Intro to AI,
Autumn, 2025

Adversarial Search and Games

Faculty of DS & AI
Autumn semester, 2025

Trong-Nghia Nguyen

Business AI Lab

2025-09

Group task

- List of topic
- Select topic
- Write report & presentation:
  - Report template (LaTeX, word)
  - Presentation template
  - All submission file should be .pdf file
- Presentation about request lecture.

Content

- Game Theory
- Optimal Decisions in Games
- Heuristic Alpha-Beta Tree Search
- Monte-Carlo Tree Search

Game Theory

Two-player Zero-sum Games – chess, Go

A game can be formally defined as a kind of search problem:

- S0: The initial state, which specifies how the game is set up at the start.
- PLAYER(s): Defines 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, which defines the result of a move.
- TERMINAL-TEST(s): which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
- UTILITY(s, p): A utility function (objective or payoff), defines the final numeric value for a game that ends in terminal state s for a player p. In chess, the outcome is a win (1), loss (0), or draw (1/2).

Game Theory

Search Tree for Tic-Tac-Toe Game

- 5,478 states
- 362,880 terminal nodes

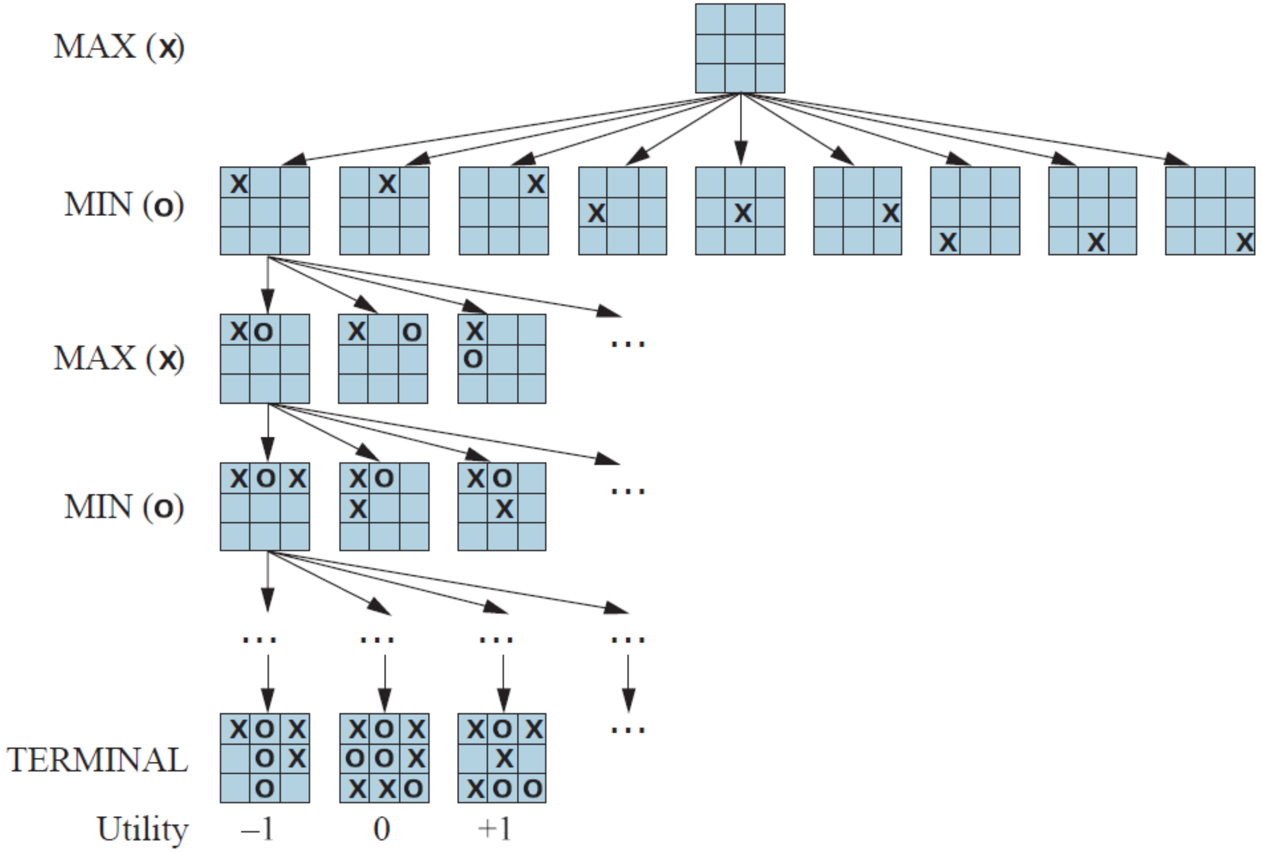


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

Optimal Decisions in Games

Minimax Search

- what is the best choice at root node, A?

