CHESS ENGINE

Alpha-Beta Pruning & PV-Splitting

Chirayu Garg

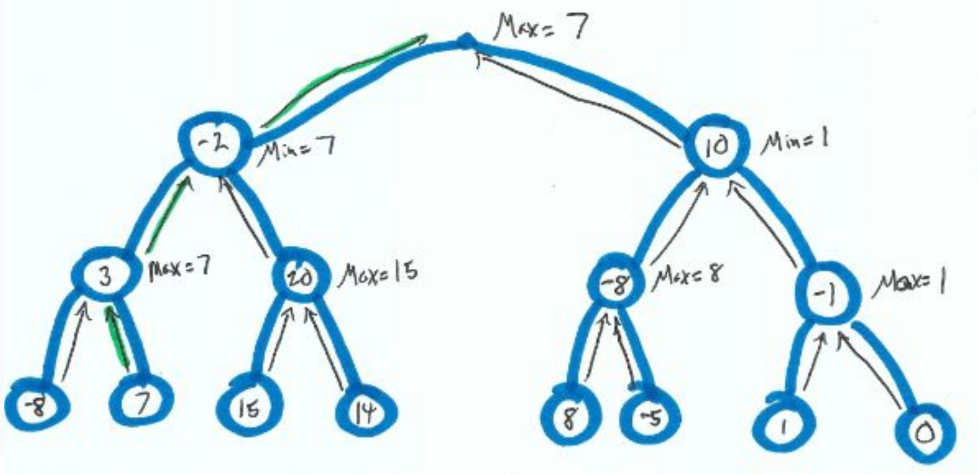
https://github.com/garg104/ChessEngine

Content

MINIMAX SEARCH

- "Minimax is a decision rule used in artificial intelligence, decision theory, game theory, statistics, and philosophy for *mini*mizing the possible loss for a worst case (*max*imum loss) scenario."
- It aims to provide optimal move for the player assuming that the opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree and is commonly used in two player games such as chess. One player is assigned as the MAX player and the other is assigned as the MIN player.
- Each player play in a way which tries to minimize the gain of the other player.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

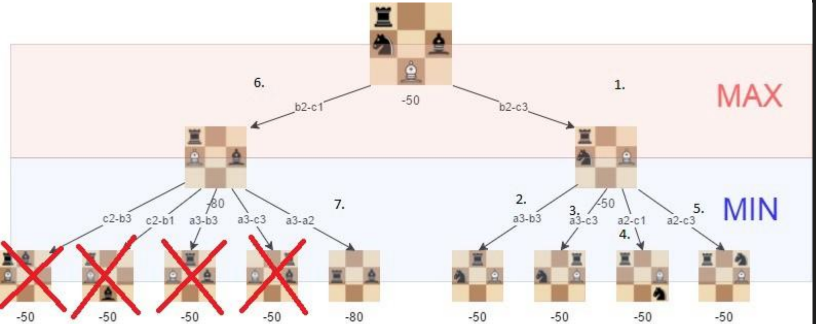


ALPHA-BETA PRUNING

- Alpha-beta pruning is an optimization to the minimax algorithm.
- As mentioned before Minimax algorithm goes over the different game states and then has to evaluate them which is exponential in depth. To make this faster we try to reduce the game states it has to evaluate by what is called pruning. Essentially cutting the tree branched off strategically so that the final answer is still the same.
- This involves two threshold parameters: Alpha and beta for future expansion, hence it is called **alpha-beta pruning**.
- The two-parameter can be defined as:
    a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.
- Alpha-beta pruning sometimes not only prune the tree leaves but also entire sub-trees. Hence by pruning these nodes, it makes the algorithm fast.

ALPHA-BETA PRUNING

- We use alpha-beta pruning in our implementation of the chess engine.
- The code for it is on the right.
- The condition alpha > beta is when we prune the tree. As shown we break out of the loop and hence never evaluate anything further in the subtree of the node we just pruned.



