Adversarial Search

CS 486/686: Introduction to Artificial Intelligence
Winter 2016

Introduction

- So far have only been concerned with single agents
- Today
  - Multiple agents planning against each other
    - Adversarial settings

Outline

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

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

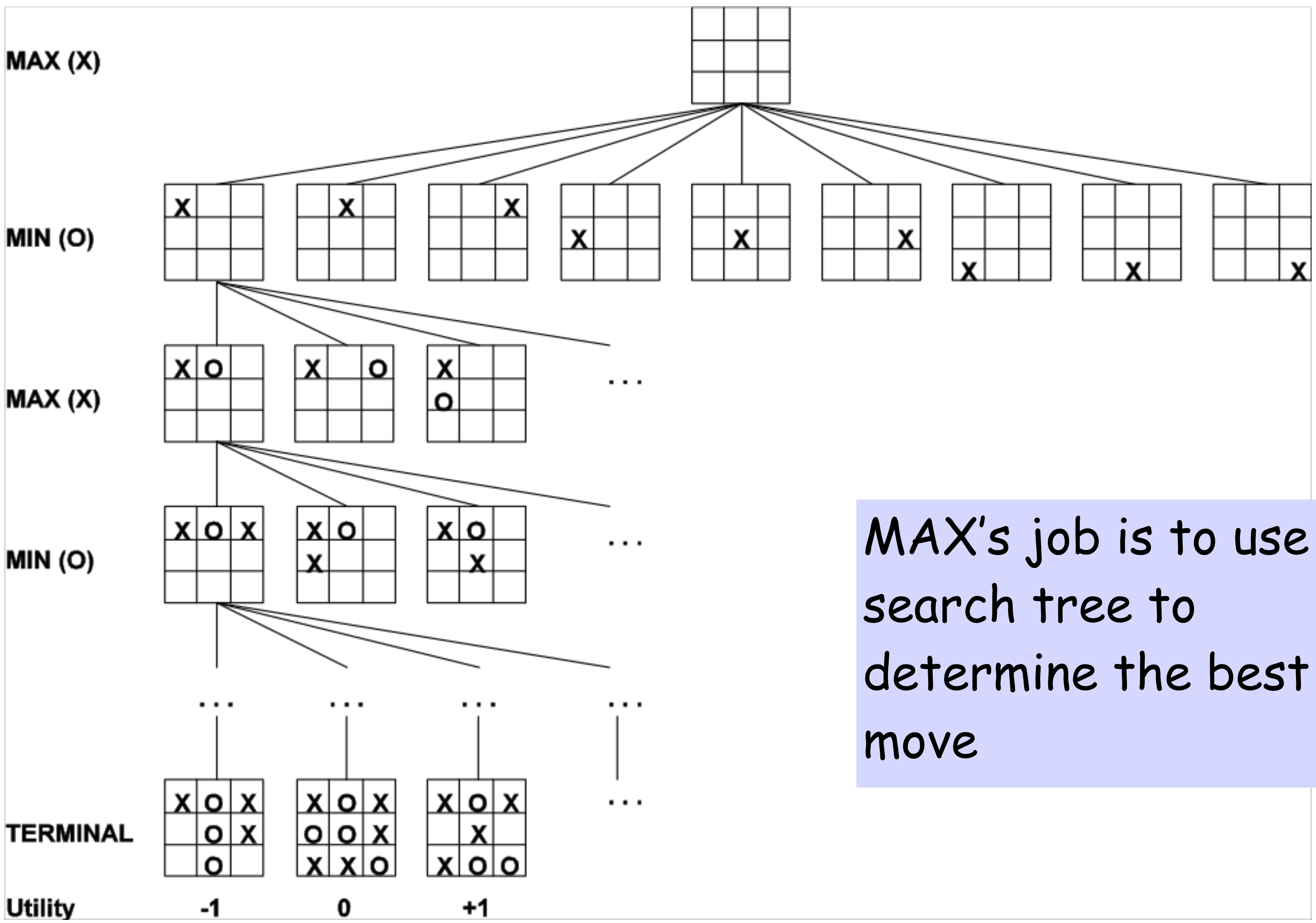
Games as Search Problems

- 2-player perfect information game
- **State**: board configuration plus the player who's turn it is to move
- **Successor function**: given a state, returns a list of (move,state) pairs indicating legal move and resulting state
- **Terminal state**: states where there is a win/loss/draw
- **Utility function**: assigns a numerical value to terminal states
- **Solution**: a strategy (way of picking moves) that wins the game

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 winning terminal 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
  - ...