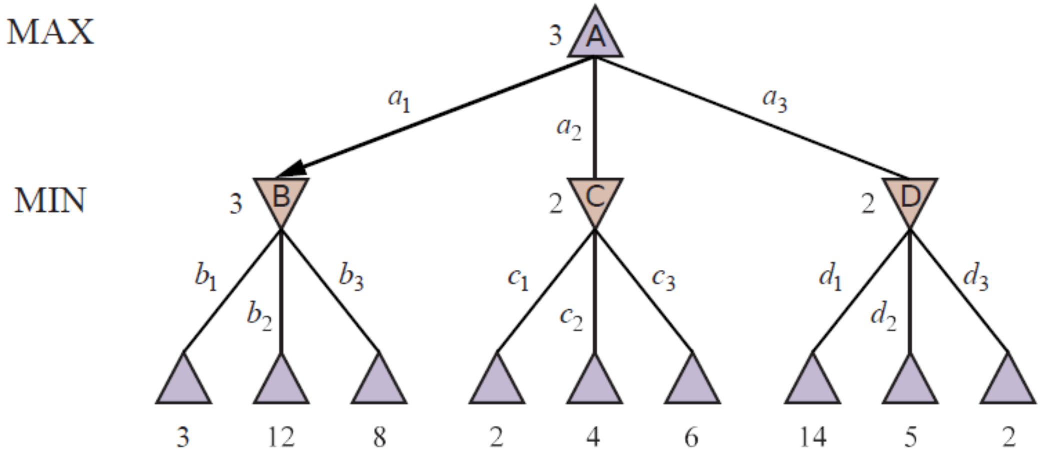


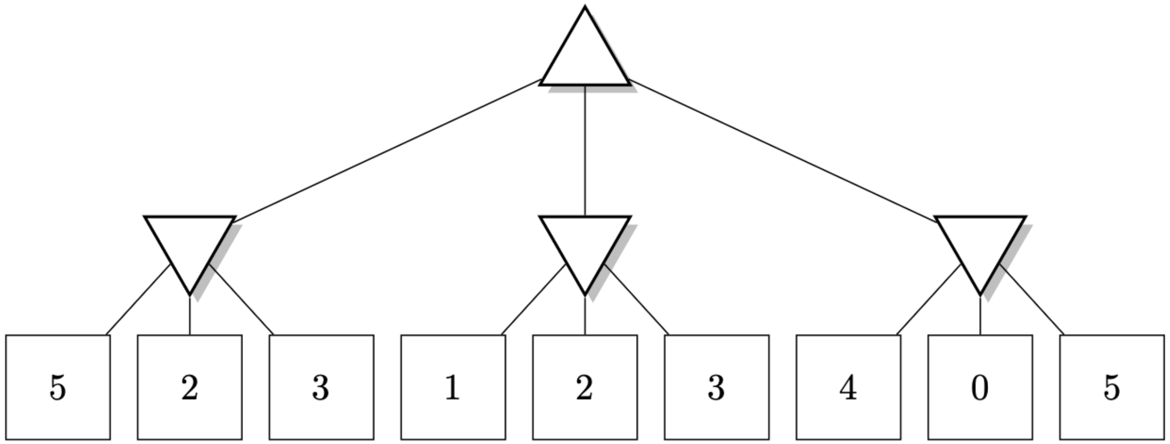
Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN's best reply is b_1, because it leads to the state with the lowest minimax value.

Optimal Decisions in Games

Minimax Search

Exercise 3.1

Consider the following tree representing a zero-sum game, where triangles pointing up are max nodes, triangles pointing down are min nodes, and squares are terminal nodes with the corresponding value of utility function for max player.



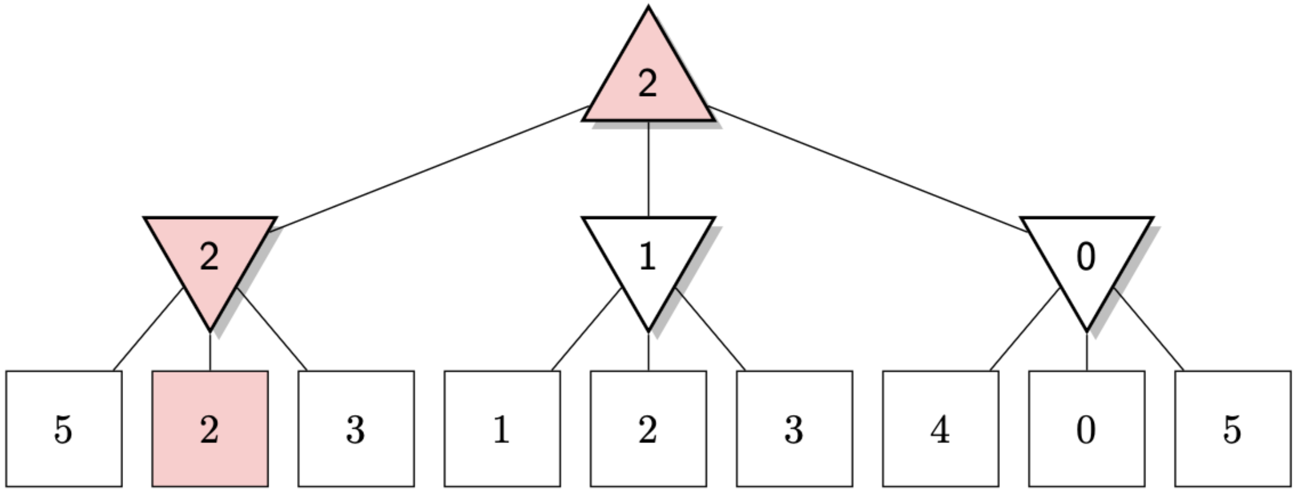
1. Apply the minimax algorithm for finding the best action for the max player at the root.
2. Apply the minimax algorithm with alpha-beta pruning for finding the best action for the max player at the root.

