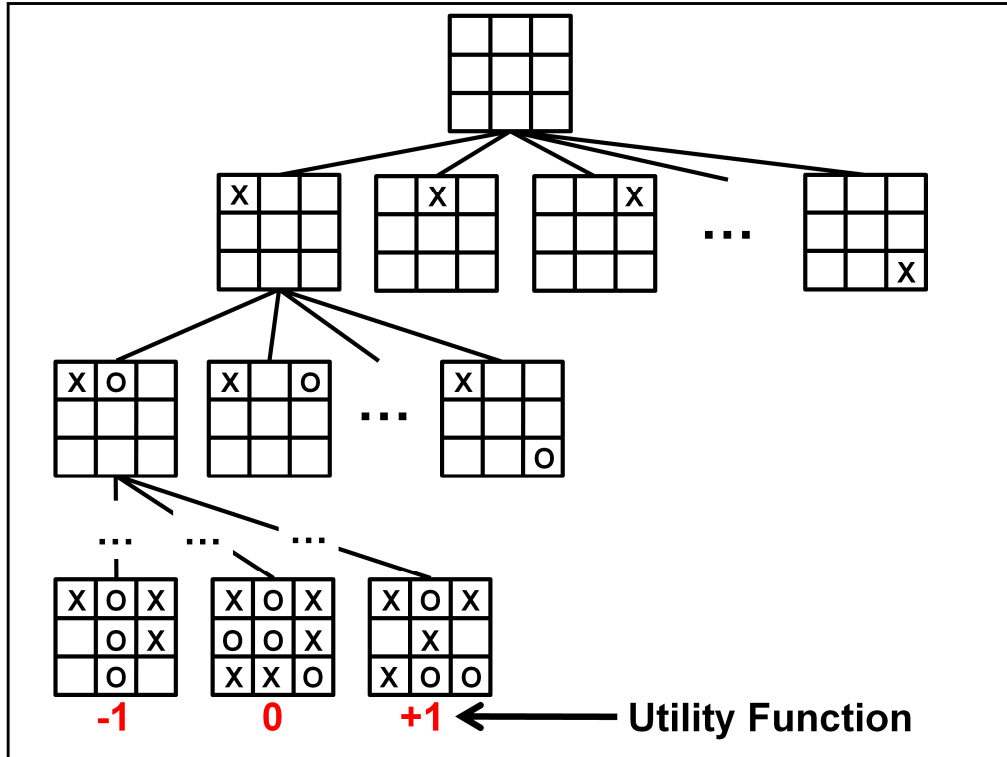


Lecture 6: Adversarial Search

(Russell and Norvig Chapter 6)

Multiplayer games where there is an adversary trying to make you lose.

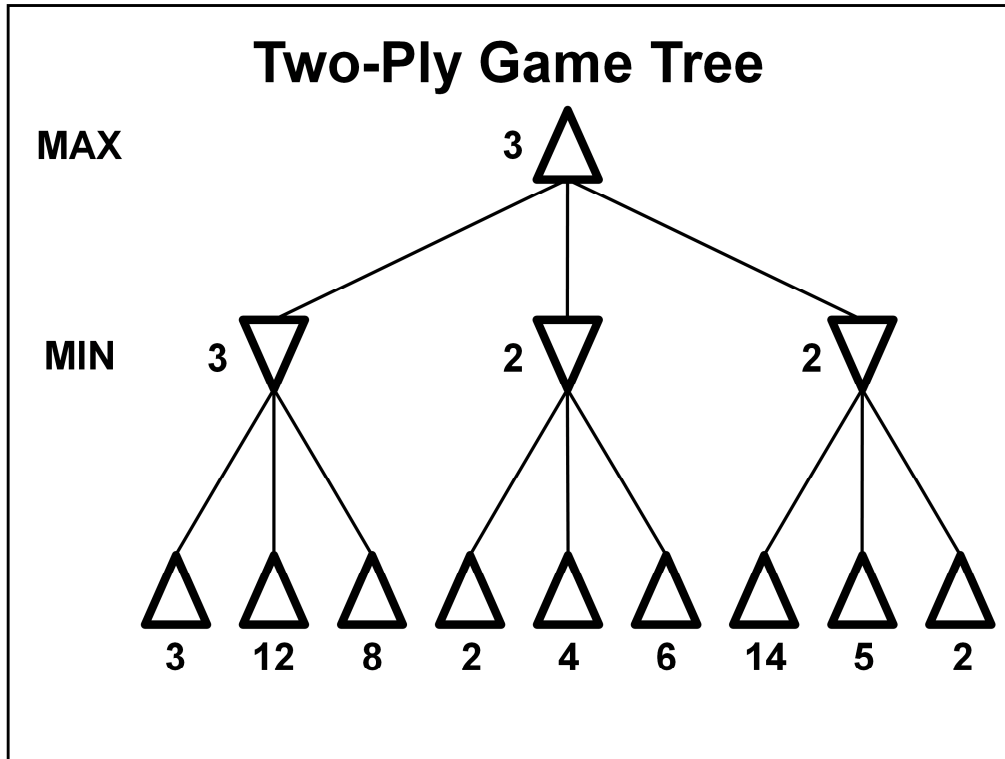


Utility function assigns scores to leaves.

For zero sum games, you have an utility function that assigns values according to point of view of player 1 (X).

Player 1 is trying to maximize this utility fn, Player 2 tries to minimize.

Important NOTE: Here we assume players can see to the bottom of the search tree and play optimally



One ply is a move from one of the players. This is a 2 ply tree: Max makes one move and Min makes one move.

Max is 1st player who tries to maximize utility. Represented by triangle up

Min is 2nd player who tries to minimize utility. Represented by triangle down

(Notation: up triangle is MAX, inverted triangle is MIN)

The Minimax Value

MINIMAX(Node n) =

$$\left\{ \begin{array}{ll} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{Succ}(n)} \text{MINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Succ}(n)} \text{MINIMAX}(s) & \text{if } n \text{ is a MIN node} \end{array} \right.$$

The Minimax Algorithm

Depth-first exploration and minimax values are backed up through the tree as the recursion unwinds

Optimal Adversaries

The definition of optimal play for MAX assumes that MIN also plays optimally. If MIN does not play optimally, then will MAX do better or worse by playing MINIMAX?

