s, call $V(s,d_{max})$

Convention: decrement depth at last player's turn

Evaluation functions



Definition: Evaluation function

- An evaluation function Eval(s) is a (possibly very weak) estimate of the value $V(s)$.

Analogy: FutureCost(s) in search problems

Evaluation functions

Example: chess

- Eval(s) = material + mobility + king-safety + center-control
- material = $10^{100}(K - K') + 9(Q - Q') + 5(R - R') + 3(B - B' + N - N') + 1(P - P')$
  - K and K': the number of kings that the agent and the opponent have
- mobility = 0.1(num-legal-moves - num-legal-moves')
- …

Function approximation

Key idea: parameterized evaluation functions

- $\mathrm{Eval}(s; \mathbf{w})$ depends on weights $\mathbf{w} \in \mathbb{R}^d$

Example: Linear evaluation function

$$\mathrm{Eval}(s) = \mathbf{w} \cdot \phi(s)$$

where $\phi(s) = \mathbb{R}^d$, e.g.,

$$\phi_1(s) = K - K'$$
$$\phi_2(s) = Q - Q'$$
$$\ldots$$

How to learn $\boldsymbol{w}$ *?*

Approximating the true value function

- If knew optimal policies $\pi_{\max}$ and $\pi_{\min}$, game tree evaluation provides best evaluation function:

$$\text{Eval}(s) = V(s)$$

Intractable!

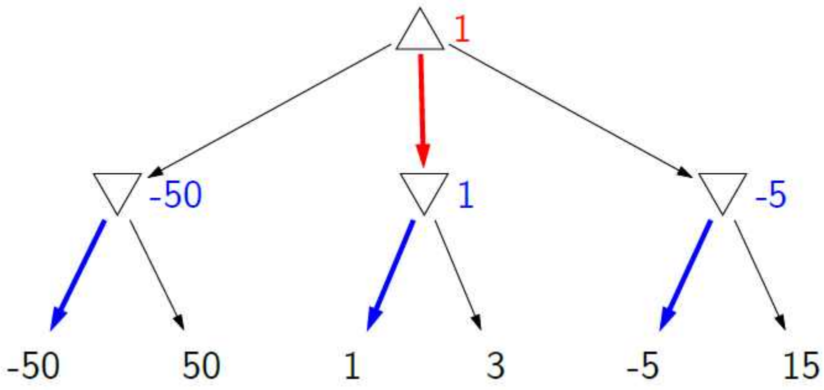
Two approximations:

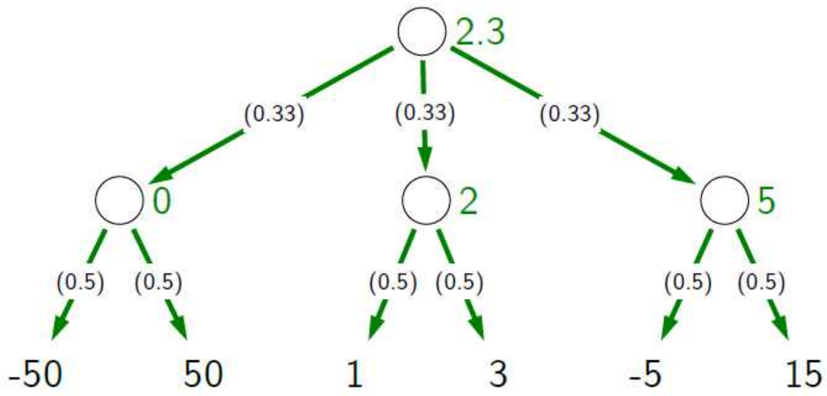
- Replace optimal policies with heuristic (stochastic) policies
- Use Monte Carlo approximation

Approximation 1: stochastic policies

Replace $\pi_{\max}, \pi_{\min}$ with stochastic $\pi_{\text{agent}}, \pi_{\text{opp}}$:

Example: game 1





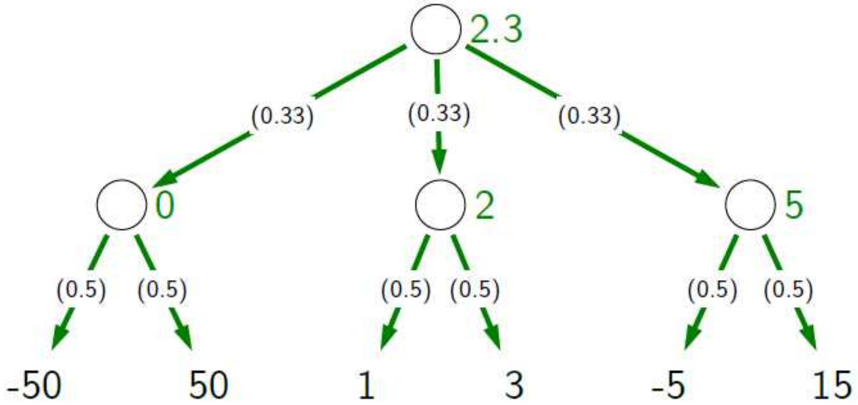
$\text{Eval}(s) = V(s)$ is still hard to compute...