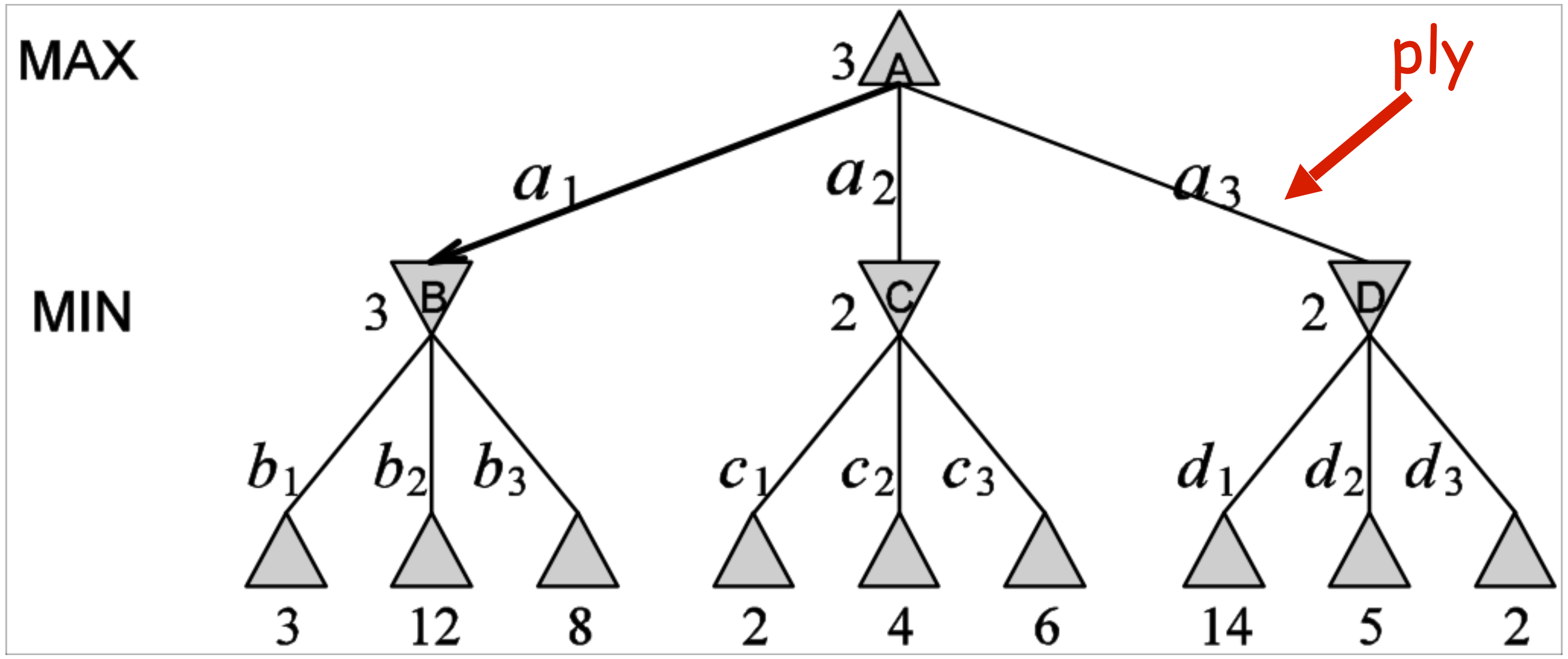
Optimal Strategies

- **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) =

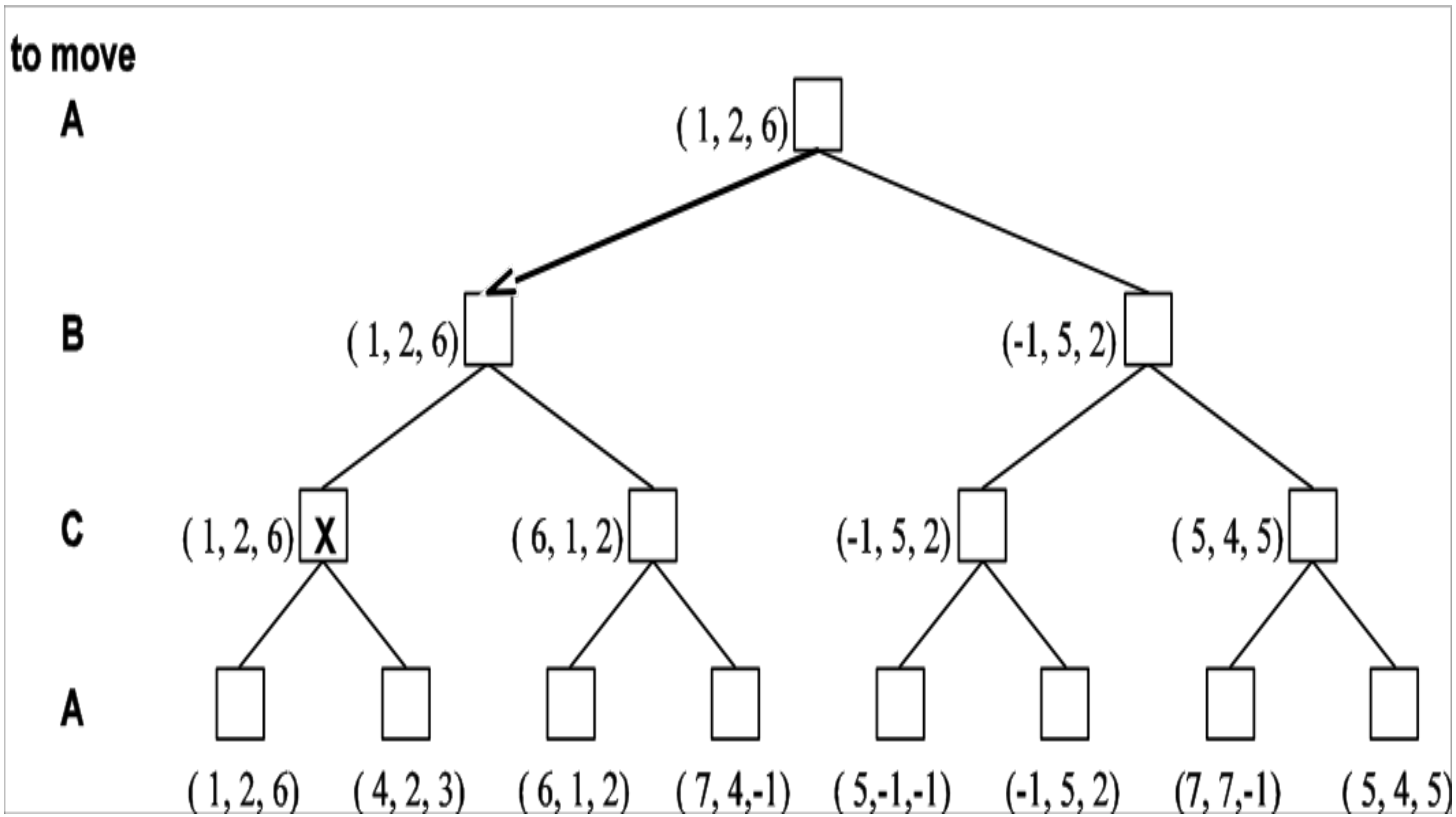
$$
\begin{cases}
\text{Utility}(n) & \text{if } n \text{ is a terminal state} \\
\text{Max}_{s \text{ in } Succ(n)} \text{ MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\
\text{Min}_{s \text{ in } Succ(n)} \text{ MINIMAX-VALUE}(s) & \text{is } n \text{ is a MIN node}
\end{cases}
$$



Properties of Minimax

- Complete if tree is finite
- Time complexity: $O(b^m)$
  - m is depth of tree
- Space complexity: O(bm)
  - It is DFS
- Optimal against an *optimal opponent*
  - If opponent is not playing optimally, then may be better off doing something else

