Adversarial Search and Games

Jihoon Yang

Machine Learning Research Laboratory
Department of Computer Science & Engineering
Sogang University

Why study games?

- Examples: chess, checkers, go, poker, backgammon, bridge
- Why study games?
  - Fun, historically entertaining
  - Interesting subject of study because they are hard
  - Easy to represent and agents restricted to small number of actions
- Games are to AI as grand prix racing is to automobile design

Games vs. Search

- Games are a form of *multi-agent environment*
    - What other agents do affect our success
- Competitive multi-agent environments give rise to *adversarial search*
- Solution is a strategy (specifying a move for every possible opponent reply)

Types of games

- Games of deterministic, perfect information: chess, checkers
- Games of chance: backgammon
- Games of imperfect information: bridge, poker

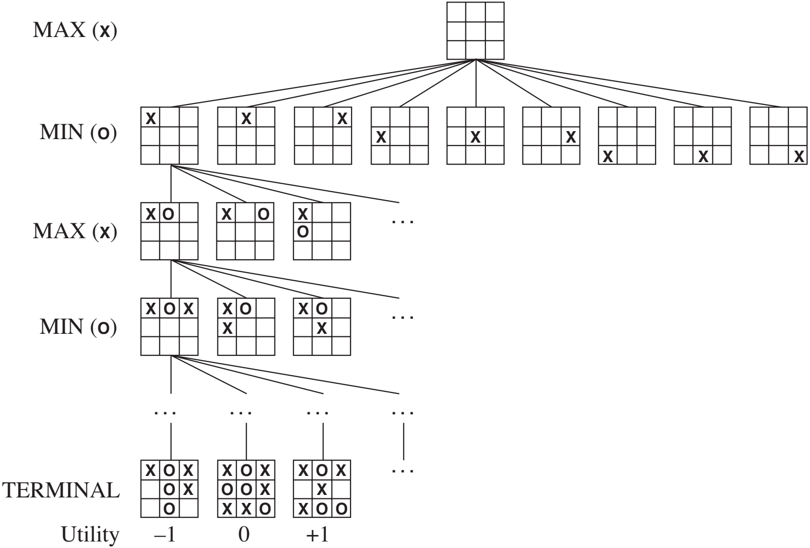
Game setup

- Perfect information: fully observable
- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
- Winner is awarded points (payoff), loser gets penalties
- *Zero-sum game*:
  - What is good for one player is just as bad for the other; there is no "win-win" outcome
  - Payoff values at the end of the game are equal and opposite (or the total payoff to all players is the same for every instance of the game)

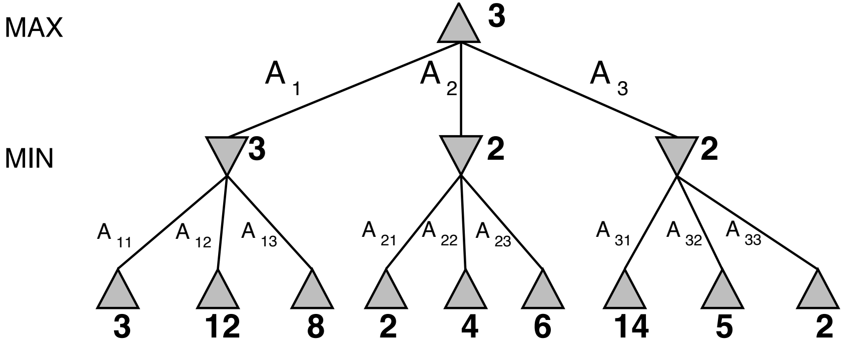
Game setup

- Games as search
  - Initial state: e.g. board configuration of chess
  - Player(s): Defines which player has the move in a state
  - Actions and transition model (or successor function): List of $(move, state)$ pairs specifying legal moves
  - Terminal test: Is the game finished (in *terminal states*)?
  - **Utility function** or **payoff function** $UTILITY(s, p)$: Gives numerical value for terminal state s for a player p; E.g. win(+1), loss(-1), and draw(0) in tic-tac-toe
- The initial state and the legal moves define the **game tree**

