Adversarial Search

CS 486/686: Introduction to Artificial Intelligence

Introduction

- So far we have only been concerned with a single agent

- Today, we introduce an adversary!

Outline

- Games
- Minimax search
- Alpha-beta pruning
- Evaluation functions
- Coping with chance

Games

- Games are the oldest, most well-studied domain in AI
- Why?
  - They are fun
  - Easy to represent, rules are clear
  - State spaces can be very large
    - In chess, the search tree has ~10^{154} nodes
  - Like the “real world” in that decisions have to be made and time is important
  - Easy to determine when a program is doing well

Types of Games

- **Perfect vs Imperfect Information**
  - Perfect information: You can see the entire state of the game
  - Imperfect information:
- **Deterministic vs Stochastic**
  - Deterministic: change in state is fully controlled by the players
  - Stochastic: change in state is partially determined by chance

Games as Search Problems

<u>2-player perfect information game</u>

- **State**:
- **Successor function**:
- **Terminal state:**
- **Utility function**:
- **Solution**:

Game Search Challenge

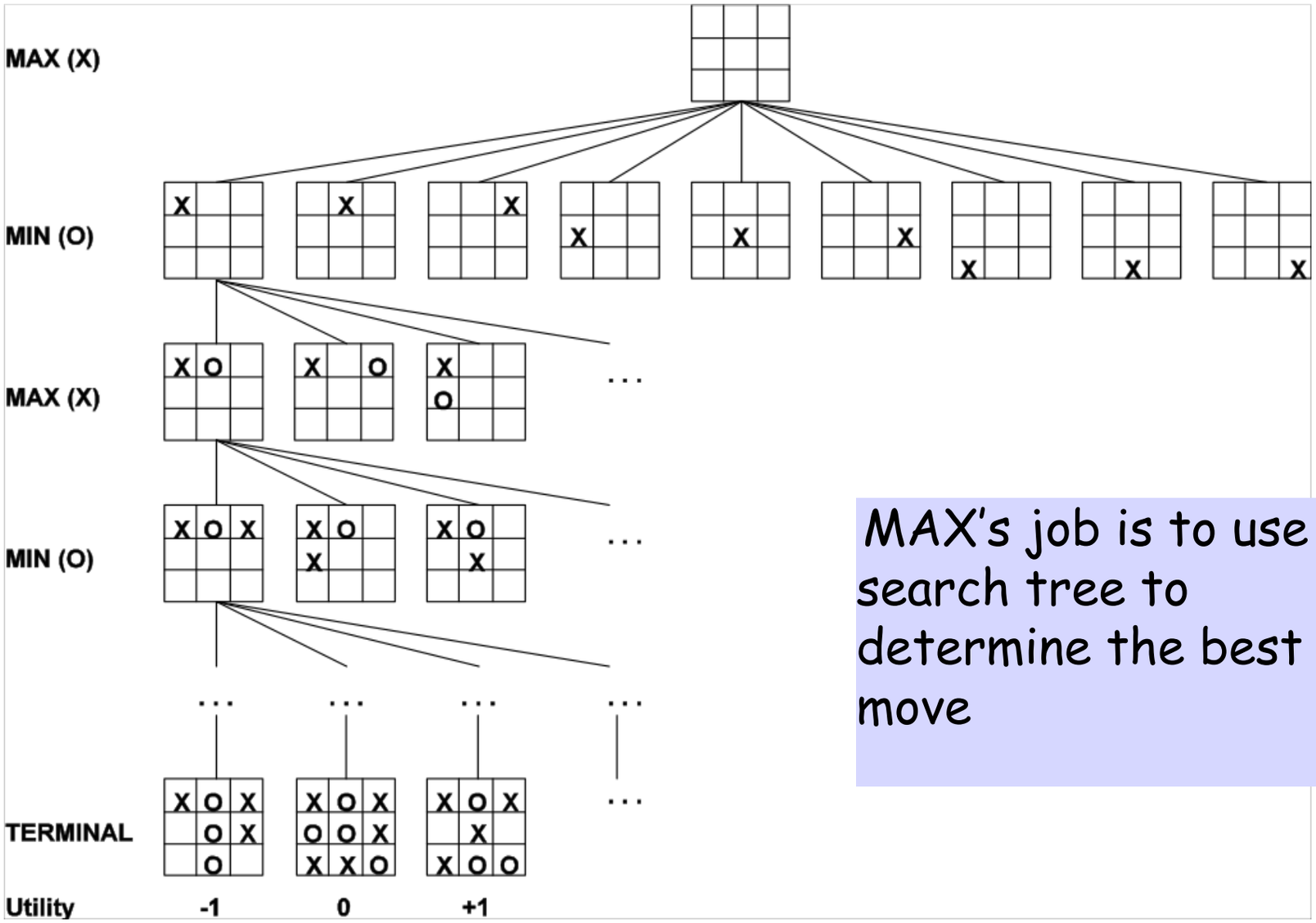
What makes game search challenging?

