Adversarial Search

Lecture 4

Adversarial Search

Adversarial search is search when there is an "enemy" or "opponent" changing the state of the problem every step in a direction you do not want.

Examples: Chess, business, trading, war.

You change state, but then you don't control the next state.

Opponent will change the next state in a way:

1. unpredictable
2. *hostile* to you

Environment

- Multi-agent environment:
  - any given agent needs to consider the actions of other agents and how they affect its own welfare introduce possible contingencies into the agent's problem-solving process cooperative vs. competitive

Primary Assumptions:

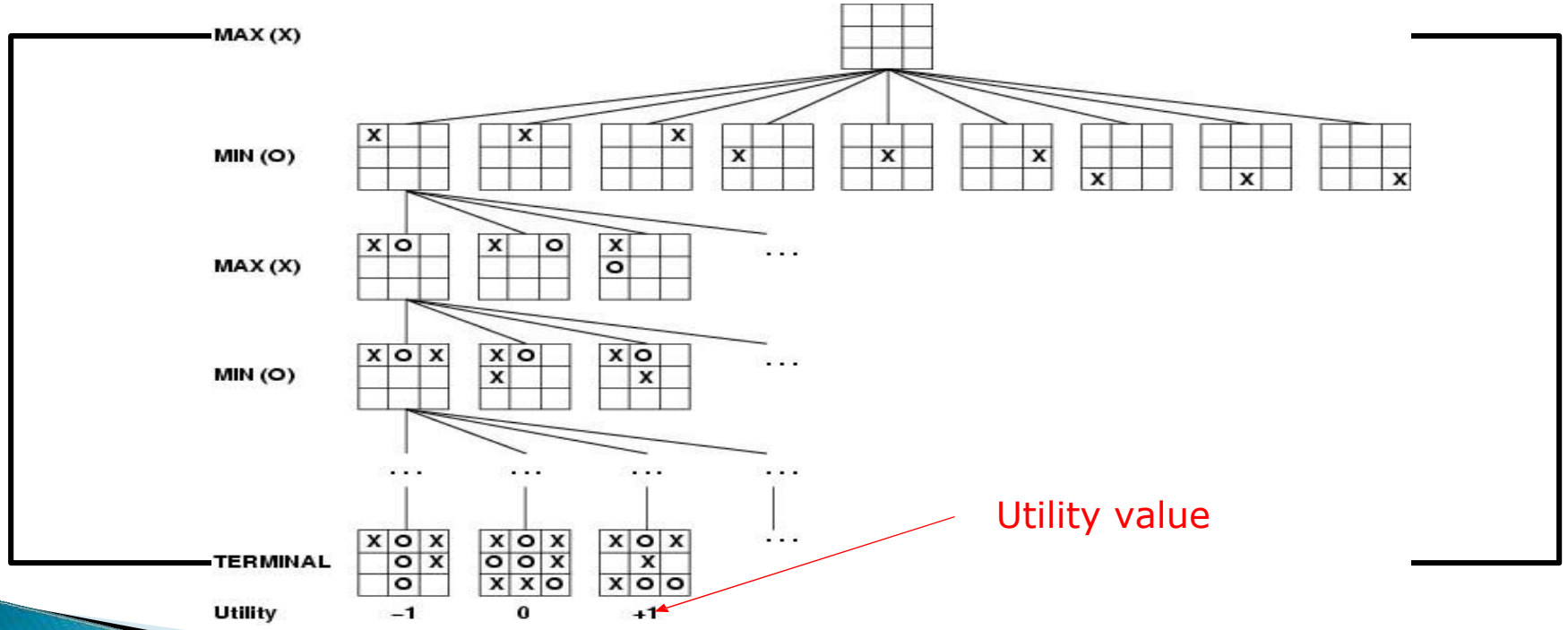
- "Game" in AI:
  - A multi-agent, non-cooperative environment
  - Zero Sum Result: describes a situation in which a participant's gain or loss is exactly balanced by the losses or gains of the other participant(s)
  - Turn Taking.
  - Deterministic.
  - *Two Player*

Game Problem Formulation

- A game with 2 players (MAX and MIN, MAX moves first, turn-taking) can be defined as a search problem with:
  - initial state: board position
  - player: player to move
  - successor function: a list of legal (move, state) pairs
  - goal test: whether the game is over – terminal states
  - utility function: gives a numeric value for the terminal states (win, loss, draw)

- Game tree = initial state + legal moves

Game Tree (2-player, deterministic)

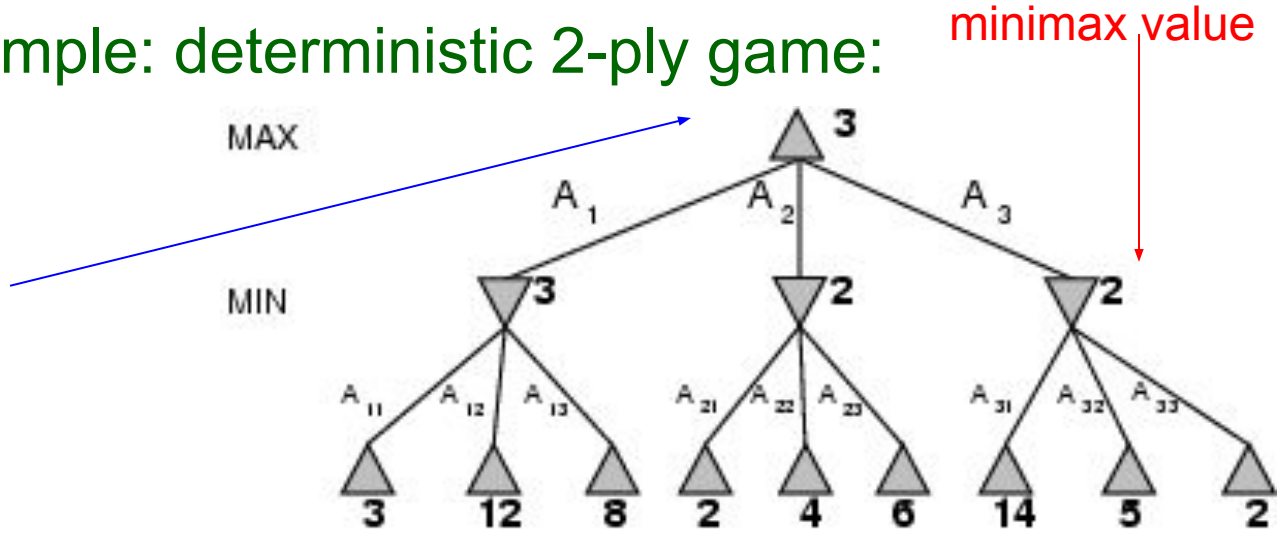


Minimax

- Idea: choose a move to a position with the highest minimax value = best achievable payoff against a rational opponent.

Minimax value is computed bottom up:
-Leaf values are given.
-3 is the best outcome for MIN in this branch.
-3 is the best outcome for MAX in this game.
-We explore this tree in depth-first manner.

Example: deterministic 2-ply game:



Properties of minimax

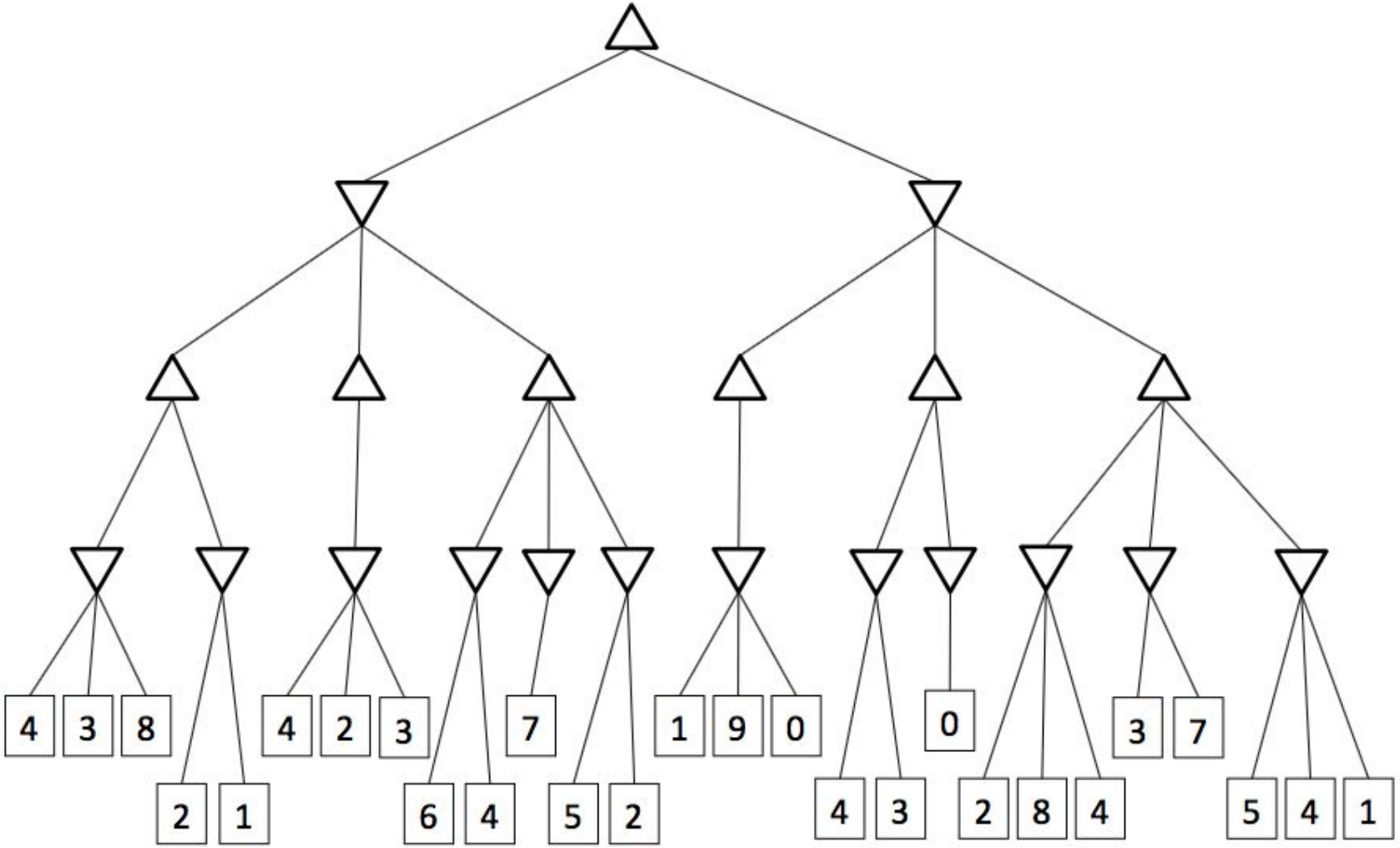
- Complete? Yes (if tree is finite)
- Optimal? Yes (against an rational opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)

Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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

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


4 3 8
2 1
4 2 3
6 4
7
5 2
1 9 0
4 3
0
2 8 4
3 7
5 4 1

Solution