

Intelligent Systems Programming

Lecture 3: Adversarial Search

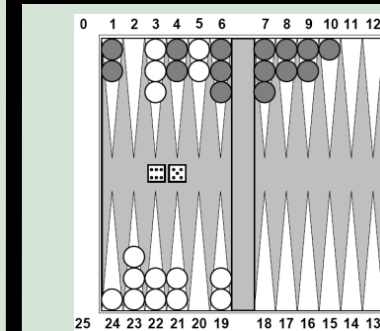
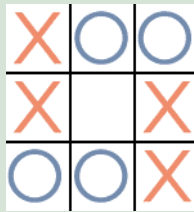
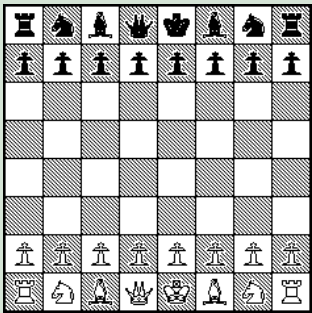
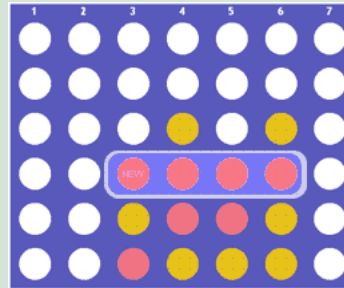
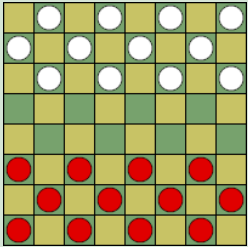


Why Learn About Game Algorithms?

- Gamification
- The algorithms are relevant control of non-deterministic systems like nuclear power plants
- Game playing is (was) considered intellectual behavior
- The algorithms give deep insight in the nature of game playing