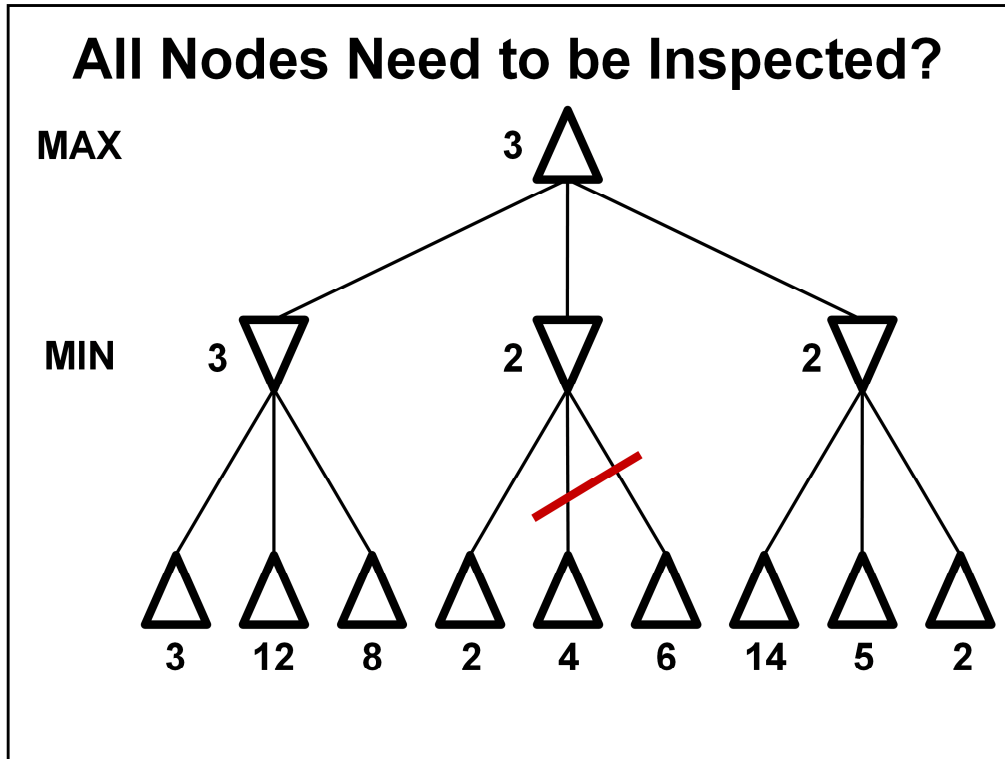
Never worse.

Basic argument: if we are facing a suboptimal opponent and they could force us to take a worse outcome by being stupid, then the adversary would be better than the optima, contradiction. More formal argument: use induction on the definition of minimax values (base case: leaves).

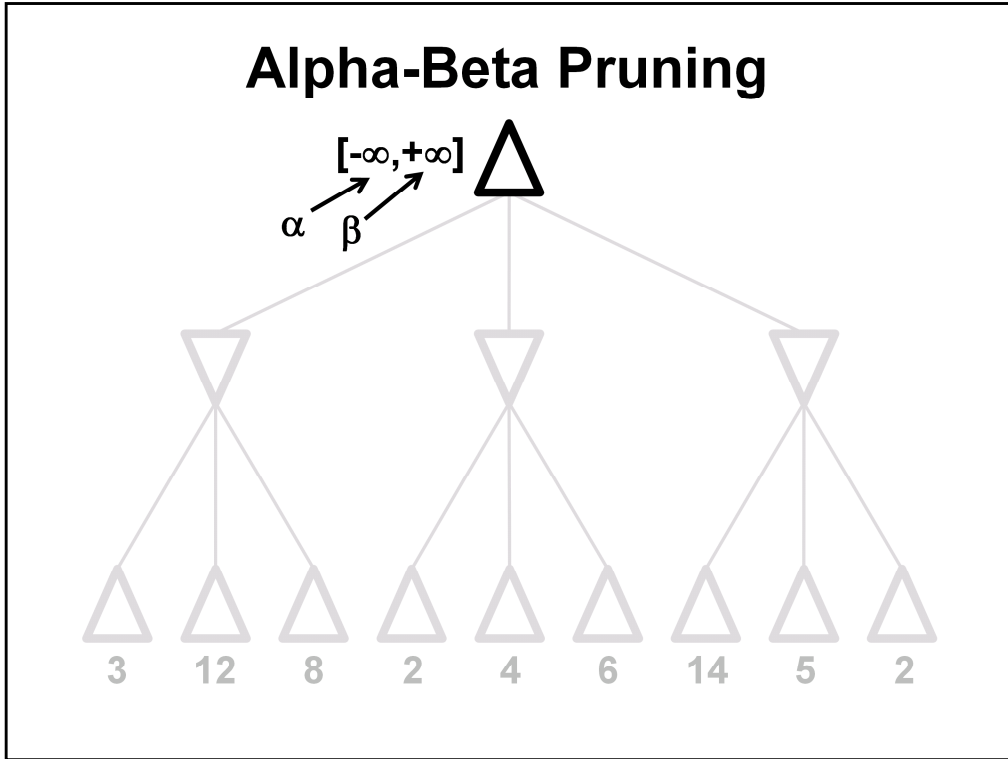
Multi-Player Games

For multi-player games, a vector is associated with each node, where the entries represent the values of the node to each player

If it is player X's turn, then player X will try to maximize his or her score.




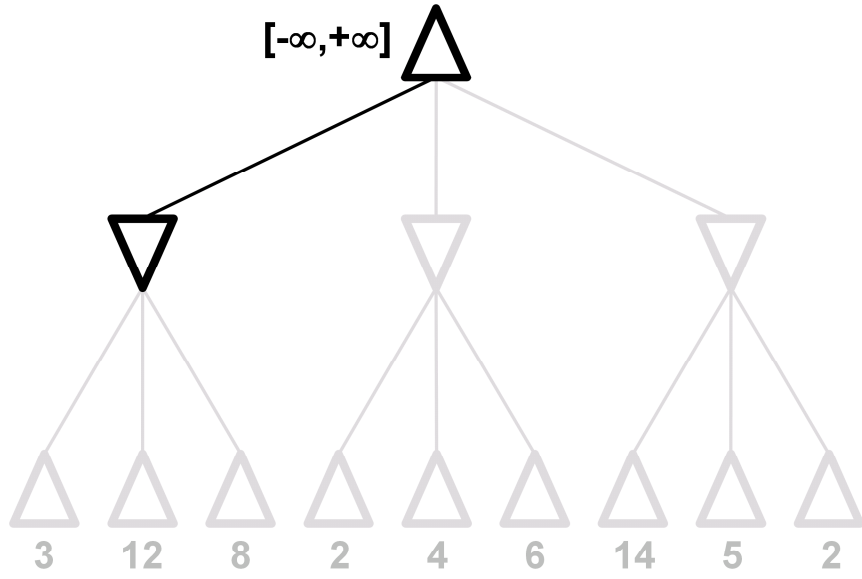
Answer to rhetorical question: "No". We do not need to look at the other nodes from the middle branch after seeing "2", because we know our MIN opponent will select something that has at most a payout of 2 for us, and we can already get 3 by going left.

