Artificial Intelligence

Game Playing (Adversarial Search)

Lecture 7

CS 444 – Spring 2019
Dr. Kevin Molloy
Department of Computer Science
James Madison University

JMU JAMES MADISON UNIVERSITY®

Outline for Today

- Games vs Search Problems
- Perfect Play
  - Minimax Decision
  - Alpha-Beta Pruning
- Games of Imperfect Information
- Game Playing Summary

Game Playing – Adversarial Search

Search in a multi-agent, competitive environment

Mathematical game theory treats any multi-agent environment as a game, with possibly co-operative behaviors (study of economies)

	Deterministic	**chance**
Perfect information	Chess, checkers, go, othello	Backgammon, monopoly
Imperfect information	Batteship, blind tictactoe	Bridge, poker, scrabble

Most games studied in AI:

deterministic, turn-taking, two-player, zero-sum games of perfect information

Game Playing – Adversarial Search

deterministic, turn-taking, two-player, zero-sum games of perfect information

Zero-sum: utilities of the two players sum to 0 (no win-win)
Deterministic: precise rules with known outcomes
Perfect information: fully observable

Search algorithms designed for such games make use of interesting general techniques (meta-heuristics) such as evaluation functions, search pruning and more.

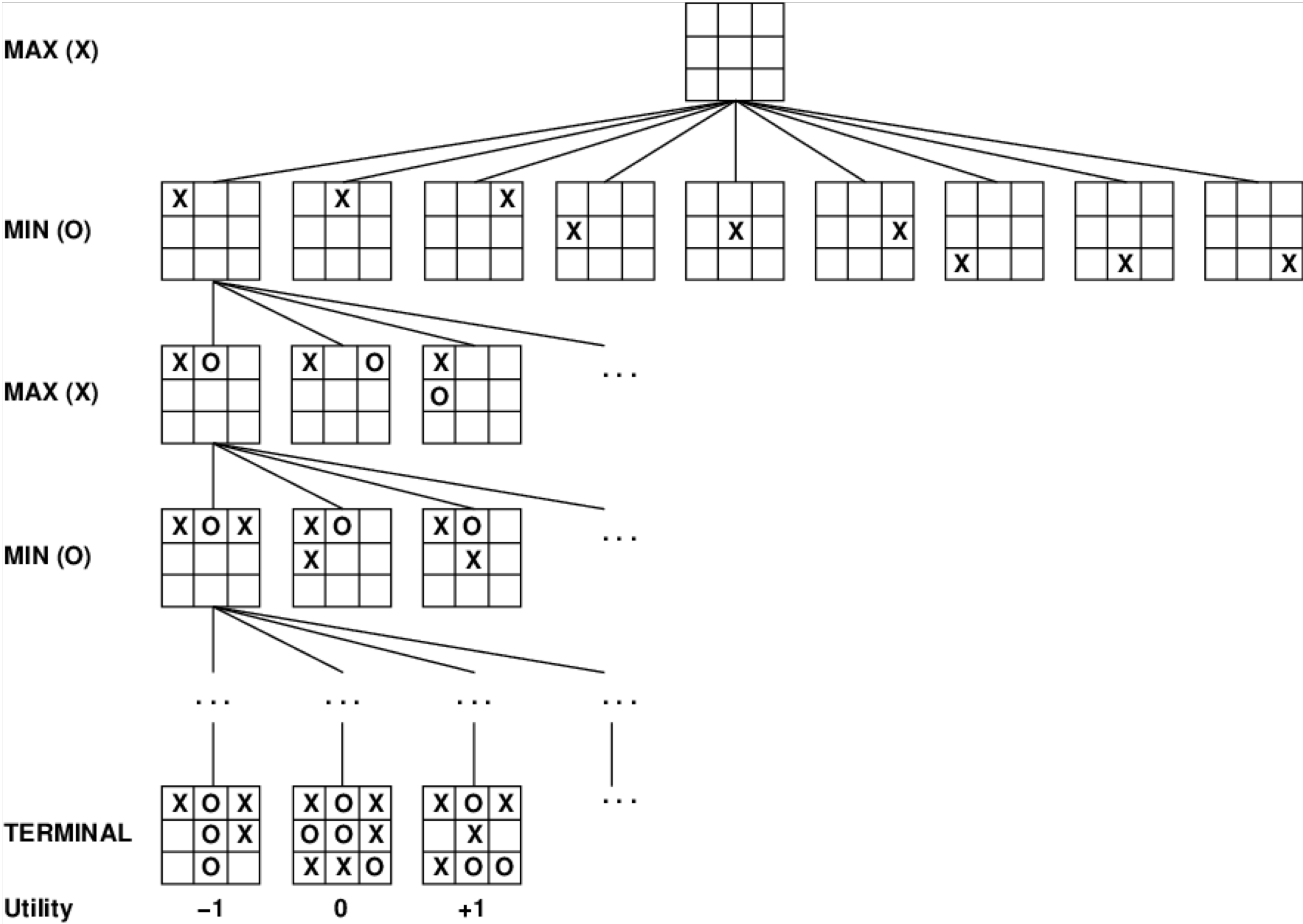
However, games are to AI what grand prix racing is to automobile design.

Our objective: study the three main adversarial search algorithms [minimax, alpha-beta pruning, and expectiminimax] and meta-heuristics they employ

Game Playing as a Search Problem

Two turn-taking agents in a zero-sum game: **Max** (starts game) and **Min**

Max's goal is to maximize it utility. Min's goal is to minimize Max's utility



JAMES MADISON UNIVERSITY

Game Playing as a Search Problem

Formal definition of a game as a search problem:

- S0 ← initial state that specifies how game starts
- PLAYER(s) ← which player has move in state *s*
- ACTIONS(s) ← returns set of legal moves in state *s*
- RESULT(s; a) ← transition model that denes result of an action a on a state s
- TERMINAL-TEST(s) ← true on states that are game enders, false otherwise
- UTILITY(s; p) ← utility/objective function denes numeric value for game that
- ends in terminal state s with player p

Concept of game/search tree valid here:

Chess: 35 moves per player → branching factor *b* = 35

Ends at typically 50 moves per player → *m* = 100

Search tree has $35^{100} \approx 10^{40}$ distinct nodes !