Types of Games

Board Games



“Physical” Games



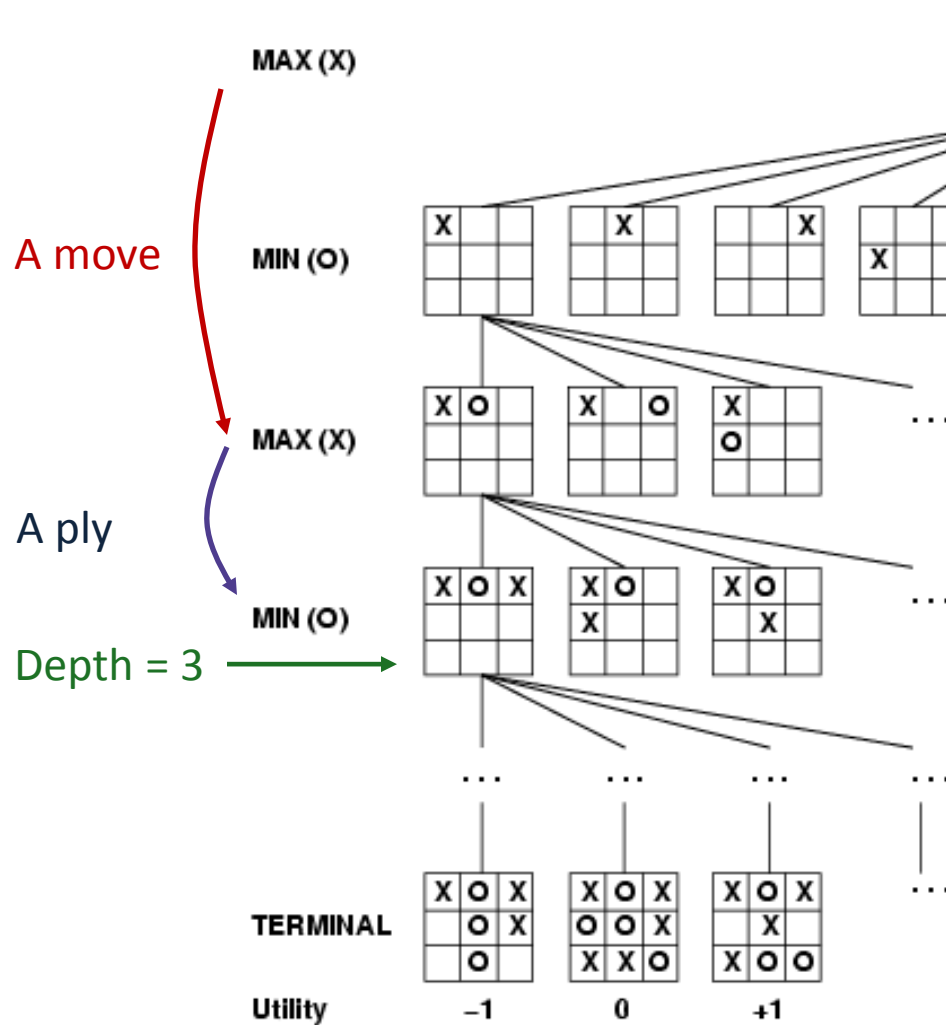
Our Assumptions

- Discrete states
- Turn-taking
- Perfect information
- Zero-Sum utility
- Mainly 2-Player
- Mainly deterministic

Today's Program

- **[10:10-11:00]**
 - Game trees
 - Minimax optimal strategy
 - MINIMAX
 - MINIMAX with alpha-beta pruning
- **[11:10-11:50]**
 - Evaluation functions
 - Cut-off strategies
 - EXPECTIMINIMAX
- **[12:00-14:00]**
 - Exercises



Tic-Tac-Toe Game Tree



- We assume the initial node to be a MAX node
- **Search tree**: part of game tree we search to decide an action

Is a solution a path to a terminal state?

Game Tree

- Initial state s_0
- **PLAYER**(s)
 - Max node  (us)
 - Min node  (opponent)
- **ACTIONS**(s)
- **RESULT** (s, a)
- **TERMINAL-TEST**(s)
- **UTILITY**(s, p) (e.g., win +1, draw 0, loss -1)

Is this an example of a search problem as defined in last lecture?

Optimal Decisions in Games

Maximize worst-case utility for Max!

$\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s, \text{PLAYER}(s)) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax decision at root =

Action to a with highest MINIMAX VALUE

Minimax Algorithm

function MINIMAX-DECISION(*state*) *returns an action*
 return $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(state, a))$

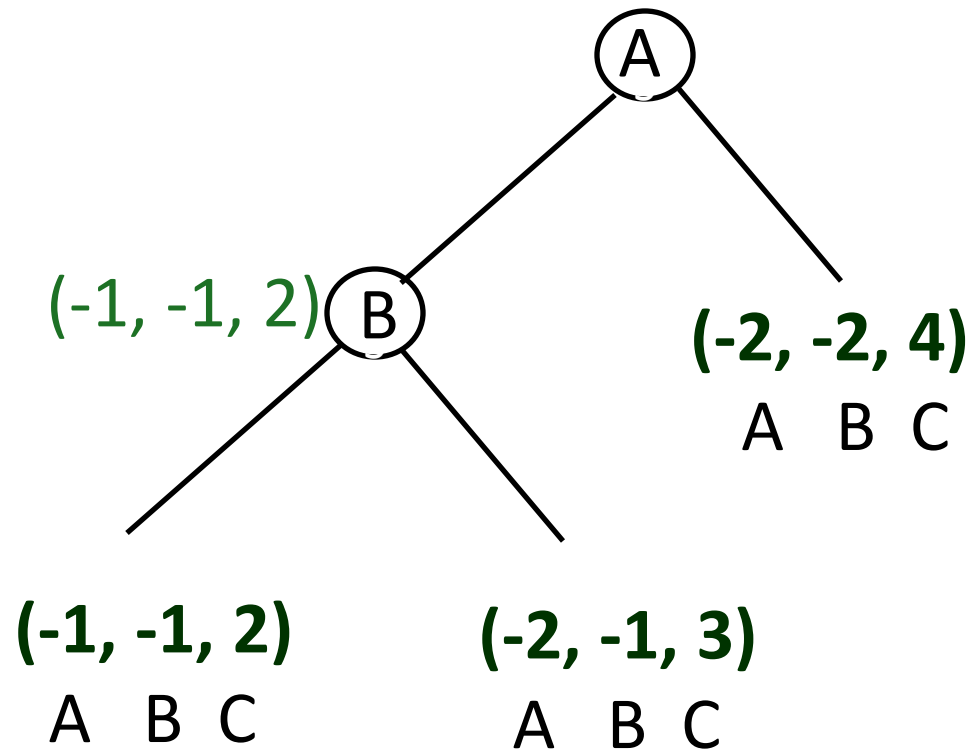
function MAX-VALUE(*state*) *returns a utility value*
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*, PLAYER(*state*))
 $v \leftarrow -\infty$
 for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$
 return *v*

