Games and Adversarial Search II
Alpha-Beta Pruning (AIMA 5.3)

Some slides adapted from Richard Lathrop, USC/ISI, CS 271
Review: The Minimax Rule

_Idea:_ Make the best move for MAX assuming that MIN always replies with the best move for MIN

1. Start with the current position as a MAX node.
2. Expand the game tree a fixed number of _ply_.
3. Apply the evaluation function to all leaf positions.
4. Calculate back-up values bottom-up:
   - For a **MAX** node, return the _maximum_ of the values of its children (*i.e. the best for MAX*)
   - For a **MIN** node, return the _minimum_ of the values of its children (*i.e. the best for MIN*)
5. Pick the move assigned to MAX at the root
6. Wait for MIN to respond and _REPEAT FROM 1_
2-ply Example: Backing up values

This is the move selected by minimax

New point: Actually calculated by DFS!
Minimax Algorithm

**function** MINIMAX-DECISION(*state*) **returns** an action

**inputs:** *state*, current state in game

\[
\nu \leftarrow \text{MAX-VALUE(*state*)}
\]

**return** an action in SUCCESSORS(*state*) with value \( \nu \)

**function** MAX-VALUE(*state*) **returns** a utility value

**if** TERMINAL-TEST(*state*) **then** return UTILITY(*state*)

\[
\nu \leftarrow -\infty
\]

**for** \( a, s \) in SUCCESSORS(*state*) **do**

\[
\nu \leftarrow \text{MAX}(\nu, \text{MIN-VALUE}(s))
\]

**return** \( \nu \)

**function** MIN-VALUE(*state*) **returns** a utility value

**if** TERMINAL-TEST(*state*) **then** return UTILITY(*state*)

\[
\nu \leftarrow \infty
\]

**for** \( a, s \) in SUCCESSORS(*state*) **do**

\[
\nu \leftarrow \text{MIN}(\nu, \text{MAX-VALUE}(s))
\]

**return** \( \nu \)
Alpha-Beta Pruning

- A way to improve the performance of the Minimax Procedure
- Basic idea: “If you have an idea which is surely bad, don’t take the time to see how truly awful it is” ~ Pat Winston

- We don’t need to compute the value at this node.
- No matter what it is it can’t effect the value of the root node.
Alpha-Beta Pruning II

• During Minimax, keep track of two additional values:
  • $\alpha$: current *lower* bound on MAX’s outcome
  • $\beta$: current *upper* bound on MIN’s outcome

• MAX will never choose a move that could lead to a worse score (for MAX) than $\alpha$

• MIN will never choose a move that could lead to a better score (for MAX) than $\beta$

• Therefore, stop evaluating a branch whenever:
  • When evaluating a MAX node: a value $v \geq \beta$ is backed-up
    — MIN will never select that MAX node
  • When evaluating a MIN node: a value $v \leq \alpha$ is found
    — MAX will never select that MIN node
Alpha-Beta Pruning Illa

- Based on observation that for all viable paths utility value $f(n)$ will be $\alpha \leq f(n) \leq \beta$

- Initially, $\alpha = -\infty$, $\beta = \infty$

- As the search tree is traversed, the possible utility value window shrinks as $\alpha$ increases, $\beta$ decreases
Whenever the current ranges of alpha and beta no longer overlap, it is clear that the current node is a dead end.
Alpha-beta Algorithm: In detail

- Depth first search (usually bounded, with static evaluation)
  - only considers nodes along a single path from root at any time

\[
\alpha = \text{current } \textit{lower} \text{ bound on MAX’s outcome} \\
\quad \text{(initially, } \alpha = -\infty) \\
\beta = \text{current } \textit{upper} \text{ bound on MIN’s outcome} \\
\quad \text{(initially, } \beta = +\infty) \\
\]

- Pass current values of \( \alpha \) and \( \beta \) \textit{down} to child nodes during search.

- Update values of \( \alpha \) and \( \beta \) during search:
  - MAX updates \( \alpha \) at MAX nodes
  - MIN updates \( \beta \) at MIN nodes

- Prune remaining branches at a node whenever \( \alpha \geq \beta \)
When to Prune

Prune whenever \( \alpha \geq \beta \).

- Prune below a Max node when its \( \alpha \) value becomes \( \geq \) the \( \beta \) value of its ancestors.
  - **Max nodes update** \( \alpha \) based on children’s returned values.
  - Idea: Player MIN at node above won’t pick that value anyway, he can force a worse value.