Game Playing as a Search Problem

Concept of game/search tree valid here:

Chess: 35 moves per player → branching factor b = 35

Ends at typically 50 moves → m = 100

Search tree has $35^{100} \approx 10^{40}$ distinct nodes !

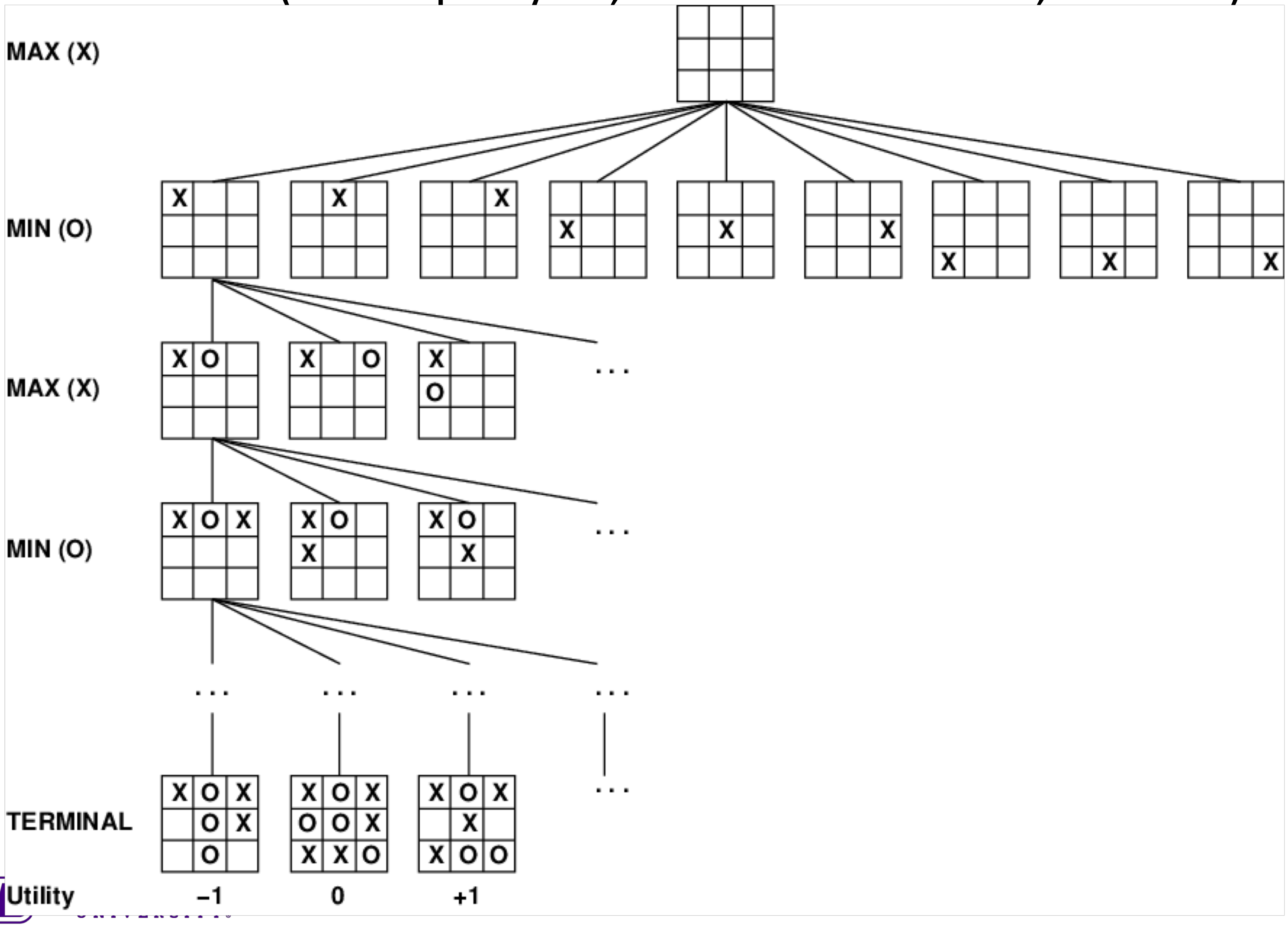
How to work with this?

- **Pruning**: how to ignore portions of the tree without impacting strategy
- **Evaluation function**: estimate utility of a state without a complete search

Some games have search trees that are too big:

- Time limits $\Longrightarrow$ unlikely to find goal, must approximate
- Many ”tricks” (meta-heuristics) employed to **look ahead**

Game Tree (two-player, deterministic, turns)

