NOVATEUR PUBLICATIONS
INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY [IJIERT]
ISSN: 2394-3696 Website: ijiert.org
VOLUME 7, ISSUE 8, Aug.-2020

IMPLEMENTATION OF SEQUENTIAL AND PARALLEL ALPHA-BETA PRUNING ALGORITHM

SRAVYA MANDADI
Software Engineer, BSH Hausgeräte, Bangalore, India
sravyamandadi97@gmail.com

TEJASHWINI B.
Product Engineer, Talisma, Bangalore, India
btejureddy1997@gmail.com

SANJAN VIJAYAKUMAR
Data Scientist, eMotionRx, Boston, USA
sanjan.svk@gmail.com

ABSTRACT
In the past, game applications were proved to be inefficient compared to present day. This is mostly because of limitation of computer computation and memory space and inadequate game tree algorithms to help determine the next move. Games that require extensive searching needed a faster and more efficacious technique. Alpha-Beta Pruning is one of the most fundamental optimization technique for Minimax algorithm. Just by passing 2 extra parameters in the Minimax function, namely alpha and beta, this algorithm helps increase the speed of the game drastically. The pruning helps in eliminating the number of searches whenever there already exists a better move available in the game tree. In this paper, the effects of Alpha-Beta Pruning on the game tree are discussed by taking into account bimatrix games such as Tic-Tac-Toe and Checkers etc.

KEY WORDS: Minimax, Alpha-Beta Pruning, Maximizer, Minimizer

INTRODUCTION
Improvement on stacked matrix games such as Chess or Tic-Tac-Toe had always been an area of development for Artificial Intelligence researchers to mimic the human intelligence. Almost all multiplayer games require a decision tree and need to identify the possible alternatives for moves as well as select the best one available. In the games that antedated this technology, the selection process involved a large and complicated tree, thus resulting in a time consuming and wastage of computational memory. In order to improve upon this, the game tree search can be associated with Alpha-Beta pruning, which helps to eliminate the number of possible outcomes that are assessed by the Minimax Algorithm. Even simple games, such as tic-tac-toe or chess, very large trees are involved for solution finding [1].
In this paper, the operation of the Minimax search procedure and the alpha-beta pruning procedure are illustrated by using the context of various stacked matrix games such as chess. The players are named Min and Max with respect to whose turn it is to play; Max being the current player and Min being his opponent. Max attempts to maximize the ultimate value of the game while Min attempts to minimize the value. Although there are myriad of strategies that already exist to determine the next move of a player, the Minimax procedure is one of the most widely used backtracking algorithm that is used in decision making.
Consider the image below, Figure 1, which depicts a simple game tree structure. Here, each node of the tree p(1) , p(2), p(3) are positions, whereas the arcs from each of the nodes p(1,1), p(1,2), p(1,3), etc are the moves that each of the players have a chance at. The square nodes indicate it is Max's turn to move while the circles indicate it is Min's turn [2,3,4]. In the Minimax procedure, the value of the Max position is calculated by comparing each of the arc values and determining a backed-up maximum value, and similarly with the successors of the minimum value, a backed-up minimum value is determined. This procedure is used time and time again until the values at the leaf nodes create a value for the root leaf node itself. For example referencing to the figure below, of all the arc nodes, the minimum is selected from each. As for the root node, the maximum of the minimum nodes is selected.