Optimal Decisions in Games

Minimax Search

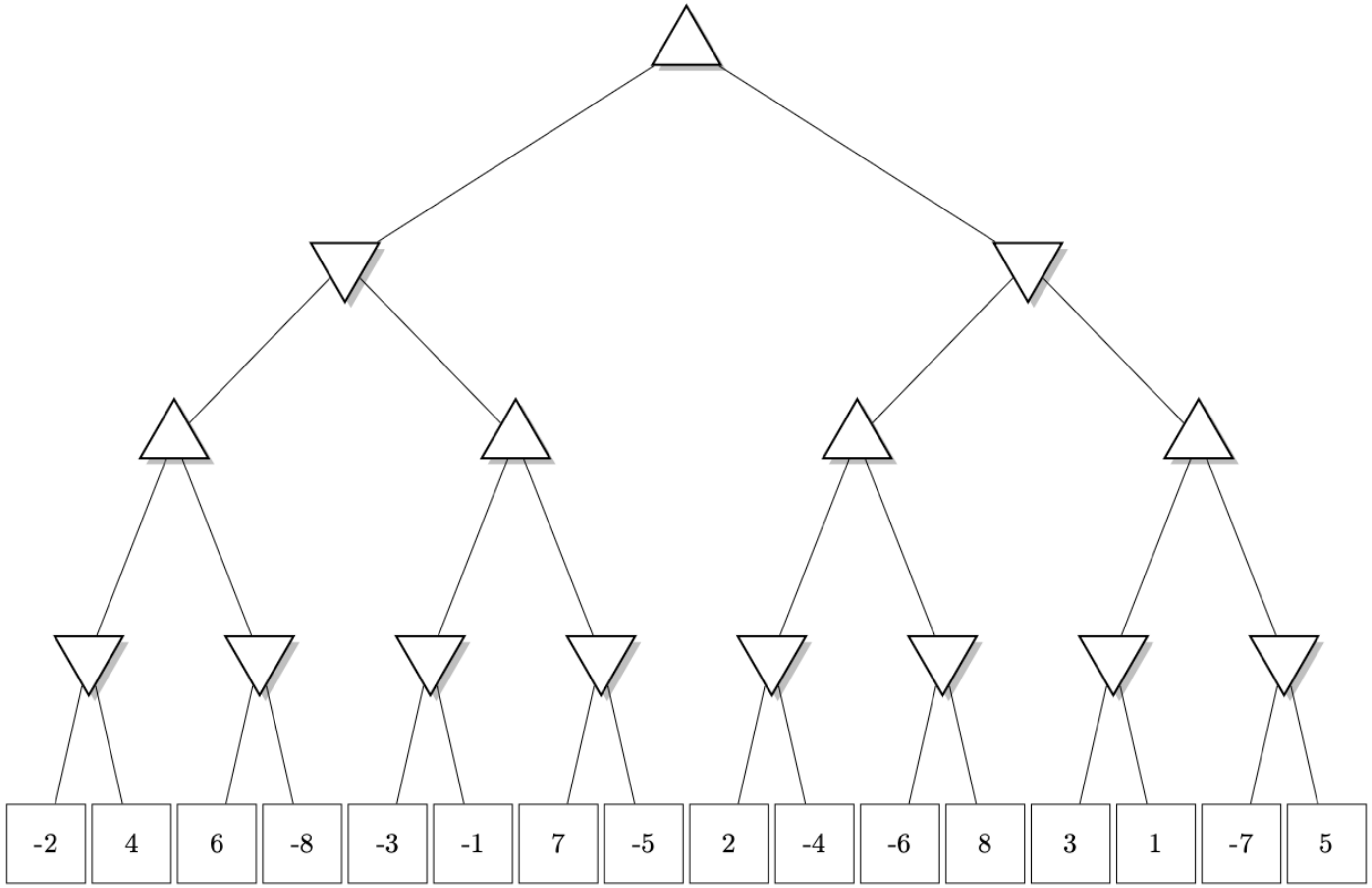
Answer of exercise 3.1

Minimax solution



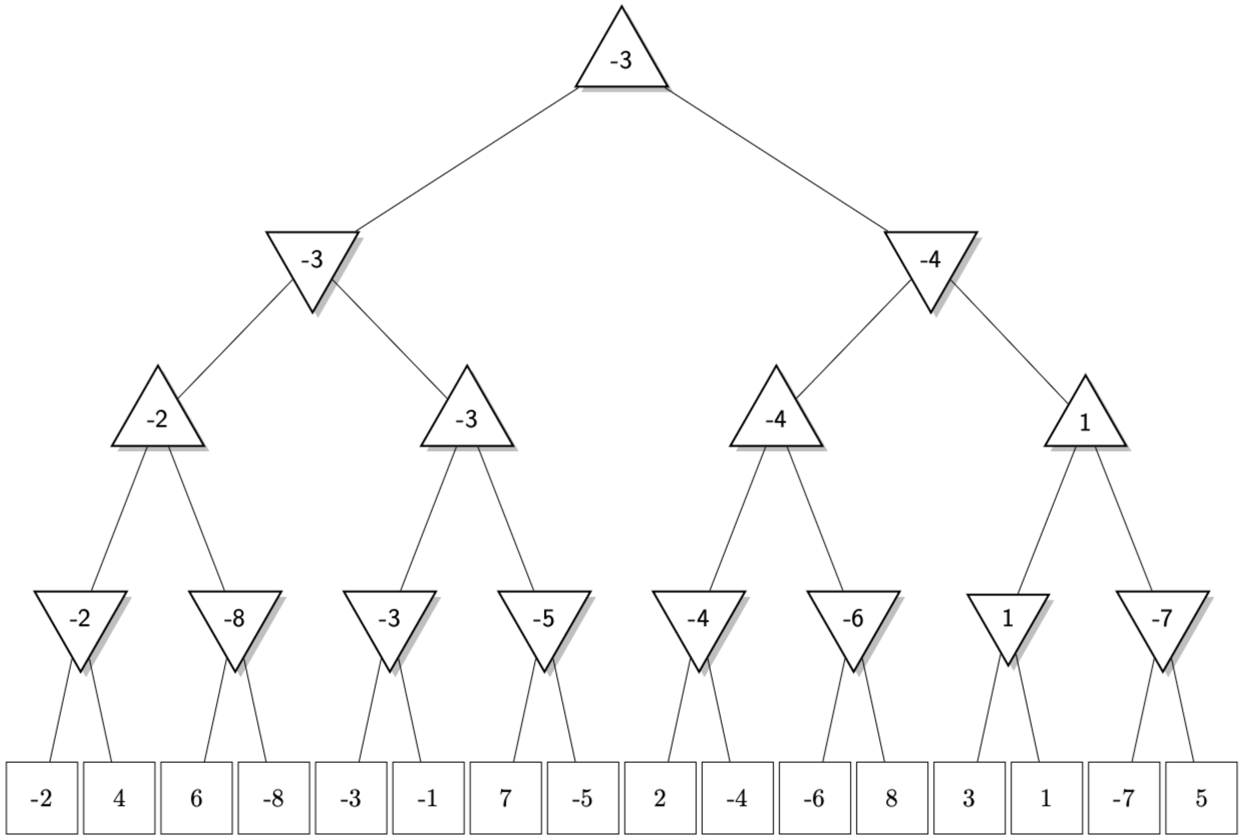
Optimal Decisions in Games

Minimax Search