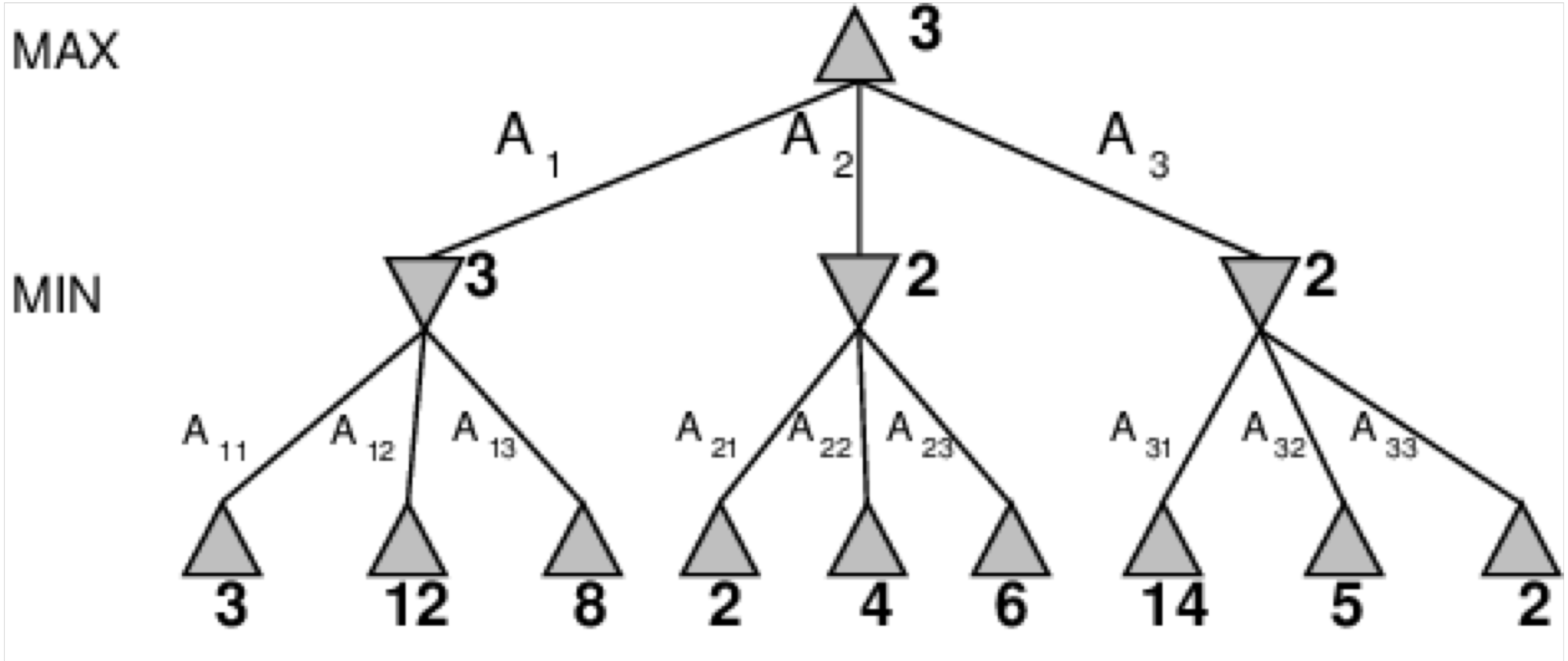


Minimax Decisions

Perfect play for deterministic, perfect-information games

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

Simple, 1 ply game



Minimax-Value Algorithm

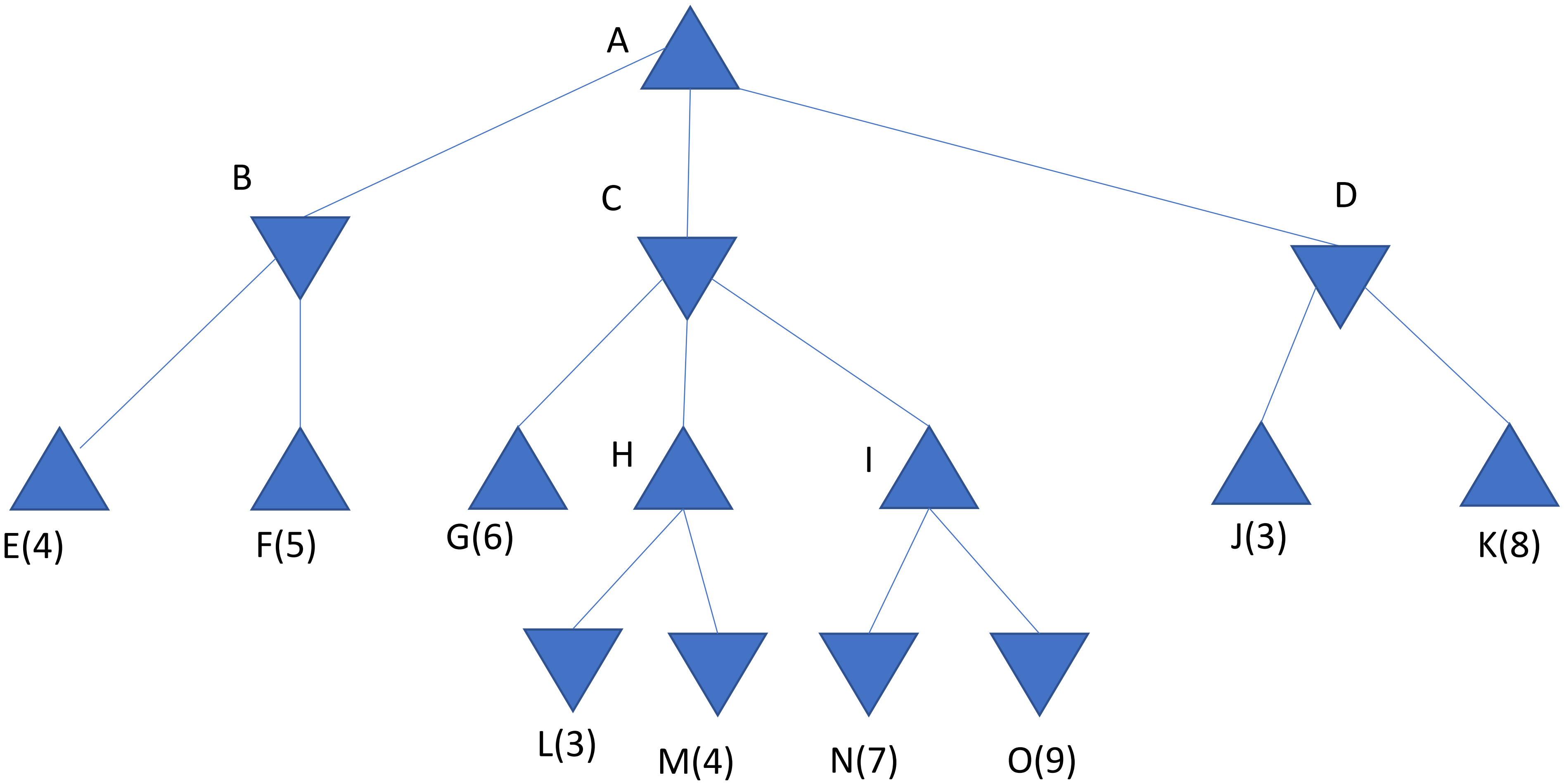
```
function Minimax-Value(state) returns minimax-value/utility
    if Terminal-Test(state) then return Utility(state)
    if NEXT-AGENT is MAX then return Max-VALUE(state)
    if NEXT-AGENT is MIN then return Min-Value(State)
```

```
function Max-Value(state) returns a utility value
    v ← - ∞
    for each successor of state
        v ← Max(v, MiniMax-Value(successor))
    return v
```

```
function Min-Value(state) returns a utility value
    v ← ∞
    for each successor of state
        Do v ← Min(v, MiniMax-Value(successor))
    return v
```