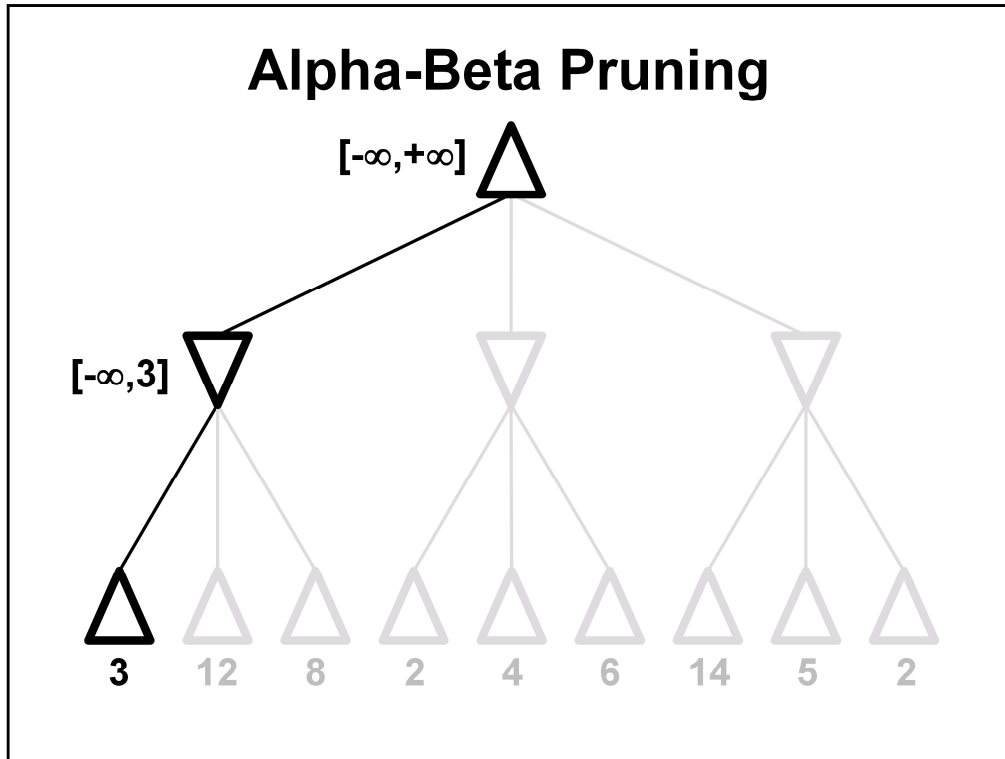


Alpha and beta are initialized to negative and positive infinity

Alpha-Beta Pruning

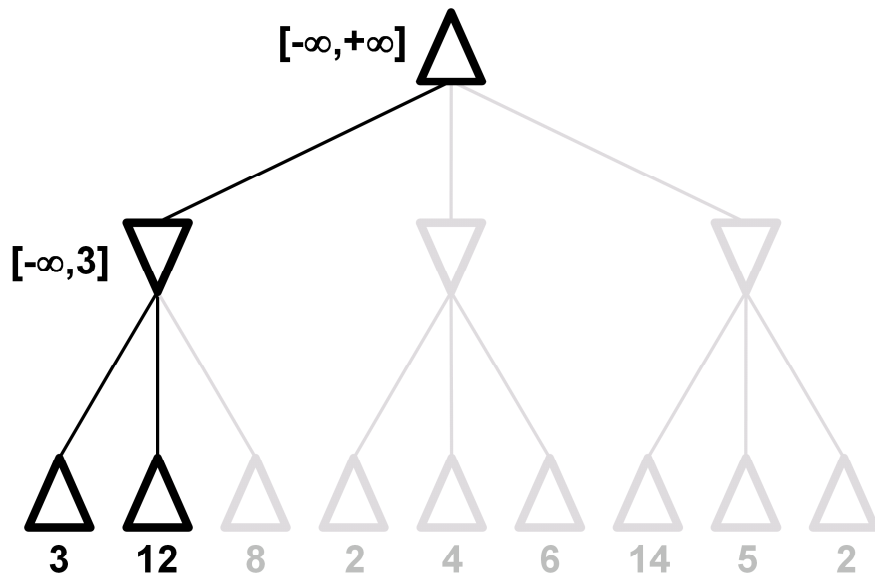
$[-\infty, +\infty]$ 

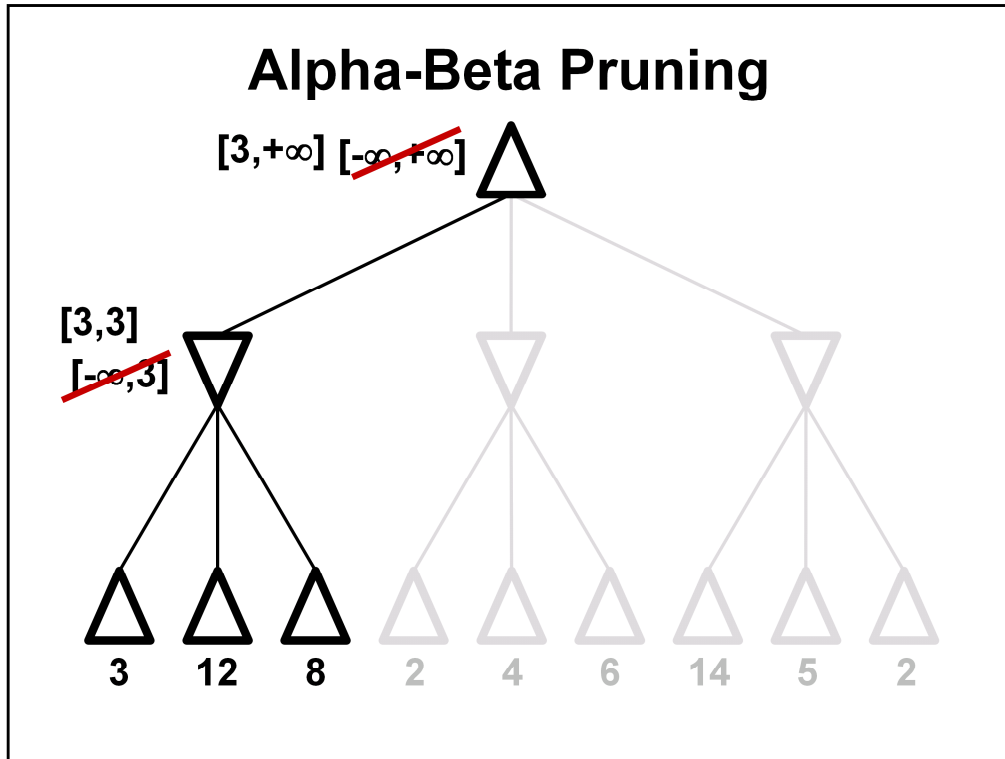




We set the beta of this min node to the max value of its successor.