function MIN-VALUE(*state*) *returns a utility value*
 if TERMINAL-TEST(*state*) **then return** UTILITY(*state*, PLAYER(*state*))
 $v \leftarrow \infty$
 for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$
 return *v*

Complexity

- b : max number of successors of a node
- m : max depth of tree
- Time: ?
- Space: ?

Multi-Player Alliances



α - β Pruning

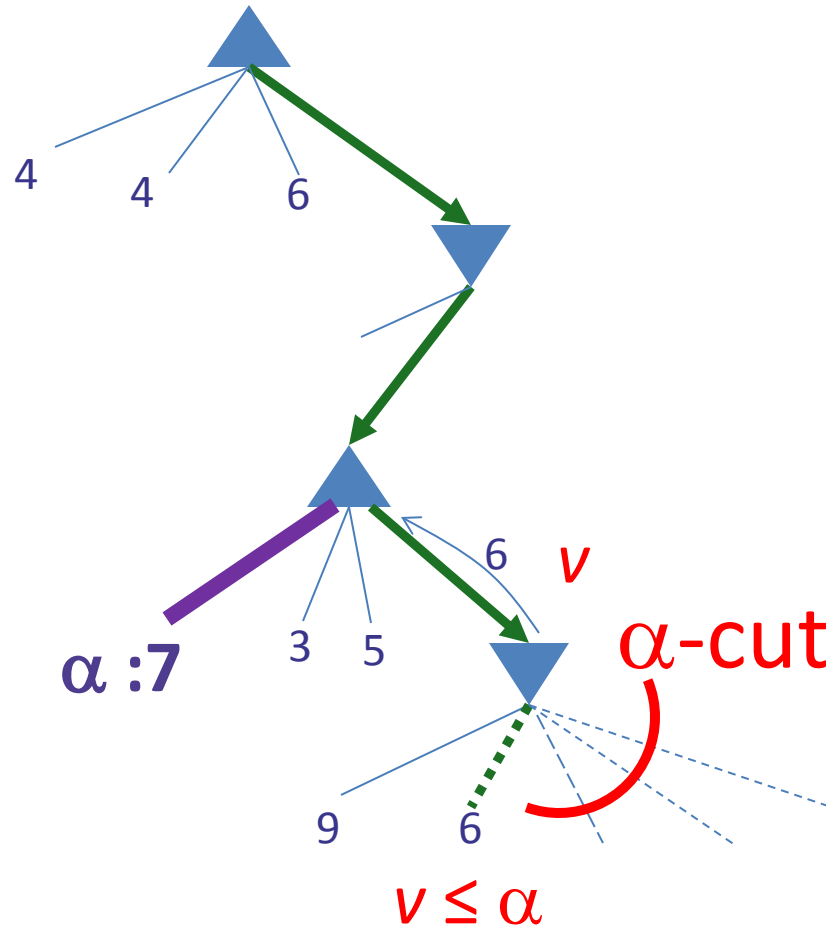
Idea: keep track of MAX and MIN's best choice found so far on the **explored path**

- α = the value (i.e. highest value) of the best choice for MAX
- β = the value (i.e. lowest value) of the best choice for MIN

