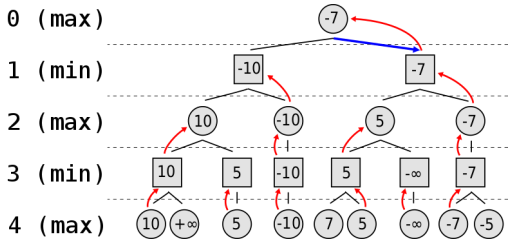
Steven Lam
Department of Computer Science,
California State, Chico
slam@csuchico.edu

Alpha - Beta Pruning Algorithm

Table of Contents

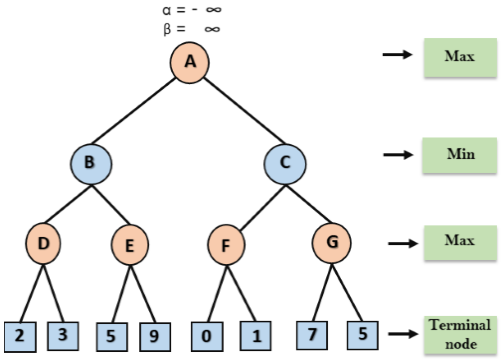
Introduction

The alpha-beta pruning algorithm is a search algorithm commonly used in decision trees and game trees to find the best possible move for a player in a game. It is a variant of the minimax algorithm, which is a backtracking algorithm that searches through all possible moves of a game and evaluates them based on a static evaluation function. The goal of the minimax algorithm is to find the move that maximizes the score for the current player while minimizing the score for the opponent. This is achieved by recursively exploring the tree of possible moves, alternating between maximizing the score for the current player and minimizing the score for the opponent at each level.



[1]

However, the minimax algorithm has a significant drawback: it explores all branches of the game tree, even those that are irrelevant to the final decision. This leads to a high computational cost, which is especially problematic in games with large branching factors, such as chess. The alpha-beta pruning algorithm addresses this issue by cutting off branches of the game tree that are determined to be irrelevant to the final decision. It does this by maintaining two values, alpha and beta, that represent the best score the current player can achieve and the best score the opponent can achieve, respectively, at any given point in the search [2].



[6]

As the alpha-beta pruning algorithm traverses the tree, it prunes branches that are guaranteed to be worse than a previously explored branch. This is achieved by comparing the current branch's score to alpha and beta and pruning it if it falls outside of the range between them. By cutting off these irrelevant branches, the alpha-beta pruning algorithm significantly reduces the number of nodes visited by the algorithm, making it much more efficient than the minimax algorithm. The result is a faster and more effective search algorithm that can be used to find the optimal move in games with large branching factors.

History

The alpha-beta pruning algorithm was first introduced by computer scientists John McCarthy and Marvin Minsky in their 1956 paper, "The Theory of Games and the Design of an Electronic Computer." McCarthy and Minsky were interested in developing algorithms for game-playing programs that could compete with human players. They recognized that the minimax algorithm, which was the dominant search algorithm at the time, was inefficient for games with large branching factors. [1]

The alpha-beta pruning algorithm was a breakthrough in game-playing algorithms, and it quickly became a standard technique in the field. However, it was not until the 1970s that researchers began to fully understand the theoretical underpinnings of the algorithm. In 1975, computer scientist Donald Knuth proved that the alpha-beta pruning algorithm was optimal in the sense that it always found the same move as the minimax algorithm, but with fewer node evaluations. [3]

Since its initial development, the alpha-beta pruning algorithm has been used in a wide range of applications beyond game playing, such as in artificial intelligence, computer vision, and natural language processing. Researchers have also continued to refine and improve the algorithm, developing variations such as the alpha-beta zero algorithm, which uses neural networks to evaluate game positions, and the parallel alpha-beta pruning algorithm, which exploits parallelism in modern hardware to speed up the search. [2][5][4]

Today, the alpha-beta pruning algorithm remains a fundamental technique in the field of artificial intelligence and game playing. Its development and refinement over the past several decades have contributed to significant advances in the field, and it continues to be a focus of research and innovation for computer scientists and game theorists alike.

Intuition

The alpha-beta pruning algorithm is a search algorithm used in game playing that reduces the number of nodes that need to be evaluated by cutting off branches of the game tree that are determined to be irrelevant to the final decision. This is achieved by maintaining two values, alpha and beta, that represent the best score the current player can achieve and the best score the opponent can achieve, respectively.

During the search, the algorithm compares the current branch's score to alpha and beta, pruning it if it falls outside of the range between them. By cutting off these irrelevant branches, the alpha-beta pruning algorithm significantly reduces the number of nodes visited by the algorithm, making it much more efficient than other search algorithms, particularly for games with large branching factors.

