Adversarial Search: Game Trees, Alpha-Beta Pruning; Imperfect Decisions
Resource Limits

**Problem:** In realistic games, cannot search to leaves!

**Solution:** Depth-limited search

- Instead, search only to a limited depth in the tree
- Replace terminal utilities with an evaluation function for non-terminal positions
Resource Limits

Example:

- Suppose we have 100 seconds and can explore 10K nodes/sec
- This means we can check 1M nodes per move
- \(\alpha-\beta\) pruning reaches about depth 8 – decent chess program

**Guarantee of optimal play is gone**
Depth Matters

Evaluation functions are always imperfect

The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters

An important example of the tradeoff between complexity of features and complexity of computation
Game Tree Pruning
Minimax Example

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Motivating Example

Image credit: Russell and Norvig
Minimax Pruning
Alpha-Beta Pruning

When applied to a standard minimax tree, it returns the same move as minimax would.

But always prunes away branches that cannot possibly influence the final decision.
What are alpha and beta?

\[ \alpha = \text{the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.} \]

\[ \beta = \text{the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.} \]
General Configuration (MIN Version)

- We’re computing the MIN-VALUE at some node $n$
- We’re looping over $n$’s children
- $n$’s estimate of the childrens’ min is dropping
- Who cares about $n$’s value? MAX
- Let $a$ be the best value that MAX can get at any choice point along the current path from the root
- If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$’s other children (it’s already bad enough that it won’t be played)

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
General Configuration (MAX Version)

The MAX version is simply symmetric
Alpha-Beta Implementation

$\alpha$: MAX’s best option on path to root
$\beta$: MIN’s best option on path to root

**def max-value(state, $\alpha$, $\beta$):**
- initialize $v = -\infty$
- for each successor of state:
  - $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$
  - if $v \geq \beta$ return $v$
  - $\alpha = \max(\alpha, v)$
- return $v$

**def min-value(state, $\alpha$, $\beta$):**
- initialize $v = +\infty$
- for each successor of state:
  - $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$
  - if $v \leq \alpha$ return $v$
  - $\beta = \max(\beta, v)$
- return $v$
This pruning has **no effect** on minimax value computed for the root!

Values of intermediate nodes might be wrong

- Important: children of the root may have the wrong value
- So the most naive version won’t let you do action selection
Demo: Minimax + Alpha-Beta Pruning

https://www.youtube.com/watch?v=_bEQJKXZ1-U
Good child ordering improves effectiveness of pruning

With “perfect ordering”:

- Time complexity drops to $O(b^{m/2})$
- Doubles solvable depth!
- Full search of, e.g., chess, is still hopeless…

This is a simple example of metareasoning (computing about what to compute)
Quiz
Evaluation Functions
It turns out that alpha-beta pruning isn’t so good...

It must search all the way to terminal states for at least a portion of the search space.

This is usually not practical, because we need to play the game in a reasonable amount of time.

Shannon’s suggestion: cutoff earlier via a heuristic **evaluation function**
Cutoff Test

\[
H\text{-MINIMAX}(s, d) =
\begin{cases}
\text{EVAL}(S) & \text{if CUTOFF-TEST}(s, d) \\
\max_{\alpha \in \text{Actions}(s)} = H\text{-MINIMAX}(\text{RESULT}(s, \alpha), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{\alpha \in \text{Actions}(s)} = H\text{-MINIMAX}(\text{RESULT}(s, \alpha), d + 1) & \text{if PLAYER}(s) = \text{MIN}
\end{cases}
\]
Evaluation Functions

Evaluation functions score non-terminals in depth-limited search.

Black to move
White slightly better

White to move
Black winning
Evaluation Functions

Ideal function: returns the actual minimax value of the position

In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g. \(f_1(s) = \text{(num white queens} - \text{num black queens)})\), etc.
Be wary of simple approaches

(b) White to move

Heuristic: Material Advantage

Image credit: Russell and Norvig
Be wary of simple approaches

(b) White to move

Probable win for black
Be wary of simple approaches
Horizon Effect

When the program is facing an opponent’s move that causes serious damage and is ultimately unavoidable, but can be temporarily avoided by delaying tactics.
Horizon Effect

Inevitable Loss

Image credit: Russell and Norvig
Horizon Effect

The loss is simply delayed

Image credit: Russell and Norvig
Search vs. Lookup

There are many standard openings and closings in chess.

Why bother with search when you can simply use a lookup table?
Search vs. Lookup

Computers are particularly good at the endgame

Example: king, bishop, and knight vs. king

462 ways a king can be placed without being adjacent

62 empty squares for the bishop, 61 for the knight, and 2 players to move next

$$462 \times 62 \times 61 \times 2 = 3,494,568$$ possible positions