Partial game tree for tic-tac-toe



Two-ply game tree



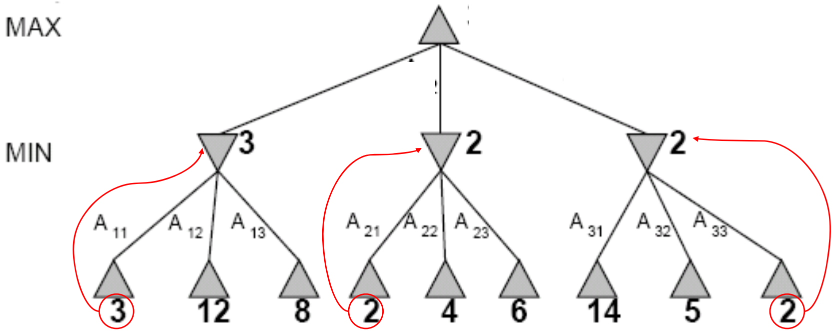
Optimal strategies

- Find the contingent *strategy* for MAX assuming an infallible MIN opponent
- Assumption: Both players play optimally!!
- Given a game tree, the optimal strategy can be determined by using the **minimax value** of each node:

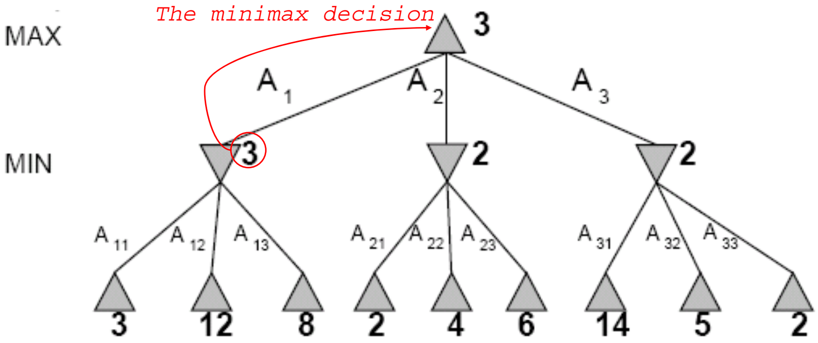
$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{MINIMAX-VALUE}(n)=} \\
\quad \text{UTILITY}(n) & \text{If } n \text{ is a terminal} \\
\quad \max_{s \,\in\, successors(n)} \text{MINIMAX-VALUE}(s) & \text{If } n \text{ is a max node} \\
\quad \min_{s \,\in\, successors(n)} \text{MINIMAX-VALUE}(s) & \text{If } n \text{ is a min node}
\end{array}
$$

- Idea: Choose move to position with highest minimax value
  = best achievable payoff against best play

Two-ply game tree



Two-ply game tree



Mimimax maximizes the worst-case outcome for max

Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the $a$ in ACTIONS(state) maximizing MIN-VALUE(RESULT($a$, state))

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

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

Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes, against an optimal opponent; Otherwise?
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$ (depth-first exploration)
- For chess, $b = 35, m = 80$ for average games → exact solution completely infeasible

What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally: maximizes worst-case outcome for MAX
- But if MIN does not play optimally, MAX will do even better [proof?]
- There may be better strategies against suboptimal opponents, but they do worse against optimal opponents

Improving minimax search

- Number of game states is exponential to the number of moves

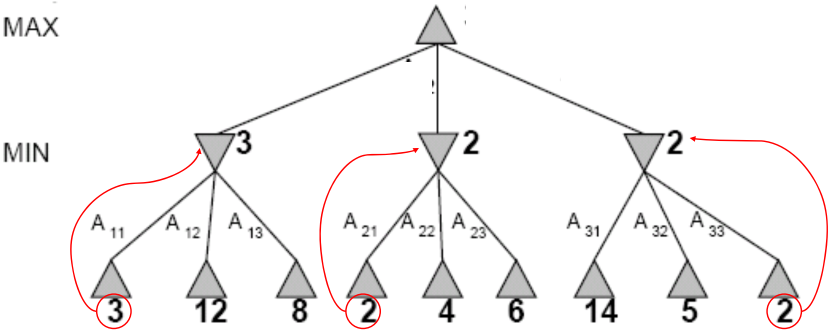
  Solution:

