Introduction to Artificial Intelligence

Adversarial Search

Jianmin Li

Department of Computer Science and Technology
Tsinghua University

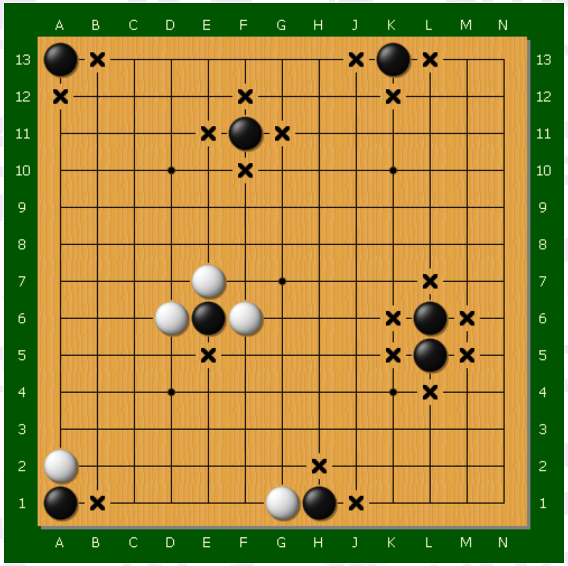
Spring, 2024

Outline

- Games
- Perfect Play
- Resource limits and approximate evaluation
- Games of chance

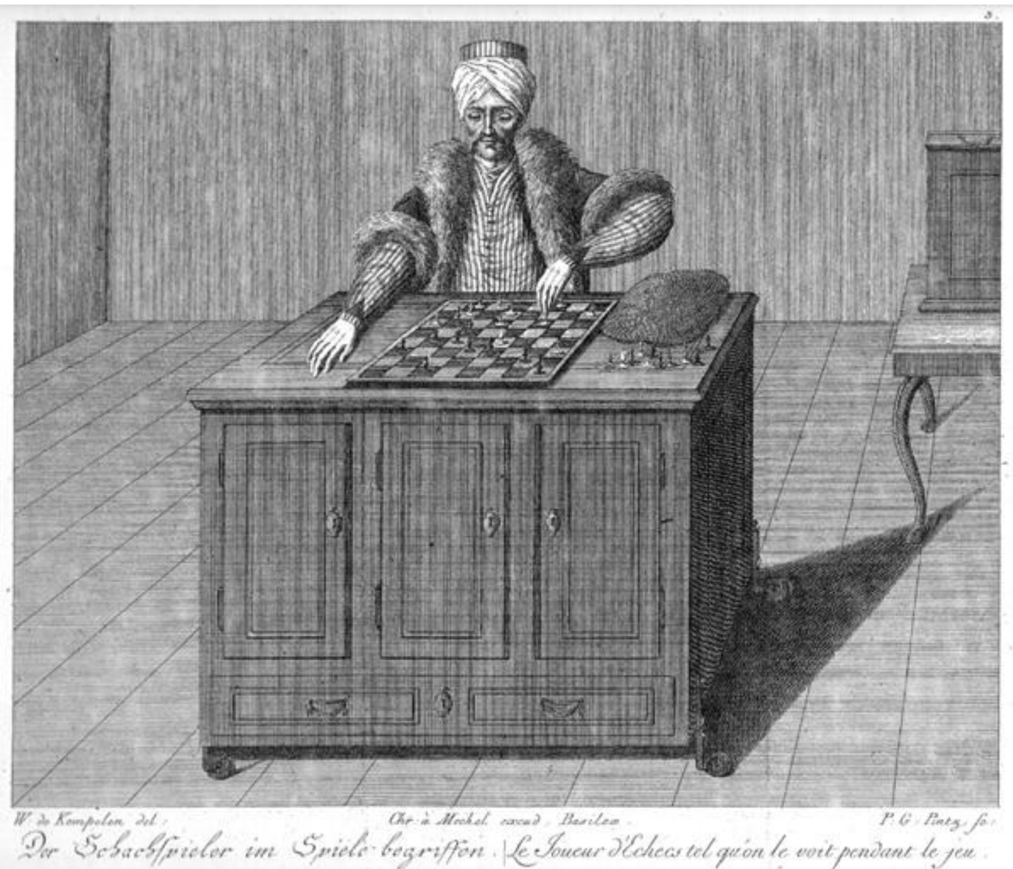
Games vs. search problems

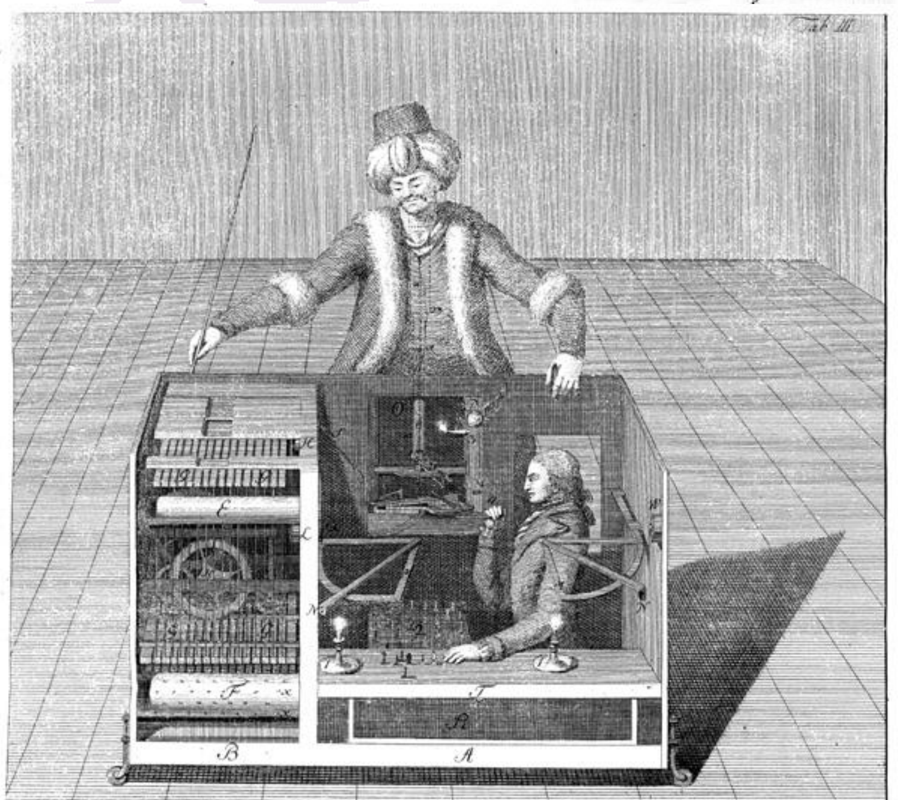
- "Unpredictable" opponent
  - solution is a strategy
  - specifying a move for every possible opponent reply
- Time limits
  - unlikely to find goal, must approximate