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

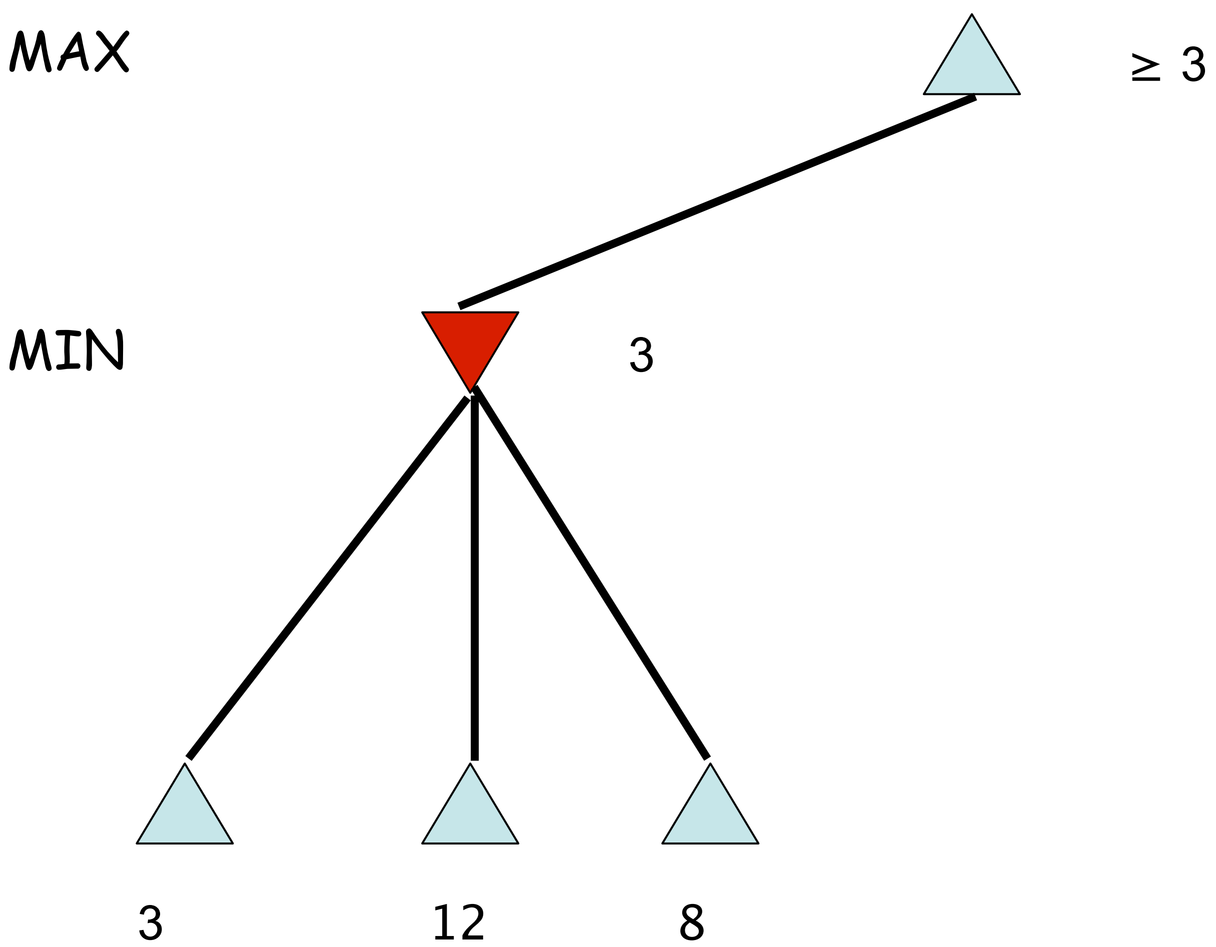
Alpha-Beta Pruning

- 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
  - Returns the same decision as minimax
  - Prunes branches that cannot influence final decision