  Max min

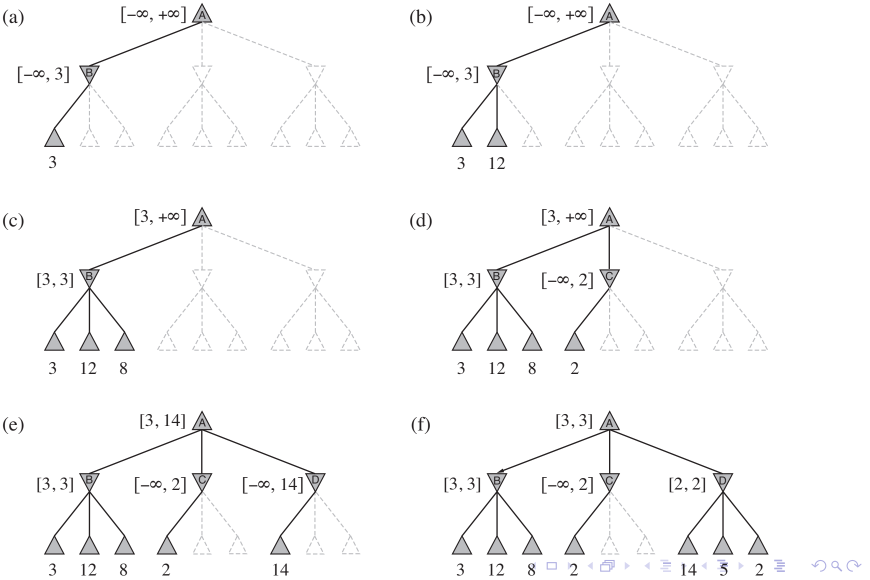
  α β

  - Do not examine every node ⇒ **Alpha-beta pruning**
  - Remove branches that do not influence final decision

