Artificial Intelligence Notes

Day:	29 December 2022
Topic:	Alpha Beta Pruning Technique
Key Message:	Every node has alpha (updated when Max turn) and beta (updated when Min turn) values. Generate tree: depth first & Left-to-Right
1st point:	Minmax algorithm return us all the nodes that are not possibly affecting the final decision, Alpha Beta Pruning return the same move as the standard one with removing (prune) all other extra nodes. • Optimization
2nd point:	• Reduces the number of nodes • Doesn't generate instead it rearranges and propagate the nodes This method is used for: • Shooting Time Complexity • Optimizing Minmax Algorithm • Reduce Size
3rd point:	If a move/node is determined worse than another move/node already examined, then there is no need for further examination of node.

4th point:	Step 1: Generate game tree and apply utility function. This utility function will generate terminal values for terminal nodes. Alpha = minus infinity Beta = plus infinity First Move: Maximizer Next Move: Minimizer
5th point:	Step 2: For max function the value of alpha will be compare and for min turn beta value will be compared.
6th point:	Step 3: Minimizer turns to compare the value with beta, algorithm backtracks to previous node and alpha value will be set to minus infinity again. While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
7th point:	Step 4: After Minimizer turn the successor node will be compared with the alpha as its max function, if alpha >= beta after first comparison then that value of alpha will be set to that node and no more traversals will occur or we can simply call it pruning. We will only pass the alpha, beta values to the child nodes.
8th point:	Effectiveness of alpha beta pruning depends upon the traversal order. • Worst Order: No pruning, best move occurs on right side, $O(b^m)$

	• Ideal Order: Lots of pruning, best move occurs on left side, $O(b^{m/2})$
9th point	Rules of thumb: • Alpha is best value for Max (Highest) • Beta is best value for Min (Lowest) • If Beta <= Alpha of some Max ancestor, then it maybe alpha-pruned in search below a Min node • If Alpha >= Beta of some Min ancestor, then it maybe beta-pruned in search below a Max node
Important vocabulary:	• Minimizer and Maximizer • Terminal Values using Utility Functions • Comparison of values for best score

What is the Alpha Beta Pruning technique?

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. We cannot eliminate the exponent depth of the tree in minmax, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm. Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also entire sub-tree.

The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Terminology

- **Alpha:** It is the best choice so far for the player MAX. We want to get the highest possible value here.
- **Beta:** It is the best choice so far for MIN, and it must be the lowest possible value.
- **Optimization:** Game trees are, in general, very time consuming to build, and it's only for simple games that it can be generated in a short time. If there are b legal moves, i.e., b nodes at each point and the maximum depth of the tree are m, the time complexity of the minimax algorithm is of the order $O(b^m)$. To curb this situation, there are a few optimizations that can be added to the algorithm. Fortunately, it is viable to find the actual minimax decision without even looking at every node of the game tree. Hence, we eliminate nodes from the tree without analyzing, and this process is called pruning.
- **Note:** Each node must keep track of its alpha and beta values. Alpha can be updated only when it's MAX 's turn and, similarly, beta can be updated only when it's MIN 's chance.

Alpha Beta Pruning Pseudocode

- Alpha = Max Value
- Beta = Min Value
- While max, cut off values lower than alpha
- While min, cut off values lower than beta