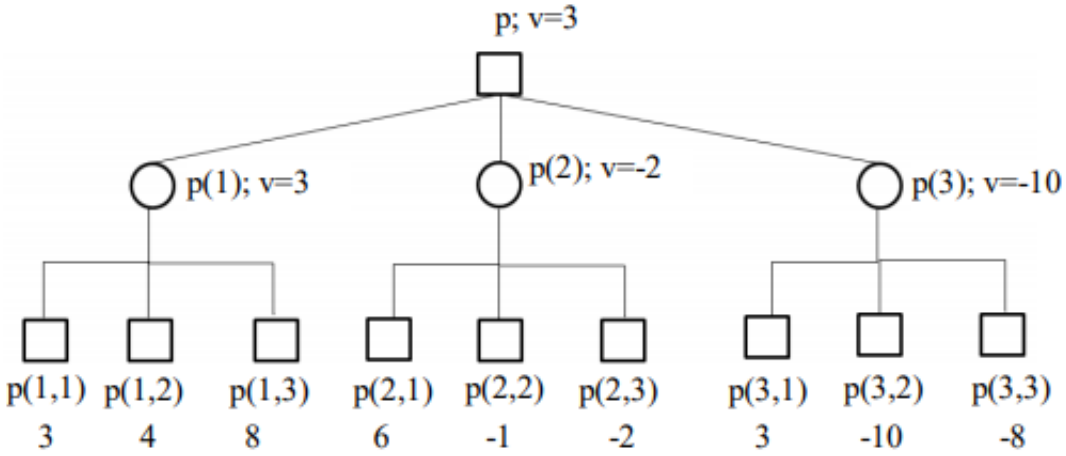

Figure 1: Game tree to understand Minimax Theorem

From the above, it can be derived that;

$$v_i = \max_{ai} \min_{s-i} v_i(a_i, a_{-i}) \ldots\ldots\ldots\ldots\ldots\ldots \text{Eq.1}$$

where: $_i$ is the index of the respective player, $_{-i}$ denotes remaining players, a_i is represented as the action taken by player $_i$, a_{-i} is the action by the remaining players and v_i is the value of the root node (value function of $_i$).

Algorithm 1: Minimax Algorithm

```
1. function minimax(node, depth, child):
2. if node is a leaf node:
3. return value of the node
4. for every child node:
5. if max:
6. value= max(child, depth[i]
7. break
8. return value
9. else:
10. value= min(child, depth[i]
11. break
12. return value.
```

ALPHA-BETA PRUNING

Alpha-Beta pruning is an extension and an optimization technique of the existing algorithm of Minimax. Though minimax is also used for the game tree, using this algorithm reduces the computational time by a large amount. By implementing this algorithm, we can search faster throughout the tree and reach the lower levels of the game tree much faster. This is because Alpha-Beta Pruning cuts off the unnecessary branches whenever a better path is available. The name Alpha-Beta is given to this algorithm since it passes 2 extra parameters, Alpha and Beta.

Sequential Alpha-Beta Pruning

The Alpha-Beta pruning algorithm proved to be an improvement over the usual MinMax algorithm that is used in most Artificial Intelligence games. Although both algorithms aim to find the best resulting path for a particular player, Alpha-Beta Pruning has a remarkably notable execution time. Since the algorithm eliminates values that are irrelevant to the final result, it is observed that it serves in a faster time period.

Refer to the sample game tree given in Figure 2. Similar to the Minimax algorithm, the maximizing player's nodes are depicted by a square, and minimizing player's nodes are depicted by a circle. The first step starts with the top of the tree structure, A. Alpha is given the value of –Infinity and Beta is given that of +Infinity.

Alpha and Beta are passed down to the next sequence of child nodes in the tree. At the position of A, the maximizer must choose the highest value between its child nodes (B and C).