- There is an opponent
- The opponent is malicious
  - it wants to win (by making you lose)
- We need to take this into account when choosing moves

Notation:

- **MAX** player wants to maximize its utility
- **MIN** player wants to minimize its utility

Example



MAX's job is to use the search tree to determine the best move

Optimal Strategies

In standard search

- Optimal solution is sequence of moves leading to a goal state

Strategy (from MAX's perspective)

- Specify a move for the initial state
- Specify a move for all possible states arising from MIN's response
- Then all possible responses to all of MIN's responses to MAX's previous moves
- ...

Optimal Strategies

Goal: Find optimal strategy

- What do we mean by optimal?
  - Strategy that leads to outcomes at least as good as any other strategy, *given that MIN is playing optimally*
    - Equilibrium (game theory)

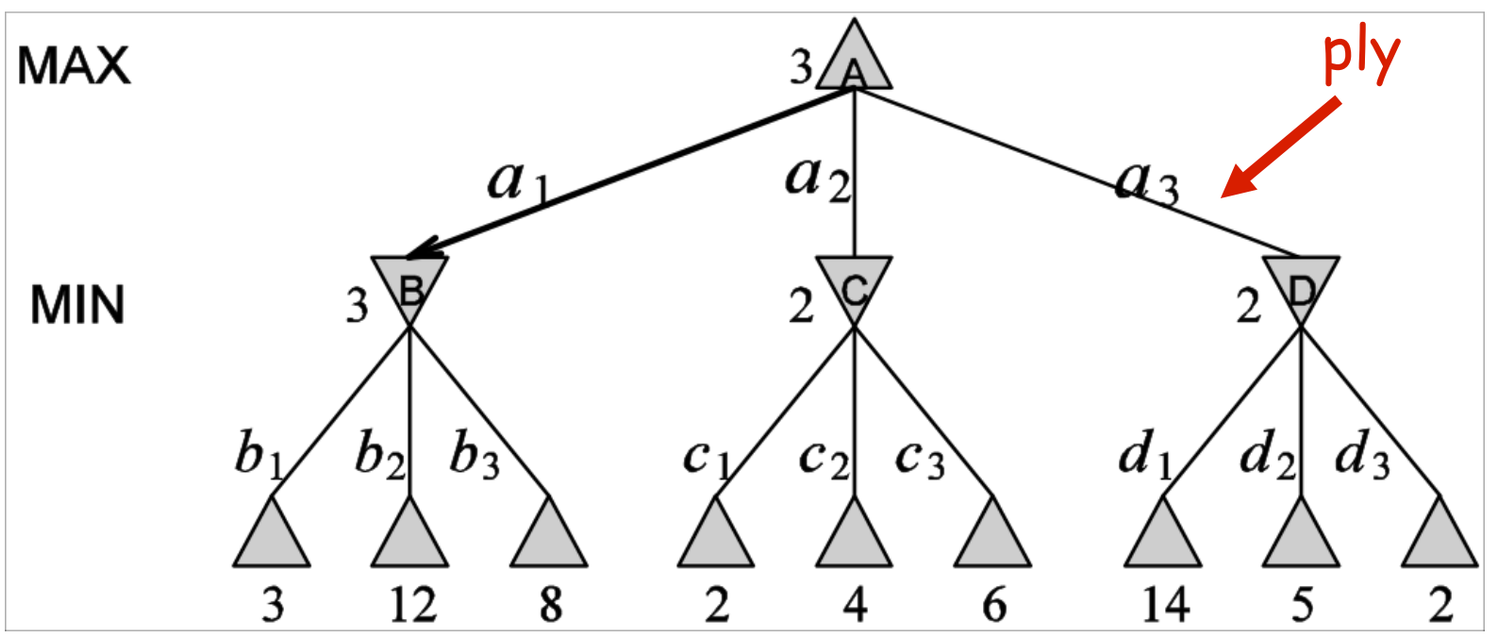
Today we focus mainly on **zero-sum games of perfect information**