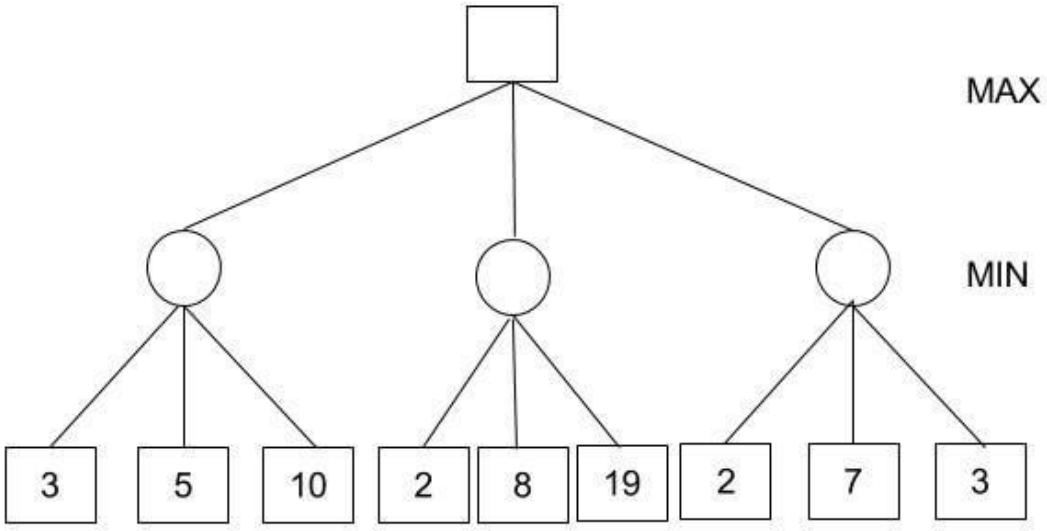
```
evaluate (node, alpha, beta)
    if node is a leaf
        return the heuristic value of node
    if node is a minimizing node
        for each child of node
            beta = min (beta, evaluate (child, alpha, beta))
            if beta <= alpha
                return beta
        return beta
    if node is a maximizing node
        for each child of node
            alpha = max (alpha, evaluate (child, alpha, beta))
            if beta <= alpha
                return alpha
        return alpha
```

When to prune?

The condition to prune a node is when alpha becomes greater than or equal to beta.

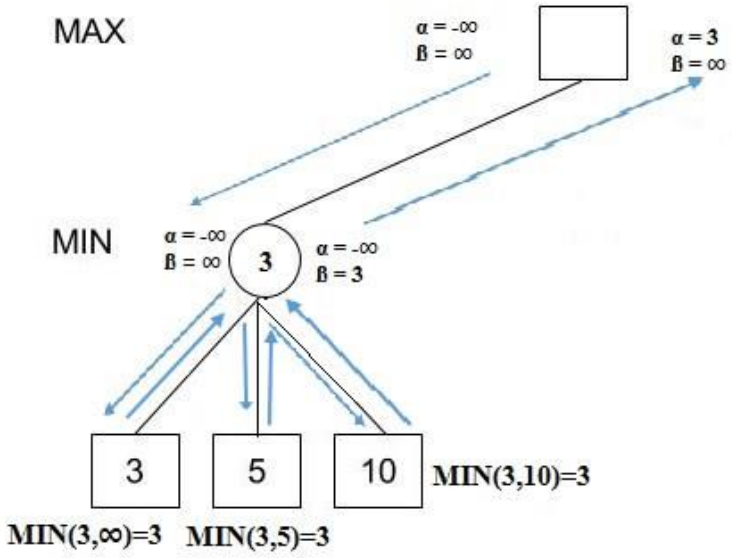
How does alpha-beta pruning work?

Step 1: Initialize alpha = -infinity and beta = infinity as the worst possible cases.



Step 2: Start with assigning the initial values of alpha and beta to root and since alpha is less than beta we don't prune it.

Step 3: Carry these values of alpha and beta to the child node on the left. And now from the utility value of the terminal state, we will update the values of alpha and be, so we don't have to update the value of beta. Again, we don't prune because the condition remains the same. Similarly, the third child node also. And then backtracking to the root we set alpha=3 because that is the minimum value that alpha can have.



Step 4: Now, alpha=3 and beta=infinity at the root. So, we don't prune. Carrying this to the center node, and calculating MIN{2, infinity}, we get alpha=3 and beta=2.

Step 5: Prune the second and third child nodes because alpha is now greater than beta.

Step 6: Alpha at the root remains 3 because it is greater than 2. Carrying this to the rightmost child node, evaluate MIN{infinity,2}=2. Update beta to 2 and alpha remains 3.

Step 7: Prune the second and third child nodes because alpha is now greater than beta.

Step 8: Hence, we get 3, 2, 2 at the left, center, and right MIN nodes, respectively. And calculating MAX{3,2,2}, we get 3. Therefore, without even looking at four leaves we could correctly find the minimax decision.