Optimal Decisions in Games

Minimax Search

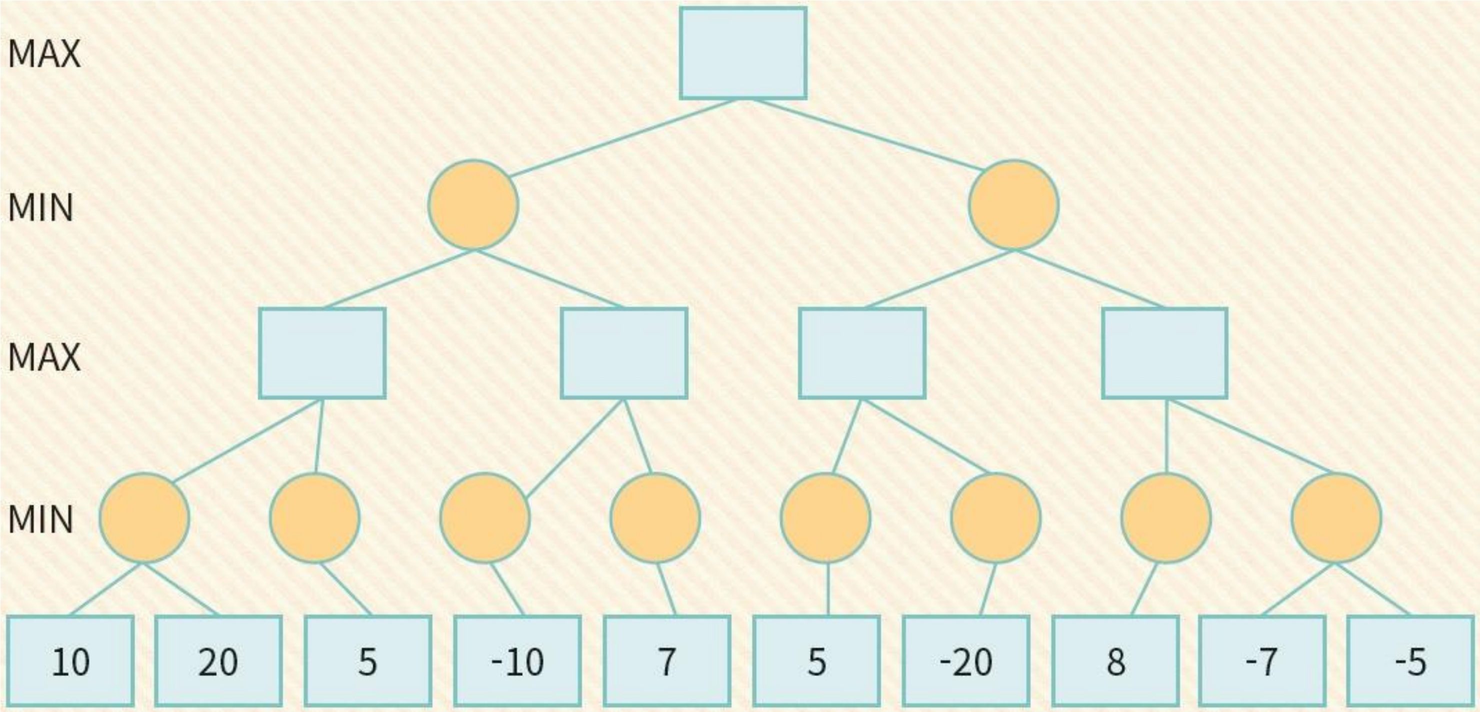


Minimax solution.

Optimal Decisions in Games

Minimax Search

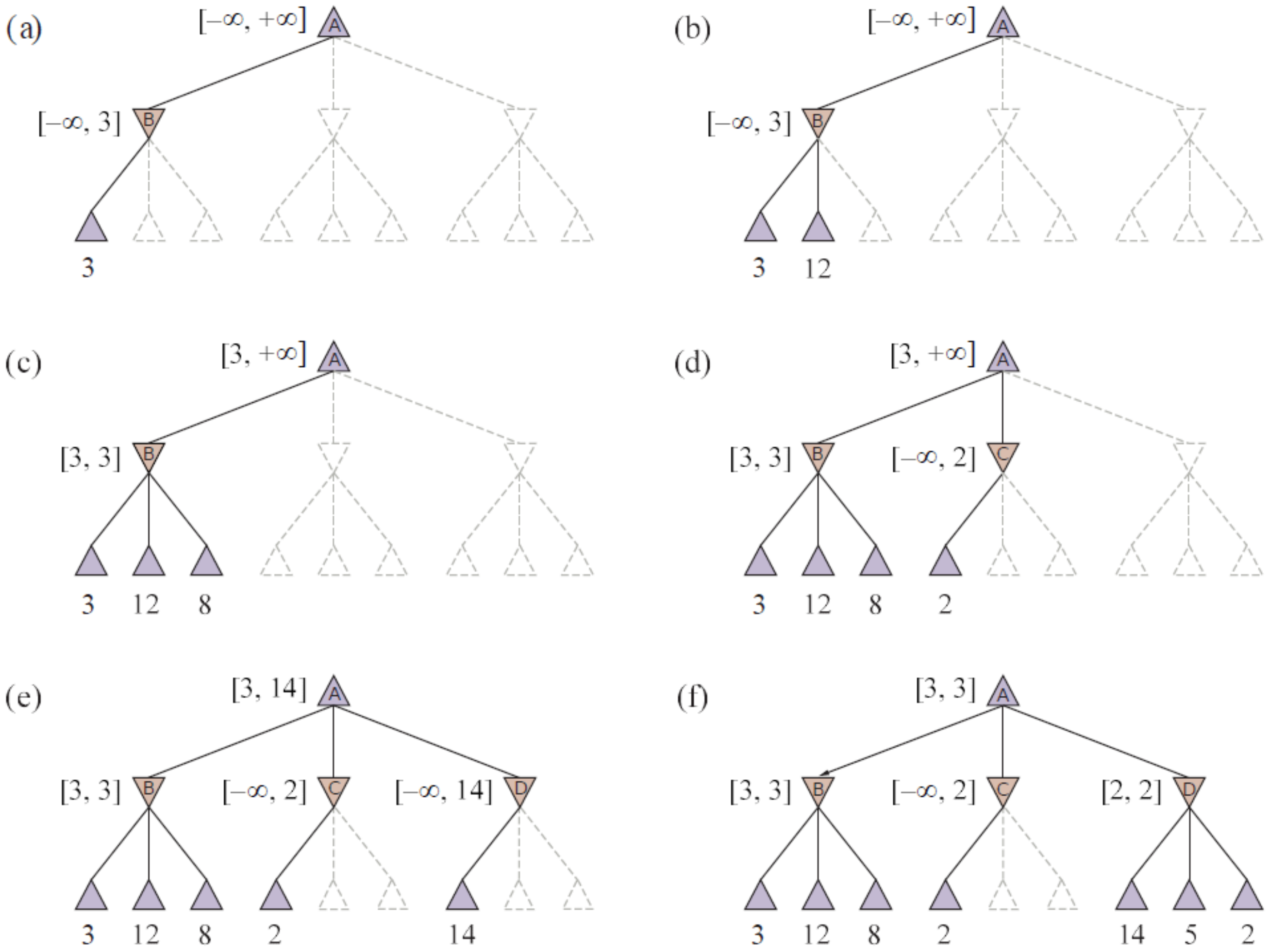
- what is the best choice at root node?