Alpha-beta pruning example



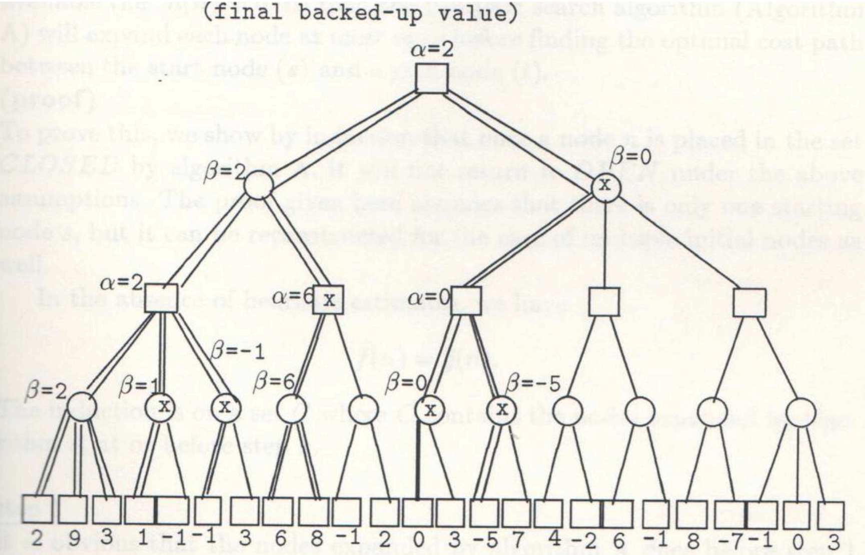
Alpha-beta pruning example



α-β search

- Assign an α-value to MAX nodes and a β-value to MIN nodes
  - α-value of a MAX node $=$ *current* largest backed-up value of its successors
  - β-value of a MIN node $=$ *current* smallest backed-up value of its successors
- Rules for discontinuing search
  - The search may be discontinued below any MIN node having a β-value $\leq$ the α-value of its MAX node ancestors. The final backed-up value of this MIN node is set to its β-value.
  - The search may be discontinued below any MAX node having an α-value $\geq$ the β-value of its MIN node ancestors. The final backed-up value of this MAX node is set to its α-value.

The α-β pruning example



The α-β algorithm

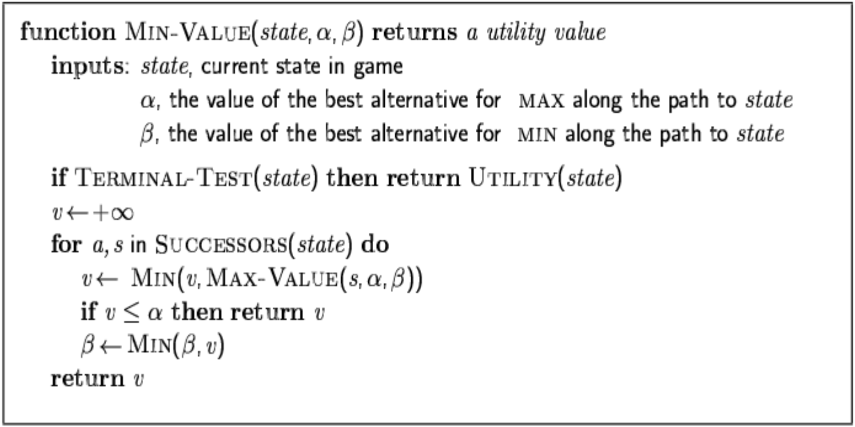
```
function ALPHA-BETA-DECISION($state$) returns an action
   return the $a$ in ACTIONS($state$) maximizing MIN-VALUE(RESULT($a$, $state$))

function MAX-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
   inputs: $state$, current state in game
           $\alpha$, the value of the best alternative for MAX along the path to $state$
           $\beta$, the value of the best alternative for MIN along the path to $state$

   if TERMINAL-TEST($state$) then return UTILITY($state$)
   $v \leftarrow -\infty$
   for $a$, $s$ in SUCCESSORS($state$) do
      $v \leftarrow$ MAX($v$, MIN-VALUE($s, \alpha, \beta$))
      if $v \geq \beta$ then return $v$
      $\alpha \leftarrow$ MAX($\alpha$, $v$)
   return $v$

function MIN-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
   same as MAX-VALUE but with roles of $\alpha, \beta$ reversed
```