History (1)

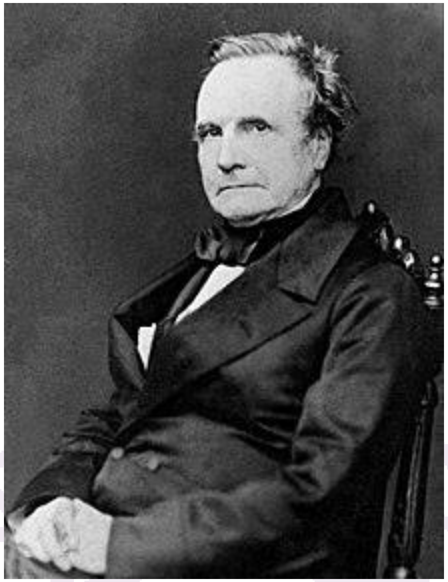
- The Turk (1770-1854)

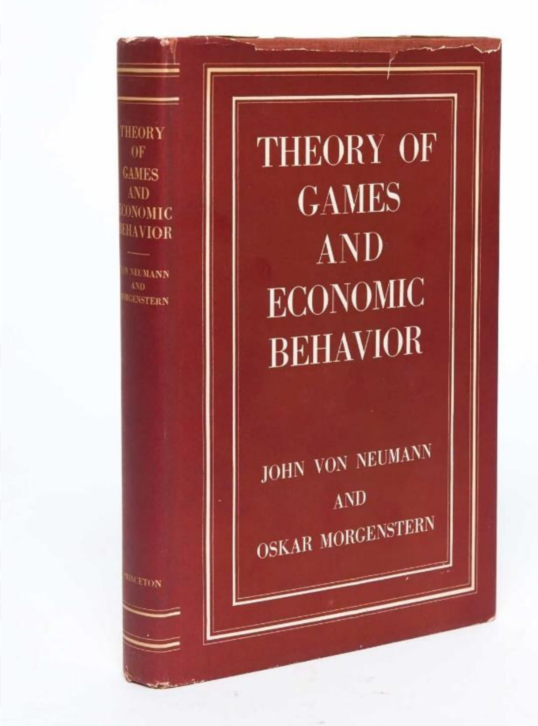


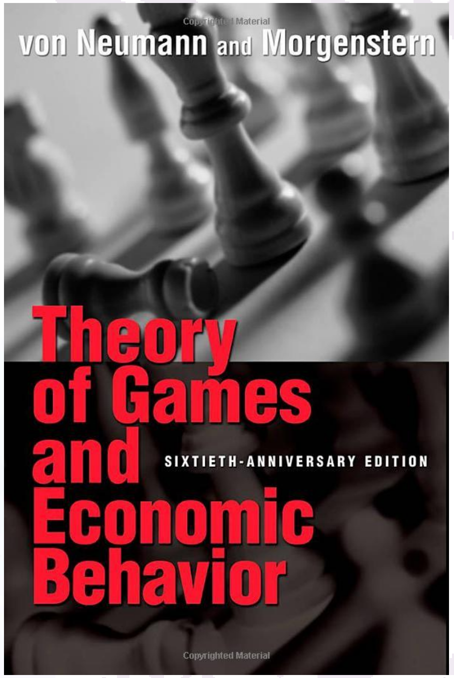


History (2)

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play
  - Zermelo’s Theory, 1912
  - minimax, Von Neumann 1928
  - game of theory: Von Neumann and Morgenstern, 1944







History (3)

- Chess
  - Zuse, 1945; Wiener, 1948; Turing, 1950;
  - Programming a Computer for Playing Chess: Shannon, 1950
  - endgame: D. G. Prinz (1952)
  - full game of standard chess: Bernstein and Roberts, 1958
- Machine learning to improve evaluation accuracy (Checker, Samuel, 1952–57)
- Pruning to allow deeper search
  - alpha-beta: McCarthy, 1956; Newell et al., 1958; Hart and Edwards, 1961; Hart et al. 1972; Knuth and Moore 1975