Tracing on the Board

Trace minimax-value on 2-ply game below updating your v's



Minimax Decision Algorithm

function Minimax-Decision(state) **returns** an action
    Return $\mathit{argmax}_{a \in actions}$ *Min-VALUE (RESULT(state,a)*

function Max-Value(state) **returns** a utility value
    If Terminal-Test(state) then return Utility(state)
    v ← *- ∞*
    for 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 a in ACTIONS(state)
        Do v ← Min(v, MAX-Value(RESULT(s,a)))
    return v

Properties of Minimax

Complete? Yes, if the tree is finite (chess has specific rules for this)

Optimal? Yes, against an optimal opponent. Otherwise?

Otherwise, even better Example?

Time complexity? $O(b^m)$

Space complexity? $O(bm)$ (depth-first exploration)

Chess: $b \approx 35$ $m \approx 100$. Exact solution completely infeasible

Do we need to explore every path?

Game Trees

In realistic games, cannot explore the full game tree.

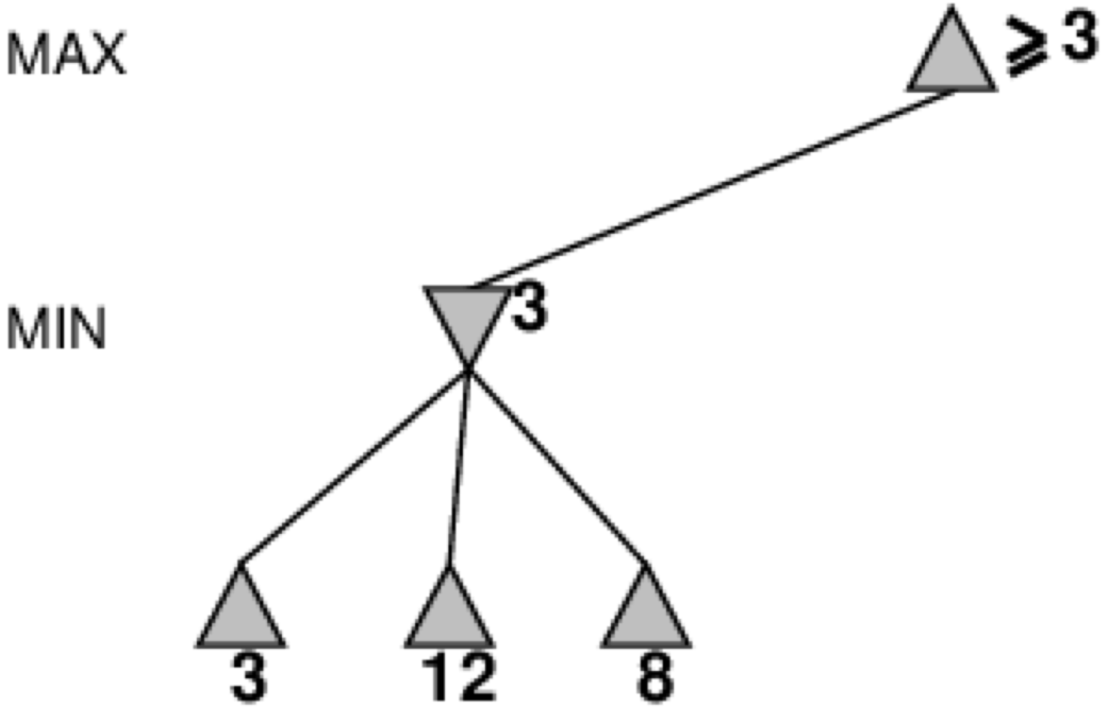
Number of game states MiniMax explores is exponential in the depth of the tree.

What to do?

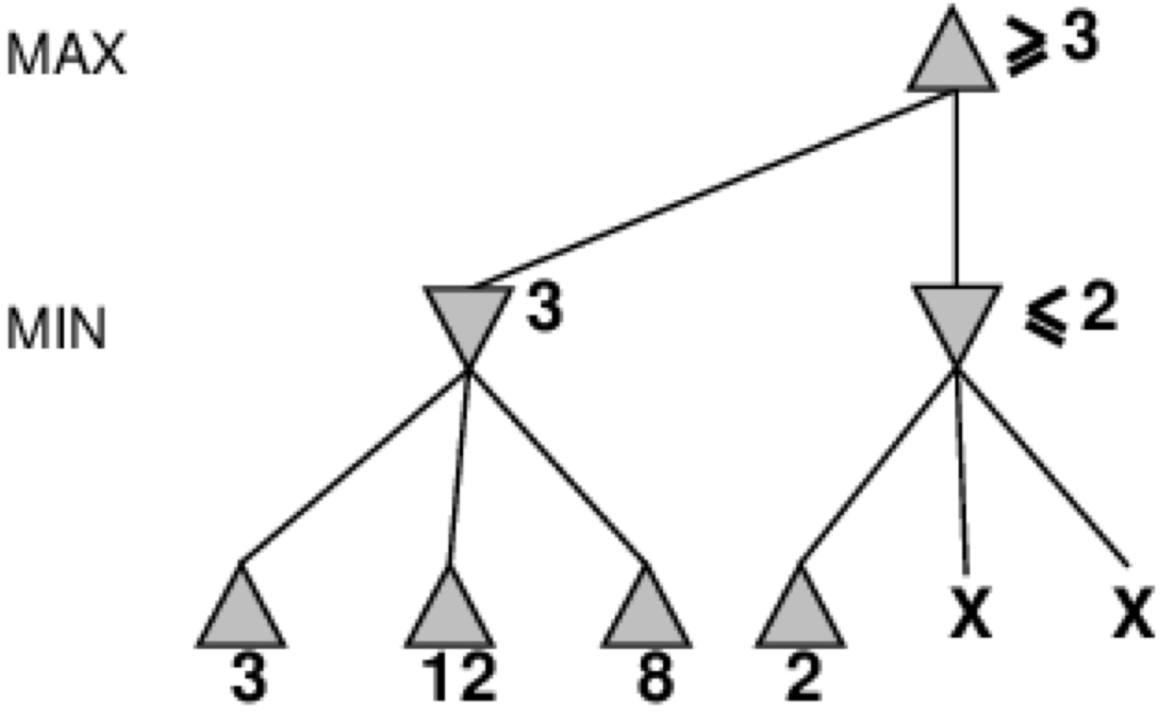
Remove from consideration entire subtrees (pruning)