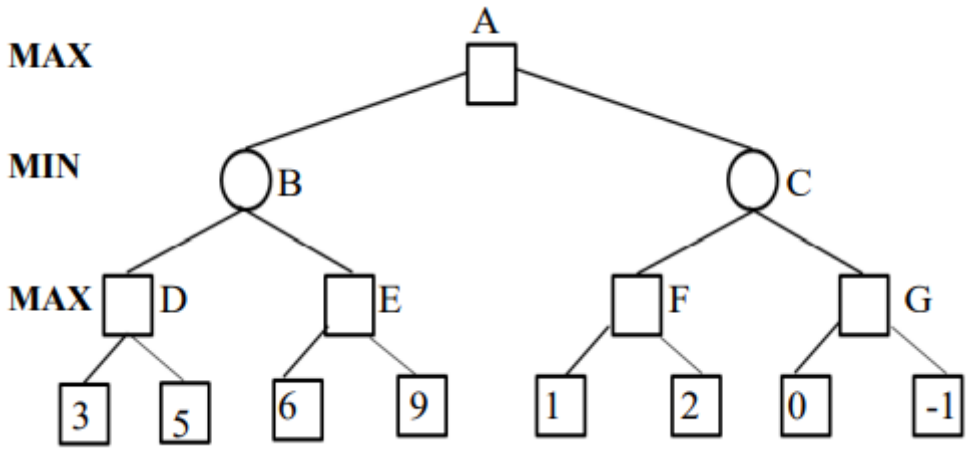

Figure 2: Game tree to understand Alpha-Beta Pruning Theorem

Let's consider the B path first. At B, it is the minimizer's turn so here the minimum of the child nodes are chosen and by his, D is called first. At D, the leaf node is considered and value of D is now to choose between Alpha and 3, and 3 is chosen as the max value. To decide whether or not to eliminate the other path, the condition beta is less than or equal to alpha is checked. In this case, this is observed to be false because Alpha is 3 and Beta is +Infinity. At D, the next leaf node is computed, here Alpha = 5. So the value of D is replaced with 5. D passes on the value of 5 to its parent node, B. At B, the minimum is computed between Beta and 5. Hence, the value of Beta is replaced with 5. B checks its other child node E for a value lesser than that of 5. The value at E is calculated between Alpha and 6, so 6 is the updated value of 6. However, since B is a minimizing point, the value is retained as 5.

Algorithm 2: Sequential Alpha-Beta Algorithm

```
function SequentialAlphaBeta(node, depth, Max, alpha, beta):
if node is a leaf node:
return value of the node
if Max:
BestValue = -Infinity
for every child node:
value = minimax(node, depth+1, false, alpha, beta)
BestValue = max( BestValue, value)
 alpha = max( alpha, BestValue)
  if beta <= alpha:
  break
  return BestValue
  else:
  BestValue = +Infinity
  for every child node :
  value = minimax(node, depth+1, true, alpha, beta)
  BestValue = min( BestValue, value)
  beta = min( beta, BestValue)
  if beta <= alpha:
  break
  return BestValue
  SequentialAlphaBeta(0, 0, true, -Infinity, +Infinity)
```

Parallel Alpha-Beta Pruning

In order to further increase the efficiency of the existing Alpha-Beta algorithm, parallelism has been introduced into this. Over the past few years, there have been different parallel Alpha-Beta Pruning Systems that have been developed. One amongst them is the classic algorithm which was implemented on sequent symmetry shared memory multiprocessor system in [9]. Principal-Variation algorithm is another approach, in which the first branch at a PV node is prioritized even before the remaining branches are searched. With the advent of this algorithm, many optimization techniques have been proposed. Young Brothers Wait Concept [11] is one which follows the master-slave approach. Here, the first sibling node is searched before spawning the remaining siblings in parallel. Dynamic Tree Splitting is similar to that of Young Brothers Wait Concept, except it disregards the use of master-slave and adopts peer-to-peer approach instead. This method uses multiple processors that process each node separately and is responsible for returning the node's evaluation to its parent.

Many concepts have been experimented with in various environments like reordering or beam search. Beam search is an algorithm which is used for efficient path finding and reduces memory usage as well. Apart from traditional Alpha-Beta Pruning, the same has been implemented on Graphics Processing Unit; in which different size boards would be used to decrease the computational time required. By using GPUs, parallel bigger boards provide a faster celerity than that of serial smaller board.

Algorithm 3: Sequential Alpha-Beta Algorithm

1. **function** ParallelAlphaBeta(node, depth, α, β, MaxP)
2. **if** depth = 0 **or** node is a leaf node
3. **return** value of node
4. **if** MaxP
5. value = -∞
6. **while** there exists child nodes in parallel
7. **do**
8. value = max(value, ParallelAlphaBeta(child, depth-1,α, β, FALSE)
9. α = max(α, value)
10. **if** β ≤ α then
11. **break**
12. **return** value
13. **else**
14. value = +∞
15. **while** there exists child nodes in parallel **do**
16. value = min(value, ParallelAlphaBeta(child, depth,–1, α, β, TRUE)
17. β = min(β, value)
18. **if** β ≤ α
19. **break**
20. **return** value

IMPLEMENTATION

As mentioned before, this algorithm could be implemented on various platforms that allow parallelism. The method used here is a shared memory model called OpenMP. For this platform we need to have some prerequisites that are needed for the installation like a device with 8GB RAM and OpenMP installed.