The alpha-beta pruning algorithm's efficiency is based on the fact that the search algorithm can avoid exploring certain parts of the game tree that are irrelevant to the final decision. This leads to a significant reduction in the number of nodes visited, making the algorithm much more efficient than other search algorithms. The alpha-beta pruning algorithm uses techniques such as recursion and backtracking to explore the game tree, keeping track of the best possible outcome at each level of the tree. It also relies on the concept of minimax, where the algorithm alternates between maximizing the score for the current player and minimizing the score for the opponent.

The alpha-beta pruning algorithm builds on these concepts by introducing the alpha and beta values, which serve as upper and lower bounds on the possible scores at each level of the tree. By maintaining these values and using them to prune irrelevant branches, the alpha-beta pruning algorithm provides a powerful tool for solving complex game-playing problems.

PseudoCode

```
function GenerateTree(node, depth, max_depth, branching_factor):
    if depth == max_depth:
        return
    for i in range(branching_factor):
        child = new Node()
        node.children.add(child)
        GenerateTree(child, depth+1, max_depth, branching_factor)

function SetTree(root, options, index)
    if node is a leaf node
        node.value = options[index]
        index = index + 1
        return node
    for each child of node
        return SetTree(child, options, index)

function AlphaBetaPruning(node, max_depth, alpha, beta,maximizingPlayer):
    if depth = 0:
        return node.value
    if maximizingPlayer:
        value = -infinity
        for child in node.children:
            value = max(value, AlphaBetaPruning(child, max_depth - 1, alpha, beta, False))
            alpha = max(alpha, value)
            if beta <= alpha:
                break
            return value
    else:
        value = infinity
        for child in node.children:
            value = min(value, AlphaBetaPruning(child, depth - 1, alpha, beta, True))
            beta = min(beta, value)
        if beta <= alpha:
            break
        return value
```

Explanation of PseudoCode Part I

GenerateTree:

The function GenerateTree takes four arguments: node, depth, max_depth, and branching_factor. It recursively generates a tree structure with node as the root node, using the specified max_depth and branching_factor. The function starts by checking if the depth has reached the max_depth, and if so, it simply returns without creating any more child nodes. If the depth is less than the max_depth, the function enters a loop that iterates branching_factor times. In each iteration, the function creates a new child node child, adds it to the set of children of the current node, and recursively calls itself with the new child node as the current node, depth+1 as the new depth, max_depth, and branching_factor. The function continues to call itself recursively until the depth reaches the max_depth, at which point it stops creating new child nodes and returns. This in turn, allows us to generate a tree dynamically that fits the user's specifications.

SetTree:

SetTree defines a recursive function that the root node of a tree, an index for the list, and a list of values to insert into the tre.. The function starts by setting the value of the current node to the given value. Then, it checks if the current node has any children. If it does, the function recursively calls itself on each child node, passing in the child node, the value, and the dictionary. The function essentially performs a depth-first traversal of the tree and sets the value of every node it visits to the given value, increasing the value of the index by 1 each time it sets a leaf node's value from left to right. This in turn, allows the tree to be generated to how the user wishes.

Explanation of PseudoCode Part II

AlphaBetaPruning:

The function takes in a node in the tree, the maximum depth of the tree to search, the alpha and beta values for the algorithm, and a boolean value indicating whether the current node is being maximized or minimized. The function first checks if the maximum depth has been reached, and if so, returns the value of the node.

If the node is being maximized, it initializes the value to negative infinity and iterates over each child of the node. For each child, it recursively calls the AlphaBetaPruning function with the child node, decreased depth, and updated alpha and beta values. It then takes the maximum of the current value and the value returned from the child. It updates the alpha value to the maximum of the current alpha and value. If the beta value is less than or equal to alpha, the function breaks out of the loop. Finally, it returns the maximum value found.