Optimal Decisions in Games

Alpha-Beta Pruning in Minimax Search

- alpha-pruning
- beta-pruning



Optimal Decisions in Games

Alpha-Beta Pruning in Minimax Search

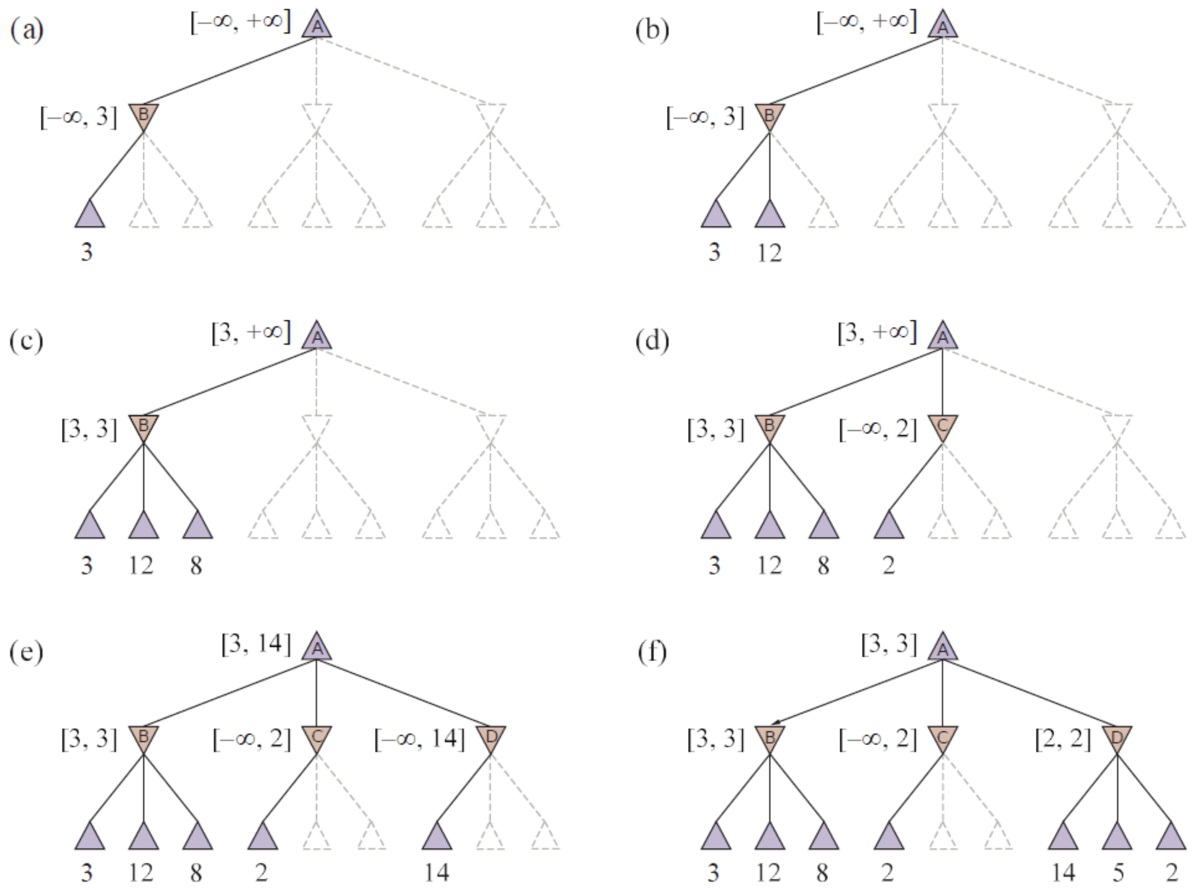


Figure 5.5 Stages in the calculation of the optimal decision for the game tree. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Optimal Decisions in Games

Alpha-Beta Pruning in Minimax Search

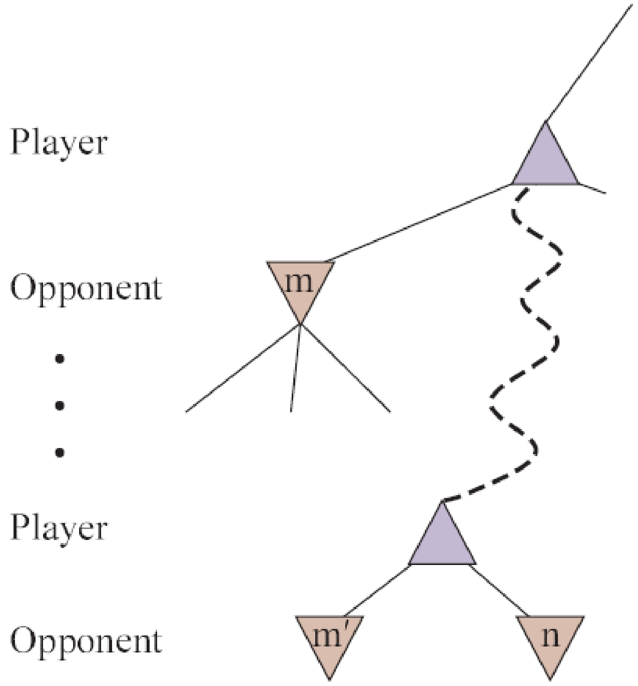
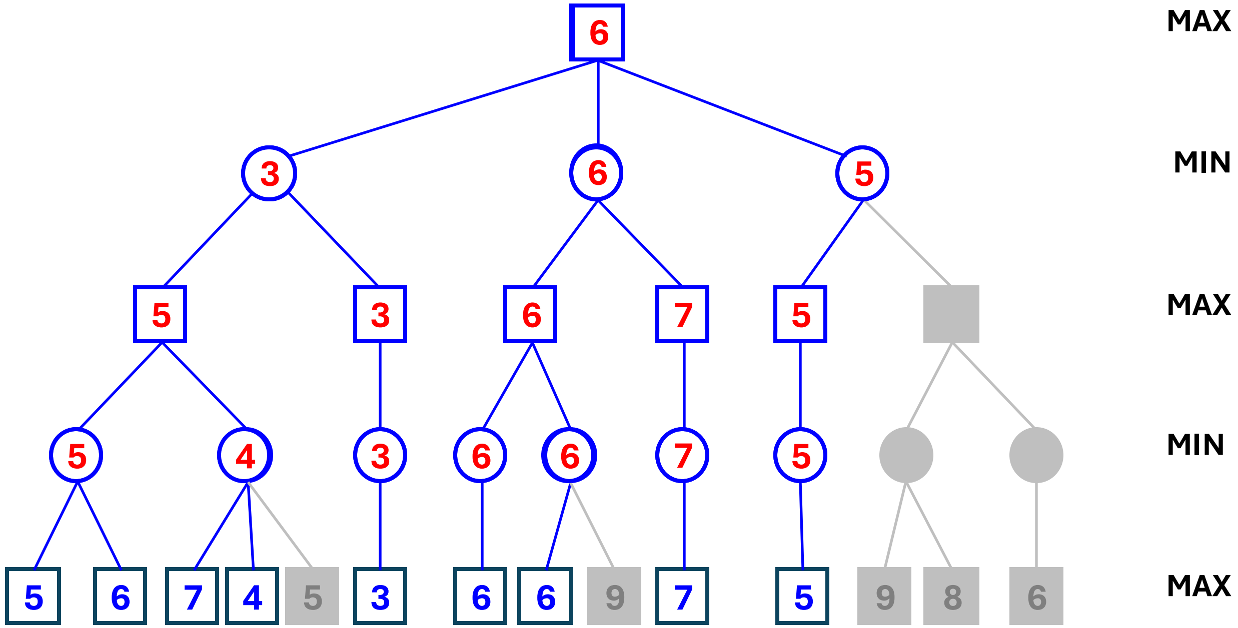


Figure 5.6 The general case for alpha–beta pruning. If m or m' is better than n for Player, we will never get to n in play.

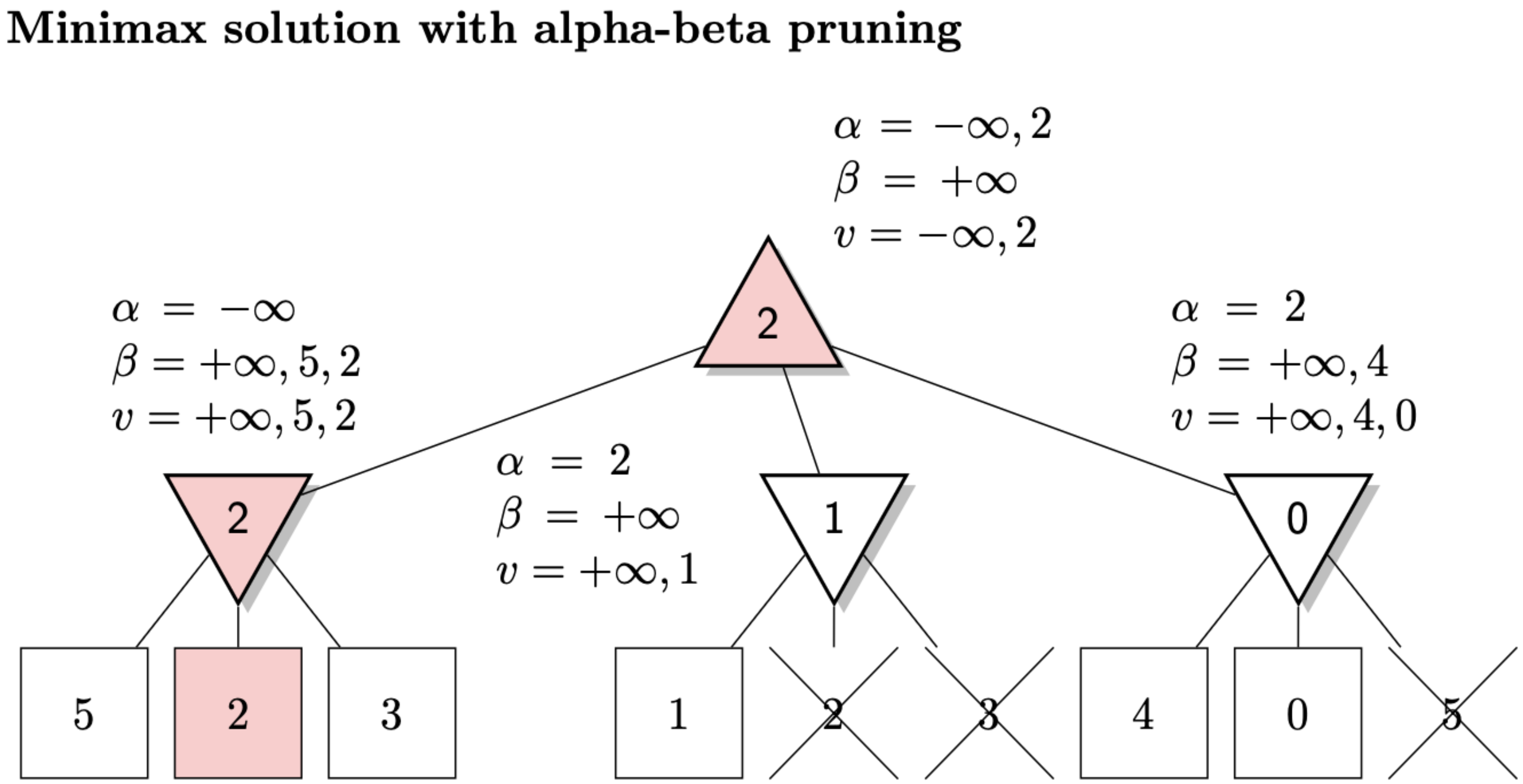
Optimal Decisions in Games

Alpha-Beta Pruning in Minimax Search



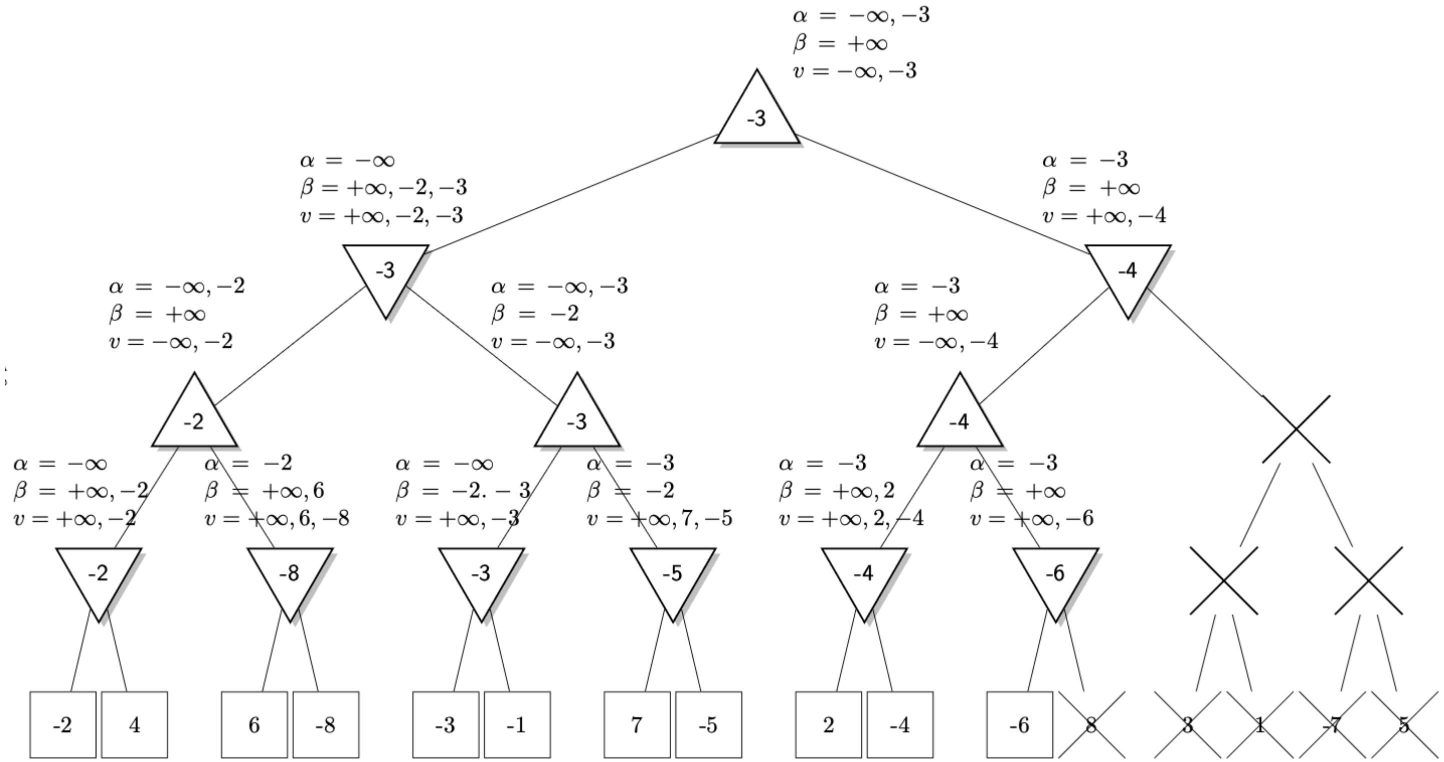
Optimal Decisions in Games

Alpha-Beta Pruning in Minimax Search



Optimal Decisions in Games

Alpha-Beta Pruning in Minimax Search



Heuristic Alpha-Beta Tree Search

Heuristic Evaluation Function, EVAL

- 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

- EVAL(s, p) estimates the Utility of state s to player p

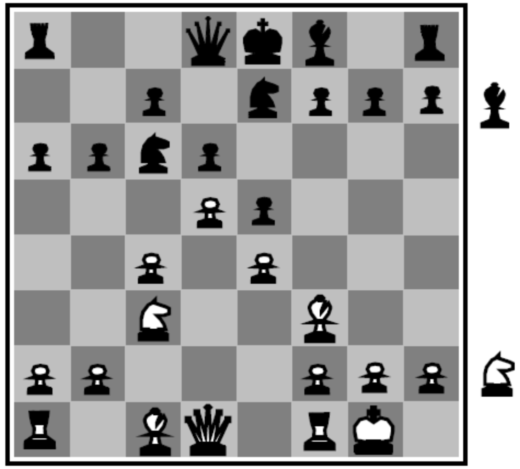
- Utility(loss, p) ≤ EVAL(s, p) ≤ Utility(win, p)

