DA221 Assignment 2 Report

By Himanshu Singhal
220150004

1 Introduction

The Minimax algorithm functions by constructing a game tree that represents all possible moves from a given game state, projecting forward until it reaches terminal states - situations where the game is won, lost, or drawn. Each terminal state is assigned a value based on its desirability, and the values are back-propagated to determine the desirability of earlier moves. In a perfectly balanced game, the algorithm can identify the move that ensures the best possible outcome for a player.

2 Alpha-Beta Pruning

Even though the minimax algorithm looks promising, it is in fact a grossly inefficient strategy. The exhaustive nature of the game tree exploration leads to a combinatorial explosion of possibilities, especially as the depth of calculation increases. This results in this algorithm being highly computationally inefficient.

Therefore, we can use an optimization technique known as Alpha-Beta Pruning which dramatically reduces the number of nodes that need to be evaluated while keeping the same good properties of the original Minimax algorithm. The main crux of this optimization is its ability to disregard entire branches of the game tree that cannot possibly affect the final decision, based on the best options already explored.

Alpha here represents the best score that the maximizing player can guarantee at a given point in the tree while Beta represents the best score that the minimizing player can guarantee. When the algorithm detects that a branch cannot improve upon these scores, it prunes the branch, sidestepping the needless evaluation of suboptimal moves.

3 Evaluation of Alpha-Beta Pruning

Suppose a tree has depth D, and every node (except a tip node) has exactly B successors. Such a tree will have precisely B^D tip nodes. Suppose an alpha-beta procedure generated successors in the order of their true backed-up value. Then it happens that this order maximizes the number of cutoffs that will occur and minimizes the number of tip nodes generated. Let us denote this minimal number of tip nodes by N_D. Therefore, it can be shown that for even D

$$N_D = 2 * B^{\frac{D}{2}} - 1$$

and for odd D,

$$N_D = B^{\frac{D+1}{2}} + B^{\frac{D-1}{2}} - 1$$

That is, the number of tip nodes of depth D that would be generated by optimal alpha-beta search is about the same as the number of tip nodes that would have been generated at depth $\frac{D}{2}$ without alpha-beta. *Therefore, for the same storage requirements, the alpha-beta procedure with perfect successor ordering allows search depth to double.*

Therefore to summarize the main benefits of using Alpha-Beta pruning are:

- **Reduction of Game Tree Size**: As discussed above, alpha-beta pruning effectively cuts down branches that will not influence the final decision by leveraging the information gathered during the search.
- **Increased Depth of Search**: Since it prunes away useless branches, the algorithm can allocate its computational resources to searching deeper into the game tree. This allows the AI to consider more future possibilities within the same amount of time or computational effort.
- **Optimized use of Time and resources**: The computational savings translate directly into the ability to make better use of limited time and computational resources. This is particularly beneficial in real-time scenarios where quick decision-making is critical.

4 Experimental Results

I tested my algorithm on 10 handpicked examples and have recorded the results as follows:

Algorithm	Avg Time Taken (ms)	Avg no. of nodes evaluated
Minimax	70268	4209
Alpha-Beta Pruning	2633	158

Table 1: *Results on 10 examples where each metric is recorded in determining the next best move*

5 References

- Principles of Artificial Intelligence by Nils J. Nilsson