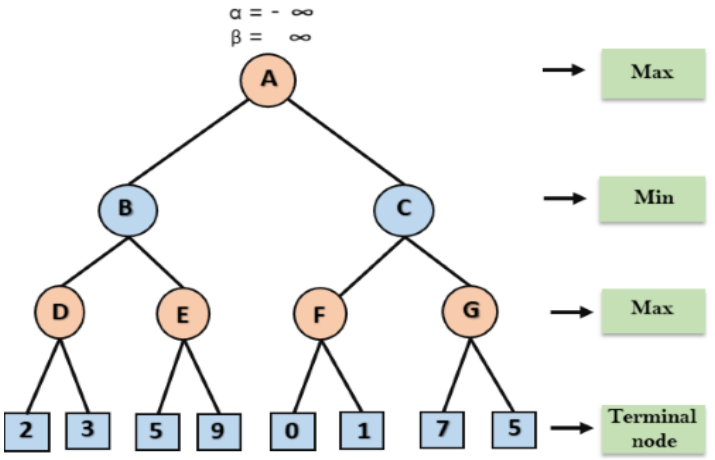
If the node is being minimized, it initializes the value to infinity and iterates over each child of the node. For each child, it recursively calls the AlphaBetaPruning function with the child node, decreased depth, and updated alpha and beta values. It then takes the minimum of the current value and the value returned from the child. It updates the beta value to the minimum of the current beta and value. If the beta value is less than or equal to alpha, the function breaks out of the loop. Finally, it returns the minimum value found.

The algorithm prunes branches of the tree that do not need to be explored by keeping track of the alpha and beta values, which represent the best values found for the maximizing and minimizing players, respectively. If the algorithm finds a node that causes the beta value to be less than or equal to the alpha value, it knows that this branch of the tree can be pruned because the other player would never choose that move.

Example of Algorithm

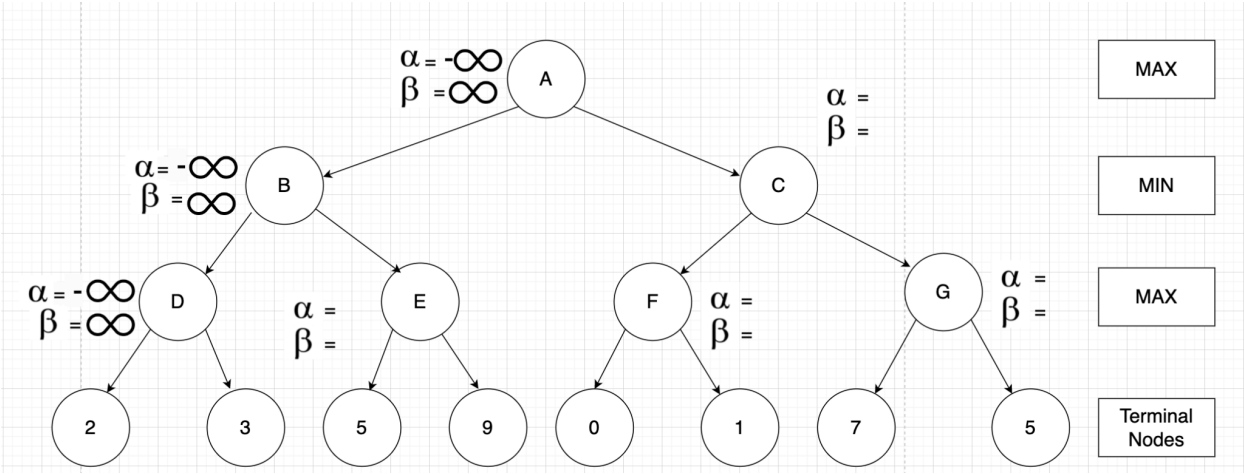
Given a tree, perform Alpha Beta Pruning on it:

Step 1: Begin by assigning the correct values and indexes to each node of the tree. Determine whether each level is calculating the max or min of the previous level or if it's a terminal node. Additionally, define alpha and beta values at the root node.