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

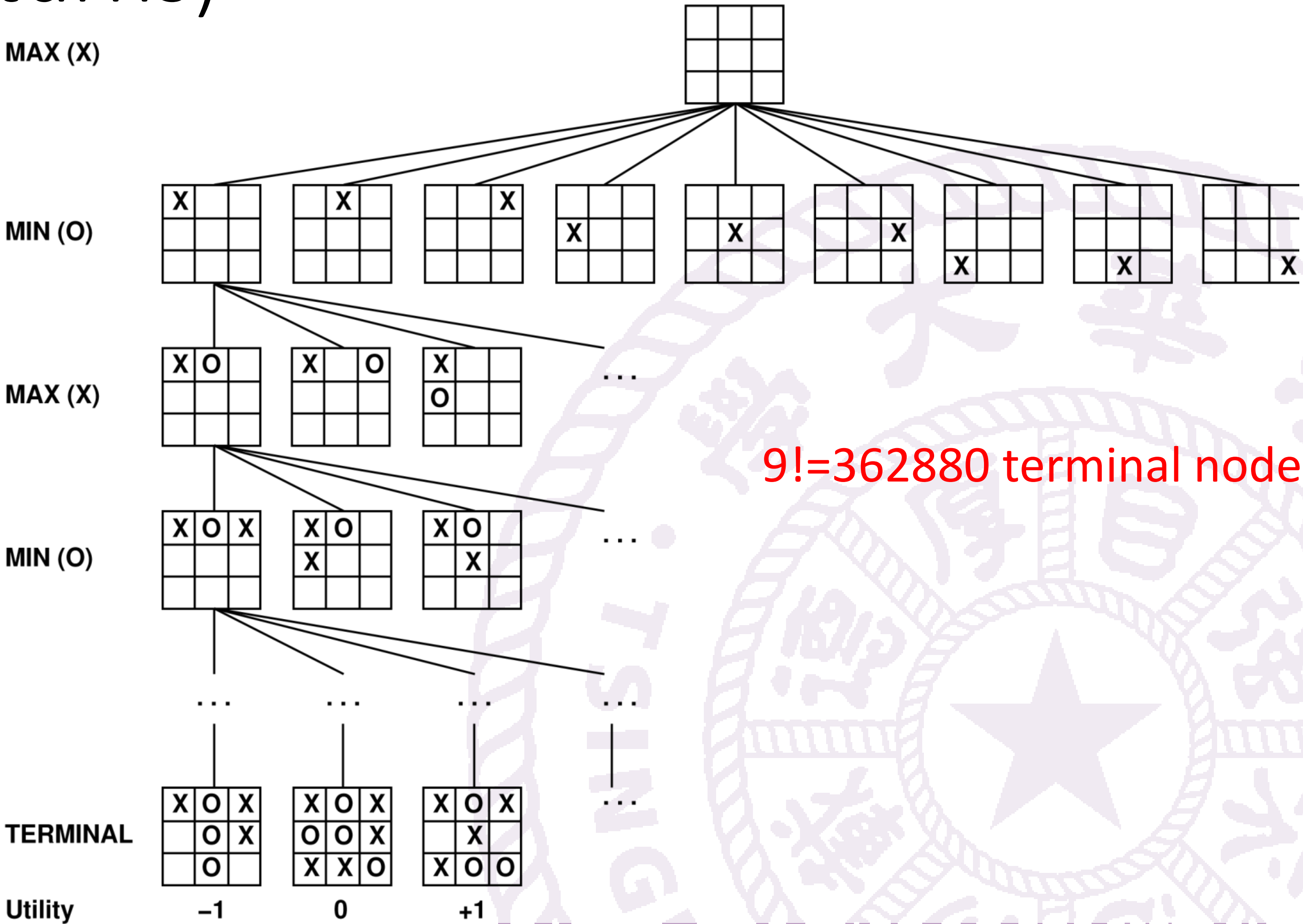
Game tree (2-player, deterministic, turns)

- The initial state
  - How the game is set up at the start
- PLAYER(s)
  - Which player has the move in a state
- ACTIONS(s)
  - Returns the set of legal moves in a state
- RESULT(s, a)
  - The transition model, defining the result of a move

Game tree (2-player, deterministic, turns)

- TERMINAL-TEST(*s*)
  - TRUE when the game is over and FALSE otherwise
  - terminal states
- UTILITY(*s*, *p*)
  - Utility function, also called an objective function or payoff function
  - final numeric value for a game that ends in terminal state *s* for a player *p*

Game tree (2-player, deterministic, turns)



9!=362880 terminal nodes

Minimax

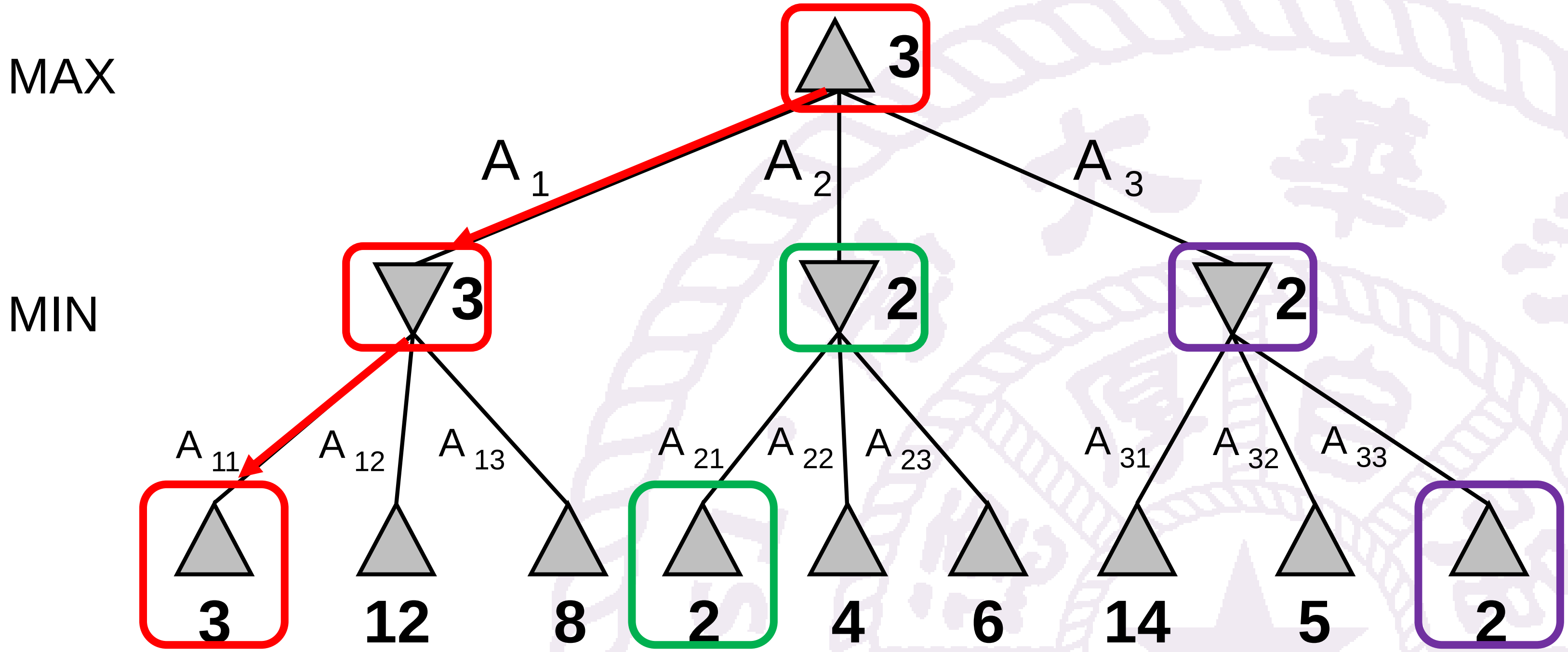
- Perfect play for deterministic, perfect-information games
- Idea: choose move to position with highest minimax value
  - best achievable payoff against best play

$$
\textsc{Minimax}(s) =
\begin{cases}
\textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\
\max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\
\min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s, a)) & \text{if } \textsc{Player}(s) = \textsc{min}
\end{cases}
$$