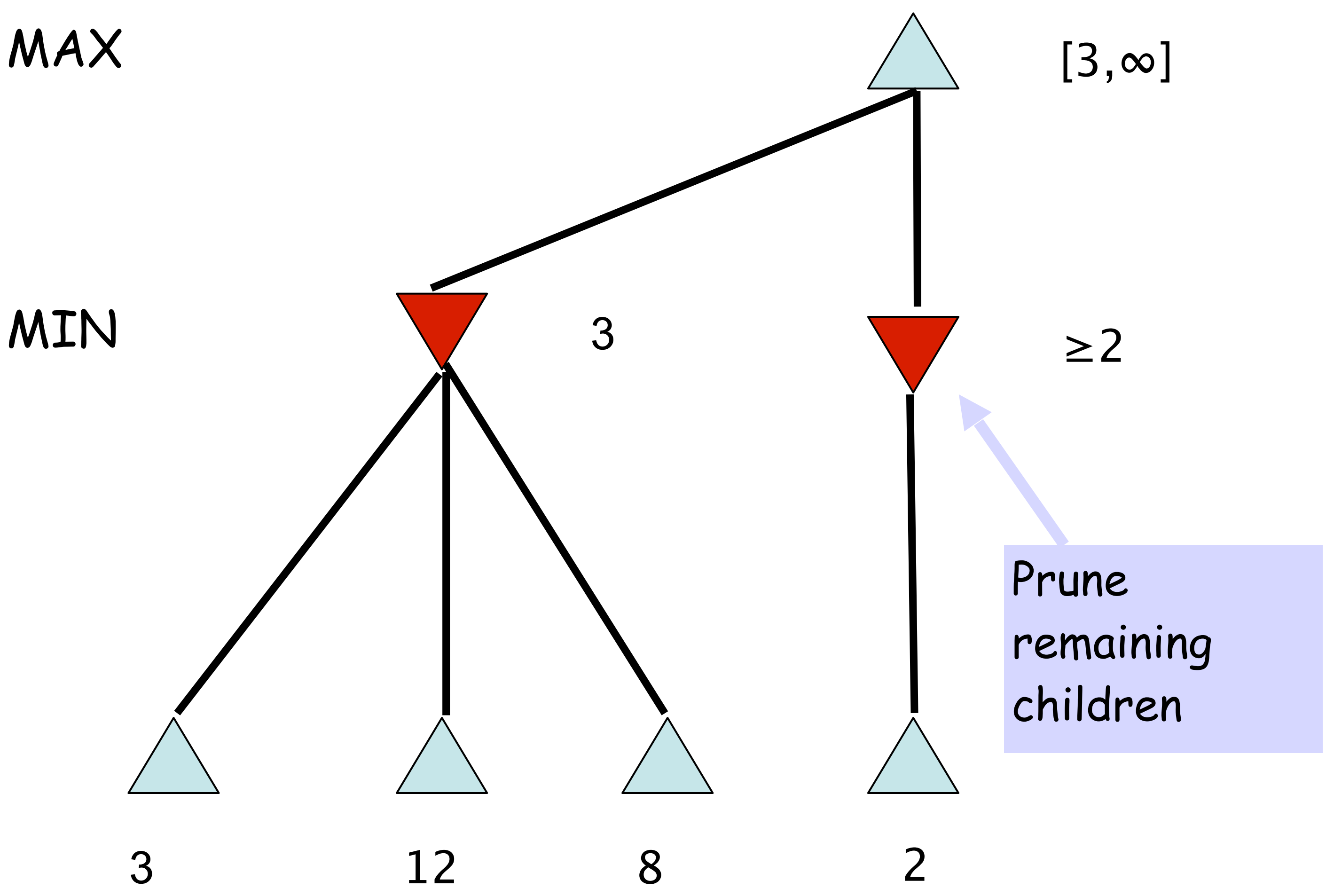
Alpha-Beta Pruding

- 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

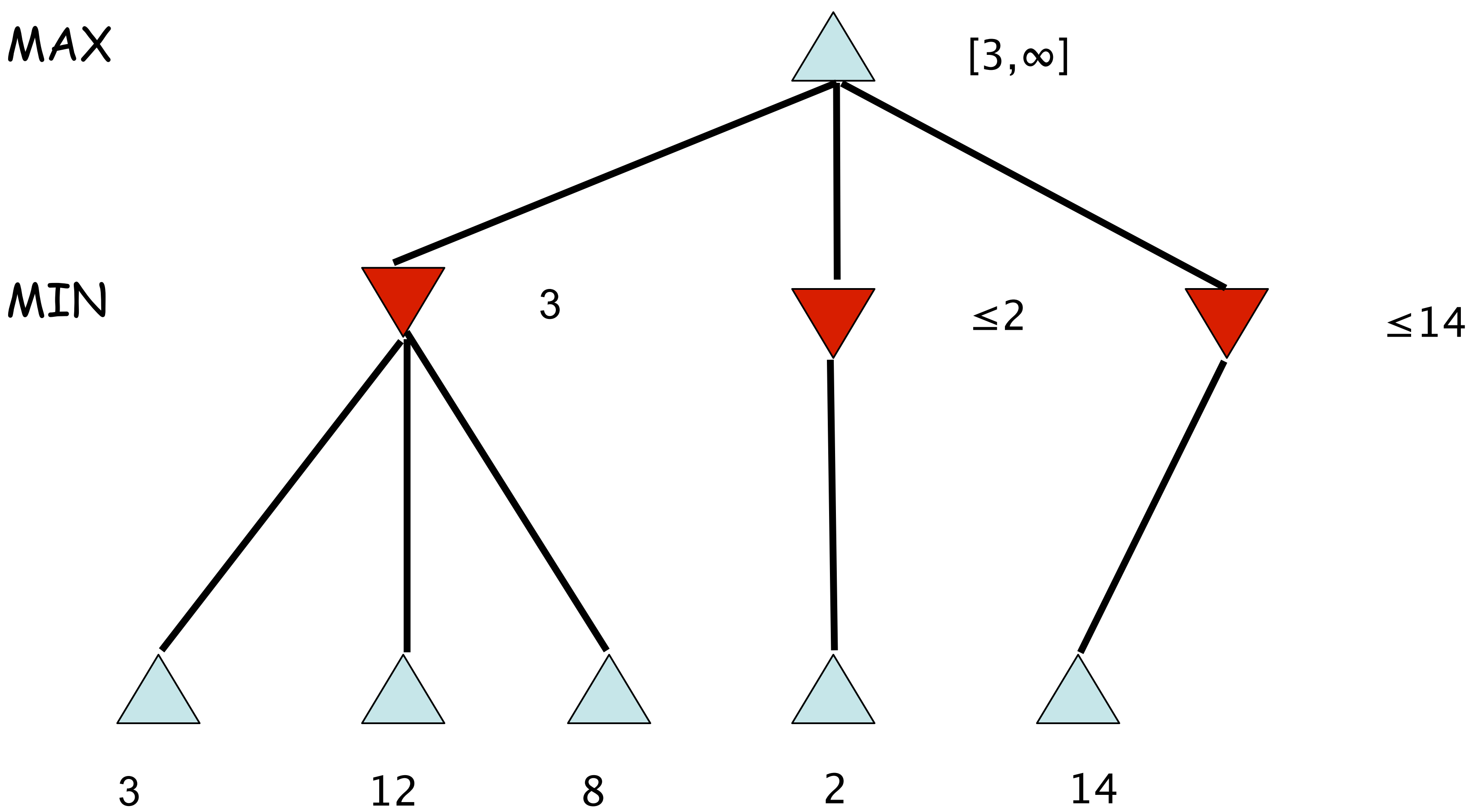
Example



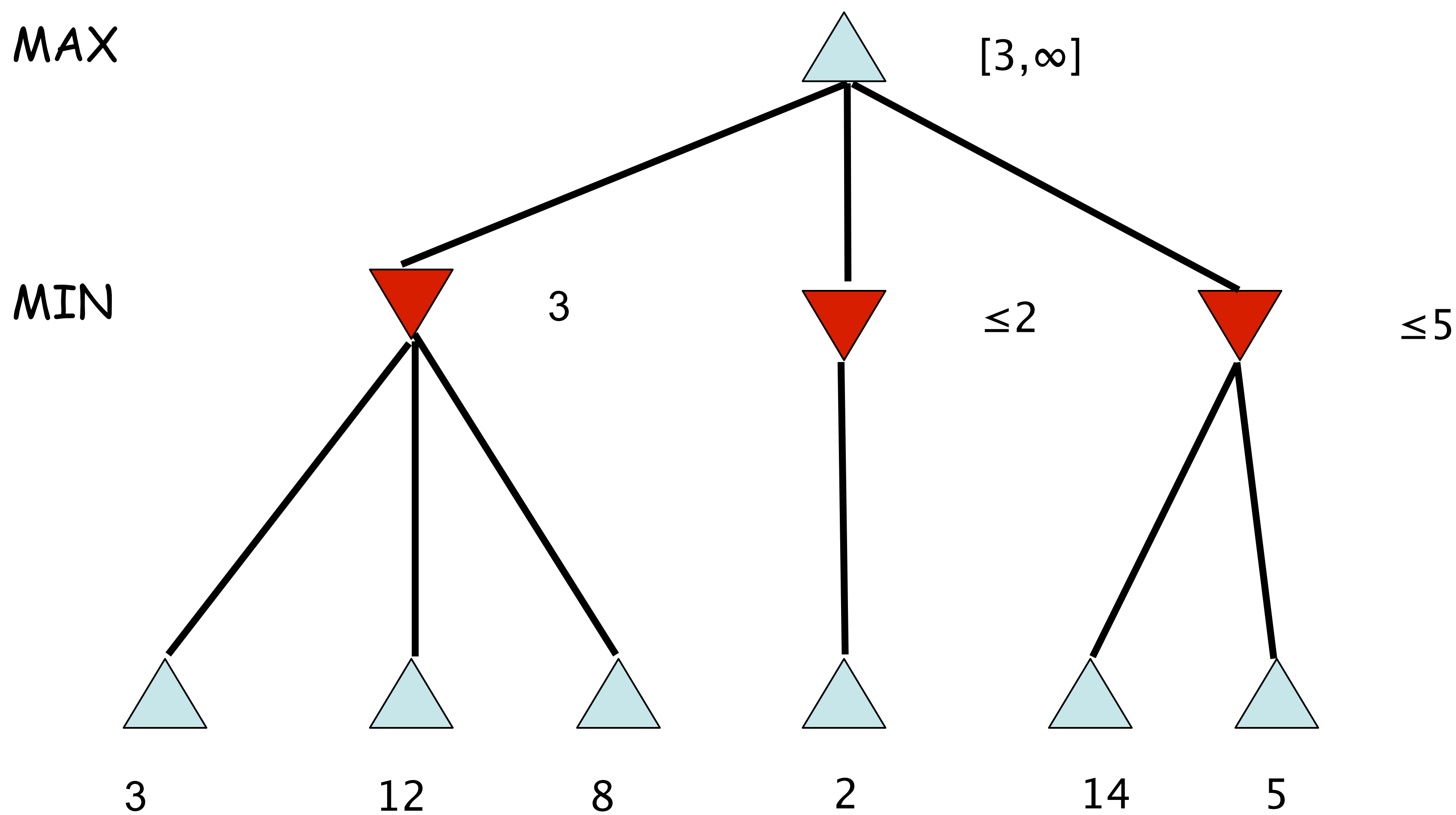
Example



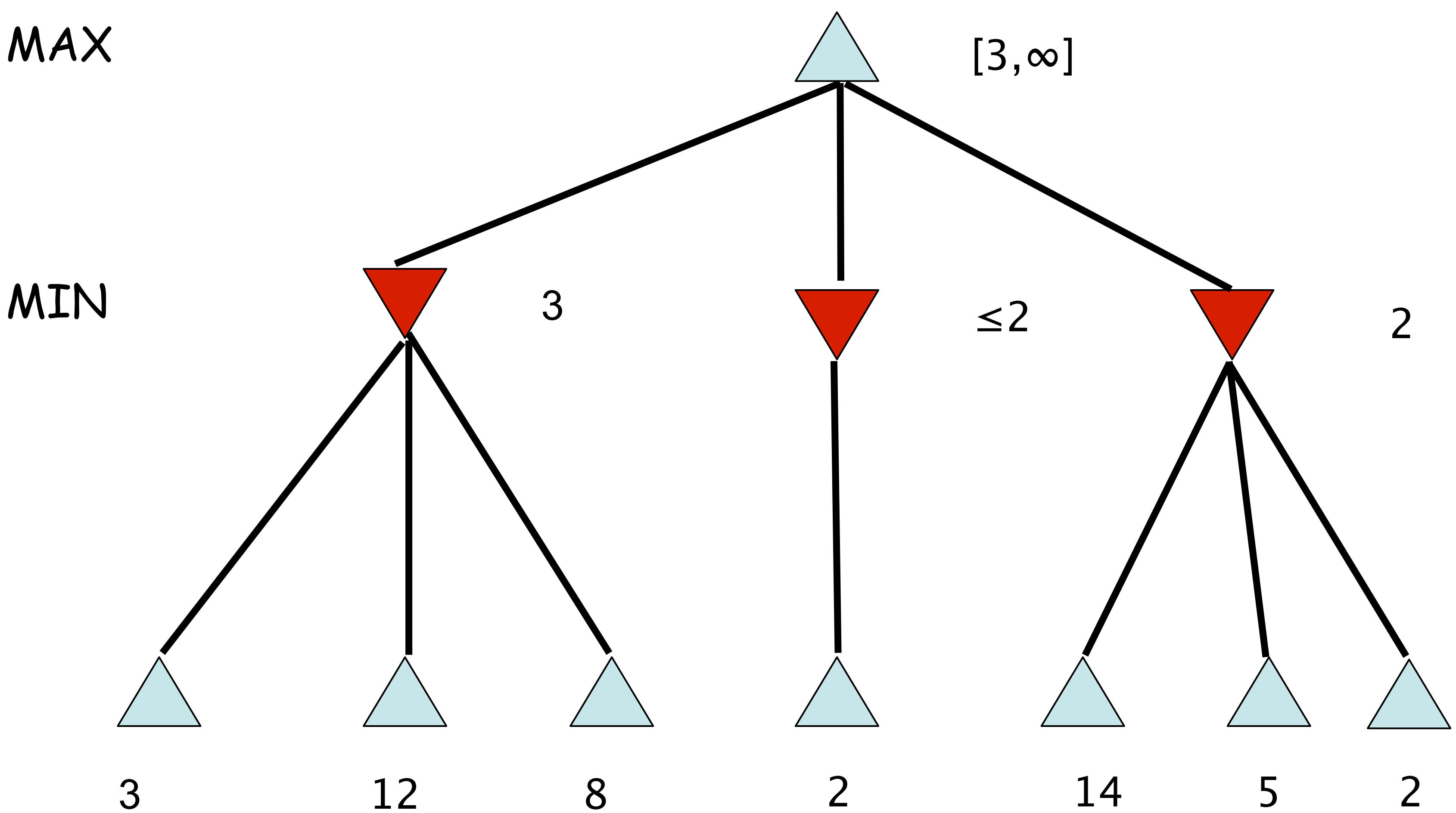
Example



Example



Example



Properties of Alpha-Beta

- Pruning does not affect the final result
  - Why?
- Move ordering is important
- Alpha-beta demonstrates the value of reasoning about which computations are important

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 $Eval(s)=\sum_i w_i f_i(s)$