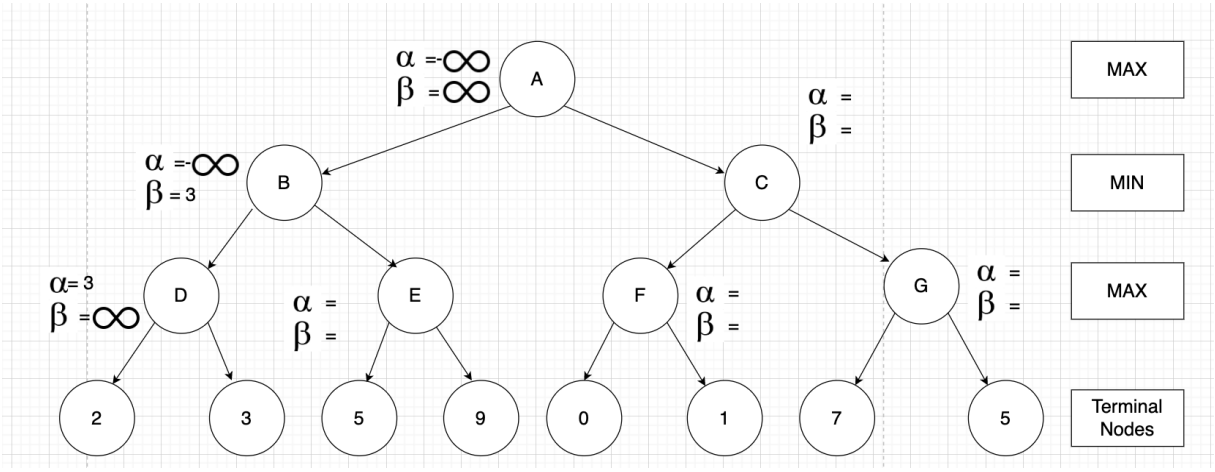


[6]

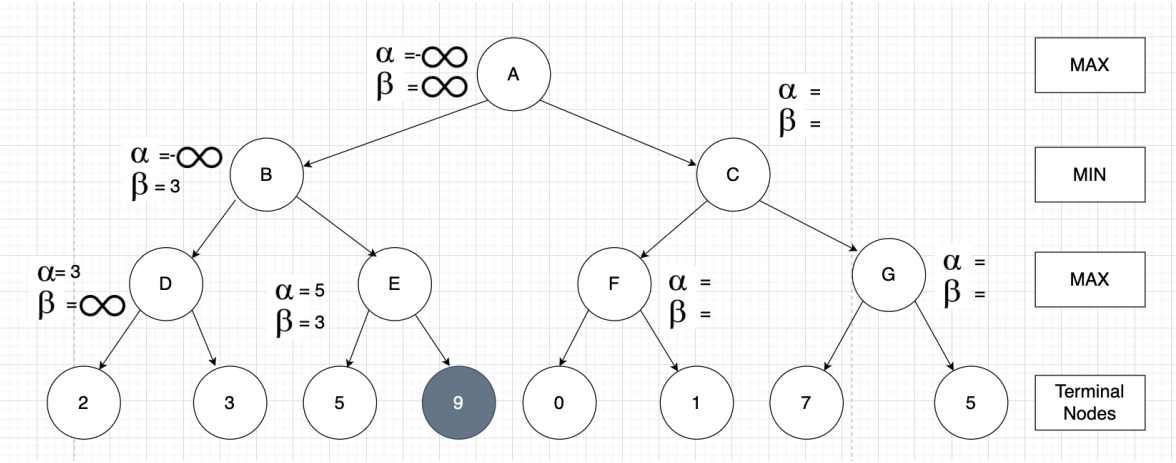
Step 2: The max player begins the first move at node A with alpha value set to negative infinity and beta value set to positive infinity. These values are then passed down to node B, which also sets its alpha and beta values to negative infinity and positive infinity, respectively. Node B then passes these values to its child node D.



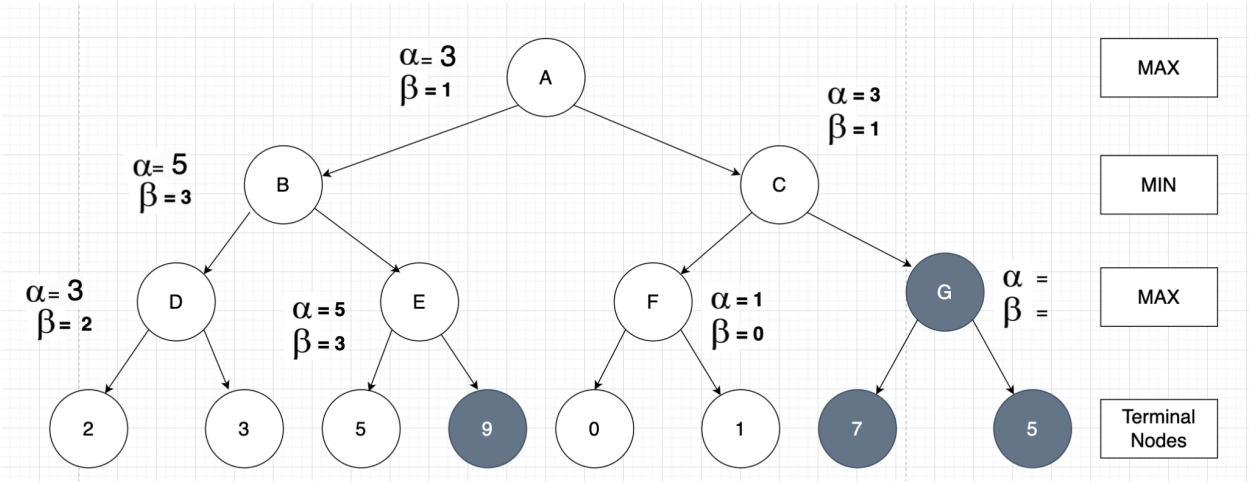
Step 3: At node D, the max value of alpha is calculated, as it is the turn of the max player. The value of alpha is compared with 2 and 3, and the maximum value (3) is assigned to alpha and the node value. The algorithm then backtracks to node B, where the value of beta is updated as it is the turn of the min player. Beta is set to positive infinity and compared with the subsequent nodes' values. The minimum value (3) is assigned to beta, and the alpha and beta values at node B become -∞ and 3, respectively.