Minimax

- 2-ply game

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



Minimax algorithm

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

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

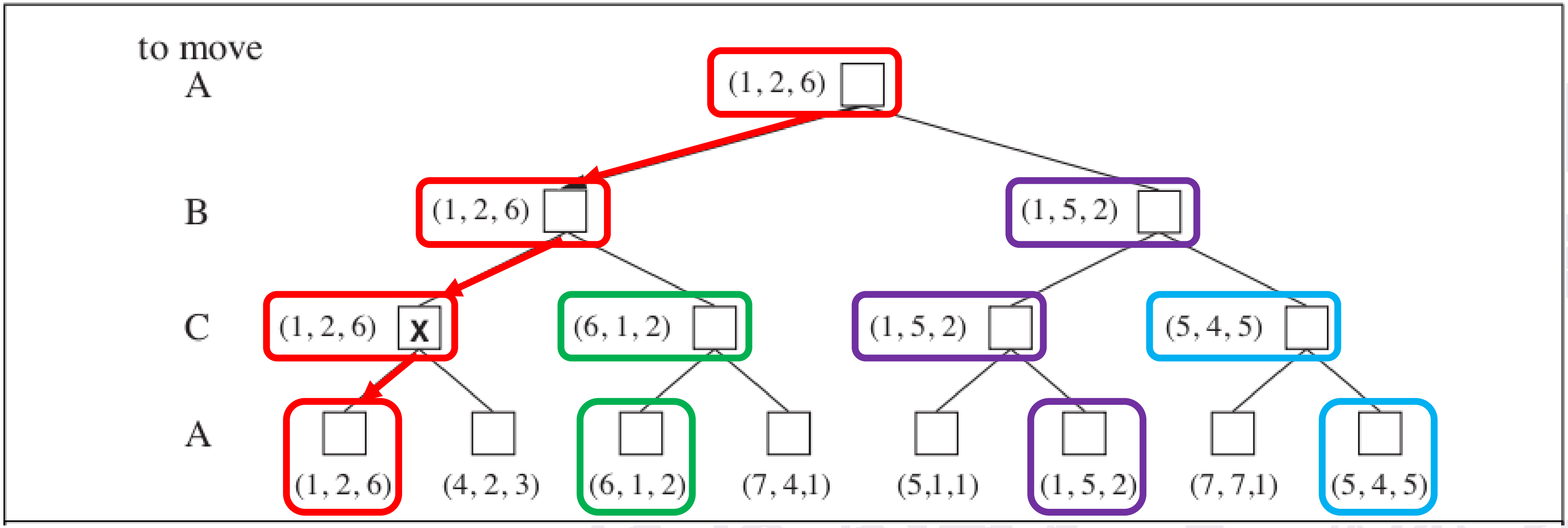
---

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

---

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

Optimal decisions in multiplayer games



MAX for all

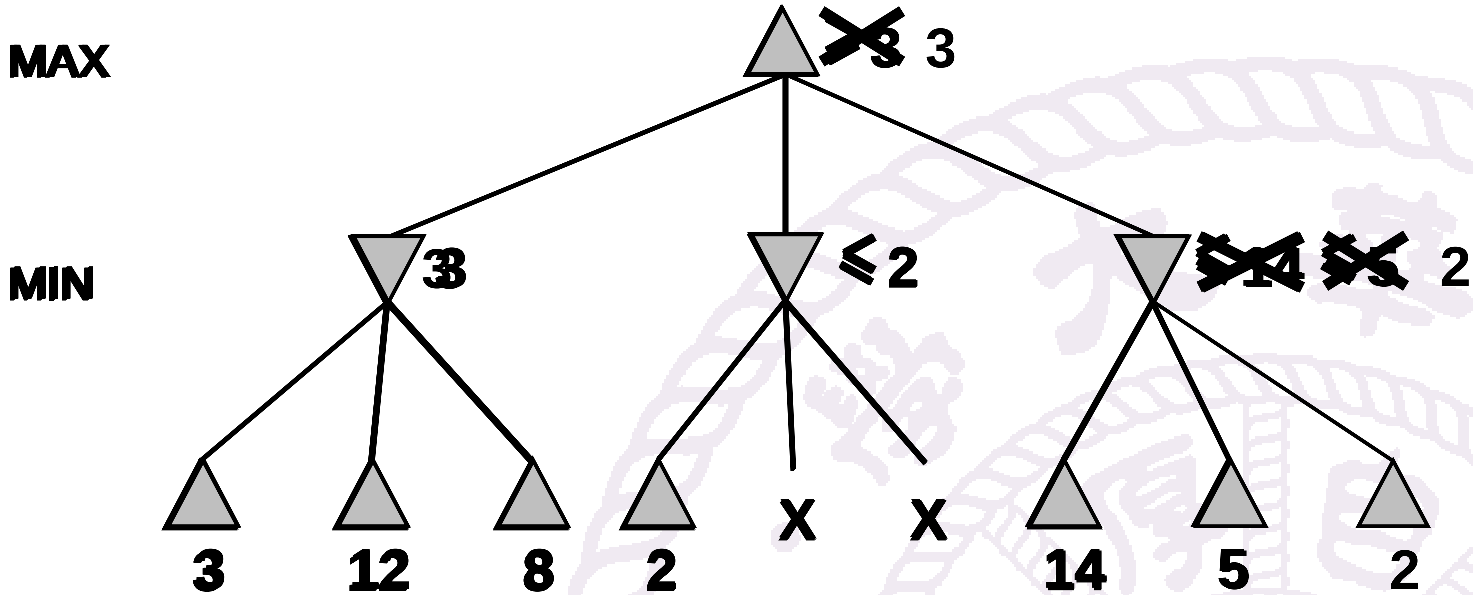
Properties of Minimax algorithm

- Complete??
  - Only if tree is finite.
- Optimal??

But do we need to explore every path?

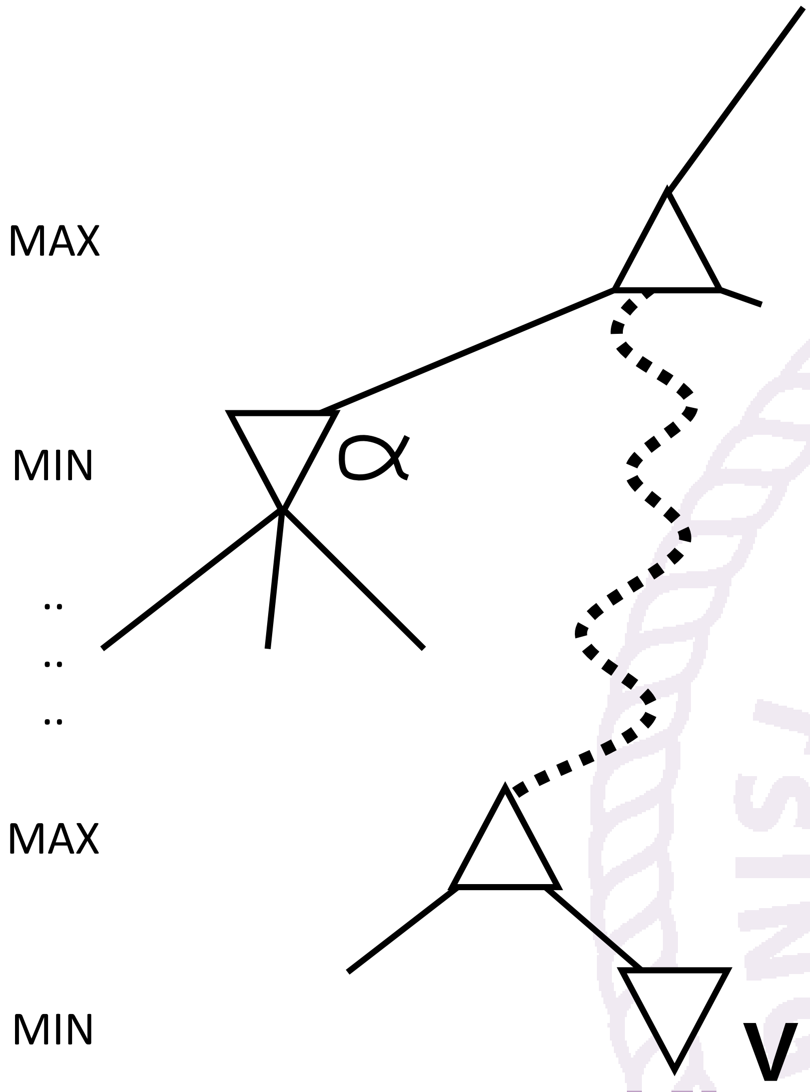
-
  - O (b^m).
- Space complexity??
  - O (bm)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  - exact solution completely infeasible

$\alpha - \beta$ prunning example



$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \leq 2 \\
&= 3.
\end{aligned}
$$

Why is it called $\alpha - \beta$?



- α is the best value (to MAX) found so far off the current path
- If V is worse than α, MAX will avoid it => prune that branch
- β is the best value (to MIN) found so far off the current path

$\alpha - \beta$ algorithm

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

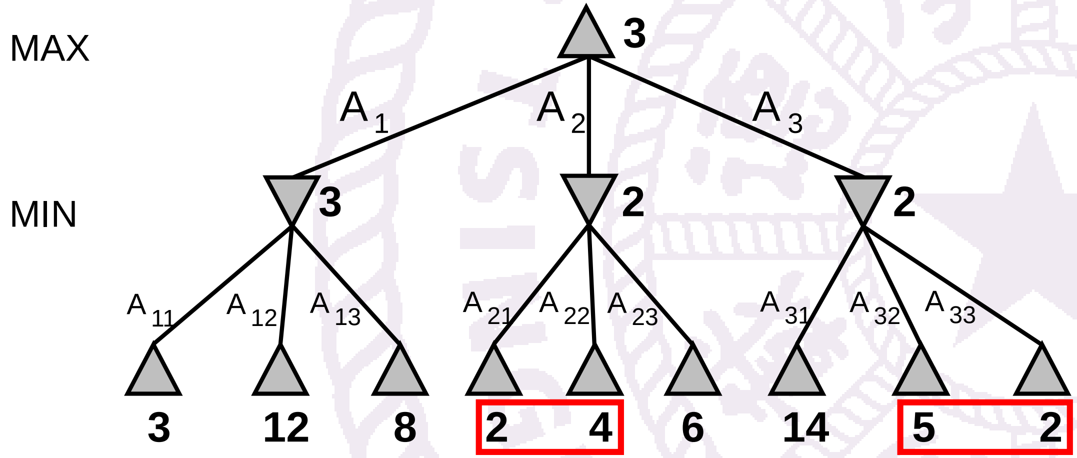
function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Properties of $\alpha - \beta$

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering", time complexity = $O(b^{m/2})$
  - doubles solvable depth
- Unfortunately, 35^{50} is still impossible!



Resource limits

- Standard approach

$$
\text{H-MINIMAX}(s,d) =
\begin{cases}
\text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\
\max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}.
\end{cases}
$$

- Use CUTOFF-TEST instead of TERMINAL-TEST
  - e.g., depth limit
  - perhaps add quiescence search
- Use EVAL instead of UTILITY
  - i.e., evaluation function that estimates desirability of position

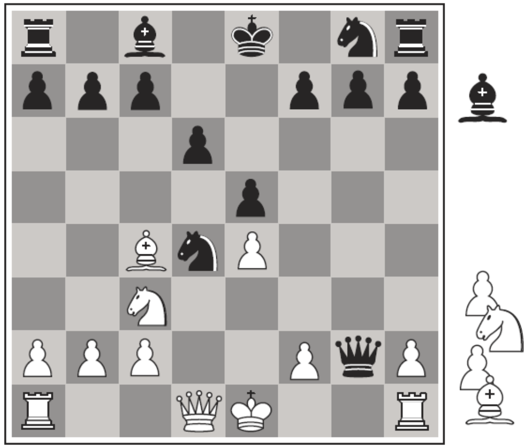
Evaluation functions

- For chess, typically linear weighted sum of features

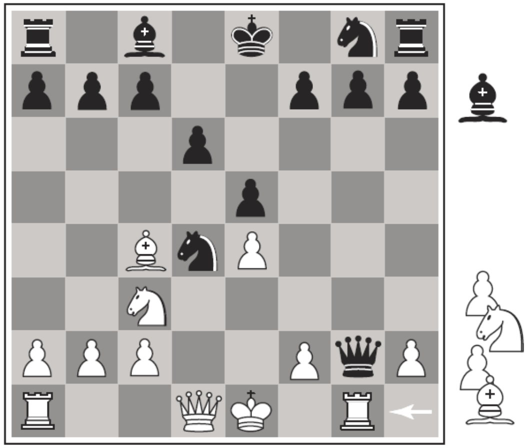
$$\text{Eval}(s) = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

- e.g., $w_1 = 9$ with

$$f_1(s) = \text{(number of whitequeens)} - \text{(number of blackqueens)}$$



White to move
Black better



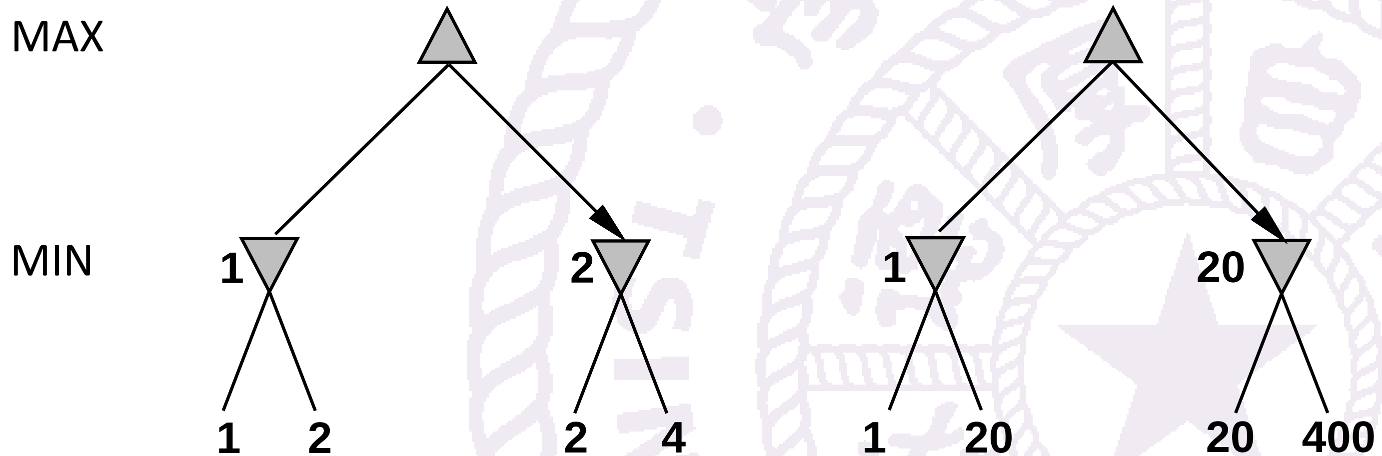
White to move
White winning

Resource limits

- Suppose we have 100 seconds, explore 10^4 nodes/second
  - 10^6 nodes per move $\approx 35^4 = 35^{8/2}$
  - $\alpha - \beta$ pruning reaches depth 8
  - pretty good chess program

Digression: Exact values don't matter

- Behavior is preserved under any monotonic transformation of EVAL
- Only the order matters:
  - payoff in deterministic games acts as an ordinal utility function

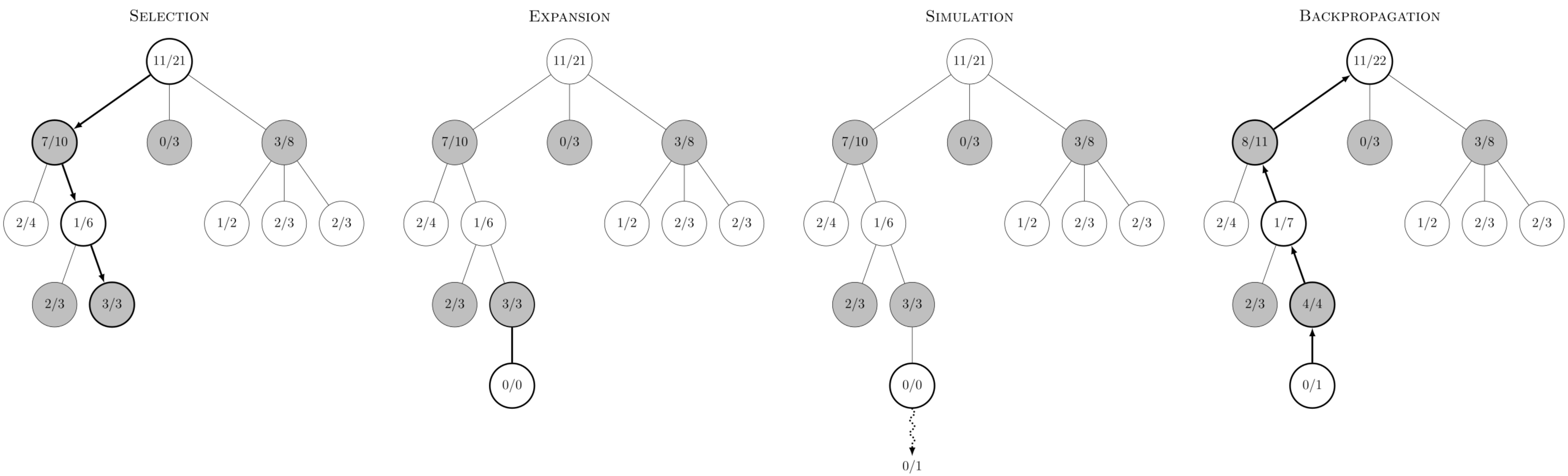


Monte Carlo Tree Search

- Combines the random sampling of traditional Monte Carlo methods with tree searching
- A probabilistic method where random simulations are used to expand the game tree
- Not always find the best move
- Reasonable success at choosing moves that lead to greater chances of winning