Use this to **prune the tree**

α -cut

$$\alpha = \max\{4, 4, 6, 7, 3, 5\} \\ = 7$$



Minimax with α - β Pruning

function ALPHA-BETA-SEARCH(*state*) *returns an action*

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

return the *action* in ACTIONS(*state*) with value v

function MAX-VALUE(*state*, α , β) *returns a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*, PLAYER(*state*))

$v \leftarrow -\infty$

for each a **in** ACTIONS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$

if $v \geq \beta$ **then return** v

$\alpha \leftarrow \text{MAX}(\alpha, v)$

return v

function MIN-VALUE(*state*, α , β) *returns a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*, PLAYER(*state*))

$v \leftarrow +\infty$

for each a **in** ACTIONS(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$

if $v \leq \alpha$ **then return** v

$\beta \leftarrow \text{MIN}(\beta, v)$

return v

Properties of α - β Pruning

- Does not affect the final result
- Exploring “best” nodes first improve pruning (killer heuristic)
- $O(b^{m/2})$ with perfect ordering
 - doubles reachable search depth!

Break



Imperfect Real-Time Decisions

Standard Approach for MINIMAX

- Use **CUTOFF-TEST** instead of **TERMINAL-TEST**
 - e.g., depth-limit
- Use **EVAL** instead of **UTILITY**
 - i.e., **evaluation function** that estimates **desirability** of position
 - 1) Order terminal state in the same way as **UTILITY**
 - 2) Value strongly correlated with chance of winning
 - 3) Efficiently computable

Minimax with EVAL and CUTOFF-TEST

function MINIMAX-DECISION(*state*) *returns an action*
 return $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$

function MAX-VALUE(*state*) *returns a utility value*
 if CUTOFF-TEST(*state*) **then return** EVAL(*state*, PLAYER(*state*))
 $v \leftarrow -\infty$
 for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$
 return *v*

function MIN-VALUE(*state*) *returns a utility value*
 if CUTOFF-TEST(*state*) **then return** EVAL(*state*, PLAYER(*state*))
 $v \leftarrow \infty$
 for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$
 return *v*

Evaluation Functions

Idea to reduce complexity

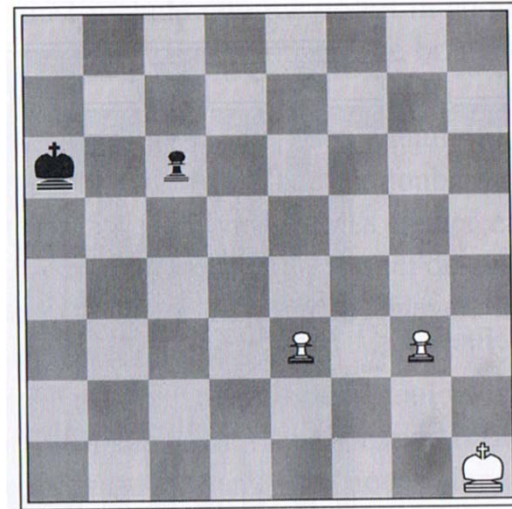
1. Simplify states by extracting **features**
2. Use features as input to evaluation functions

Examples

- Chess: # of pawns, rooks, queens,... of each side
- Tic-tac-toe: # adjacent tokens of each side
- Etc...

Expected Utility

- **Eval** = expected utility of a category (=all states with same state features)
- Category example:
two pawns vs. one

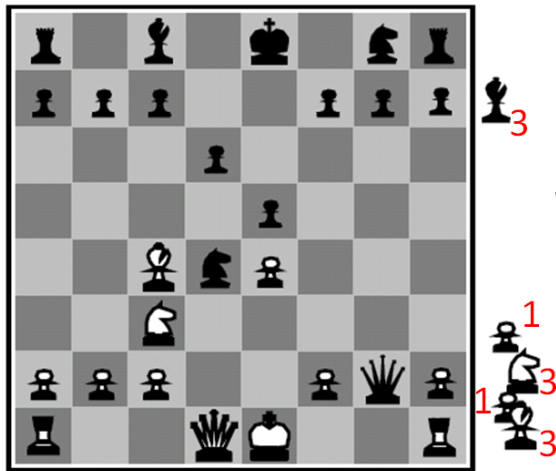


From experience : 72% win, 20% loss 8% draw

Expected Value: $(0.72 * 1) + (0.20 * 0) + (0.08 * \frac{1}{2}) = 0.76$

But: too many states of each category in practice!