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

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

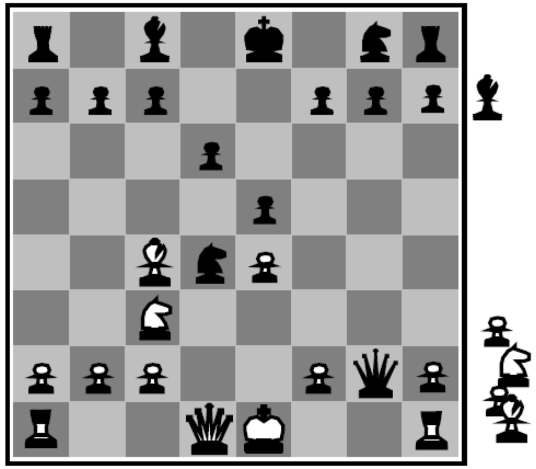
Heuristic Alpha-Beta Tree Search

Heuristic Evaluation Function – an example



Black to move

White slightly better



White to move

Black winning

Equivalent classes of states: two-pawn/one-pawn. $Eval(s) = expected\ value$

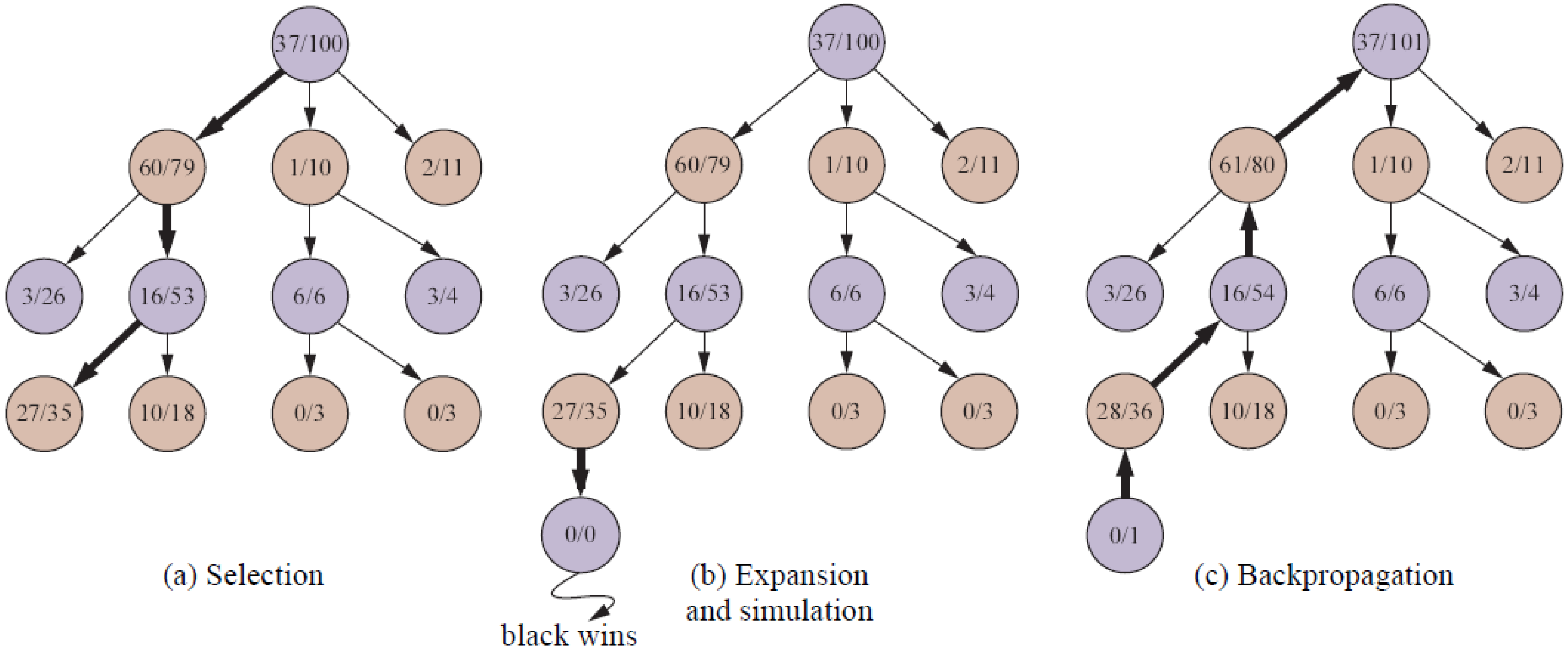
Linear weighted sum of features $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$, e.g., $w_1 = 9$ with $f_1(s) =$ (no of white Q) – (no of black Q), etc.

Assumes that contribution of each feature is independent of the values of the other features

Monte Carlo Tree Search

Four Steps of Selection – Expansion – Simulation - Back-propagation