Parallel Alpha-Beta Pruning for Stacked Matrix Games

Unlike Minimax, Alpha-Beta algorithm isn't one that incorporates parallelism on its own. By processing multiple child nodes concurrently in the game tree, parallelism has been introduced into this algorithm. If the best move is found and can be evaluated first, the others can be eliminated. However, on doing this, it might be possible that this technique takes longer amount of time that classical sequential Pruning. In order to avoid this, each node is allocated to a processor, following the root splitting algorithm. Clusters are formed with each

of the children nodes and are held responsible by a processor. So effectively, each child is processed along with its roots and is reported back to the root node, where the best path is determined. A similar effective method is reordering the child nodes of a leaf node to explore the sub-tree on their own, hence reducing the time and number of paths that are encountered. Beam search is yet another algorithm which sorts the best of paths. It simplifies and priorities the nodes in breadth first order. The algorithm decides the least effective ones and discards them and explores the remaining path until the best one is found. This is also another way to increase the productivity of the algorithm and reducing computational time.

In order to implement the traditional sequential Alpha-Beta Pruning algorithm, the language used is C++. In order to parallelize, the algorithm must be implemented in Open MP in order to get an effective result. Open MP Multi Processing is one application programming interface which supports parallelization and consists of a multi top platform based shared memory. Open MP is a multithreading framework that is used for parallelization. In this framework, master-slave approach is followed, wherein the slaves run simultaneously, each performing their own task in the runtime environment. This is a shared memory system where a single space is accessible by multiple processors, but each slave/string has an id which cannot be accessed by others. The master process has an address space of 0 whereas the other processes have the address space of a whole number. On executing, the district processes join the master process, since each process performs the parallelized segment of code on its own. In this experiment, Open MP has been used to implement Alpha-beta pruning onto a shared memory.

Open MP is chosen since the API has the ability to perform parallelization as well as partition the code among the different processes to execute at a much faster and efficient manner. These processes can utilize work sharing and partition the code and execute their own task and later be joined to the master to be evaluated. During runtime, the partitioned segments of the code are assigned to different processors. On associating Alpha-Beta pruning with Open MP, the root split algorithm is assigned to a thread from the pool of free threads. Every thread is assigned to individual process in the running state which decides which branch is the best and is returned to the master process. The threads perform the parallelization of the segmented codes by using omp_get_thread_num(). This is used to get the address of each of the processors and the master process.

EXPERIMENTAL RESULTS

Figure 3 displays the computational time that is needed for the implementation of parallel and sequential Alpha-Beta Pruning distinctively in Tic-Tac-Toe. As shown, the time needed to compute the best path by using parallelization is significantly lesser than that of the sequential Alpha-Beta Pruning. It is observed that the serial time decreases when the player's moves are decreasing. This is because in that case, there would be lesser branches to compute the optimal path from. Because of this, from the graph it is observed that for the last few moves, both parallelization and sequential method of Alpha-Beta Pruning have almost the same computational factor. Sequential Alpha-Beta Pruning time decreases linearly when the number of sub-trees to be checked is decreased. When the sub-trees are minimal, the slaves are free and this causes overhead in the multithread process. In almost all cases, sequential Alpha-Beta Pruning took longer than that of parallel. Parallel Alpha-Beta Pruning is 1.8times faster than Sequential.

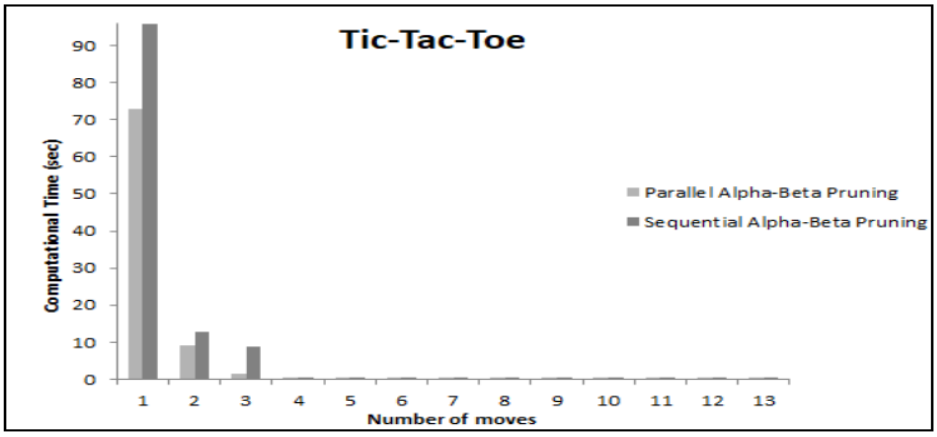

Figure 3: Comparison study between Parallel and Sequential Alpha-Beta Pruning using Tic-Tac-Toe