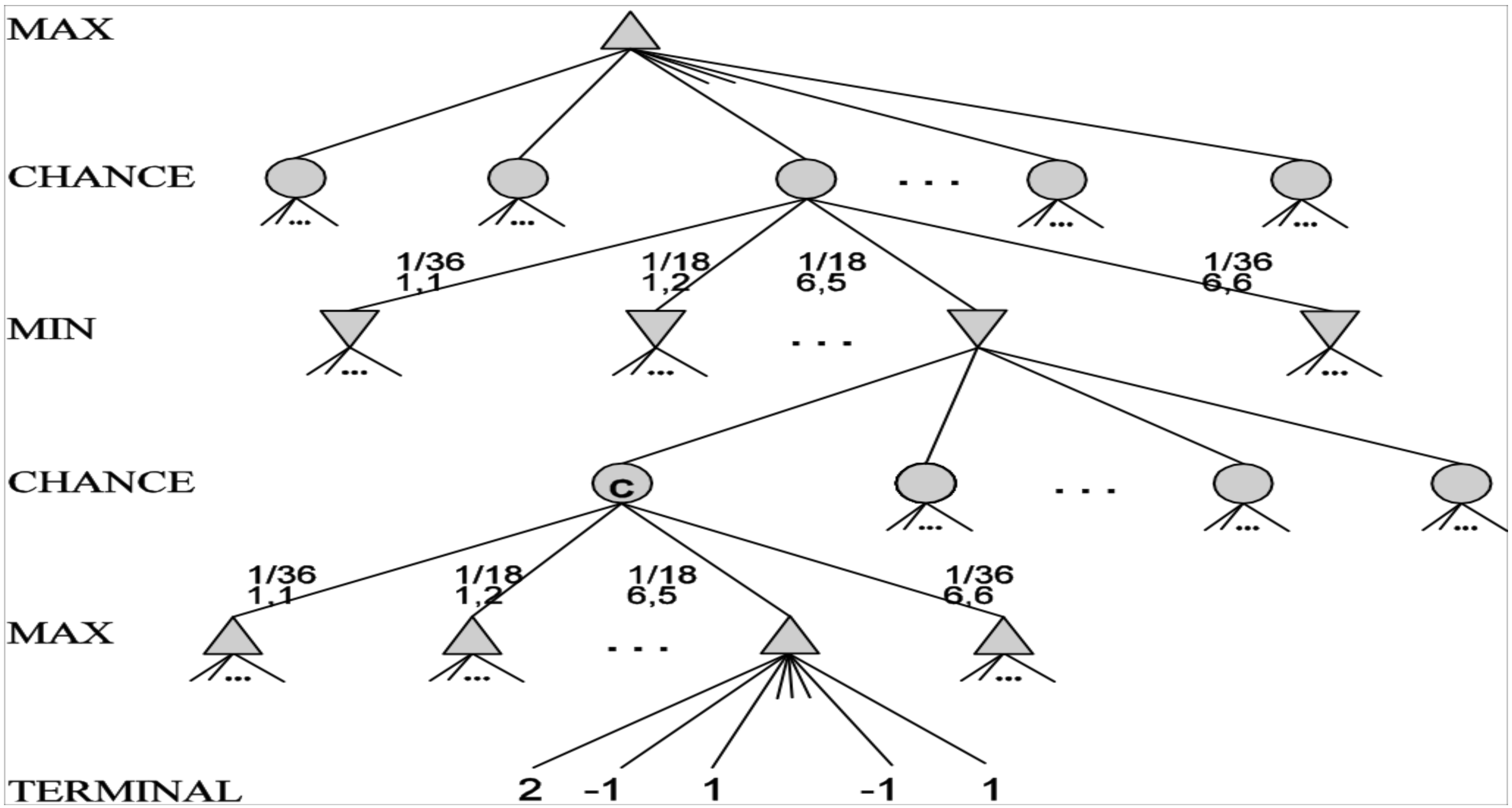
Cutting Off Search

- 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,...