- after the tree construction, decide the next move

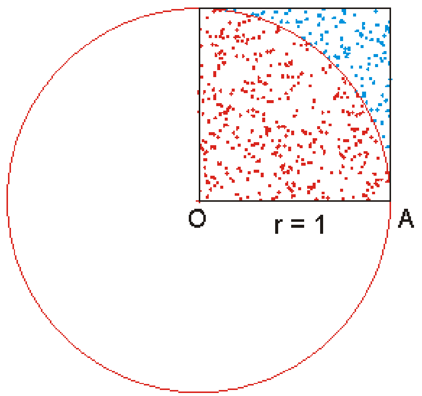


(a) Selection (b) Expansion and simulation (c) Backpropagation

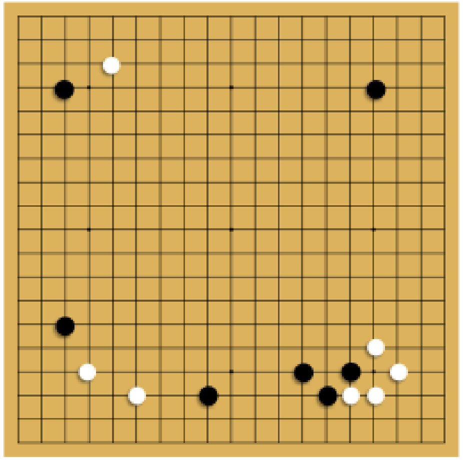
Monte Carlo Tree Search

Monte Carlo Simulation

- random sampling from a particular probability distribution
- check the result
- simulate via a repetition of the two steps



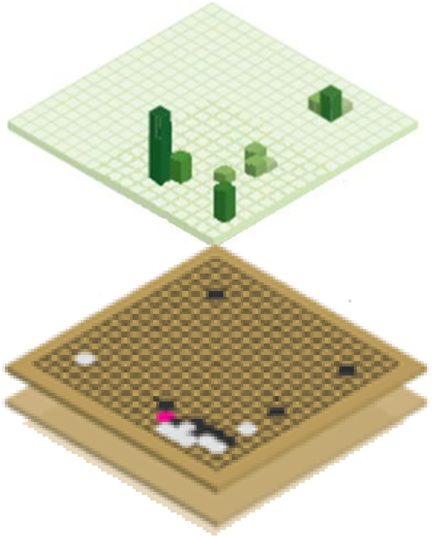
Area of the quarter circle

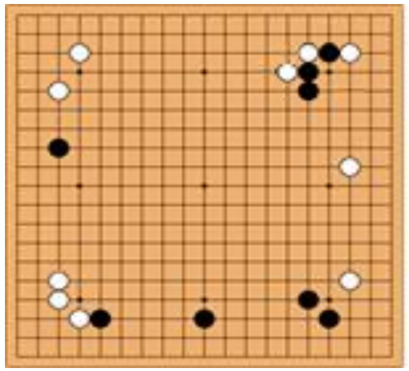


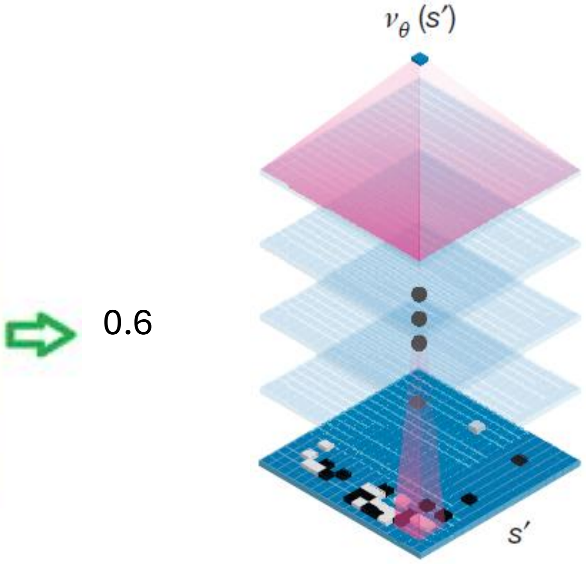
Utility value of a state, EVAL(s)

Monte Carlo Tree Search

MCTS in AlphaGo







Policy Network
for the next movement

Value Network
for Utility estimation

Thank you!

You're now ready to explore the exciting world of AI!