Monte Carlo Tree Search

- Selection
- Expansion
- Simulation
- Backpropagation



Monte Carlo Tree Search

- Exploration
  - Promotes exploring unexplored areas of the tree
  - Expand the tree's breadth more than its depth
- Exploitation
  - Stick to one path that has the greatest estimated value
  - Extend the tree's depth more than its breadth

$$UCT(node) = \frac{W(node)}{N(node)} + \boxed{c}\sqrt{\frac{ln(N(parentNode))}{N(node)}}$$

Deterministic games in practice

- Checkers
  - Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 position

Deterministic games in practice

- Chess
  - Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.


Deterministic games in practice

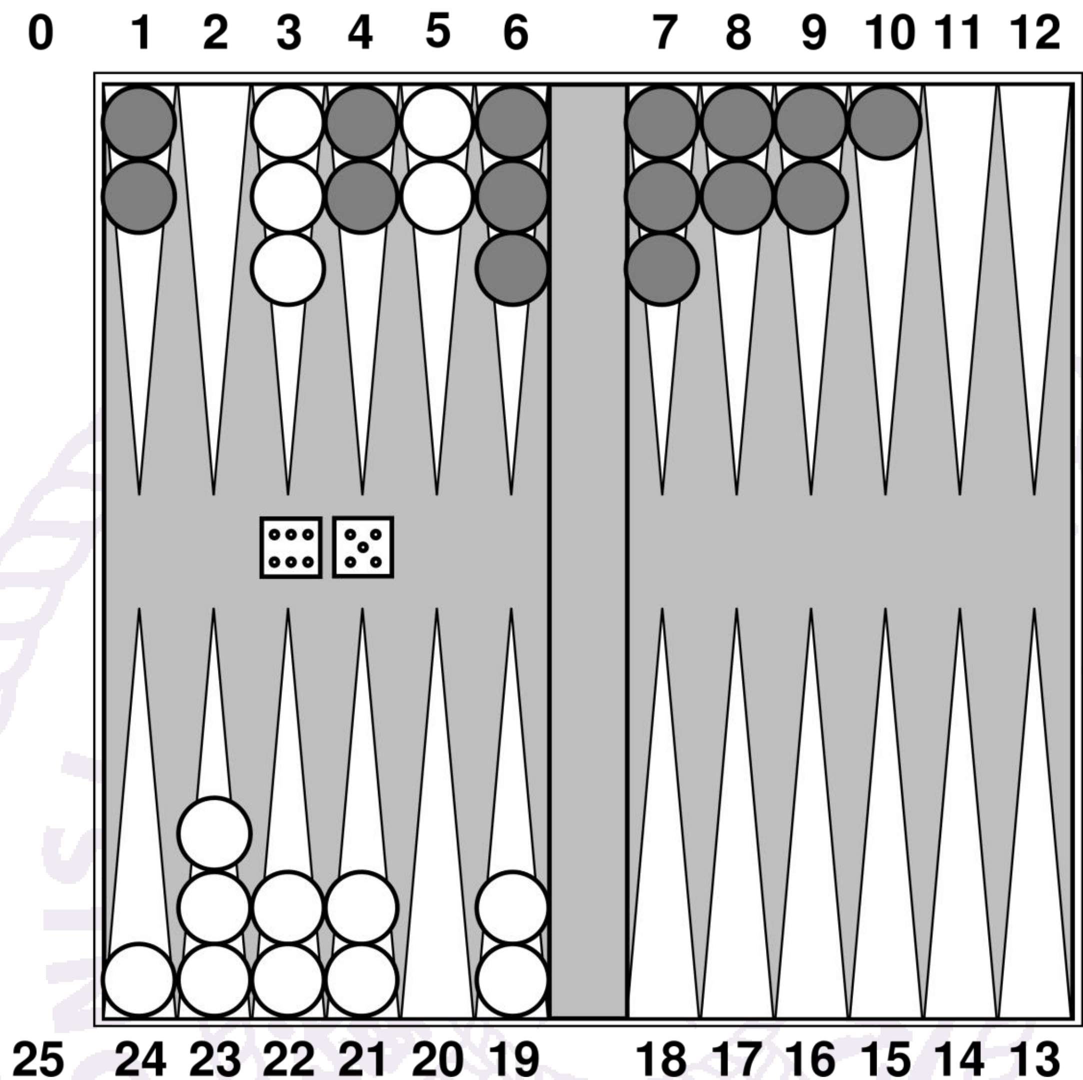
- Othello
  - human champions refuse to compete against computers, who are too good.

Deterministic games in practice

- Go
  - human champions refused to compete against computers, who were too bad
  - But