Find a way not to have to reach the leaves to determine the value of a state

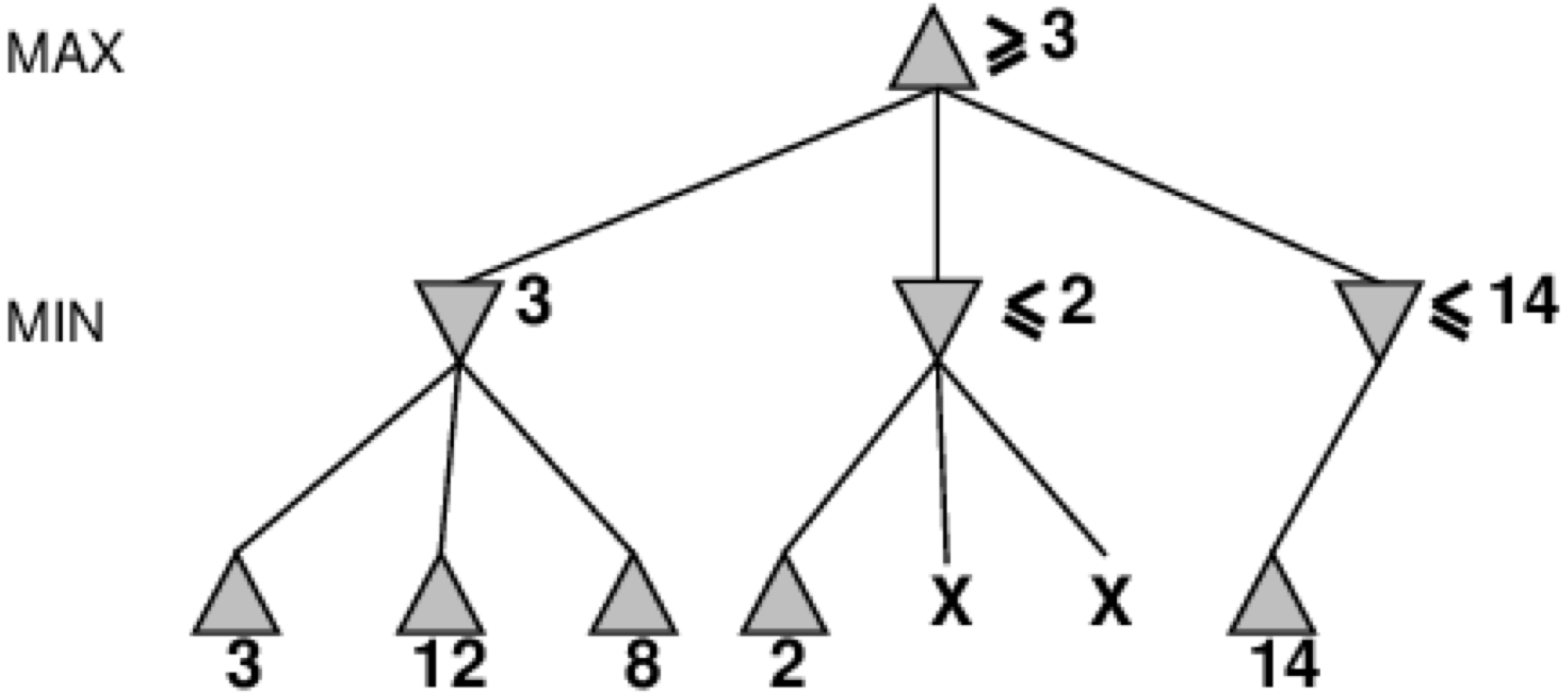
Remove from Consideration Entire Subtrees -- α - β Pruning Example