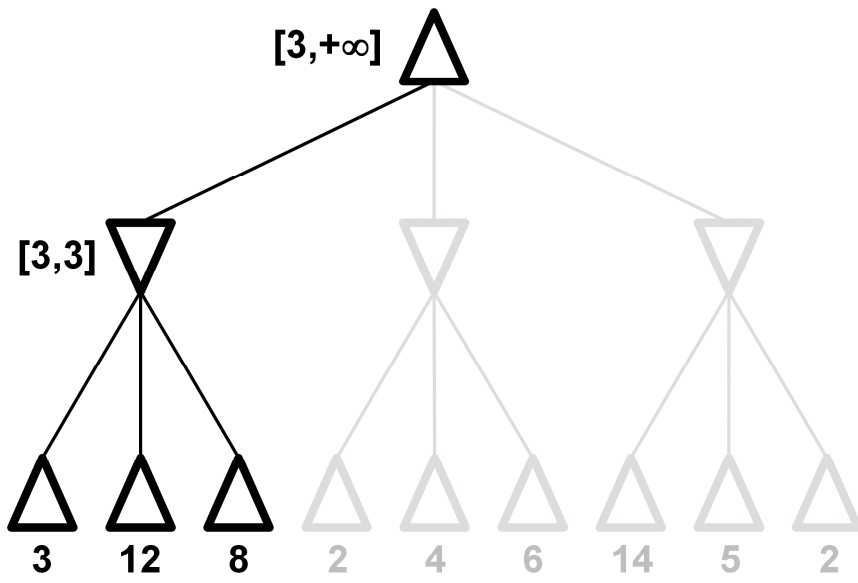
Alpha-Beta Pruning

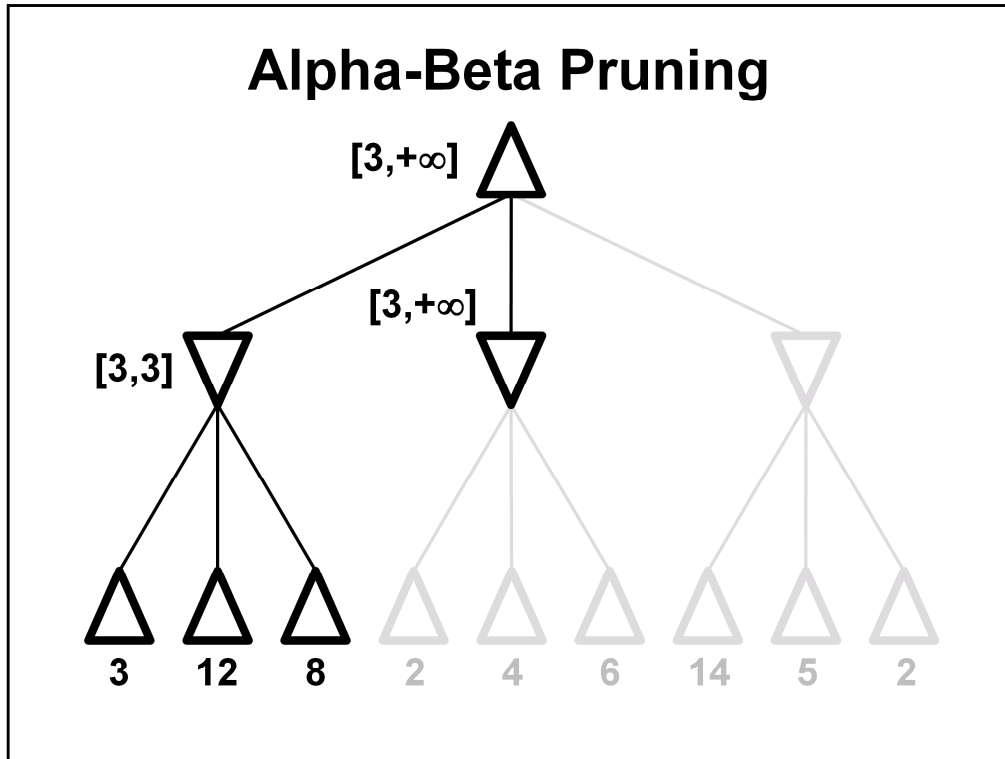




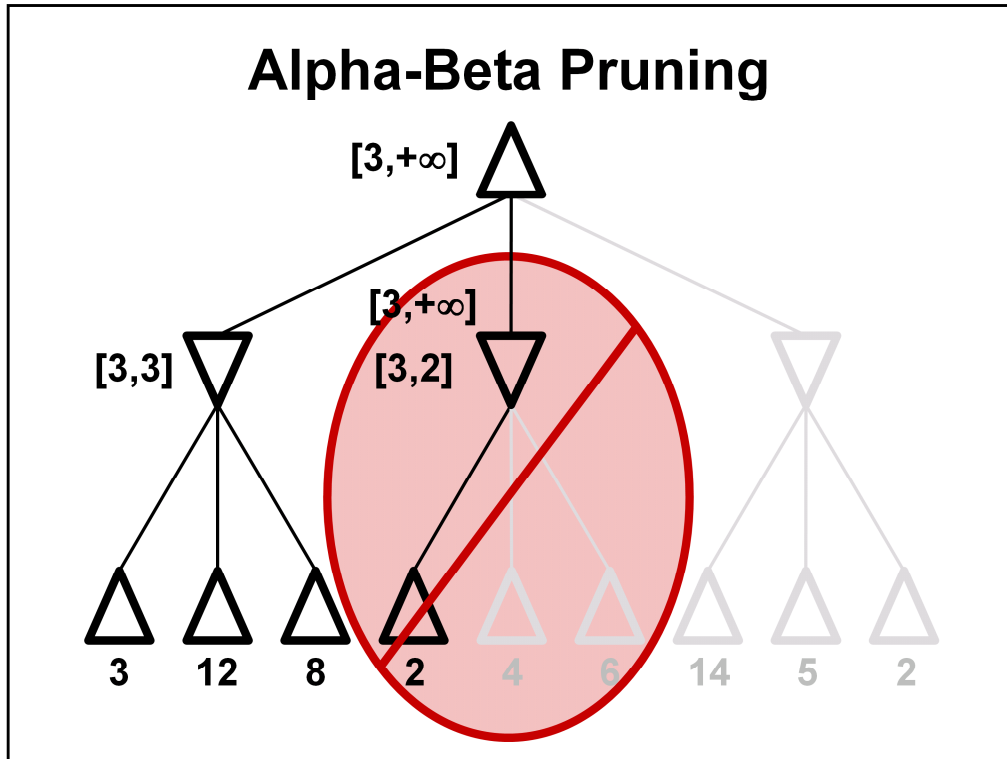
We have determined the value of the left move node is 3, so we set alpha = beta = 3 and propagate upwards.

