Game-playing AIs: Games and Adversarial Search FINAL SET (w/ $\alpha\beta$ pruning study examples)

AIMA 5.1-5.2
Games: Outline of Unit

Part I: Games as Search
• Motivation
• Game-playing AI successes
• Game Trees
• Evaluation Functions

Part II: Adversarial Search
• The Minimax Rule
• Alpha-Beta Pruning
The Minimax Rule (AIMA 5.2)
The Minimax Rule: `Don’t play hope chess’

*Idea*: Make the best move for **MAX** assuming that **MIN** always replies with the best move for **MIN**

Easily computed by a recursive process

- The *backed-up value* of each node in the tree is determined by the values of its children:
  - For a **MAX** node, the backed-up value is the *maximum* of the values of its children (i.e. the best for **MAX**)
  - For a **MIN** node, the backed-up value is the *minimum* of the values of its children (i.e. the best for **MIN**)

CIS 391 - Intro to AI
The Minimax Procedure

Until game is over:

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 the leaf positions.
4. Calculate back-up values bottom-up.
5. Pick the move assigned to MAX at the root
6. Wait for MIN to respond
2-ply Example: Backing up values

Evaluation function value

This is the move selected by minimax
What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally:
  - *Maximizes worst-case outcome* for MAX.
  - (Classic game theoretic strategy)

- But if MIN does not play optimally, MAX will do even better. [Theorem-not hard to prove]
Comments on Minimax Search

• Depth-first search with fixed number of ply $m$ as the limit.
  • $O(b^m)$ time complexity – *As usual!*
  • $O(bm)$ space complexity

• Performance will depend on
  • the quality of the static evaluation function (expert knowledge)
  • depth of search (computing power and search algorithm)

• Differences from normal state space search
  • Looking to make *one* move only, despite deeper search
  • No cost on arcs – costs from backed-up static evaluation
  • MAX can’t be sure how MIN will respond to his moves

• Minimax forms the basis for other game tree search algorithms.
Alpha-Beta Pruning (AIMA 5.3)

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

=2

>=2

=>1

<=1

=2, 7, 1, ?

- 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:
  - $\alpha$: MAX’s current *lower* bound on MAX’s outcome
  - $\beta$: MIN’s current *upper* bound on MIN’s outcome

- MAX will never allow a move that could lead to a worse score (for MAX) than $\alpha$
- MIN will never allow 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
Alpha-Beta Pruning IIIc

- Whenever the current ranges of alpha and beta no longer overlap, it is clear that the current node is a dead end.
Tic-Tac-Toe Example with Alpha-Beta Pruning

Figure 4.17  Two-ply minimax applied to the opening move of tic-tac-toe.
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

\[ \alpha = \text{highest-value found at any point of current path for MAX} \]
\[ \text{(initially, } \alpha = -\infty) \]
\[ \beta = \text{lowest-value found at any point of current path for MIN} \]
\[ \text{(initially, } \beta = +\infty) \]

- 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 when \( \alpha \geq \beta \)
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

$\nu \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

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

\begin{algorithm}
\begin{algorithmic}
\Function{ALPHA-BETA-SEARCH}{state} \FunctionRet{an action}
\Inputs{state, current state in game}
\State $v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$
\Return{an action in ACTIONS(state) with value $v$}
\EndFunction
\Function{MAX-VALUE}{state, $\alpha$, $\beta$} \FunctionRet{a utility value}
\If{\text{TTERMINAL-TEST}(state)} \Return{UTILITY(state)} \EndIf
\State $v \leftarrow -\infty$
\For{$a$ in ACTIONS(state)}
\State $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{Result}(s,a), \alpha, \beta))$
\If{$v \geq \beta$} \Return{$v$} \EndIf
\State $\alpha \leftarrow \text{MAX}(\alpha, v)$
\EndFor
\Return{$v$}
\EndFunction
\end{algorithmic}
\end{algorithm}
function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
$v \leftarrow +\infty$
for $a,s$ in SUCCESSORS(state) do
$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$
if $v \leq \alpha$ then return $v$
$\beta \leftarrow \text{MIN}(\beta, v)$
return $v$
An Alpha-Beta Example

Do DF-search until first leaf

$\alpha, \beta, initial \ values$

$\alpha = -\infty$

$\beta = +\infty$

$\alpha, \beta, passed \ to \ kids$

$\alpha = -\infty$

$\beta = +\infty$
MIN updates $\beta$, based on kids

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

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

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

$\alpha = 3$

$\beta = +\infty$

3 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.

$\alpha = 3$

$\beta = +\infty$

$\alpha = 3$

$\beta = 2$
Alpha-Beta Example (continued)

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

\[ \alpha = 3 \quad \beta = 2 \]
\[ \alpha \geq \beta, \text{ so prune.} \]
Alpha-Beta Example (continued)

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.

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

$$\alpha = 3, \quad \beta = 5$$
Alpha-Beta Example (continued)

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

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

Max now makes its 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?
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).