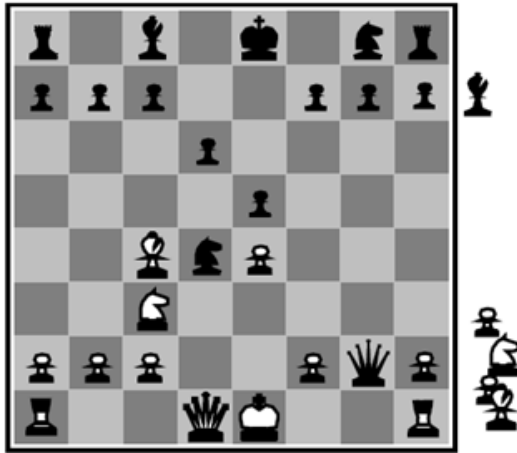
Material Value



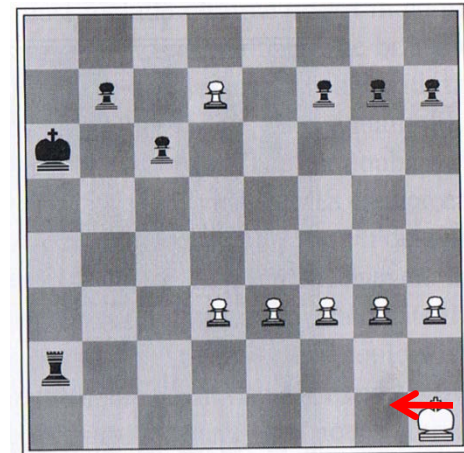
- Eval = sum of feature values
- For chess typically linear weighted sum of features
 - $EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
- E.g. $w_1 = 9$ with
 - $f_1(s) = (\# \text{ of white queens}) - (\# \text{ of black queens}), \text{ etc.}$