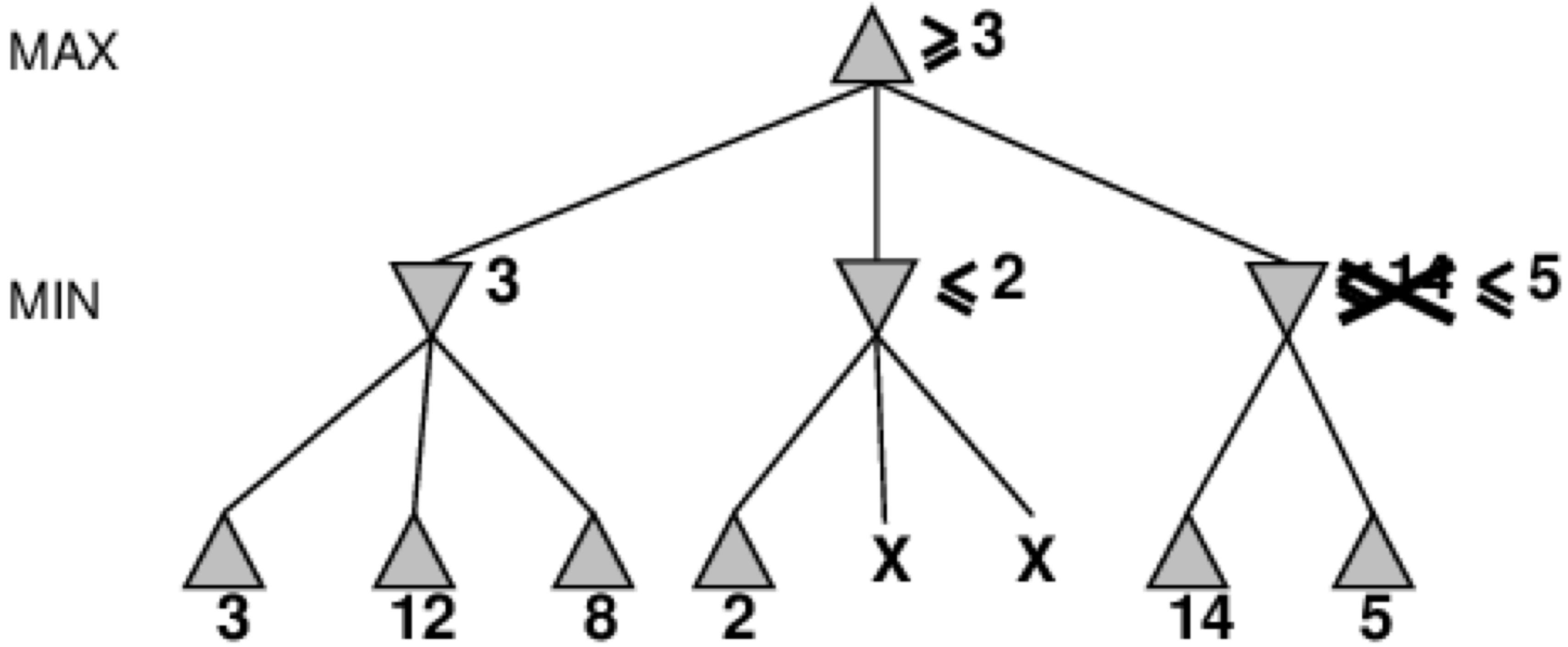
Remove from Consideration Entire Subtrees -- α - β Pruning Example



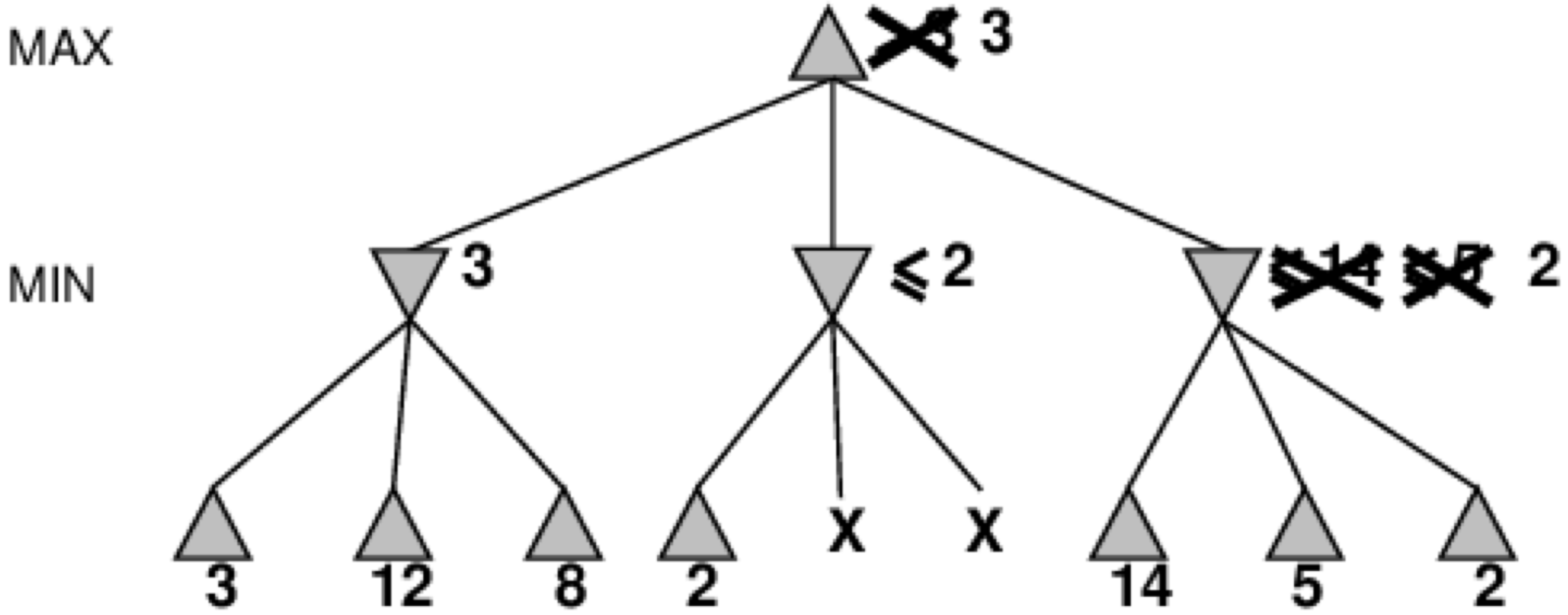
Remove from Consideration Entire Subtrees -- α - β Pruning Example



Remove from Consideration Entire Subtrees -- α - β Pruning Example



Remove from Consideration Entire Subtrees -- α - β Pruning Example

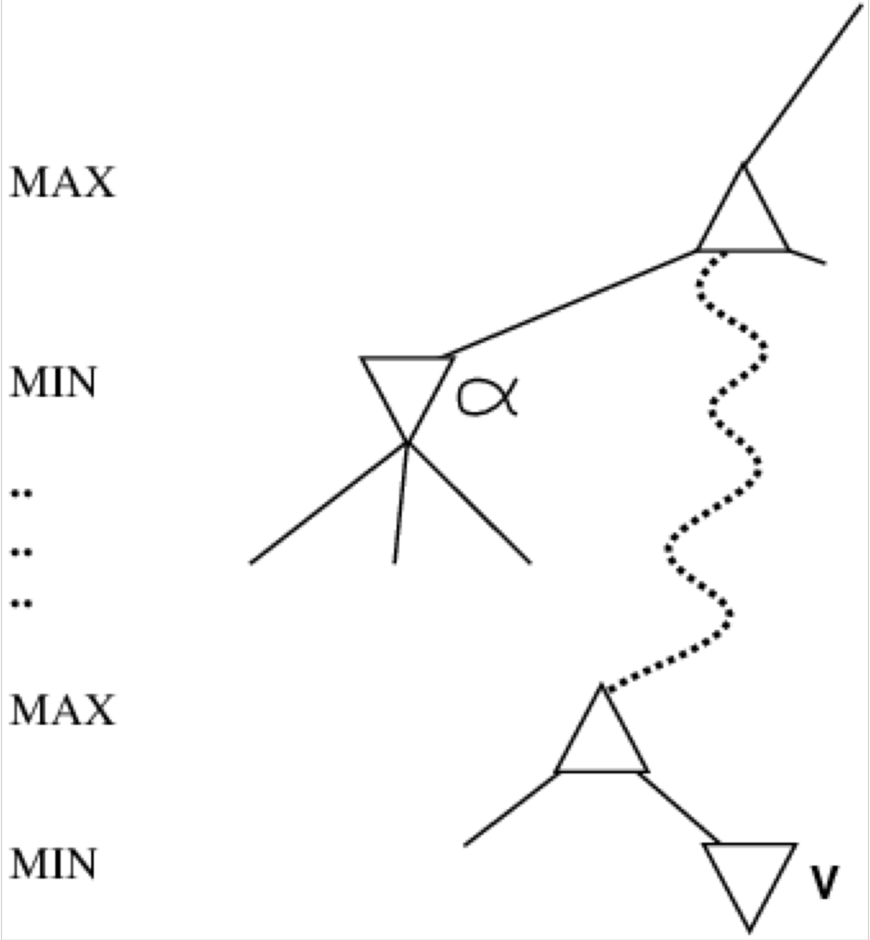


α - β Pruning Example

α is the best value (to MAX) found so far off the current path.

If V is worse than α, MAX will avoid it $\Longrightarrow$ prune that branch.

Similarity, β for MIN



α : MAX's best option on path to root

β: MIN's best option on path to root

Pruning by Maintaining α and β

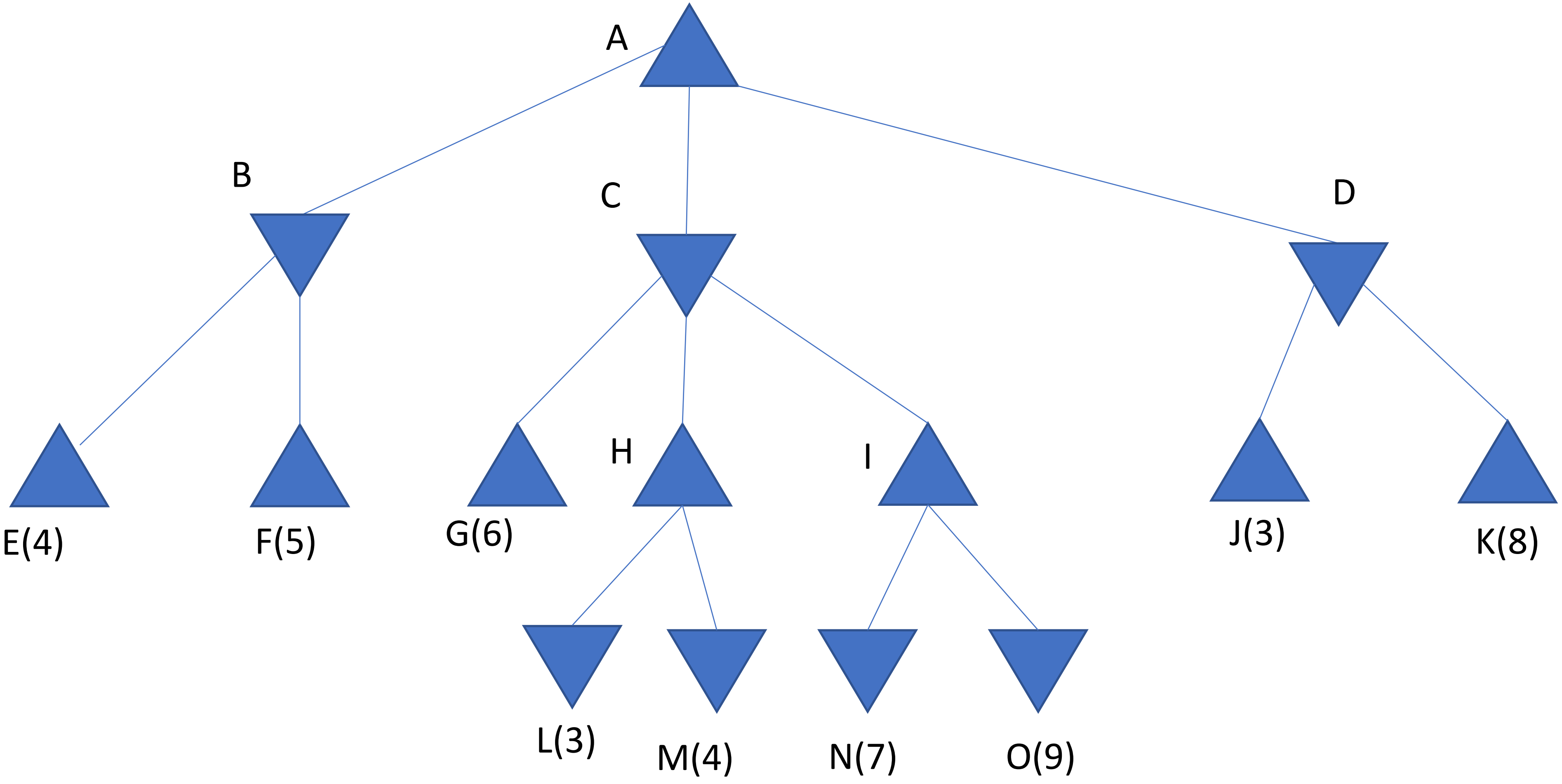
function Alpha-Beta-Value(state, α, β) **returns** value/utility
If Terminal-Test(state) then return Utility(state)
If NEXT AGENT is MAX then return MAX-VALUE(state, α, β)
If NEXT AGENT is MIN then return Min-Value(State α, β)

function Max-Value(state α, β) **returns** a utility value
v ← - ∞
for each successor of state
v ← Max(v, Alpha-Beta-Value(successor α, β))
if v ≥ β then return v
α ← MAX(α, v)
return v

function Min-Value(state, α, β) **returns** a utility value
v ← ∞
for each successor of state
v ← Alpha-Beta-Value(successor α, β))
If v ≤ α then **return** v
β ← MIN(β, v)
return v