Figure 4 depicts the comparison of computational values between Sequential and Parallel execution of Alpha-Beta Pruning Algorithm in the game Checkers. By this figure, it is observed that Sequential Alpha-Beta Pruning always took considerably large amount of time compared to the Parallel implementation of the same. As the moves in the game progressed, the time taken to compute the best possible path has also increased in both Sequential and Parallel. Parallel Alpha-Beta Pruning is averaged to be 2.97 times the Sequential methodology.

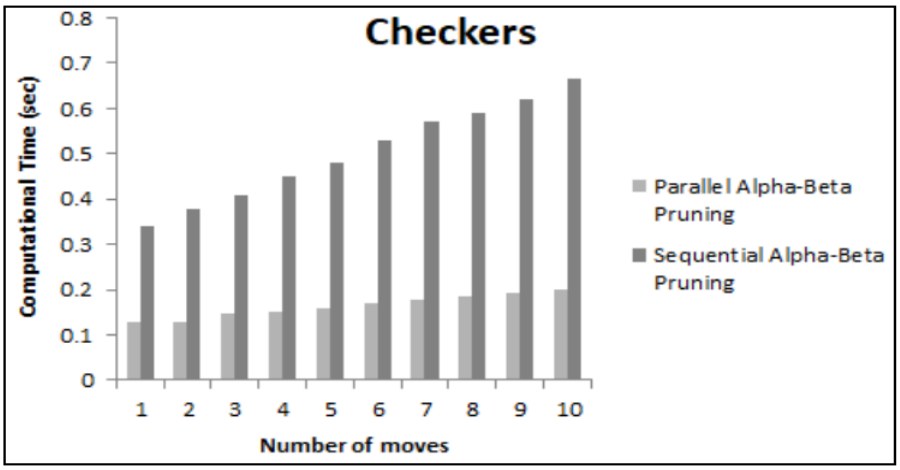

Figure 4: Comparison study between Parallel and Sequential Alpha-Beta Pruning using Checkers

Apart from this, the difference between using reordering technique and search beam optimization has also been explored. It can be observed from Figure 5 that Beam Search is 2.45 times more efficient that of reordering implementation. The number of nodes is indirectly proportional to the performance. The more number of nodes, the lesser the performance of the algorithm and vice versa. From this, it can be said that the number of nodes visited by the search beam optimization technique is lesser than the reordering approach, and thus decreases the computational time.