The α-β algorithm



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

    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for a, s in SUCCESSORS(state) do
        $v \leftarrow$ MIN($v$, MAX-VALUE($s, \alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta, v$)
    return $v$
```

Properties of alpha-beta pruning

- Pruning does not affect final results
- The effectiveness of pruning is highly dependent on the order in which the successors are checked
  - Try to examine first the successors that are likely to be best
  - Ordering heuristics (e.g. killer move heuristic with IDS)
- With "perfect ordering", time complexity is $O(b^{m/2})$
  - Branching factor of $\sqrt{b}$ (instead of b)!
  - Alpha-beta pruning can look twice as far as minimax in the same amount of time
- Even with alpha-beta pruning and clever move ordering, minimax won't work for games like chess and Go; Two strategies for move exploration: Type A (wide but shallow) & Type B (deep but narrow)

Real-time decisions

- Minimax and alpha-beta pruning require leaf-node evaluations
- May be impractical within a reasonable amount of time
- Suppose we have 100 secs, explore 10^6 nodes/sec
  $\rightarrow$ 10^8 nodes per move $\sim 35^{10/2}$
  $\rightarrow$ Alpha-beta pruning reaches depth 10
  $\Rightarrow$ pretty good chess program

Real-time decisions

Standard approach

- Cut off search earlier (replace TERMINAL-TEST by CUTOFF-TEST)
  - E.g. Introduce a fixed depth limit (selected so that the amount of time will not exceed what the rules of the game allow), or iterative deepening
- Apply heuristic *evaluation function* EVAL (replacing utility function)
- Change:
  - **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

  into
  - **if** CUTOFF-TEST(*state, depth*) **then return** EVAL(*state*)

Evaluation functions

- Idea: produce an estimate of the expected utility of the game from a given position
- Performance depends on quality of EVAL
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY
  - Computation may not take too long
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning
- Exact values don't matter, only the order matters

Evaluation functions

- Most evaluation functions work by calculating various features of the state
  - Compute numerical contributions from each feature and combine them to find the total value

- For chess, *material value* for each piece: pawn=1, knight=bishop=3, rook=5, queen=9
- *Linear* weighted sum of *features*

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

E.g. $w_1 = 9$ with $f_1(s) =$(number of white queens) $-$ (number of black queens)

Real-time decisions

- Extensively tuned evaluation function
- More sophisticated cutoff test (e.g. avoiding wild swings in value in near future)
- Forward pruning by beam search, probabilistic cut, late move reduction
- A large *transposition table*: hash table of previously seen positions
- A large database of optimal opening and endgame moves: table lookup instead of search

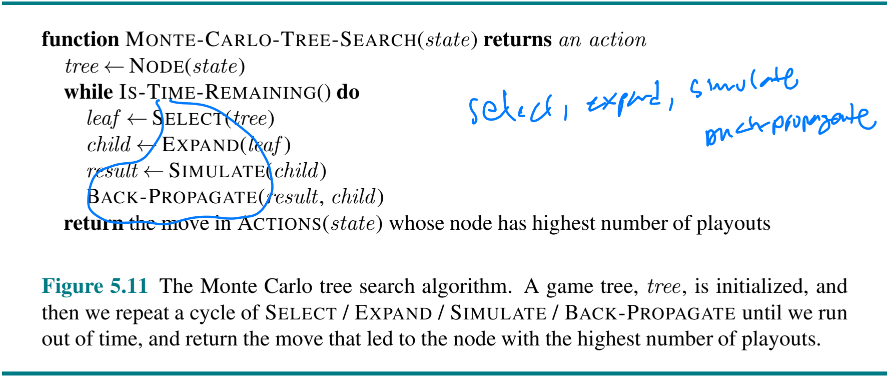
Monte Carlo Tree Search (MCTS)

- The value of a state is estimated as the average utility over a number of simulations (or *playout* or *rollout*) of complete games starting from the state
- At a terminal position, the rules of the game determine who has won or lost, and by what score
- Need a playout policy that biases the moves towards good ones
- Need a selection policy that balances two factors of *exploration* and *exploitation* (e.g. Upper confidence bounds applied to trees (UCT) based on UCB1)

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(Parent(n))}{N(n)}}$$

- MC search has an advantage over alpha-beta when branching factor is high, or good evaluation function is hard to define

The MCTS algorithm

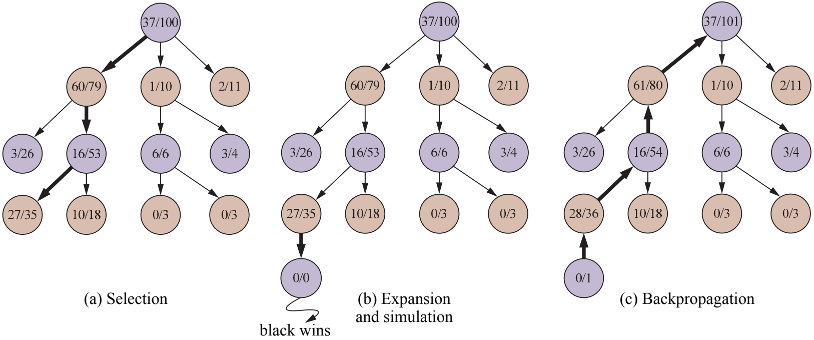


```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

select, expand, simulate
backpropagate

Figure 5.11 The Monte Carlo tree search algorithm. A game tree, *tree*, is initialized, and then we repeat a cycle of SELECT / EXPAND / SIMULATE / BACK-PROPAGATE until we run out of time, and return the move that led to the node with the highest number of playouts.

MCTS example



(a) Selection

(b) Expansion and simulation

(c) Backpropagation

Figure 5.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b) we expand the selected node and do a simulation (playout), which ends in a win for black. In (c), the results of the simulation are back-propagated up the tree.

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions. In July 2007, Chinook's developers announced that the program has been improved to the point where it cannot lose a game.
- Chess: Deep Blue defeated human world champion Garry 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.
- Othello/Reversi: Human champions refuse to compete against computers, who are too good.

Deterministic games in practice

- Go: Google DeepMind
  - AlphaGo: defeated human world champion Sedol Lee in 2016 and Ke Jie in 2017
  - AlphaGo Zero: defeated the champion-defeating versions AlphaGo Lee/Master in 2017 100-0 by 3 days learning without teacher (i.e. self-learning without prior human knowledge except the game rules)
  - AlphaZero: a General Game Playing (GGP) program, achieved within 24h a superhuman level of play in the games of Chess/Shogi/Go (defeated AlphaGo Zero) in Dec. 2017

Stochastic games

- Includes a random element (e.g. throwing of dice)
- Must include *chance nodes* in addition to MAX & MIN nodes
- Need to compute *Expectminimax value* for games with chance nodes

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

Stochastic game tree for a backgammon position

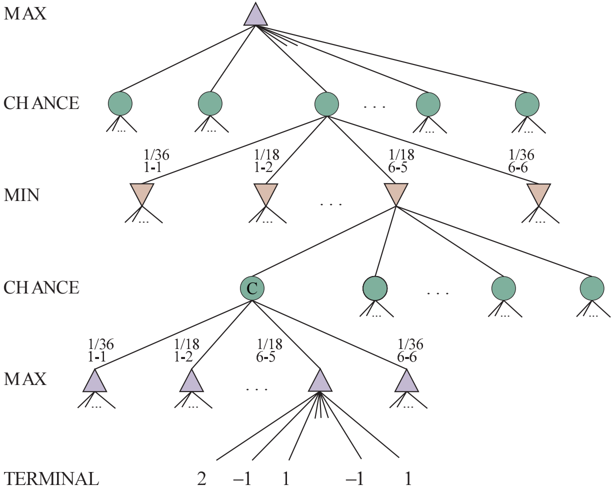


Figure 5.13 Schematic game tree for a backgammon position.

Stochastic games

- Complexity of minimax: $O(b^m)$ (m: maximum depth of the game tree)
- Complexity of expectminimax: $O(b^m n^m)$ (n: number of distinct chances)

- Alpha-beta pruning can be applied to game trees with chance nodes by placing upper bounds without looking at all children

- In games where the branching factor for chance nodes is high
    - Consider forward pruning that samples a smaller number of the possible chance branches, OR
    - Avoid using evaluation function altogether, and opt for MCTS instead where each playout includes random dice rolls

Imperfect information games in practice

- Poker: surpass human experts in the game of heads-up no-limit Texas hold'em, which has over 10^{160} decision points
  - DeepStack: beat top poker pros in limit Texas hold'em in 2008, and defeated a collection of poker pros in heads-up no-limit in 2016
  - Libratus: current two-time champion of the Annual Computer Poker Competition in heads-up no-limit, and defeated a team of top heads-up no-limit specialist pros in 2017
- StarCraft II (real-time strategy games): DeepMind AlphaStar attained Grandmaster status in 2019, ranking the top 0.2 percent of human players