Alpha-Beta Pruning



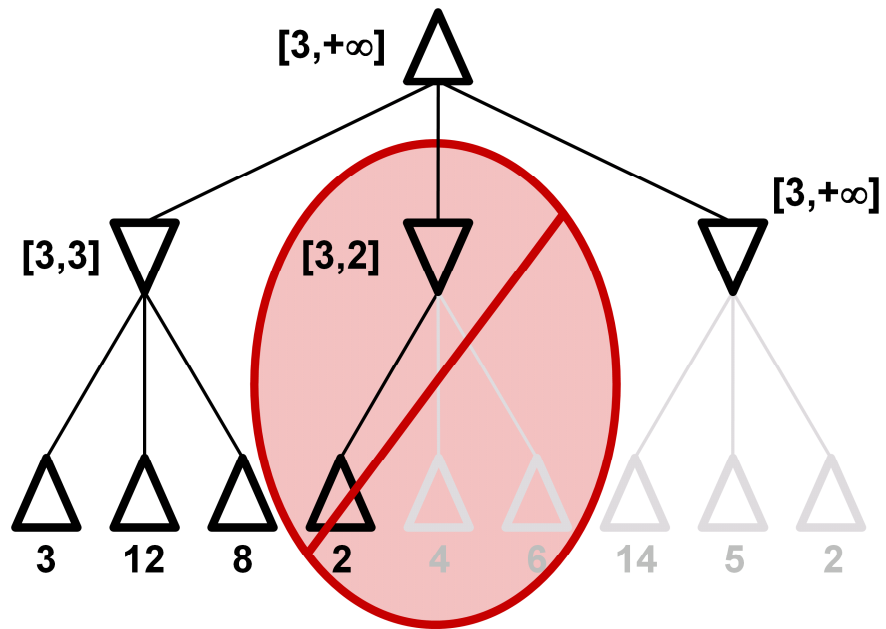


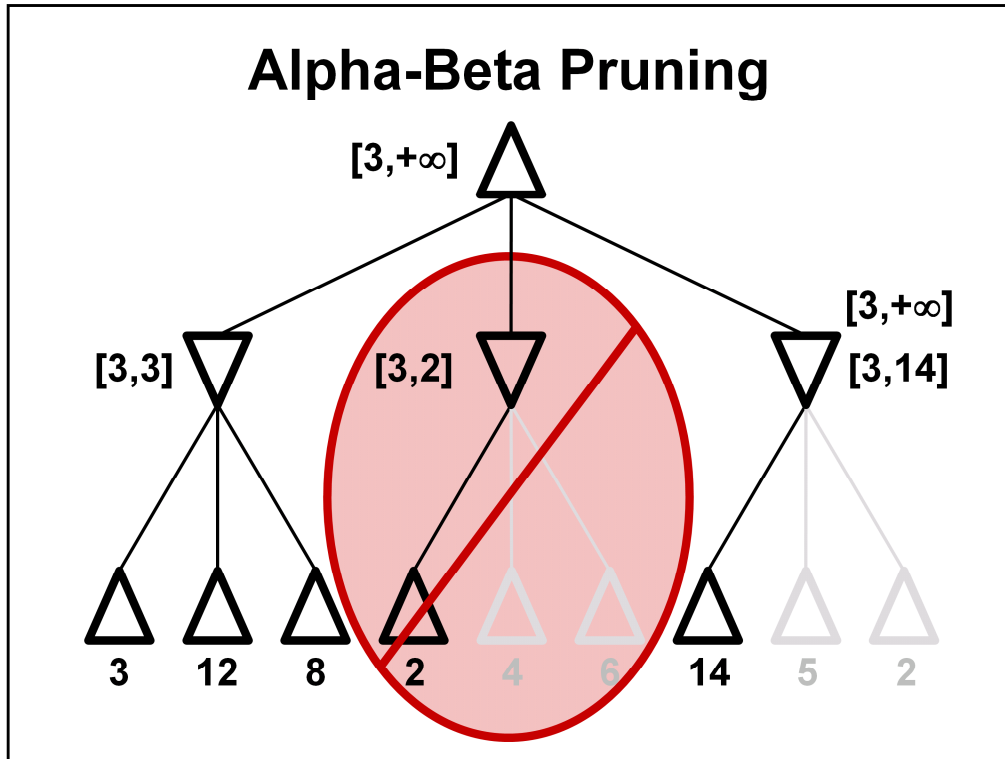
We pass down the 3 to our middle node search, because we know we can achieve it.



If $\alpha \geq \beta$, then prune!

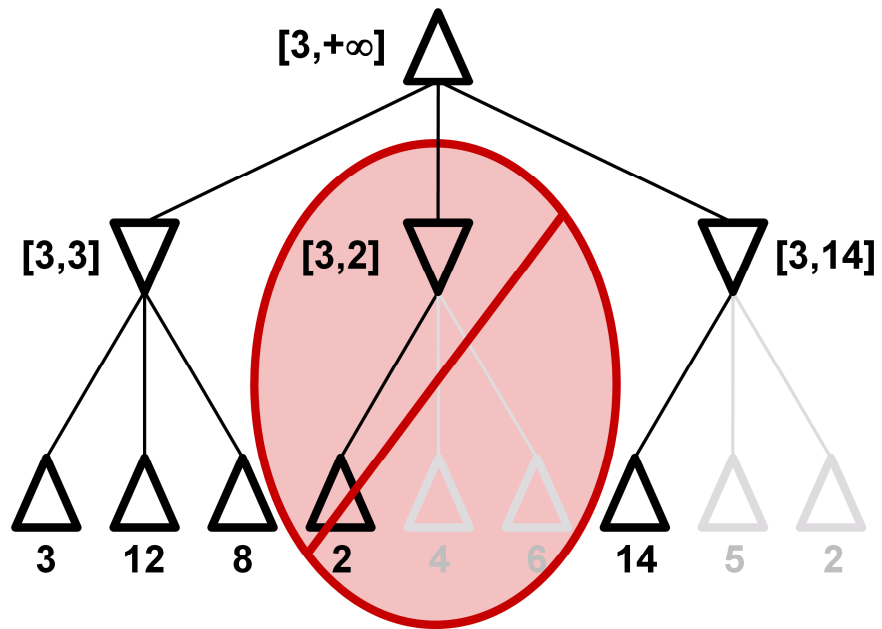
Alpha-Beta Pruning

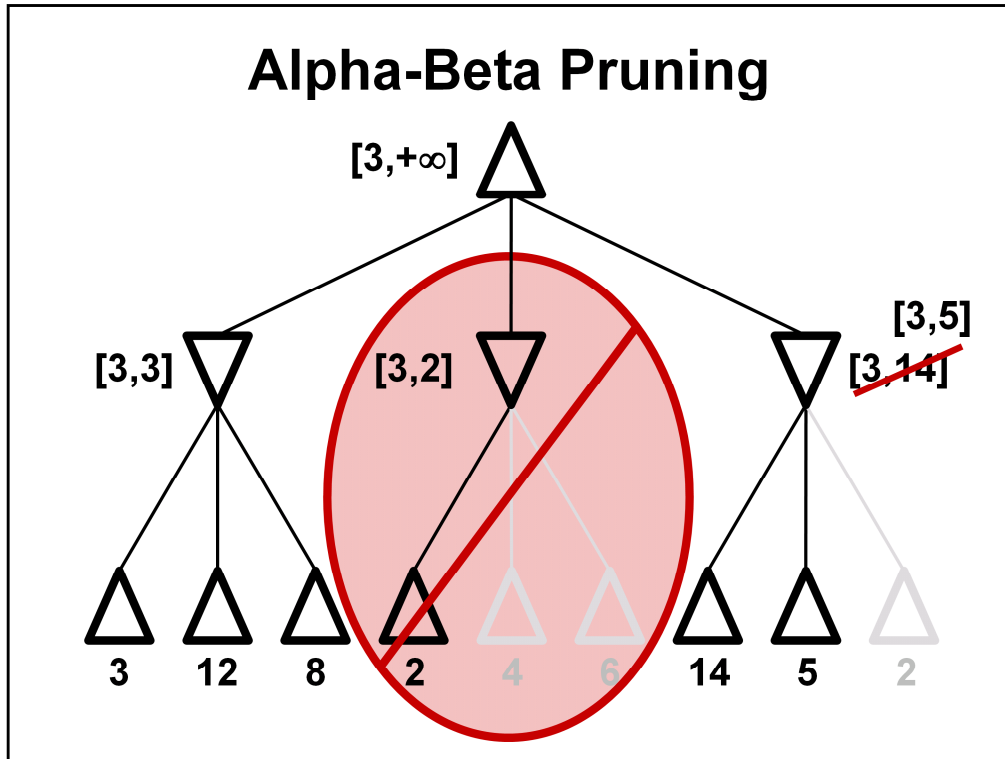




In this branch, we might be able to get 14

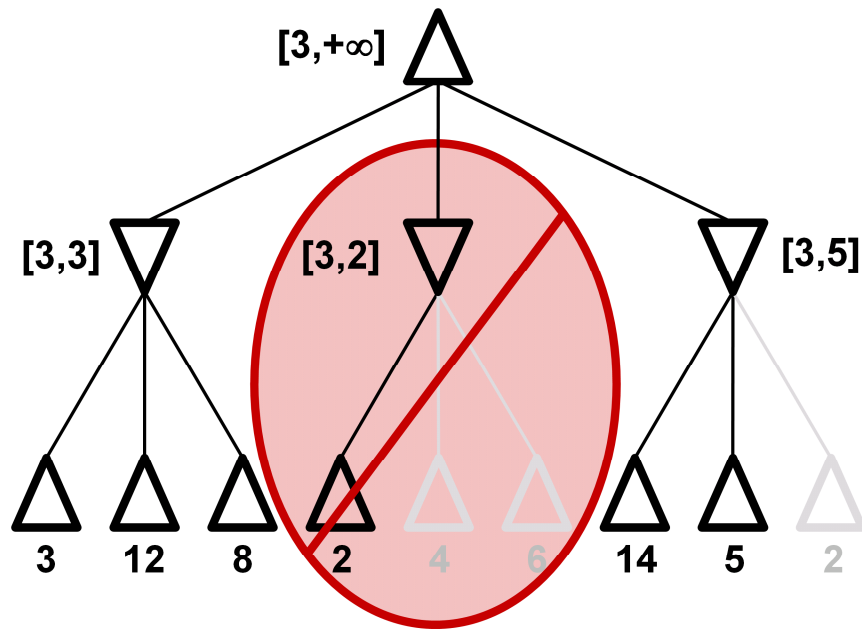
Alpha-Beta Pruning

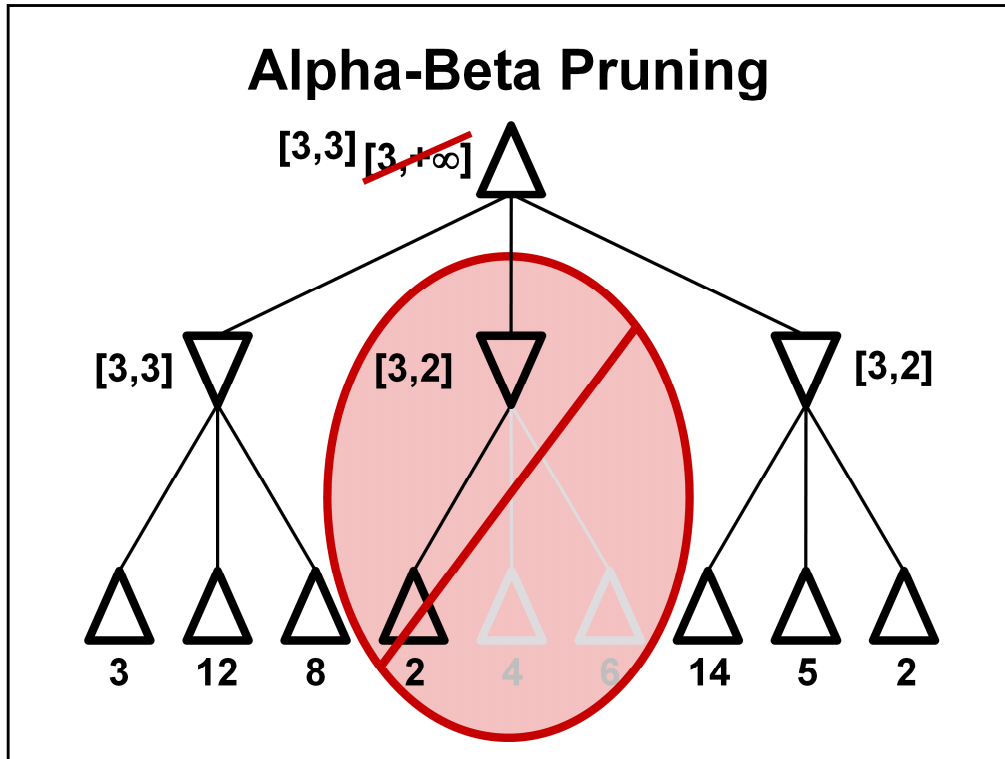




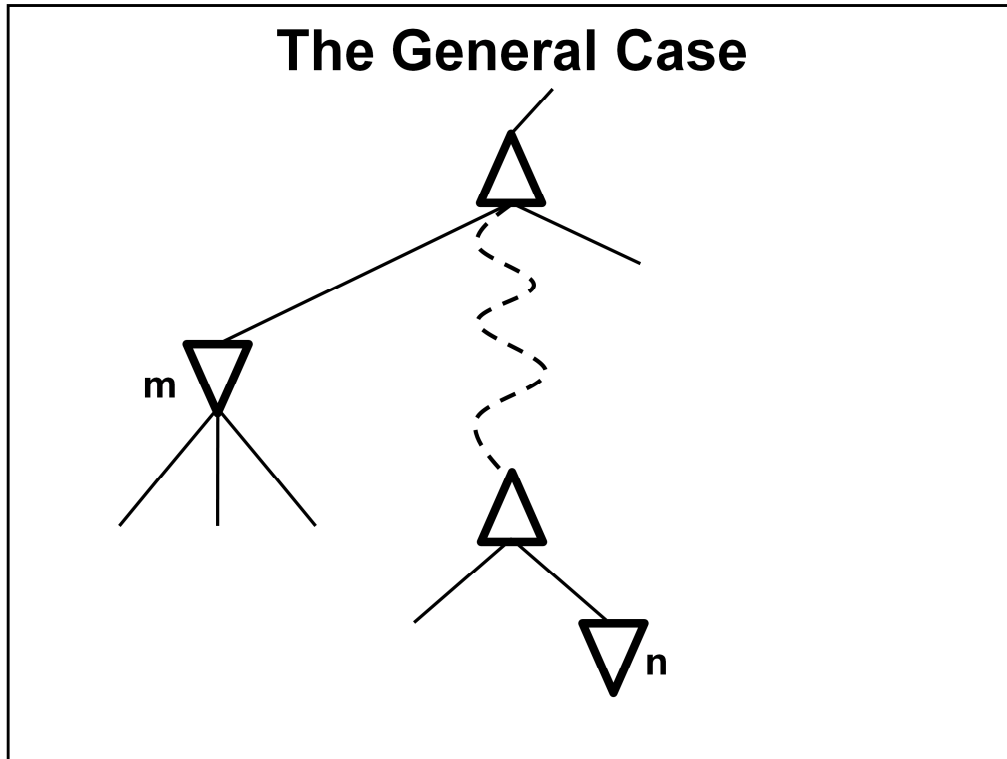
Now we see 5.

Alpha-Beta Pruning

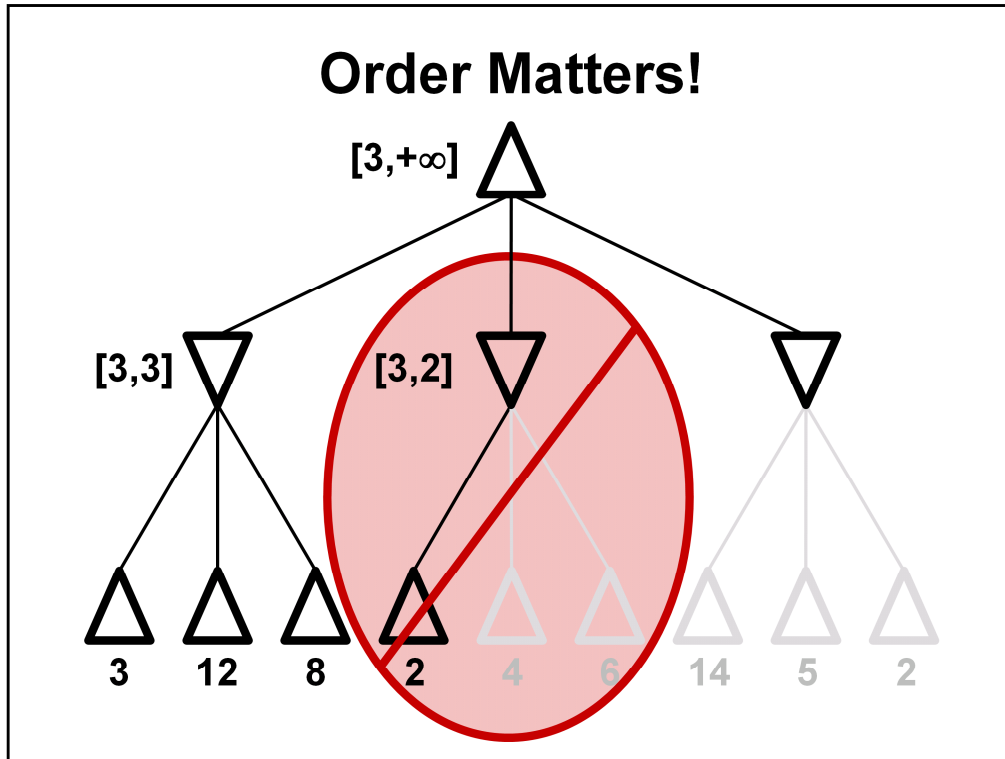




Now we see 2. If the right branch had more nodes, or if we looked at that node first we could prune as our alpha \geq beta



Imagine the $m > n$. We can terminate our search in the squiggly branch.



Here we do not need to look at 4 or 6, because we already have found $2 < 3$. If the order were 6,4,2 we would need to expand all the nodes in the middle branch.

Complexity

Suppose tree has average branching factor b and d ply:

Standard MINIMAX: $O(b^d)$

Worst Case Alpha-Beta: $O(b^d)$

Best Case Alpha-Beta: $O(b^{d/2})$

Average Case Alpha-Beta: $O(b^{3d/4})$

Can we prune at every level? No. Only every other level. Need to do half the branching as before.

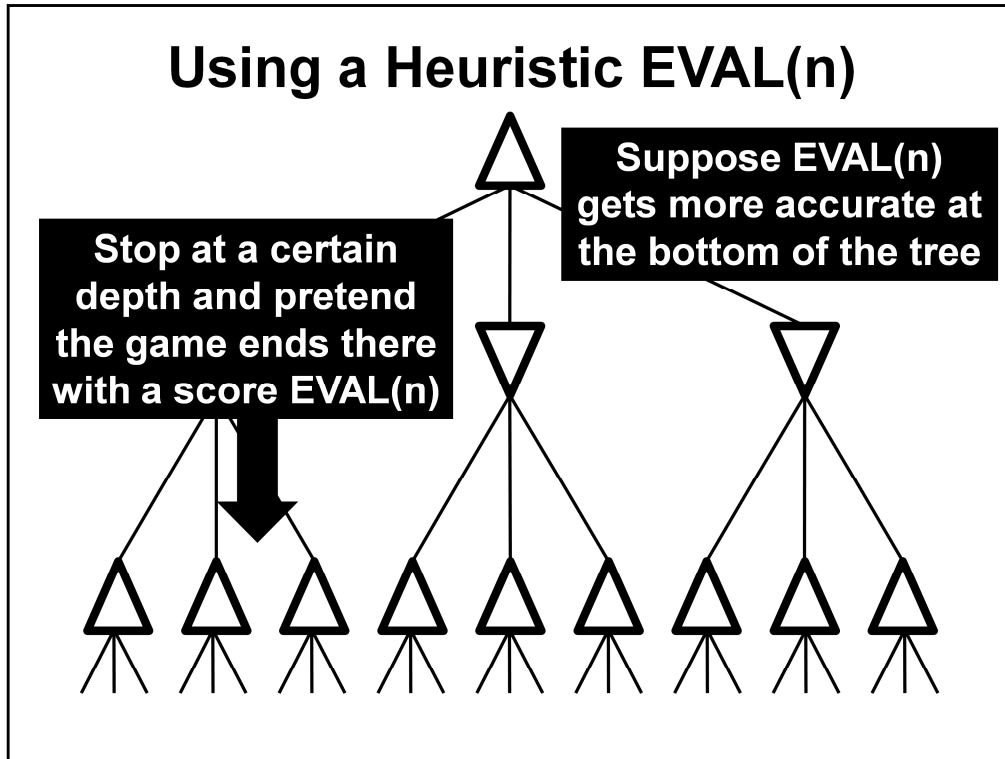
Can establish average-case bounds by considering that we'll find the max or min value after $b/2$ expansions (halfway between best and worst).

Chess

Average branching factor of chess: 35

35^{10} is very large!

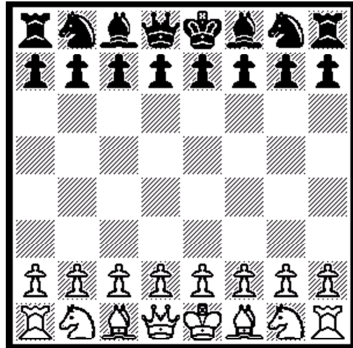
35^{10} is 10-ply lookahead.



Characteristics we want in our heuristic: fast, good (not necessarily an underestimate).

In general, eval gets better closer to the leaves. If eval was consistently accurate, then there would be little incentive to look far down the tree. You would just look at successors

Heuristics for Chess



Assign a value to each piece: Pawn = 1; Knight = 3; Bishop = 3; etc. Score of a position is the sum of the values of all my pieces minus all opponent's pieces.

Thought experiment: where do the values of pawns, knights, bishops, etc. come from? Can you think of ways to generate these values?

Deep Blue



Based on a research project from CMU

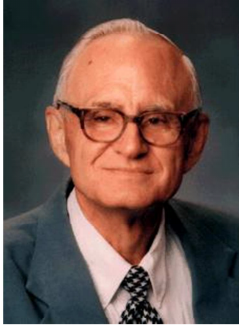
Alpha-Beta search with heuristics

Evaluation heuristic looked at over 8,000 features

Search looked 40 moves ahead

According to Manuela, the first big-time computer chess competition took place in the Wean Hall lounge.

Checkers



Dr. Marion Tinsley

vs CHINOOK

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: Perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times as complex as Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as Deep Blue for chess. Solving a game takes this to the next level by replacing the heuristics with perfection.

Games That Include Chance

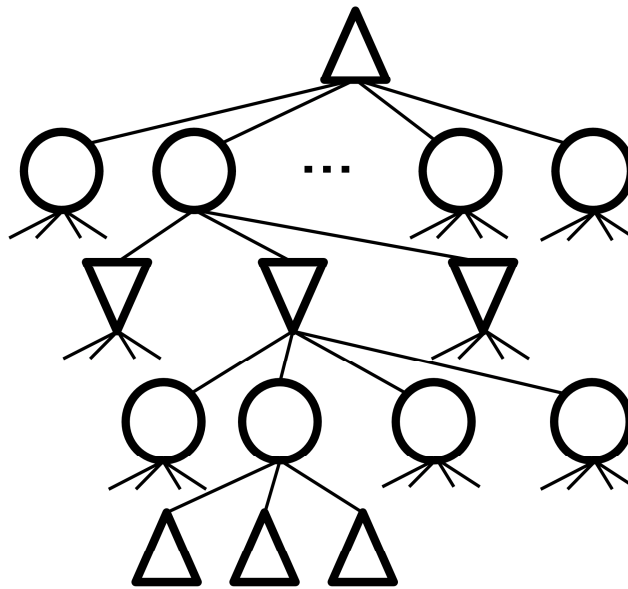
MAX

CHANCE

MIN

CHANCE

MAX



Sometimes we call chance nodes moves by the “nature” player.

The Expected Minimax Value

EXP-MINIMAX(Node n) =

$$\left\{ \begin{array}{ll} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{Succ}(n)} \text{E-MINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Succ}(n)} \text{E-MINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in \text{Succ}(n)} P(s) * \text{E-MINIMAX}(s) & \text{if } n \text{ is a CHANCE node} \end{array} \right.$$

Calculate the expected values at chance nodes.

FIN