Tracing on the Board

Trace alpha/beta on game below



Properties of α -β

Complete? Yes, if the tree is finite

Optimal? Yes, although intermediate nodes may have wrong values when subtrees are pruned

Time complexity? $O(b^{m/2}) \Longrightarrow$ doubles solvable depth; with "perfect ordering"

With random ordering, time complexity $\approx O(b^{3m/4})$

Unfortunately, 35^{50} or chess is still impossible.

Games of Imperfect Information

E.g., card games, where opponent's initial cards are unknown

Typically, we can calculate a probability for each possible deal

Seems just like having one dig dice roal at the beginning of the game

Idea:

Compute the minimax value of each action in each deal

Special case: if an action is optimal for all deals, it's optimal

GIB (best bridge program) approximates this idea by:

1. Generating 100 deals consistent with bidding information
2. Picking the action that wins most tricks on average

Game Playing Summary

Games are fun to work on! (and dangerously obsessive)

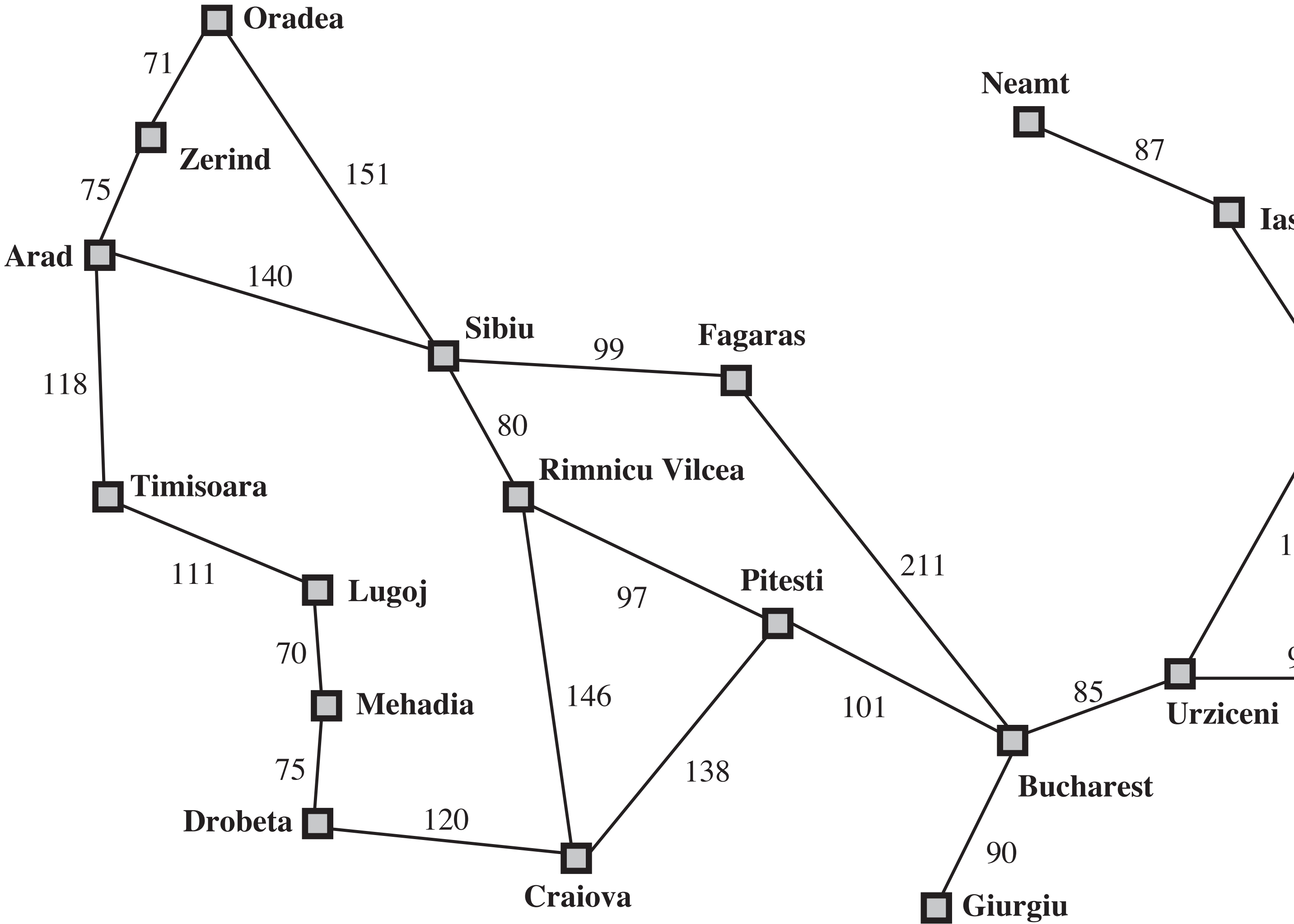
Illustrate several important points about AI

- Perfection is usually unattainable $\Longrightarrow$ most approximate
- Good idea to think about what to think about
- Uncertainty constrains the assignment of values to states
- Optimal decisions depend on information state, not the real state
- Domain-specific tricks can be generatlized to meta-heuristics of possible relevance for search of

Problem A* Search

Map out the tree that A* would use utilizing the straight line distance heuristic for a trip from Sibiu to Bucharest.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



JMU JAMES MADISON UNIVERSITY.

Problem 4.1

Give the name of the algorithm that results from each of the following special cases: local beam search with $k = 1$