- Prune below a Min node when its \( \beta \) value becomes \( \leq \) the \( \alpha \) value of its ancestors.
  - **Min nodes update** \( \beta \) based on children’s returned values.
  - Idea: Player MAX at node above won’t pick that value anyway; she can do better.
Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(state) returns an action

inputs: state, current state in game

ν ← MAX-VALUE(state, -∞, +∞)

return an action in ACTIONS(state) with value ν
Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(state) returns an action

inputs: state, current state in game

\[
v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)
\]

return an action in ACTIONS(state) with value \( v \)

---

function MAX-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

\[
v \leftarrow -\infty
\]

for a in ACTIONS(state) do

\[
v \leftarrow \text{MAX}(v, \text{MIN-VALUE(Result(s,a), } \alpha \text{, } \beta))
\]

if \( v \geq \beta \) then return \( v \)

\[
\alpha \leftarrow \text{MAX}(\alpha \text{,} v)
\]

return \( v \)
Alpha-Beta Algorithm II

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a,s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
    β ← MIN(β, v)
    return v
```
An Alpha-Beta Example

Do DF-search until first leaf

\( \alpha, \beta, \text{initial values} \)

\( \alpha = -\infty \)

\( \beta = +\infty \)

\( \alpha, \beta, \text{passed to kids} \)

\( \alpha = -\infty \)

\( \beta = +\infty \)
MIN updates $\beta$, based on kids
Alpha-Beta Example (continued)

MIN updates $\beta$, based on kids.
No change.
Alpha-Beta Example (continued)

MAX updates $\alpha$, based on kids.

$\alpha = 3$

$\beta = +\infty$

3 is returned as node value.
Alpha-Beta Example (continued)

\[
\begin{align*}
\alpha &= 3 \\
\beta &= +\infty
\end{align*}
\]

\[a = 3 \quad b = +\infty\]

\(\alpha, \beta, \text{passed to kids}\)

\[
\begin{align*}
\alpha &= 3 \\
\beta &= +\infty
\end{align*}
\]
Alpha-Beta Example (continued)

MIN updates $\beta$, based on kids.

$\alpha = 3$
$\beta = +\infty$

$\alpha = 3$
$\beta = 2$

CIS 521 - Intro to AI
Alpha-Beta Example (continued)

\[ a = 3 \]
\[ b = 2 \]
\[ a \geq b, \text{ so prune.} \]
MAX updates $\alpha$, based on kids.
No change.

2 is returned as node value.
Alpha-Beta Example (continued)

\[ \alpha = 3 \]
\[ \beta = +\infty \]

\(\alpha, \beta, \text{passed to kids}\)

\[ \alpha = 3 \]
\[ \beta = +\infty \]
Alpha-Beta Example (continued)

MIN updates $\beta$, based on kids.

$\alpha = 3$
$\beta = +\infty$

$\alpha = 3$
$\beta = 14$
Alpha-Beta Example (continued)

MIN updates $\beta$, based on kids.

$\alpha = 3$

$\beta = +\infty$
Alpha-Beta Example (continued)

\[ \alpha = 3 \]
\[ \beta = +\infty \]

2 is returned as node value.
Alpha-Beta Example (continued)

Max now makes it’s best move, as computed by Alpha-Beta
Effectiveness of Alpha-Beta Pruning

- Guaranteed to compute same root value as Minimax
- **Worst case:** no pruning, same as Minimax ($O(b^d)$)
- **Best case:** when each player’s best move is the first option examined, examines only $O(b^{d/2})$ nodes, allowing to search twice as deep!
When best move is the first examined, examines only $O(b^{d/2})$ nodes.

- So: run Iterative Deepening search, sort by value last iteration.
- So: expand captures first, then threats, then forward moves, etc.

- $O(b^{(d/2)})$ is the same as having a branching factor of $\sqrt{b}$,
  - Since $(\sqrt{b})^d = b^{(d/2)}$
  - e.g., in chess go from $b \sim 35$ to $b \sim 6$

- For Deep Blue, alpha-beta pruning reduced the average branching factor from 35-40 to 6, as expected, doubling search depth
Real systems use a few tricks

- Expand the proposed solution a little farther
  - Just to make sure there are no surprises
- Learn better board evaluation functions
  - E.g., for backgammon
- Learn model of your opponent
  - E.g., for poker
- Do stochastic search
  - E.g., for go
Chinook and Deep Blue

- **Chinook**
  - the World Man-Made Checkers Champion, developed at the University of Alberta.
  - Competed in human tournaments, earning the right to play for the human world championship, and defeated the best players in the world.

- **Deep Blue**
  - Defeated world champion Gary Kasparov 3.5-2.5 in 1997 after losing 4-2 in 1996.
  - Uses a parallel array of 256 special chess-specific processors
  - Evaluates 200 billion moves every 3 minutes; 12-ply search depth
  - Expert knowledge from an international grandmaster.
  - 8000 factor evaluation function tuned from hundreds of thousands of grandmaster games
  - Tends to play for tiny positional advantages.
FOR STUDY....
Example

-which nodes can be pruned?

3 4 1 2 7 8 5 6
Answer to Example

-which nodes can be pruned?

Answer:  **NONE!**  Because the most favorable nodes for both are explored last (i.e., in the diagram, are on the right-hand side).
Second Example
(the exact mirror image of the first example)

-which nodes can be pruned?
Answer to Second Example (the exact mirror image of the first example)

-which nodes can be pruned?

Answer: **LOTS!** Because the most favorable nodes for both are explored **first** (i.e., in the diagram, are on the left-hand side).