CUT-OFF Function

White to
move



Quiescence State

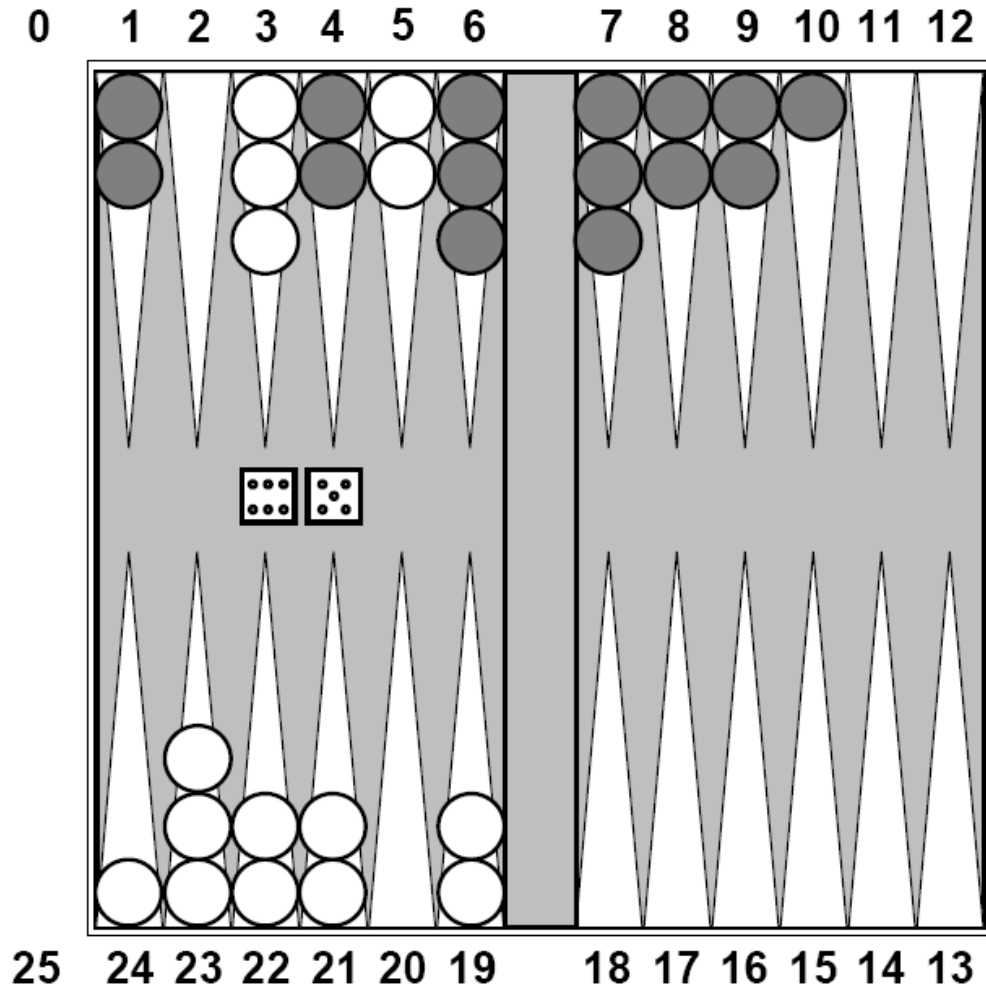


Horizon Effect

Iterative Deepening, possibly extended with

- **Quiescence search**: only make cut-off in **quiescence states**
- **Singular extensions**: apply singular extension to avoid horizon effect
- **Transposition table**: memorize previously evaluated states