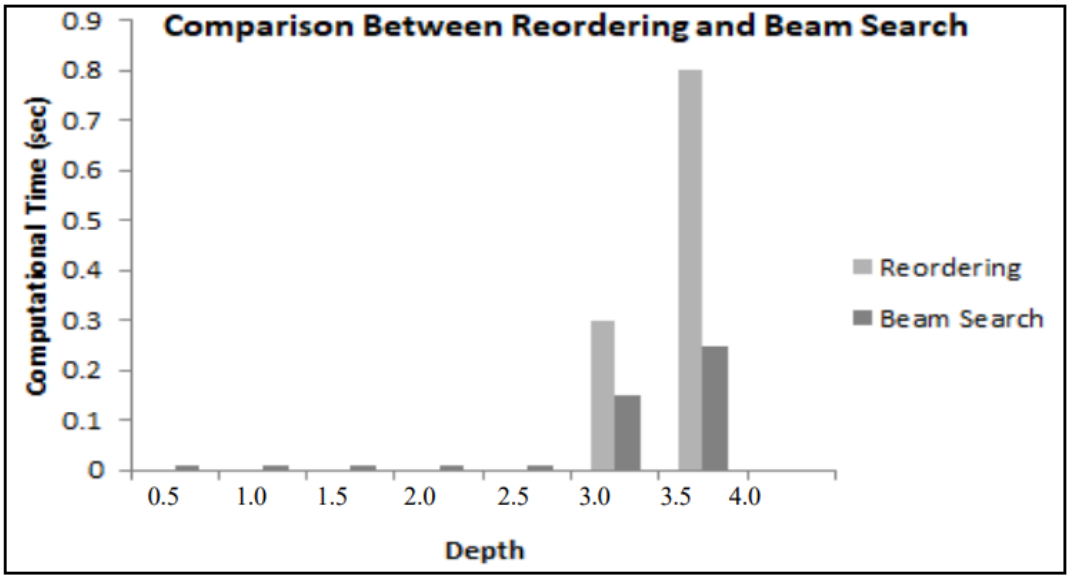

Figure 5: Comparison Between Reordering and Beam Search

CONCLUSION

Simple stacked matrix games have been used to identify the difference in computational speed and efficiency between Sequential and Parallel Alpha-Beta Pruning techniques. Experimental results of Tic-Tac-Toe and Checkers have been observed along with the approach of reordering and search beam optimization technique. From the results of this experiment, it is been observed that Parallel Alpha Beta Pruning is considerably more efficient and consumes lesser time than that of Sequential. From the 2 games that have been experimented, different computational factors have been derived based on the board size and different number of nodes.

In this paper, the functionality of Minimax algorithm and Alpha-Beta Pruning Algorithm have been studied with an example. By experimenting with real time game trees, which are of stacked matrix games, simulations of each game have been displayed. These results can be extended based on other algorithms that can further enhance the efficiency of game trees by using neural networks.

REFERENCES

1) Ballard, B. W. 1983. The Minimax search procedure for trees containing chance nodes. Artificial Intelligence 21(3):327–350.
2) Sturtevant, N. R., and Korf, R. E. 2000. On pruning techniques for multi-player games. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, AAAI/IAAI 2000, 201–207.
3) Sturtevant, N. R. 2005. Leaf-value tables for pruning nonzero-sum games. In IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, 317–323. Edinburgh, Scotland, UK: Professional Book Center.
4) C. M. M Guidry, "Techniques to parallelize chess." [Online]. Available: https://pdfs.semanticscholar.org/8e1c/9f70aa4849475199e26ccd3e21c66 2e2d801.pdf
5) C. wiki, "Alpha-beta," 7 June 2019. [Online]. Available: https://www.chessprogramming.org/Alpha-Beta
6) S. D. G. V. Akansha Kumari, Shreya Singh, "Parallelization of alpha beta pruning algorithm for enhancing the two player games," vol. 4, pp. 74–81, Feb 2017.
7) R. S. Kamil Rocki, "Parallel minimax tree searching on gpu," 2010, . (Eds.): PPAM 2009.
8) N. G. Damjan Strnad, "Parallel alpha-beta algorithm on the GPU," Journal of Computing and Information Technology, vol. 19, no. 4, pp. 269–274, 2011.
9) H. L. N. Robert M Hyatt, Bruce W Suter, "A parallel alpha/beta tree searching algorithm," IEEE Transaction on Parallel and Distributed Systems, vol. 10, pp. 299–308, May 1989, dOI : 10.1016/0167-8191(89)90102-6.
10) Shubhendra Pal Singhal, M. Sridevi, Comparative study of performance of parallel Alpha Beta Pruning for different architectures, 2019 IEEE 9th International Conference on Advanced Computing (IACC), 13-14 Dec, 2019.
11) Hyatt, R.M. (1988). A High-Performance Parallel Algorithm to Search Depth-First Game Trees. Ph.D. Thesis, University of Alabama, Birmingham.
12) Hyatt, R.M. (1997). The Dynamic Tree-Splitting Parallel Search Algorithm. ICCA Journal, Vol. 20, No. 1, pp. 3 – 19.
13) Parallel Alpha-Beta Search on Shared Memory Multiprocessors by Valavan Manohararajah (Thesis)
14) Parallel Alpha-Beta Algorithm on the GPU Damjan Strnad and Nikola Guid Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia