Mini-Max vs Alpha-Beta

15-150
Principles of Functional Programming
April 9, 2019

Michael Erdmann & Karl Crary
Mini-Max at a Maxie Node

$v = \max\{v_1, \ldots, v_k\}$
alpha-beta pruning

• Rather than blindly evaluate children of a node, we pass an accumulator argument:

$$\alpha , \beta$$  (REQUIRE \(\alpha < \beta\))

• \(\alpha\) is the best Maxie can achieve in the tree explored so far.
• \(\beta\) is the best Minnie can achieve in the tree explored so far.

Think of \((\alpha , \beta)\) as an open interval of potential game values that both players would like better than their current best options.
alpha-beta pruning

• Rather than blindly evaluate children of a node, we pass an accumulator argument:

   $$(\alpha, \beta)$$  (REQUIRE $\alpha < \beta$$)

• $\alpha$ is the best Maxie can achieve in the tree explored so far.
• $\beta$ is the best Minnie can achieve in the tree explored so far.

**KEY POINT:** If Maxie sees a move to a node with value $v \geq \beta$,
then Maxie should stop searching from the current node.

Q: Why?
A: Minnie can prevent the game from reaching the current node.

(Dually, with roles of Maxie and Minnie reversed, now using “value $v \leq \alpha$".)
$\alpha_1 = \alpha$

$\alpha_2 = \max(\alpha_1, v_1)$

$\vdots$

$\alpha_i = \max(\alpha_{i-1}, v_{i-1})$

$v = \max\{v_1, \ldots, v_i\}$, with $i$ the smallest index such that $v_i \geq \beta$, if that occurs, and $i=k$ otherwise.

No need to explore these children.
Let’s hide the leaf values until the search needs them.
We’ll label nodes with values $v$ and edges with $(\alpha, \beta)$ intervals and update these dynamically as the search progresses.
Maxie

Minnie

Maxie

7

7 6 3

(−∞, +∞)

(−∞, +∞)
Maxie

Minnie

Maxie

$(\infty, +\infty)$

$(\infty, 7)$

$(8, 7)$

$(7, 7)$
Maxie

Minnie

Maxie

\[ (\geq 9) \]

\[ \leq \]

\[ (\geq 9) \]

\[ (\infty, +\infty) \]

\[ (\infty, 7) \]

\[ \beta \]

means “pruned” (in this case because at a Maxie level \( v \geq 9 > 7 = \beta \)).
Maxie

Minnie

Maxie

7

7

7

(\leq 9)

(8, 7)

(\infty, 7)

(\infty, +\infty)

7 6 3 9 2
Maxie

Minnie

Maxie
Maxie

Minnie

Maxie

\[(4, +\infty)\]

\[(4, 5)\]

\[(\geq 9)\]

\[4\]

\[7\]

\[6\]

\[3\]

\[9\]

\[2\]

\[4\]

\[1\]

\[1\]

\[3\]

\[5\]

\[3\]

\[7\]

\[6\]

\[3\]

\[9\]
If value here is 5, (rather than 8), Maxie still prunes and Minnie selects leftmost 5.
Maxie

Minnie

Maxie

\[
\begin{align*}
7 & \leq 9 \\
4 & \leq 8 \\
5 & \leq 6
\end{align*}
\]
Maxie

Minnie

Maxie

7
(≥)9
4
1
(≥)8
5
(≥)6
3

(5, +∞)

7
6 3 9
2 4 1
1 3
5 3
8
6
1 2 3
The “deep pruning” on the right occurs because at a Minnie level $v \leq 3 < 5 = \alpha$. 
Alpha-Beta saved 11 of 27 node evaluations

Maxie

Minnie

Maxie
Alpha-Beta saved 11 of 27 node evaluations

(Optimal game play indicated by arrows.)
functor AlphaBeta (Settings : SETTINGS) : PLAYER =
struct
  structure Game = Settings.Game

  Much as before, with some new helper types and functions.

  datatype bound = NEGINF | Bound of Game.est | POSINF
  type alphabeta = bound * bound

  Remember: (alpha,beta) describes knowledge that a search has.
  (EXAMPLE: POSINF and Definitely(Winner(Maxie)) are different.
  POSINF means the search does not yet know how well Minnie can do.)

  Combination of (alpha,beta) knowledge and move est guides search.

  datatype orderAB = BELOW | INTERIOR | ABOVE

  (* compareAB : alphabeta -> Game.est -> orderAB *)

You will write the new search and evaluation code.