Nondeterministic Games



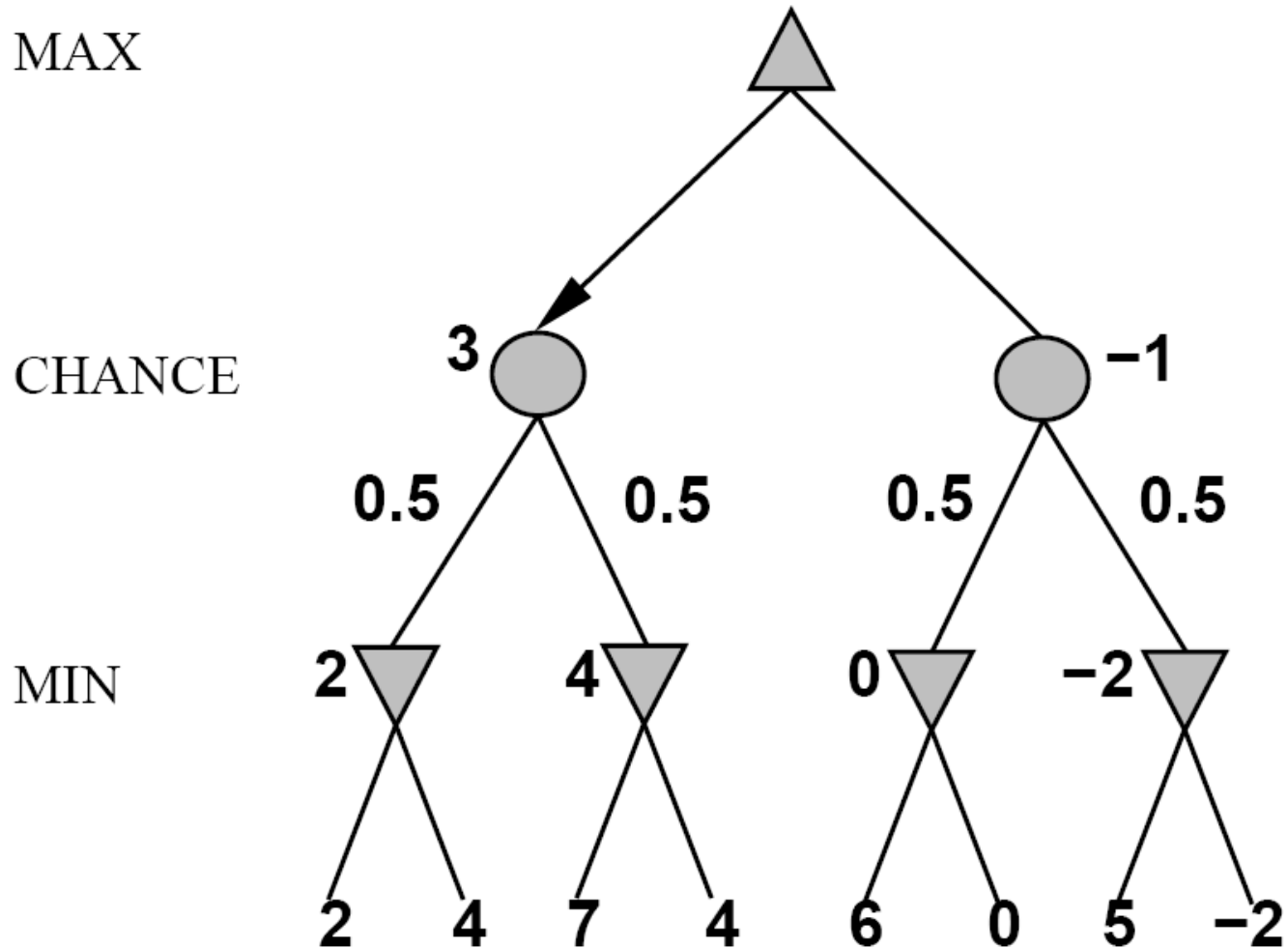
Backgammon

Branching factor ≈ 20

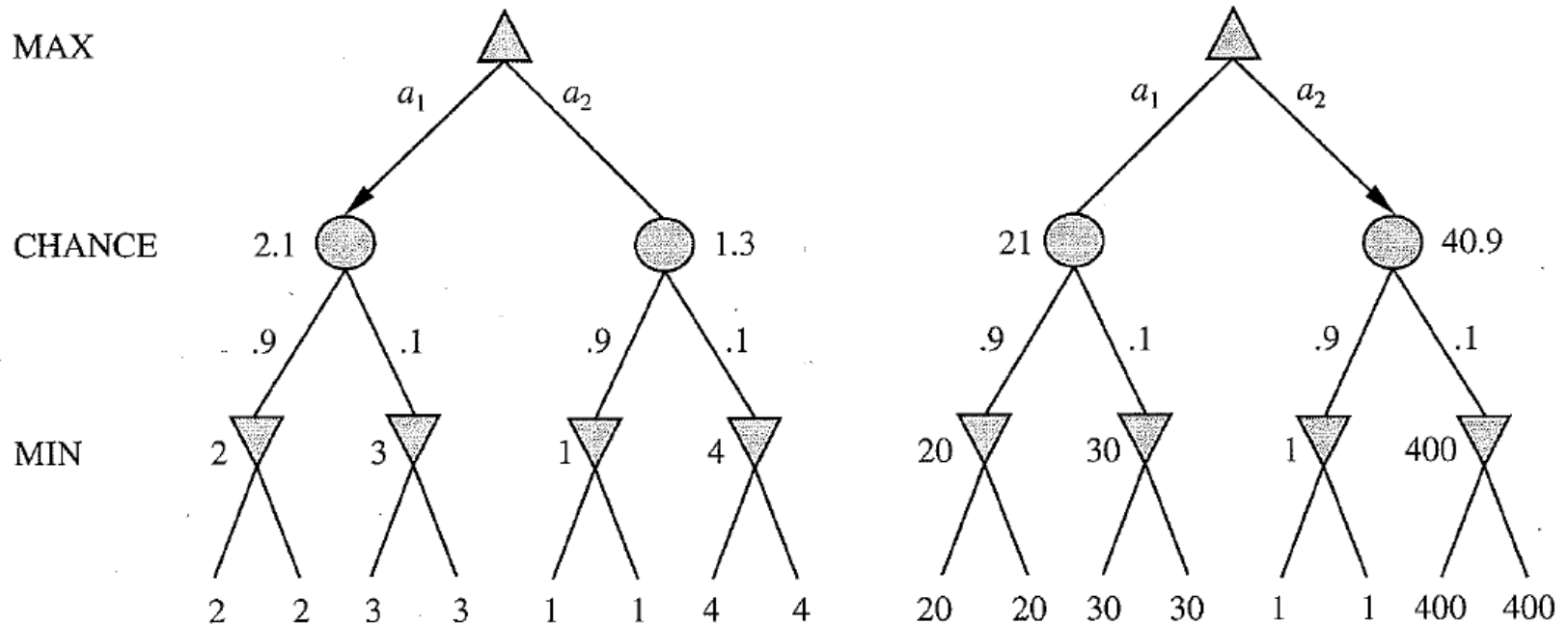
Distinct dice rolls = 21

Total outcomes 420!

Expectiminimax



Evaluation Function



- Only **linear transformations** of utility function allowed!

Expectiminimax

EXPECTIMINIMAX(s) =

UTILITY(s , PLAYER(s))

$\max_{a \in \text{ACTIONS}(s)} \text{EXPECTIMINIMAX}(\text{RESULT}(s, a))$

$\min_{a \in \text{ACTIONS}(s)} \text{EXPECTIMINIMAX}(\text{RESULT}(s, a))$

