Artificial Intelligence

Adversarial Search

Motivation

- Kasparov vs Deep Blue
- Deep Blue wins by 3 wins, 1 loss and 2 draws in 1997

- AlphaGo by Google Deep Mind

Adversarial Search: To find the optimal move or strategy for a player in a two-player game where each player is trying to maximize their chances of winning. In adversarial search, the algorithm assumes that the opponent is also playing optimally, and therefore, it tries to find the best possible move that will maximize the player's chances of winning, given the opponent's best possible move.

Game Playing

- Game Playing is a search problem defined by:
  - Initial State: board configuration of chess
  - Successor Function:
    - Player(s): two players A & B;
    - Actions: Legal moves in a state
    - Results: It defines the result of move
  - Terminal-Test: End of Game?

  Terminal test, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
  - Utility: Objective Function: it defines final numeric value for a game that ends in terminal state. E.g. win (+1), lose (-1) and draw (0) in chess

- Initial State, Actions and Results function define the game tree for a game.
  - Nodes are state
  - Edges are moves

Perfect Information (Zero-Sum) Game

- AI Games
  - Perfect Information
    - Deterministic
    - Fully Observable
  - Zero Sum Game
    - Utility Functions of each player at the end of the game are EQUAL and OPPOSITE
      - e.g, WINNER (1) , LOSER (-1) or multi-valued utility values
  - Giving rise to adversary … agents need to maximize their utility values

Zero Sum: any gain by one player is exactly balanced by a loss by the other player or players. In other words, the total payoff is constant and any increase in one player's payoff is necessarily accompanied by a decrease in the other player's payoff.

Games vs. search problems

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

Game tree (2-player, deterministic, turns)

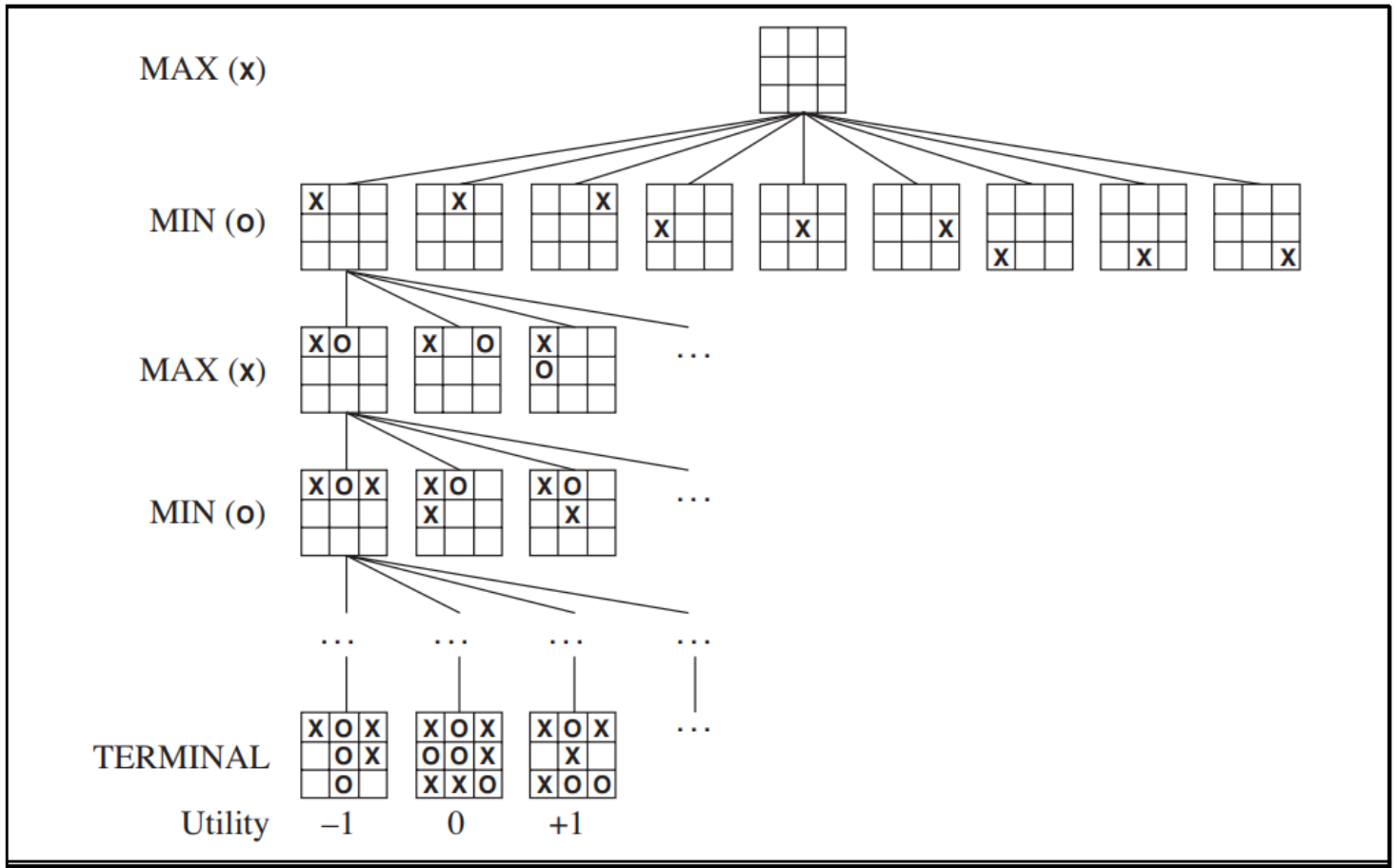

Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities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

Optimal Decisions in Games

- Search problem formulation for MINMAX
  - Initial State / Successor Function / Terminal State / Utility Fn
- Explanation
  - MAX starts first
  - Play alternates b/w MAX (places X) & MIN (places O)
  - Terminal State (WIN/LOSS/DRAW)
  - Number each leaf node w.r.t MAX
    - High values are GOOD for MAX, BAD for MIN
    - It is MAX's job to use search-tree to determine "best move"
- Normal Search solution vs. Game's Solution
    - Normal sequence of moves leading to goal vs. MIN has a say
    - Requires contingent OPTIMAL strategy (in response to MIN's move)

Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value=best achievable payoff against best play
- Back-tracking Algorithm
- Use DFS for searching
- Why not BFS?

Minimax

- Minimax Value=
  - Utility (n) if n is a terminal state
  - Max(n) set of successors if n is a MAX node
  - Min(n) set of successors if n is a MIN node

MiniMax “Search”

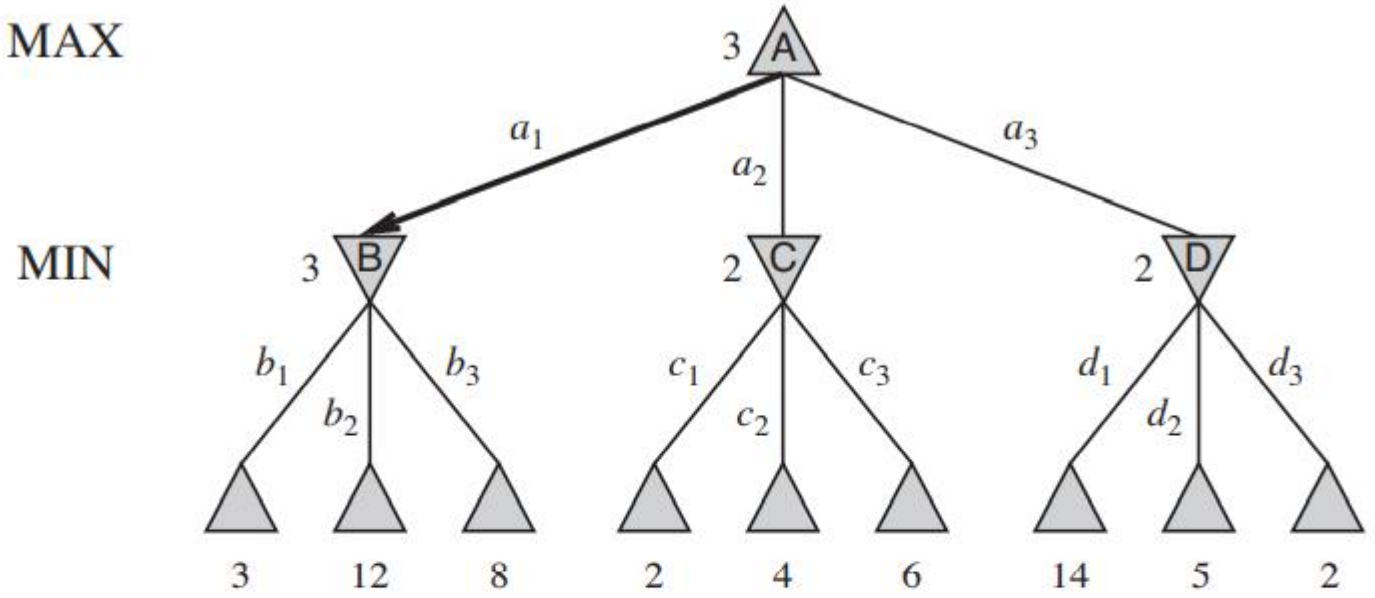

Figure 5.2 A two-ply game tree. The △ nodes are “MAX nodes,” in which it is MAX’s turn to move, and the ▽ nodes are “MIN nodes.” The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX’s best move at the root is a_1, because it leads to the state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

Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← 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 ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\text{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^d)$
- Space complexity? O(bd) (depth-first exploration)
- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

MiniMax (Multiplayer)

Multiplayer games usually involve alliances, whether formal or informal, among the players. Alliances are made and broken as the game proceeds

suppose A and B are in weak positions and C is in a stronger position. Then it is often optimal for both A and B to attack C rather than each other, lest C destroy each of them individually

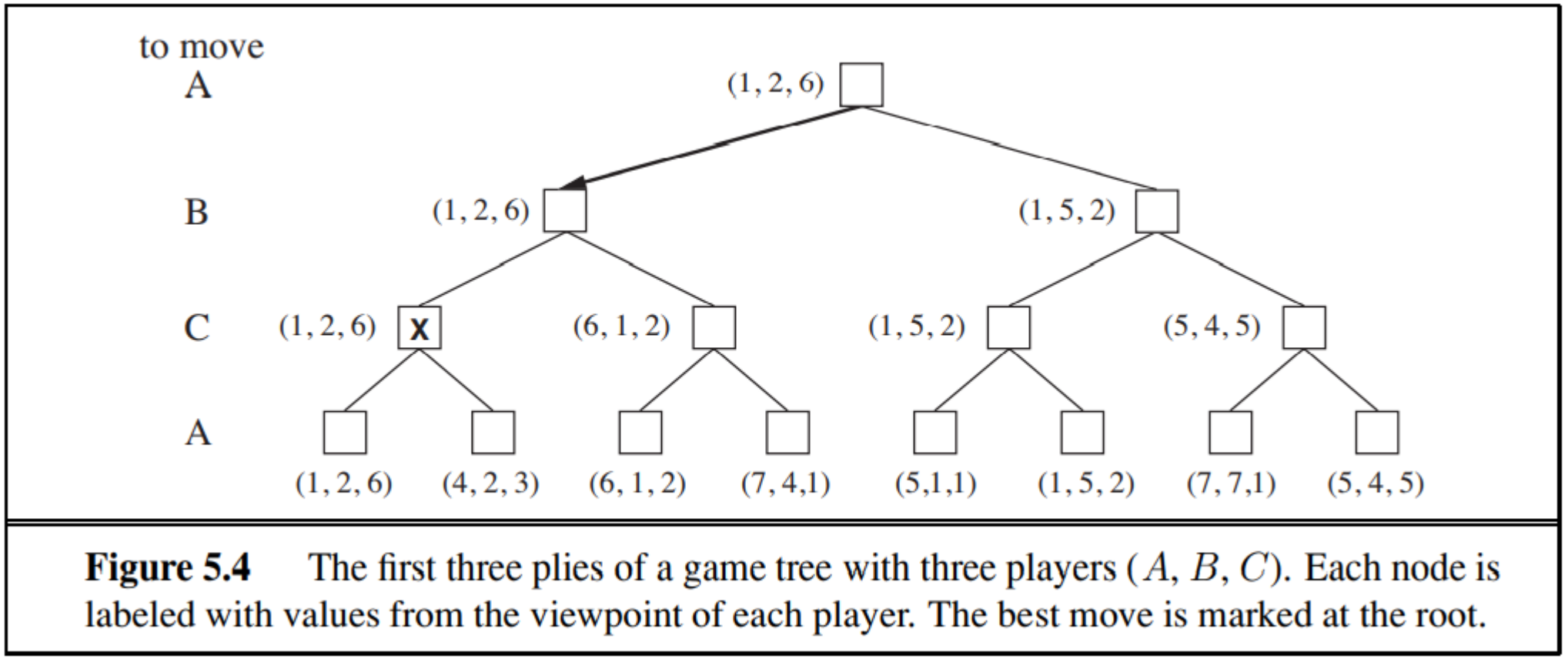

Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.