Step 4: At node E, the max player takes its turn, and the value of alpha is updated. The current value of alpha is compared with 5, and the maximum value (5) is assigned to alpha. The alpha value at node E becomes 5, and the beta value remains at 3. Since alpha>=beta, the right successor of E is pruned, and the algorithm does not traverse it. The value of node E is updated to 5.



Step 5: Continue these steps until the whole tree is searched. The best option is found to be 3, and the algorithm traces back to 3 to determine the best possible option without having to explore the entire tree.



Analysis of Implementation

The time complexity of the AlphaBetaPruning algorithm is typically expressed in terms of the size of the game tree being searched. The size of the game tree is determined by the number of nodes it contains. The time complexity of the algorithm can be expressed as the number of nodes that need to be evaluated during the search. In the worst-case scenario, the game tree is a complete tree where every non-leaf node has b children. In this case, the total number of nodes in the tree can be expressed as follows:

$N = 1 + b + b^2 + ... + b^(d-1)$

where N is the total number of nodes in the tree, b is the branching factor of the tree, and d is the maximum depth of the tree. This is a geometric series, which can be simplified as follows:

$N = (b^d - 1) / (b - 1)$

Therefore, the total number of nodes in the tree is exponential in the depth of the tree, with a base of the branching factor.

During the alpha-beta pruning algorithm, the search space is pruned based on the current best move found so far. If the algorithm discovers that a node does not improve the current best move, it does not need to evaluate its children. Therefore, the actual number of nodes evaluated during the algorithm is often much smaller than the total number of nodes in the tree. In practice, the actual number of nodes evaluated depends on the specific structure of the tree and the quality of the utility function being used.

In the best case, where the tree is perfectly balanced, the time complexity is $O(b^(d/2))$. In the worst case, where the tree is a linear chain, the time complexity is $O(b^d)$, as defined by the geometric series above. However, due to the pruning technique used in the algorithm, it can often achieve a significant reduction in the search space compared to the minimax algorithm so b^d is more often than worst case than the average time.

Conclusion

In conclusion, the alpha-beta pruning algorithm is a powerful search algorithm that has revolutionized game playing and decision-making in various fields of artificial intelligence. Its ability to reduce the number of nodes evaluated in a game tree search has made it an essential tool for solving problems with large branching factors. The algorithm's history dates back to the 1950s, and it has since undergone numerous refinements and variations. Despite its simplicity, the alpha-beta pruning algorithm remains a fundamental technique in the field of artificial intelligence, with numerous applications in game playing, computer vision, natural language processing, and more. Its continued development and innovation will undoubtedly lead to exciting new advances in the field of artificial intelligence and decision-making.

Works Cited

[4] Abdelbar, Ashraf M. "Alpha-Beta Pruning and Althöfer's Pathology-Free Negamax Algorithm." *Algorithms*, vol. 5, no. 4, 2012, pp. 521–28, https://doi.org/10.3390/a5040521.

[3] Felstiner, Carl. "Alpha-Beta Pruning." *Alpha-Beta Pruning*, 9 May 2019, https://www.whitman.edu/documents/Academics/Mathematics/2019/Felstiner-Guichard.pdf. Accessed 15 March 2023.

[2] GeeksforGeeks. "Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)." *GeeksforGeeks*, 16 January 2023, https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/. Accessed 15 March 2023.

[6] JavaTPoint. "Artificial Intelligence | Alpha-Beta Pruning." *Javatpoint*, https://www.javatpoint.com/ai-alpha-beta-pruning. Accessed 15 March 2023.

[5] Singhal, Shubhendra Pal, and M. Sridevi. "Comparative study of performance of parallel Alpha Beta Pruning for different architectures." *Arxiv*, 29 10 2019, https://arxiv.org/pdf/1908.11660.pdf. Accessed 15 March 2023.

[1] Wikipedia. "Alpha–beta pruning." *Wikipedia*, 19 12 2002, https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning. Accessed 15 March 2023.