$\sum_r P(r) \cdot \text{EXPECTIMINIMAX}(\text{RESULT}(s, r))$

if TERMINAL-TEST(s)

if PLAYER(s) = MAX

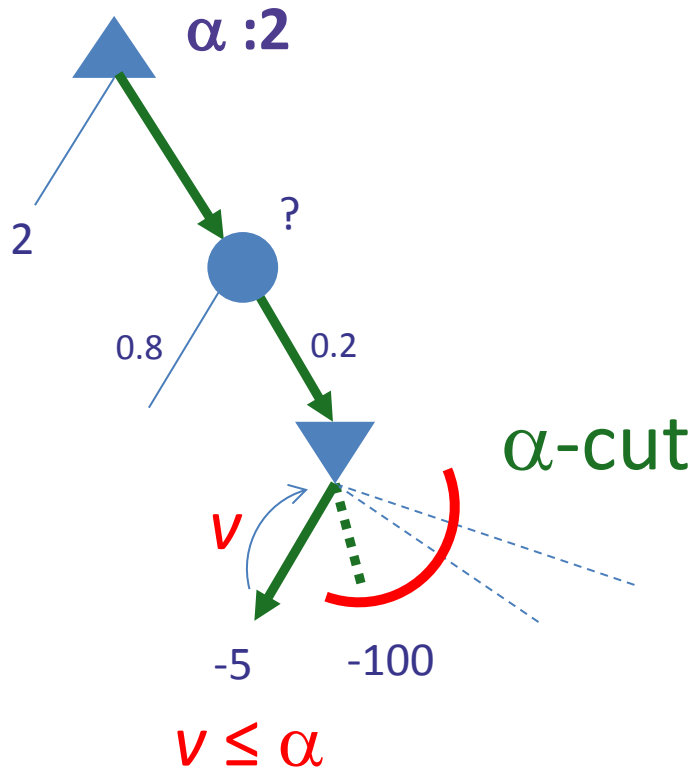
if PLAYER(s) = MIN

if PLAYER(s) = CHANCE

Complexity

- b : max number of successors of a node
- m : max depth of tree n : number of distinct dice rolls
- Time: ?
- Space: ?

Does simple α - β pruning still work?



Problem: due uncertainty the value of the chance node may be better than -5.

So MIN cannot conclude that the cut can be made!