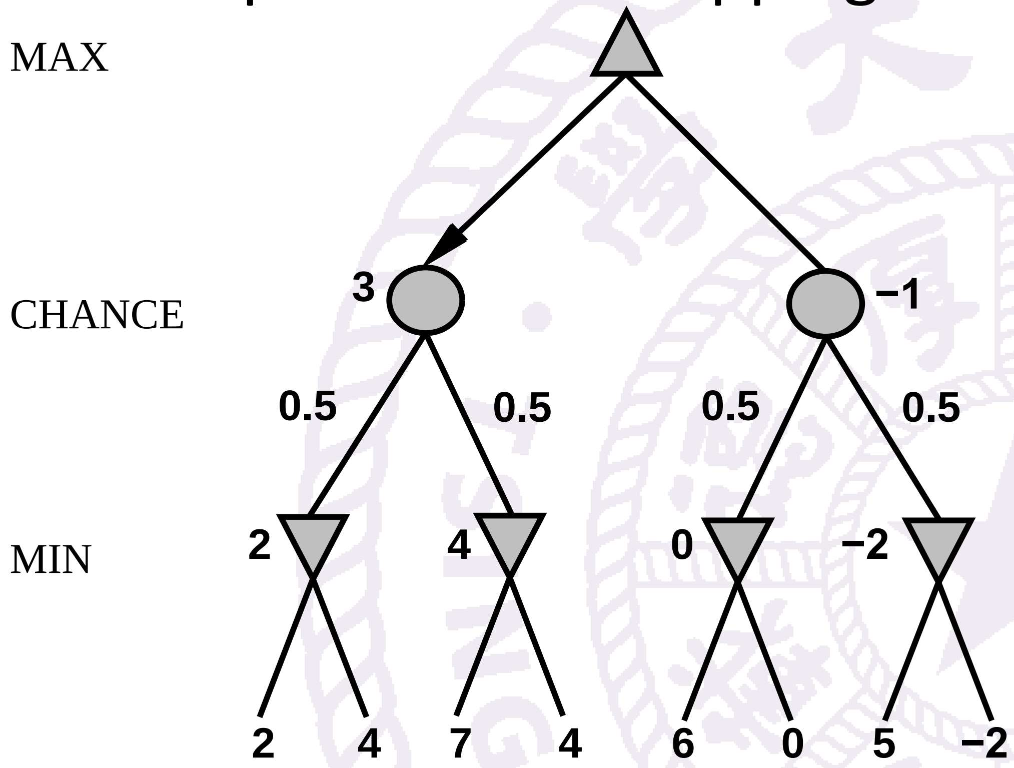
Nondeterministic games: backgammon

- 5 => 10, 5 => 11
- 5 => 11, 19 => 24
- 5 => 10, 10 => 16
- 5 => 11, 11 => 16



Nondeterministic games in general

- In nondeterministic games, chance introduced by dice, card-shuffling
- Simplified example with coin-flipping



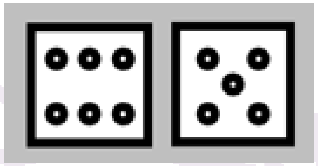
Algorithm for nondeterministic games

- EXPECTIMINIMAX gives perfect play
- Just like MINIMAX, except we must also handle chance nodes

$$\textsc{Expectiminimax}(s) =$$
$$\begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \boxed{\sum_r P(r)} \textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

$$\textsc{Minimax}(s) =$$
$$\begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

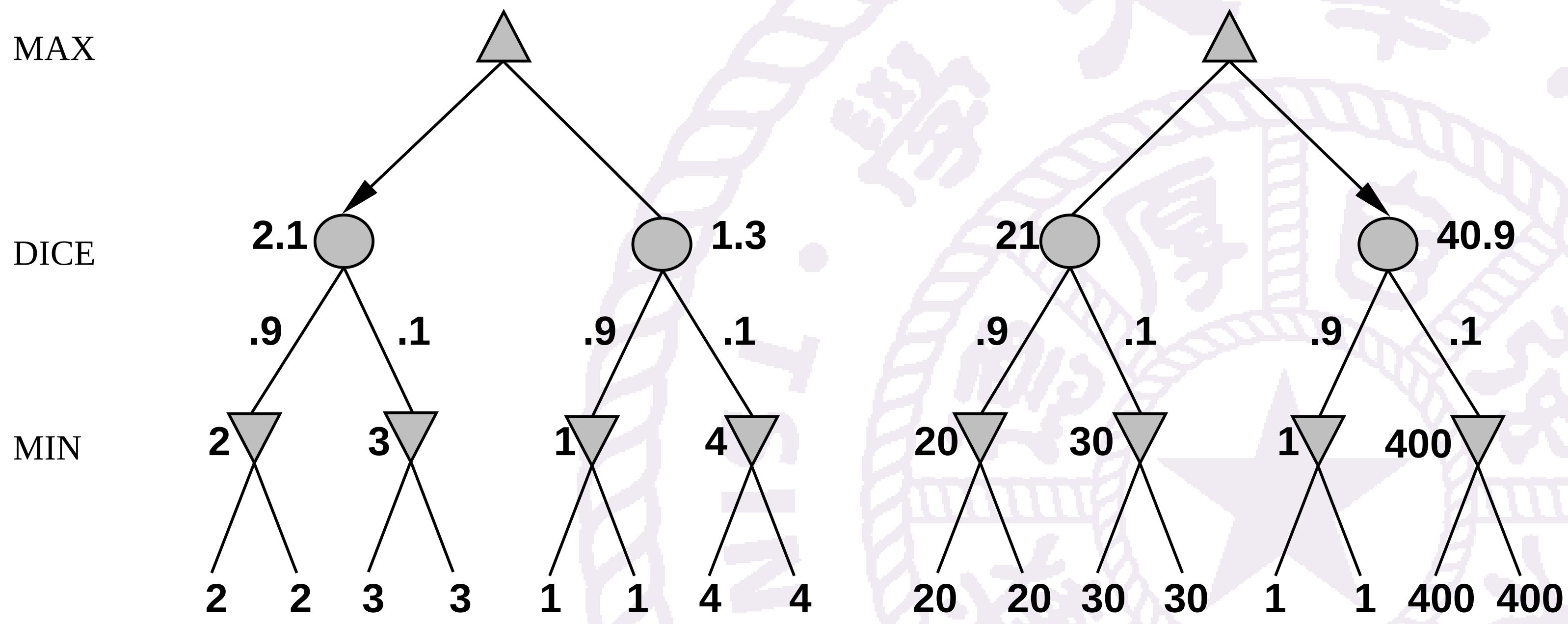
Nondeterministic games in practice



- Dice rolls increase *b*:
  - 21 possible rolls with 2 dice
- Backgammon ≈ 20 legal moves
  - depth 4 = $20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
  - value of lookahead is diminished
- TDGAMMON uses depth-2 search + very good EVAL
  - ≈ world-champion level

Digression: Exact values DO matter

- Behaviour is preserved only by positive linear transformation EVAL
- EVAL should be proportional to the expected payoff



Summary

- Minimax
- H-Minimax
- ExpectMinimax
- Alpha-beta pruning
- Monte Carlo Tree Search

谢谢！