- Easy games according to game theory

Minimax Value

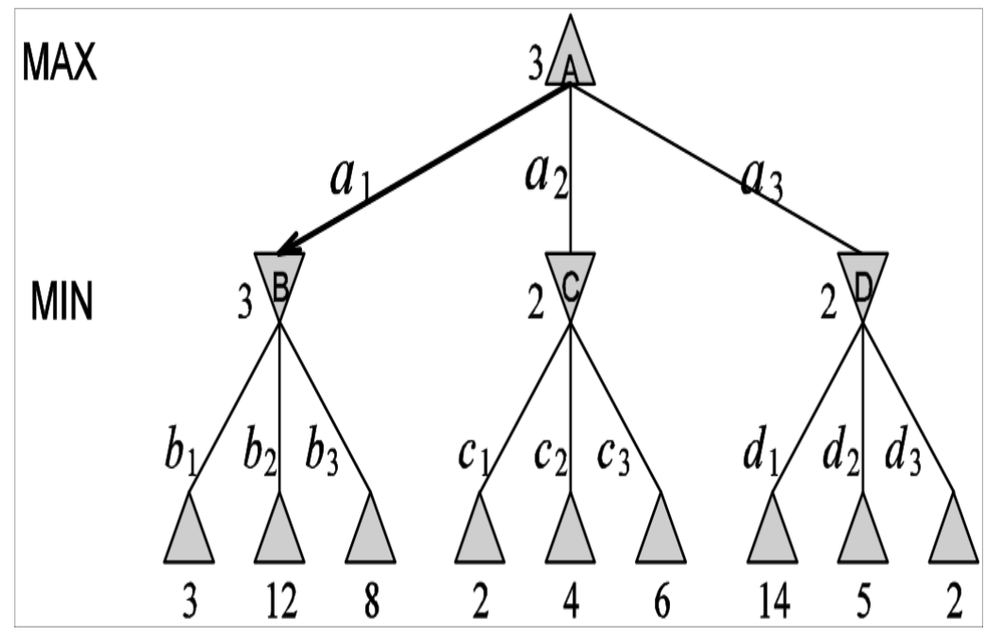
MINIMAX-VALUE(n) =

$$\begin{cases} \text{Utility}(n) & \text{if } n \text{ is a terminal state} \\ \text{Max}_{s \text{ in Succ}(n)} \text{ MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \text{Min}_{s \text{ in Succ}(n)} \text{ MINIMAX-VALUE}(s) & \text{is } n \text{ is a MIN node} \end{cases}$$