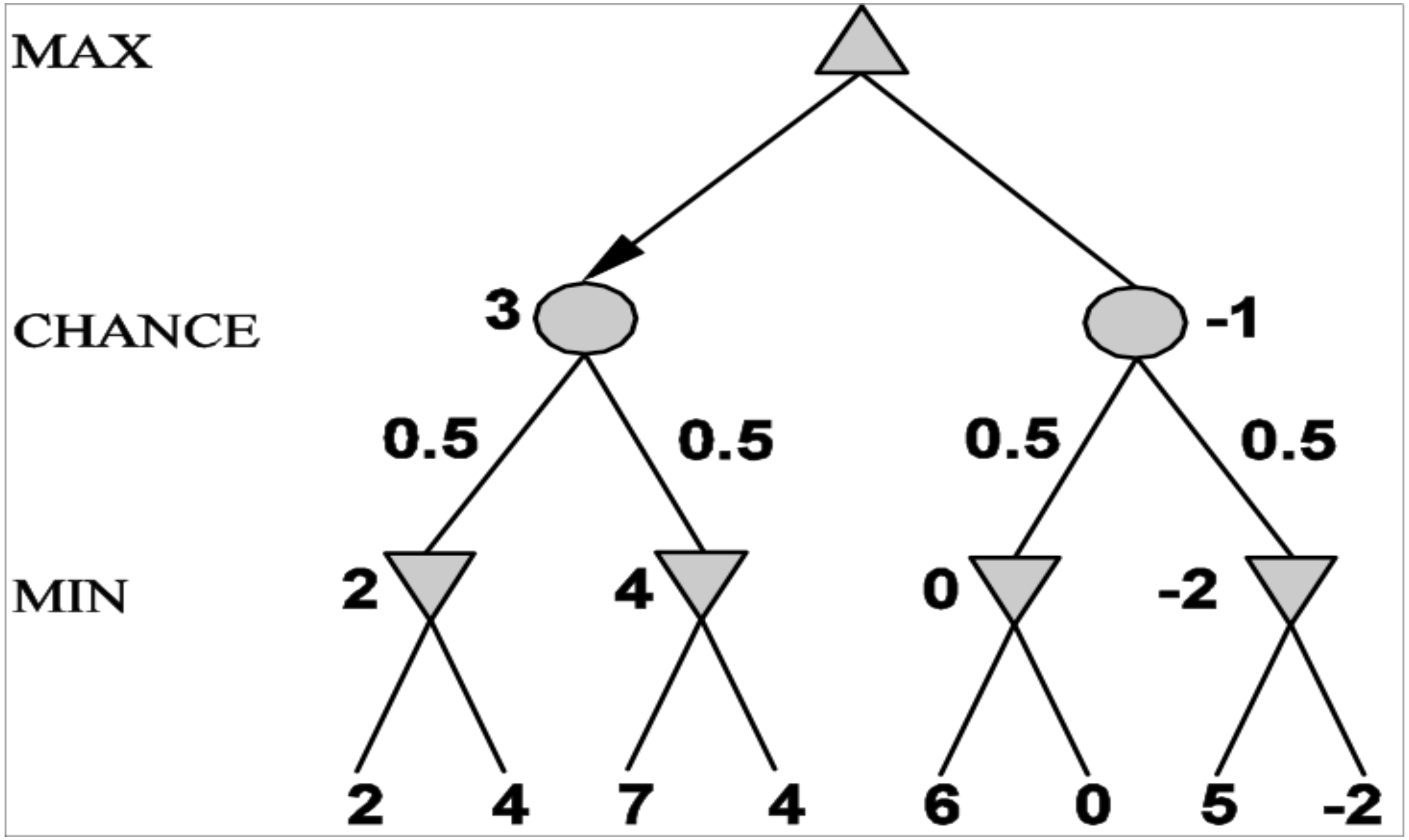
Stochastic Games



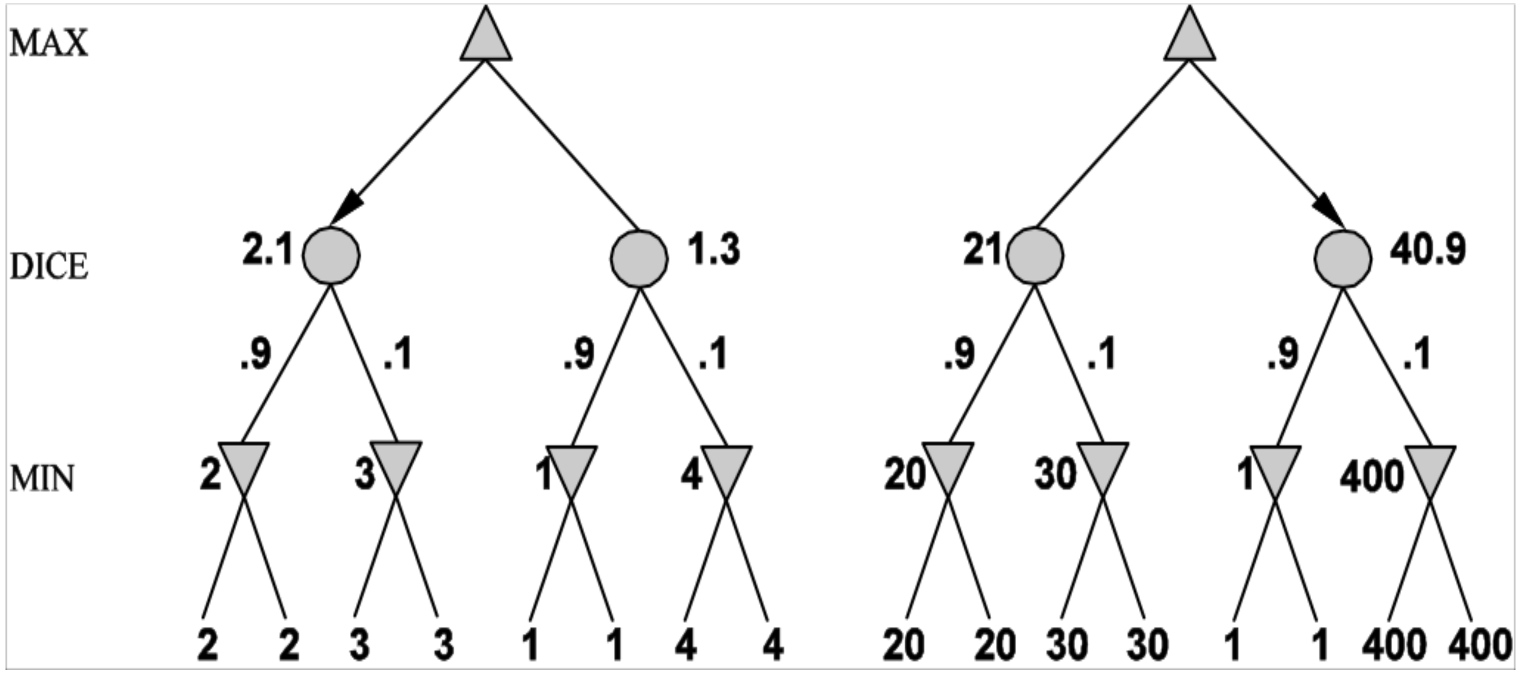
Stochastic Games

- 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

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!

Some Game Programs

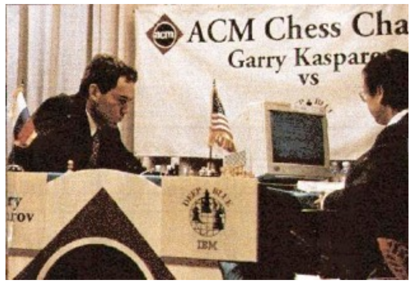
Checkers

Mr. Tinsley suffered his 4th and 5th losses **<u>ever</u>** against Chinook

Checkers

- Chinook (University of Alberta)
  - World Man-Machine Checkers Champion
  - Alpha-beta search
  - Opening database
- Secret weapon: **Endgame database**
  - Perfect knowledge into the search
- Checkers is now dominated by computers
  - Checkers is (weakly) solved

Chess: Kasparov vs. Deep Blue



1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Kasparov		Deep Blue
5'10"	Height	6' 5"
176 lbs	Weight	2,400 lbs
34 years	Age	4 years
50 billion neurons	Computers	32 RISC processors + 256 VLSI chess engines
2 pos/sec	Speed	200,000,000 pos/sec
Extensive	Knowledge	Primitive
Electrical/chemical	Power Source	Electrical
Enormous	Ego	None

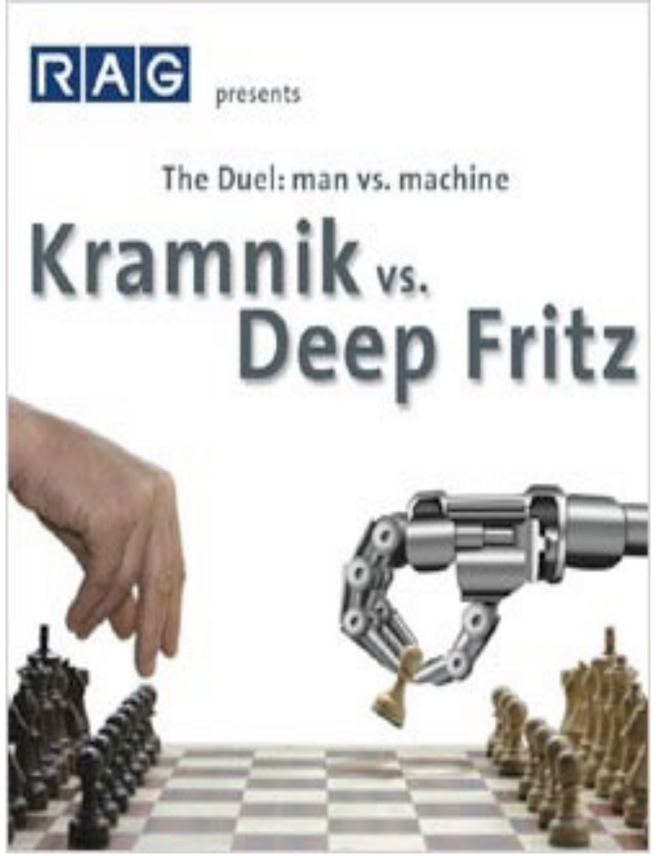
Jonathan Schaeffer

Chess

- Its secret:
  - Specialized chess processor + special-purpose memory optimization
  - Very sophisticated evaluation function
    - Expert features and hand-tuned weights
  - Opening and closing books
  - Alpha-beta + improvements (searching up to 40 ply deep)
  - Searched over 200 million positions per second

Chess

- There are now apps that are on par with human champions
- Is Chess still a human game or have computers conquered it?

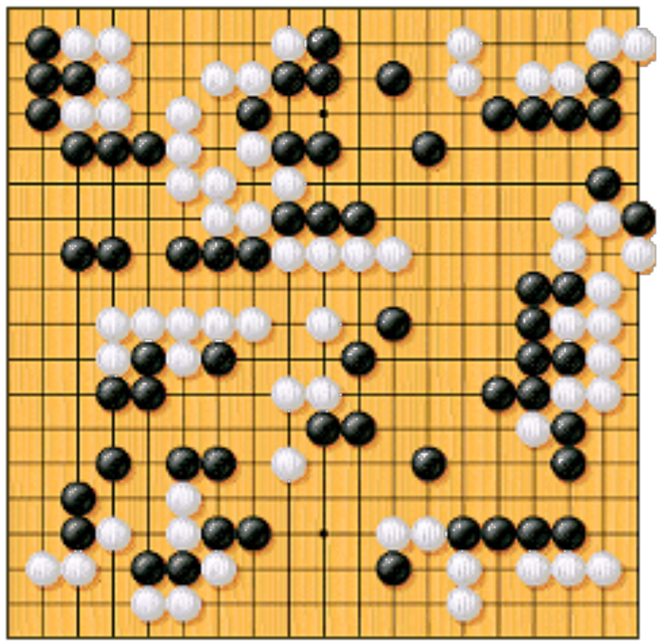


Backgammon

- TD-Gammon (Gerry Tesauro at IBM)
- One of the top players in the world
- Searches only two moves ahead!
- Its secret: **One amazing evaluation function**
  - Neural network trained with reinforcement learning during ~1 million games played against itself
  - Humans play backgammon differently now, based on what TD-Gammon learned about the game
  - Very cool AI ☺

Go

- Large branching factor makes Go too large to solve by classic search methods
    - pieces added to the board
    - evaluation function
    - ...
- Limited progress for decades

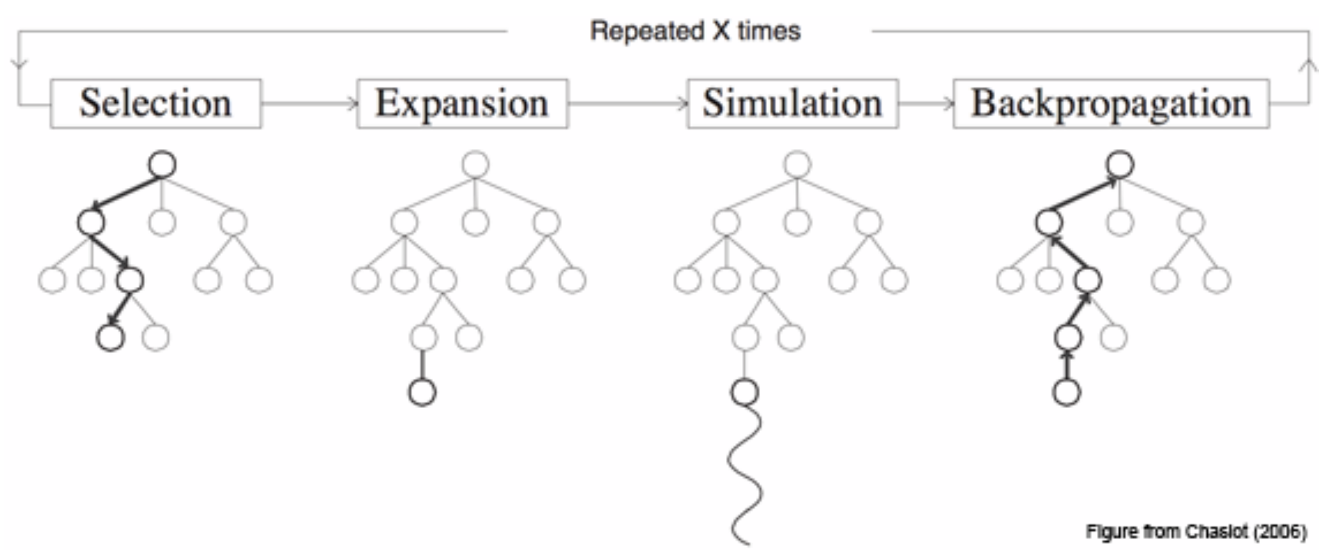


Go

- BUT computer Go has undergone a revolution in the past ~5 years
  - Close to perfection on 7x7 games
  - Reached top human level on 9x9 games
  - Still weaker than top humans on 19x9 boards

Go

- Monte-Carlo Tree Search (MCTS)
  - Build search tree according to outcomes of simulated plays



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

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

Card Games

- Focus has been on Bridge and Poker
  - Humans are still winning...
  - But machines are catching up!
- Issues
  - Stochastic and partially observable
    - Ideas discussed today don't work well
    - New approaches are being developed