CmSc310  Artificial Intelligence

Alpha-Beta Pruning

Basic idea:
Depth-first traversal
Each node has a value:
leaf node: score of the game (determined by the static evaluation function)
max node: alpha = max of beta values of its children
min node: beta = min of alpha values of its children

If alpha value of a max node is greater than the beta value of its parent – the max
node is irrelevant, should be cut off
If beta value of a min node is less than the alpha value of its parent – that min node is
irrelevant – should be cut off.

Two issues:
a) alpha and beta values are defined as max / min values of the children. This
implies that we need to find all children, i.e. expand the node down to its leaves.
However that is not always necessary.
   We compute the children in a loop over the possible moves at the current
node. For a max node, when we find a child whose value is greater than or equal
to the beta value of the node's parent, we can stop computing the remaining children –
the node is irrelevant.
   For a min node, when we find a child whose value is less than or equal to
the alpha value of the node's parent, we can stop computing the remaining children – the
node is irrelevant.

b) The algorithm outlined so far needs to have access to the parent of the currently
processed node. We can avoid looking up in the tree by passing to the generated child the
value of its parent. Thus each node has two values: alpha and beta.

• The root is initialized with alpha – the smallest possible value, beta – the largest
possible value
• At generation, both alpha and beta values are inherited from the parent.
• If max node: alpha is computed based on its children and compared with the beta
value (if greater, we stop)
• If min node: beta is computed based on the children and compared with the alpha
value. (if smaller, we stop)
method int MINIMAX_AB(N, A, B)
    begin
        AlphaN ← A
        BetaN ← B

        if N is a leaf then
            return the estimated score of this leaf
        else
            // process min nodes
            if N is a Min node then
                For each successor Ni of N
                    Val ← MINIMAX_AB(Ni, AlphaN, BetaN) // recursive call
                    BetaN ← Min(BetaN, Val)
                    If BetaN <= AlphaN then exit the “for” loop;
                Return BetaN;
            // process max nodes
            Else
                For each successor Ni of N
                    Val ← MINIMAX_AB(Ni, AlphaN, BetaN) // recursive call
                    AlphaN ← Max(AlphaN, Val)
                    If AlphaN >= BetaN then exit the “for” loop;
                Return AlphaN;
    end MINIMAX_AB;
Questions:

1. What information should be stored in each node? How do we know which node is a max node and which node is a min node? How do we know which node is a leaf?

2. What additional information should each non-leaf node have so that the results of the minimax algorithm can be used when playing the game?

3. What would be the structure of an algorithm that actually plays the game?

4. What happens if the opponent makes a mistake?