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

Minimax Algorithm

function MINIMAX-DECISION(state) returns an action
inputs: state, current state in game
v = MAX-VALUE(state)
return an action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v = -∞
for a in SUCCESSORS(state) do
  v = MAX(v, MIN-VALUE(a))
return v

function MIN-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v = +∞
for a in SUCCESSORS(state) do
  v = MIN(v, MAX-VALUE(a))
return v

Alpha-Beta Pruning (AIMA 5.3)

Some slides adapted from Richard Lathrop, USC/ISI, CS 271

Minimax Algorithm

function MINIMAX-DECISION(state) returns an action
inputs: state, current state in game
v = MAX-VALUE(state)
return an action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v = -∞
for a in SUCCESSORS(state) do
  v = MAX(v, MIN-VALUE(a))
return v

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

Alpha-Beta Pruning

- 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 IIIa

- 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

Alpha-Beta Pruning IIIb

- 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
  - \( \alpha \) = current lower bound on MAX’s outcome
    - (initially, \( \alpha = -\infty \))
  - \( \beta \) = current upper bound on MIN’s outcome
    - (initially, \( \beta = \infty \))
- Pass current values of \( \alpha \) and \( \beta \) 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
    \( v \leftarrow \text{MAX-VALUE} (\text{state}, -\infty, +\infty) \)
    return an action in ACTIONS(state) with value \( v \)
```
Pseudocode for Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(state) returns an action
inputs: state, current state in game

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

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

function MAX-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value
inputs: state, \( \alpha \), \( \beta \), \( -\infty \) \( +\infty \)
if TERMINAL-TEST(state) then return UTILITY(state)
\[ \alpha \leftarrow \infty \]
for \( a \) in ACTIONS(state) do
\[ \alpha \leftarrow \max(\alpha, \text{MIN-VALUE}(\text{Result}(s, a), \alpha, \beta)) \]
if \( \alpha \geq \beta \) then return \( \alpha \)
\( a \leftarrow \max(a, \alpha) \)
return \( \alpha \)

CIS 521 - Intro to AI

An Alpha-Beta Example

Do DF-search until first leaf

\( \alpha, \beta \), initial values

\( \alpha, \beta \), passed to kids

MAX

\( \alpha = -\infty \)

\( \beta = +\infty \)

MIN

\( \alpha = -\infty \)

\( \beta = +\infty \)

\( a \), \( b \), passed to kids

CIS 521 - Intro to AI

Alpha-Beta Algorithm II

function MIN-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value
inputs: state, \( \alpha \), \( \beta \), \( -\infty \) \( +\infty \)
if TERMINAL-TEST(state) then return UTILITY(state)
\[ \beta \leftarrow -\infty \]
for \( a \) in SUCCESSORS(state) do
\[ \beta \leftarrow \min(\beta, \text{MAX-VALUE}(\text{Result}(s, a), \alpha, \beta)) \]
if \( \beta \leq a \) then return \( \beta \)
\( \beta \leftarrow \min(\beta, b) \)
return \( \beta \)

CIS 521 - Intro to AI

Alpha-Beta Example (continued)

MAX

\( \alpha = -\infty \)

\( \beta = +\infty \)

MIN

\( \alpha = -\infty \)

\( \beta = +\infty \)

\( \alpha = \beta = 3 \)

\( \beta = 3 \)

MIN updates \( \beta \), based on kids.

CIS 521 - Intro to AI

Alpha-Beta Example (continued)

MAX

\( \alpha = -\infty \)

\( \beta = +\infty \)

MIN

\( \alpha = -\infty \)

\( \beta = +\infty \)

\( \alpha = 3 \)

\( \beta = 3 \)

MAX updates \( \alpha \), based on kids.

MIN updates \( \beta \), based on kids.

3 is returned as node value.

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

MAX

MIN

\( \alpha = 3 \)
\( \beta = +\infty \)

\( \alpha, \beta \) passed to kids

\( \alpha = 3 \)
\( \beta = +\infty \)

\( a \geq b \), so prune.

2 is returned as node value.

MAX updates \( \alpha \), based on kids. No change.

MIN updates \( \beta \), based on kids.

\( a \geq b \), so prune.

\( a = 3 \)
\( b = +\infty \)

2 is returned as node value.

MAX updates \( \alpha \), based on kids.

MIN updates \( \beta \), based on kids.
Alpha-Beta Example (continued)

\[
\begin{array}{c}
\text{MAX} \\
\Downarrow \\
\text{MIN} \\
\Downarrow \\
3 \\
\Downarrow \\
8 \\
\Downarrow \\
2 \\
\Downarrow \\
12 \\
\Downarrow \\
\alpha = 3 \\
\Downarrow \\
\beta = +\infty \\
\Downarrow \\
\beta = 5 \\
\Downarrow \\
2
\end{array}
\]

MIN updates \( \beta \) based on kids. \( \alpha = 3 \)

\( 2 \) is returned as node value.

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 ~ 35 to b ~ 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?

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).