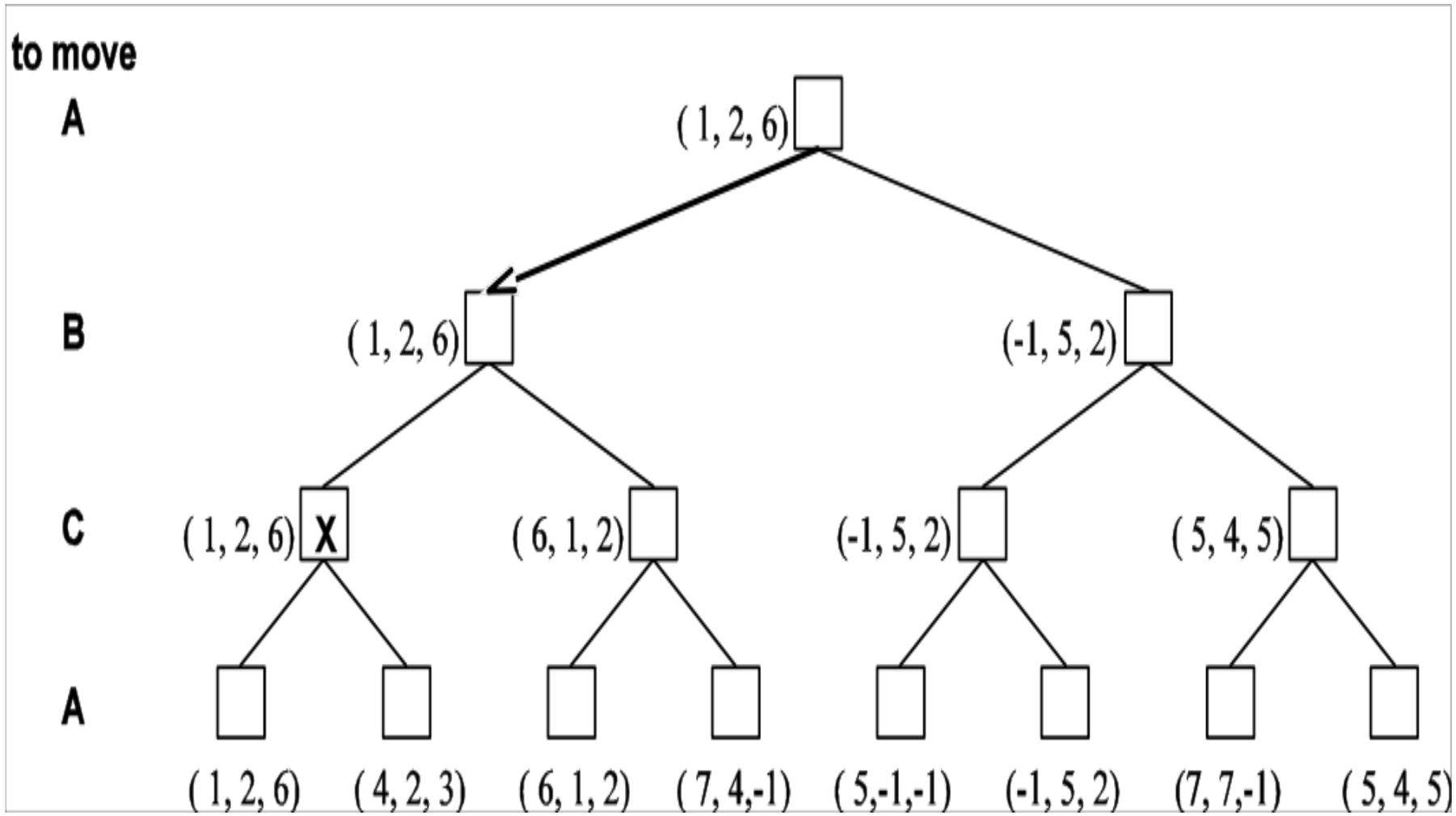


Properties of Minimax

- Complete:
- Time complexity:
- Space complexity:
- Optimal:



Minimax and Multi-Player Games



Question

Can we now write a program that will play chess reasonably well?

Question

Can we now write a program that will play chess reasonably well?

For chess b~35 and m~100

Alpha-Beta Pruning

If we are smart (and lucky) we can do **pruning**

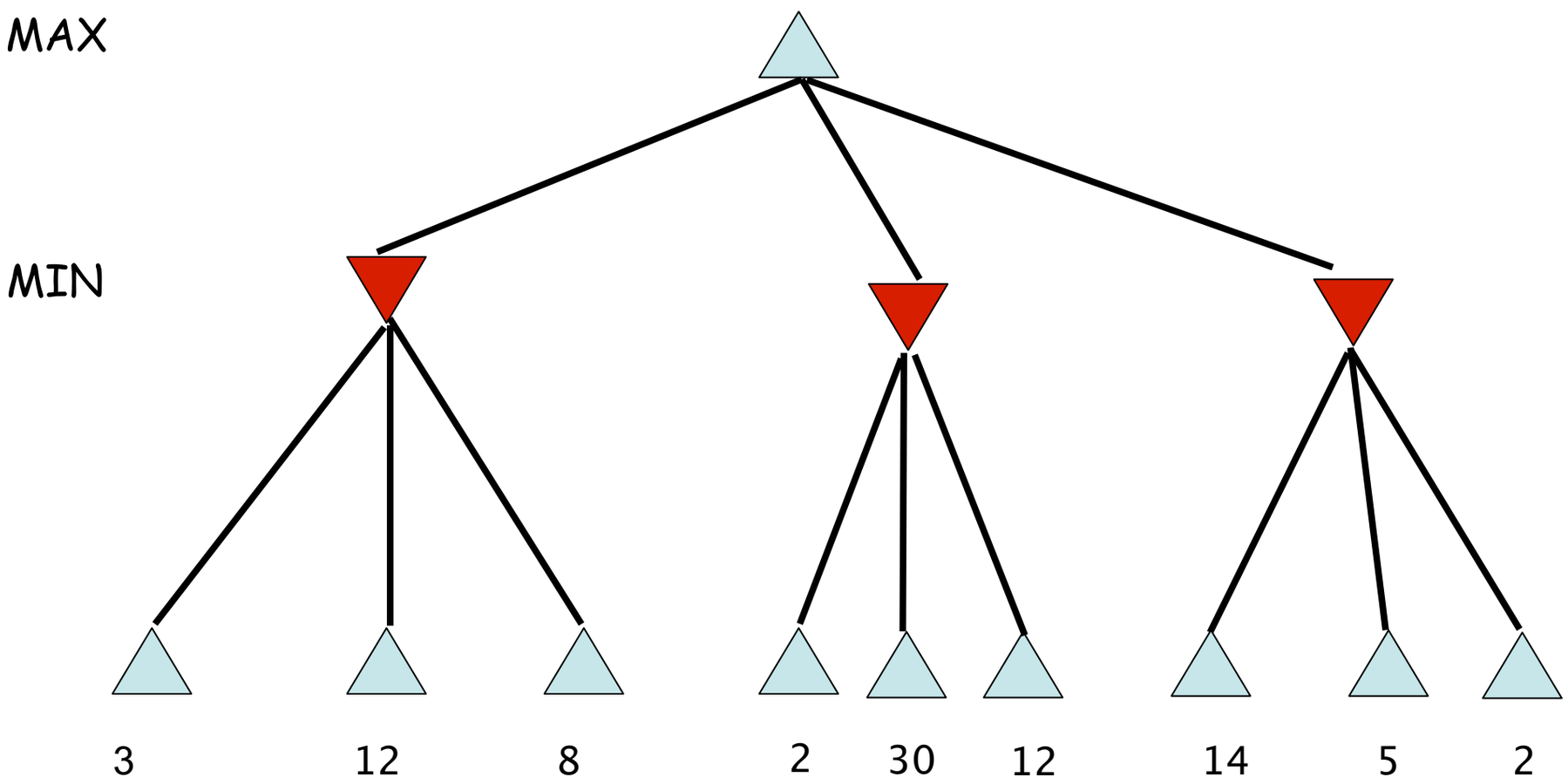
- Eliminate large parts of the tree from consideration

Alpha-beta pruning applied to a minimax tree

Alpha-Beta Pruding

- Alpha:
  - Value of best (highest value) choice we have found so far on path for MAX
- Beta:
  - Value of best (lowest value) choice we have found so far on path for MIN
- Update alpha and beta as search continues
- Prune as soon as value of current node is known to be worse than current alpha or beta values for MAX or MIN

Example



Properties of Alpha-Beta

- Can pruning result in a different outcome than minimax search?

- How much can be pruned when searching?

Real-Time Decisions

Alpha-Beta can be a huge improvement over minimax

- Still not good enough
  - Need to search to terminal states for at least part of search space
  - Need to make decisions quickly

Solution

- Heuristic evaluation function + cutoff tests

Evaluation Functions

- Apply an evaluation function to a state
  - If terminal state, function returns actual utility
  - If non-terminal, function returns estimate of the expected utility
- **Function must be fast to compute**

Evaluation Functions

- How do we get evaluation functions?
  - Expert knowledge
  - Learned from experience
- Look for features of states
  - Weighted linear function $Eval(s)=\sum_i w_i f_i(s)$

Cutting Off Search

Do we have to search to terminal states?

- No! Cut search early and apply evaluation function

When?

- Arbitrarily (but deeper is better)
- Quiescent states
  - States that are “stable”
- Singular extensions
  - Searching deeper when you have a move that is “clearly better”
  - Can be used to avoid the **horizon effect**

Cutting Off Search

How deep?

Novice player

- 5-ply (minimax)

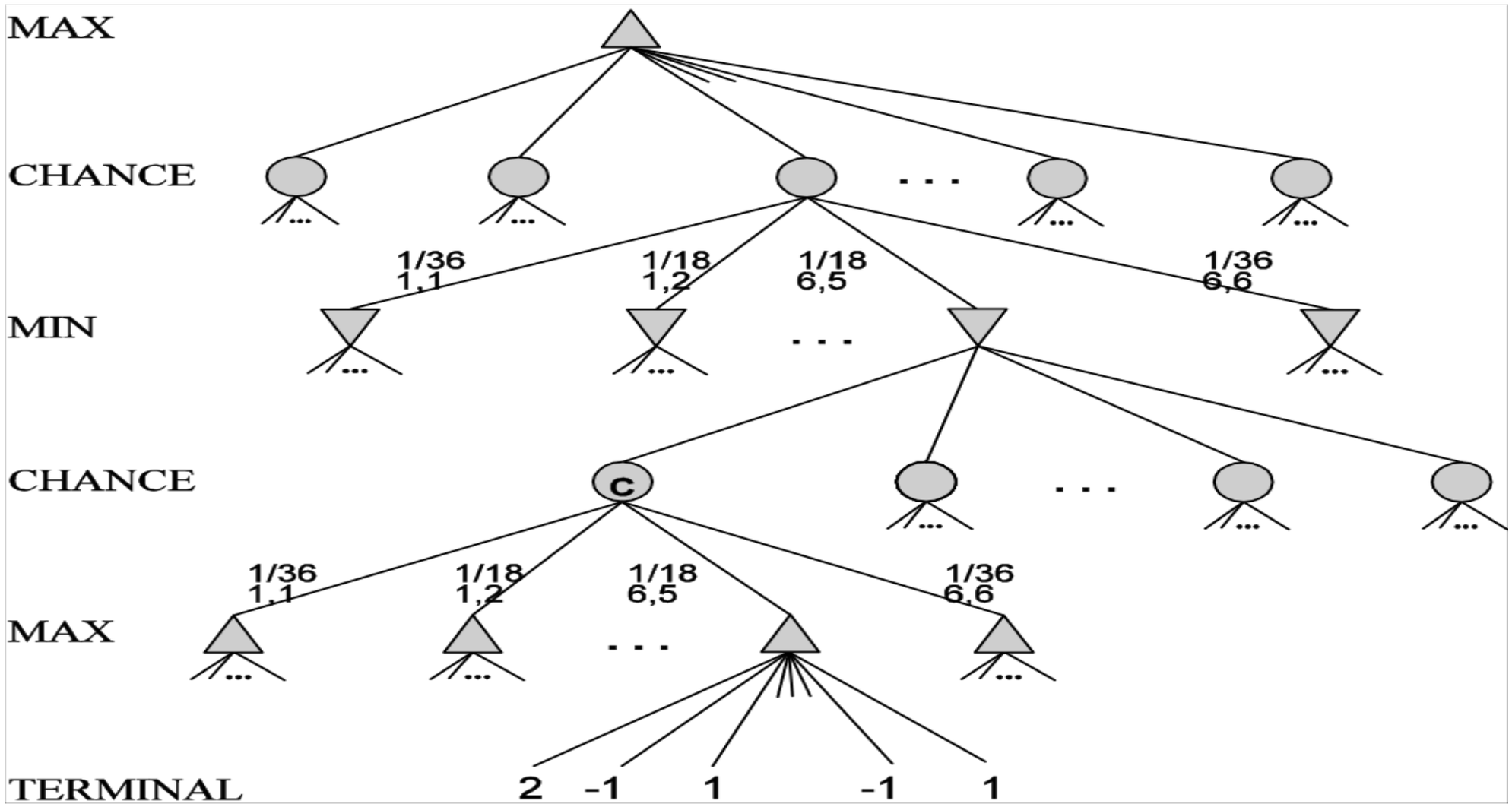
Master player

- 10-ply (alpha-beta)

Grandmaster

- 14-ply + fantastic evaluation function, opening and endgame databases,...

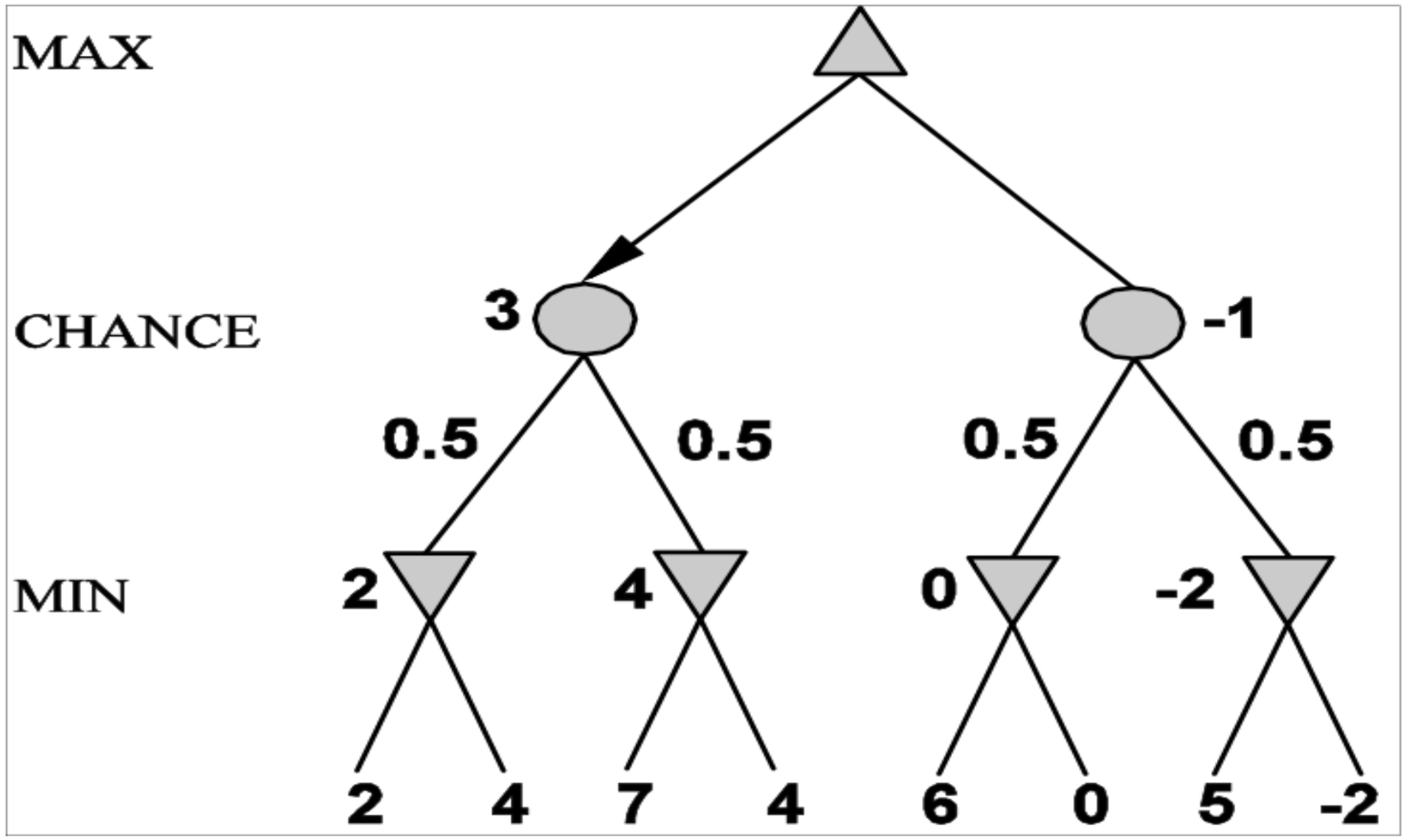
Stochastic Games



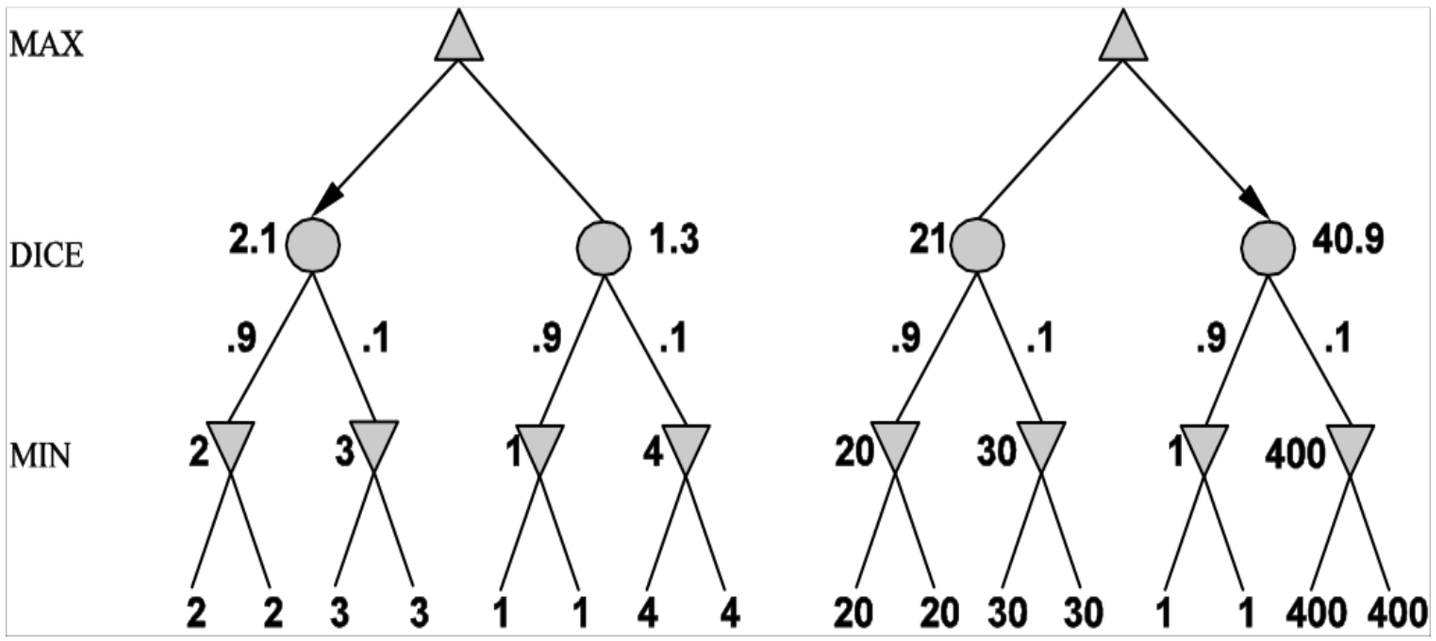
Stochastic Games

- Need to consider **best/worst cases + probability** they will occur
- Recall: Expected value of a random variable x $E[x]=\sum_{x \text{ in } X} P(x)x$
- **Expectiminimax**: minimax but at chance nodes compute the expected value

Expectiminimax



Expectiminimax



WARNING: exact values do matter! Order-preserving transformations of the evaluation function can change the choice of moves. Must have **positive linear transformations** only