Approximation 2: Monte Carlo

Approach:

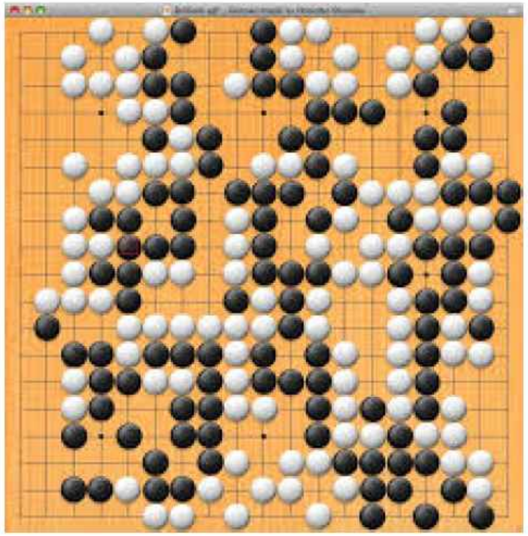
- Simulate n random paths by applying the policies
- Average the utilities of the n paths

Example: game 1



$$\text{Eval}(s) = V(s) = \frac{1}{10}[(1) + (3) + (50) + (50) + (50) + (-50) + (-50) + (50) + (15) + (-5)] = 11.4$$

Monte Carlo Go



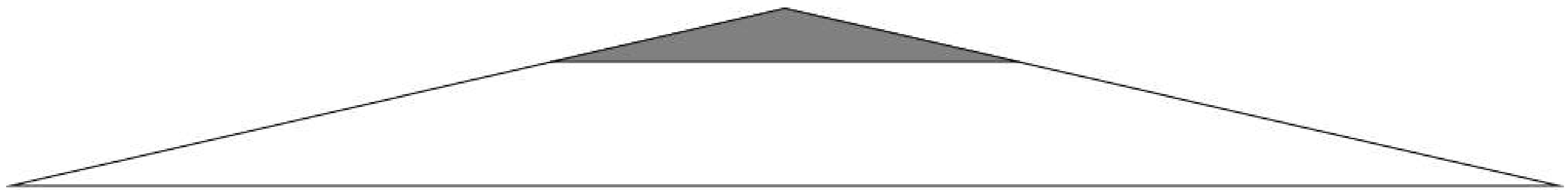
- Go has branching factor of 361, depth of 361
- Example heuristic policy: if stone is threatened, try to save it; otherwise move randomly
- Monte Carlo is responsible for recent successes

Google's AlphaGo (March 2016)

- Monte Carlo Tree Search: for exploring the game tree
- Policy network (CNN): used as the stochastic policy to guide the search
- Value network (CNN): used as the evaluation function.

Summary: evaluation functions

Depth-limited exhaustive search: $O(b^{2d})$ time



Rely on evaluation function:

- Function approximation: parameterize by $\mathbf{w}$ and features
- Monte Carlo approximation: play many games heuristically (randomize)

Roadmap

Games, minimax

Evaluation functions

Alpha-beta pruning

Pruning principle

Choose A or B with maximum value:

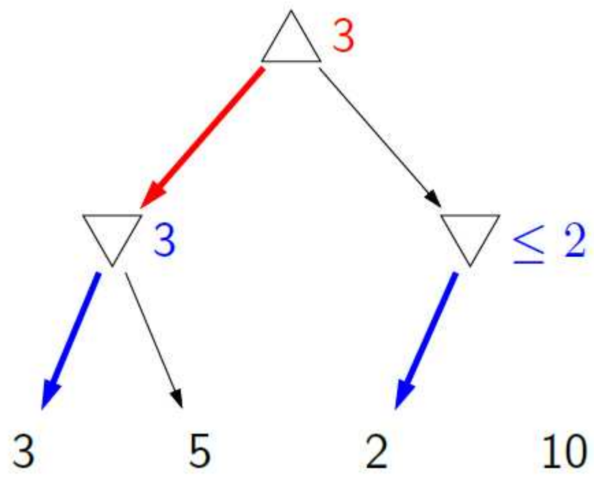
A: [3, **5**] B: [**5**, 100]

Key idea: branch and bound (refer to BnB in Wiki)

- Maintain lower and upper bounds on values.
- If intervals don't overlap non-trivially, then can choose optimally without further work.

Alpha-beta pruning is a specialization of BnB for minimax tree search.

Pruning game trees



Once see 2, we know that value of right node must be ≤ 2

Root computes $\max(3, \leq 2) = 3$

Since branch doesn't affect root value, can safely prune

Alpha-beta pruning

Key idea: optimal path

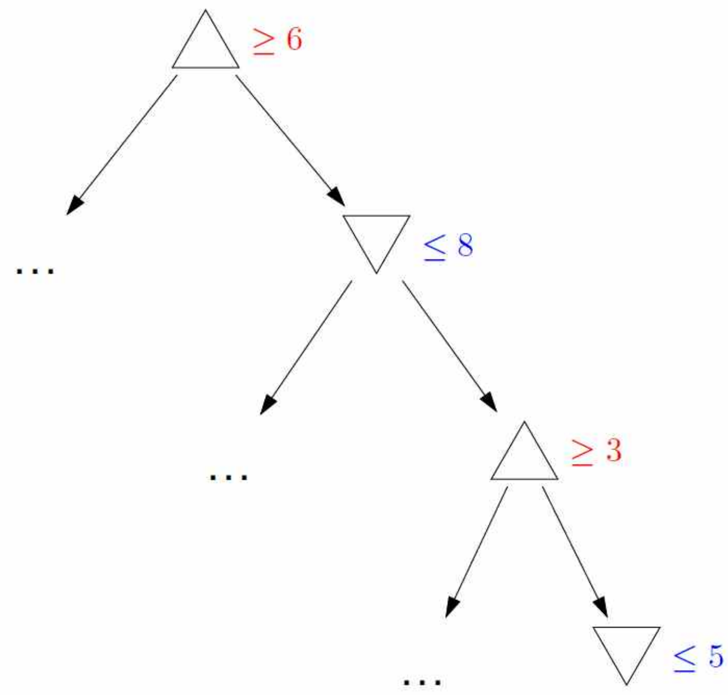
- The optimal path is path that minimax policies take.
- Values of all nodes on the optimal path are the same.

While doing DFS, maintaining a_s or b_s

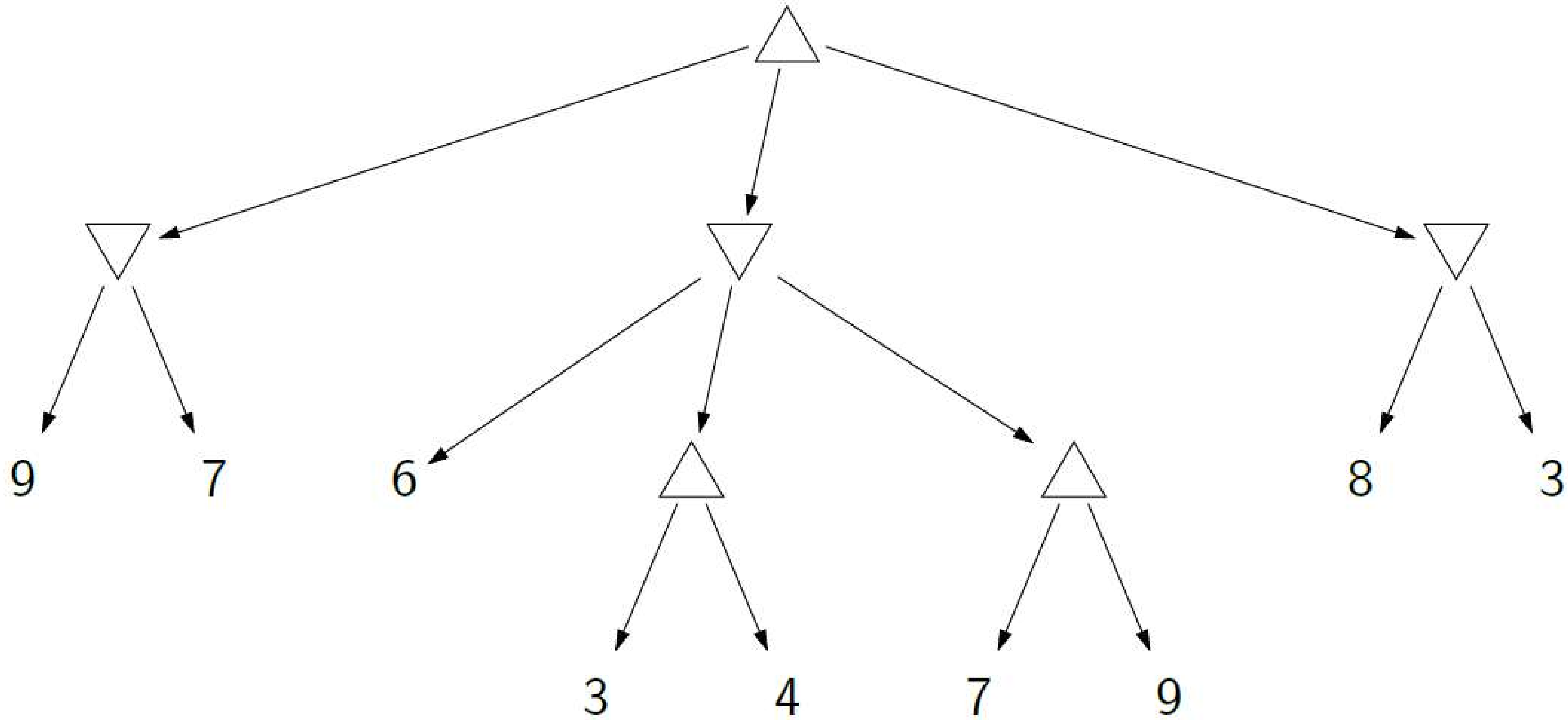
- a_s: lower bound on value of max node s
- b_s: upper bound on value of min node s

Prune a node if its interval doesn't have non-trivial overlap with <u>every ancestor</u>

- Implementation note: for each max or min node, store $\alpha_s = \max_{s' \preccurlyeq s} a_{s'}$, and $\beta_s = \min_{s' \preccurlyeq s} b_{s'}$, ($s'$ is every ancestor.)



Alpha-beta pruning example

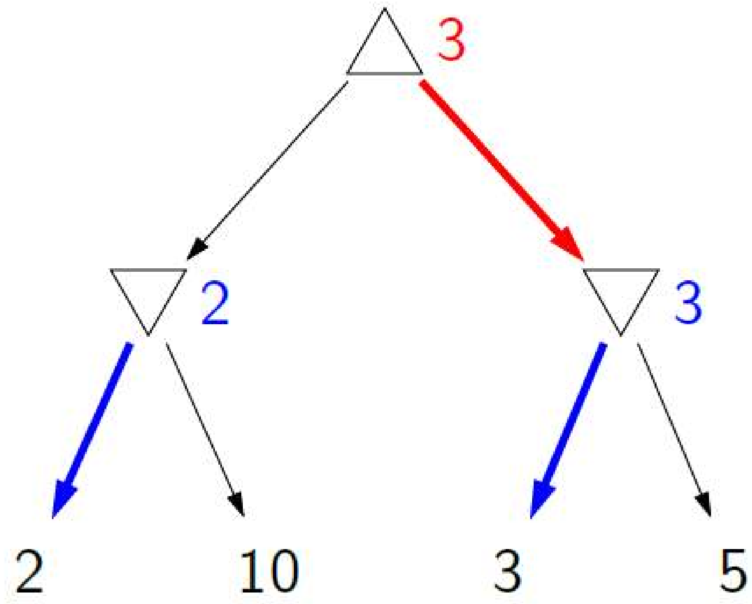


```
function alphabeta(node, depth, α, β, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        v := -∞
        for each child of node
            v := max(v, alphabeta(child, depth – 1, α, β, FALSE))
            α := max(α, v)
            if β ≤ α (* if lowerbound is larger than upperbound *)
                break
        return v
    else
        v := ∞
        for each child of node
            v := min(v, alphabeta(child, depth – 1, α, β, TRUE))
            β := min(β, v)
            if β ≤ α (* if lowerbound is larger than upperbound *)
                break
        return v
…
alphabeta(origin, depth, -∞, +∞, TRUE)
```

Move ordering

Pruning depends on order of actions.

Can't prune the 5 node:



Move ordering

Which ordering to choose?

- Worst ordering: $O(b^{2 \cdot d})$ time (= $O(b * b * b * b \dots)$)
- Best ordering: $O(b^{2 \cdot 0.5d})$ time (= $O(b * 1 * b * 1 \dots)$)
- Random ordering: $O(b^{2 \cdot 0.75d})$ time

In practice, can use evaluation function $\text{Eval}(s)$:

- Max nodes: order successors by decreasing $\text{Eval}(s)$
- Min nodes: order successors by increasing $\text{Eval}(s)$
- But need time for computing $\text{Eval}(s)$ and sorting nodes according to $\text{Eval}(s)$