What about Go?

What about Go?

Monte-Carlo Tree Search (MCTS)

- Build search tree according to outcomes of simulated plays

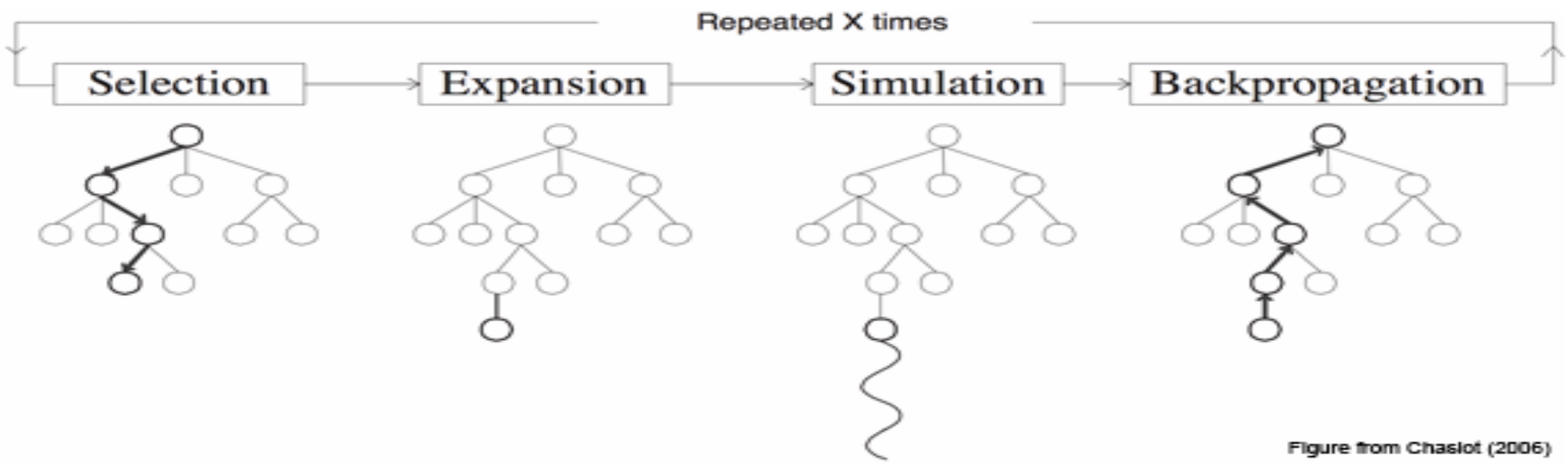


Figure from Chaslot (2006)

Upper Confidence Bounds for Trees (UCT): "Minimax search" using UCB

$$v_i + C\sqrt{\frac{\ln N}{n_i}}$$

Summary

- Games pose lots of fascinating challenges for AI researchers
- Minimax search allows us to play optimally against an optimal opponent
- Alpha-beta pruning allows us to reduce the search space
- A good evaluation function is key to doing well
- Games are fun!