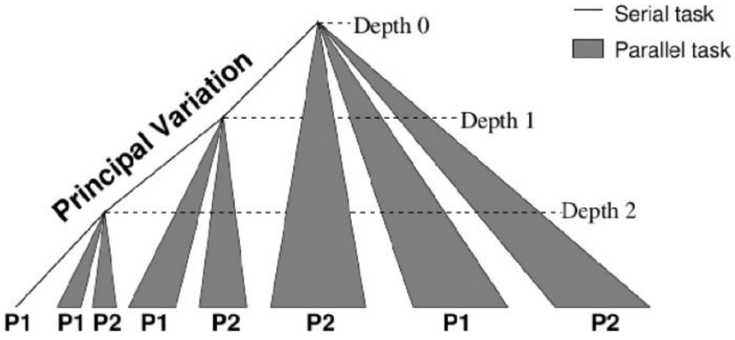

```cpp
ChessBoard* alphaBetaPruning(int maxDepth, ChessBoard* board, int depth, int alpha, int beta, int action) {
    // if maximum depth is reached
    // action 1 is maximizing which is black and action 0 is minimizing with is also white
    // get action of the child node
    int childAction = (action == 1 ? 0 : 1);
    if (depth >= maxDepth) {
        return board;
    }
    vector<ChessBoard*> possibleMoves = board->getAllPossibleMoves(action);
    ChessBoard* bestMove = NULL;
    int bestMoveIndex = 0;

    // go over all the possible moves
    for (int i = 0; i < (int) possibleMoves.size(); i++) {
        // make tree
        ChessBoard* tempMove = alphaBetaPruning(maxDepth, possibleMoves[i], depth + 1, alpha, beta, childAction);
        // see if the move is the best move or not
        if (bestMove == NULL || isMoveBetterThan(tempMove, bestMove, action)) {
            // replace the current bestMove with the better one
            bestMove = tempMove;
            bestMoveIndex = i;
            // update alpha-beta by getting scores
            if (childAction == 1) {
                // Black
                beta = tempMove->evaluate(BLACK);
            } else {
                // White
                alpha = tempMove->evaluate(WHITE);
            }
        }
        // prune
        if (alpha > beta) {
            break;
        }
    }
}       You, now • Uncommitted changes
```

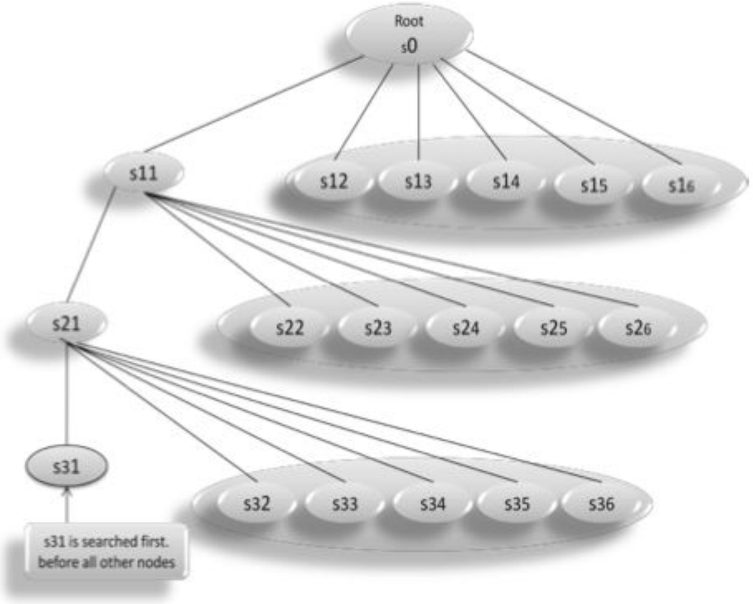
PV-SPLITTING

- Principal Variation Search or PVS was first introduced by Tony Marsland and Murray Campbell in 1982.
- This follows the idea that in most of the nodes/ game state we just need a bound which proves that a move is unacceptable and not the exact score.
- With respect to chess we can visualize it as we just need to get a bound by the first branch of a tree and then the remaining branched can be run in parallel with the bounds and can update the bounds as in when with proper mutex locks.
- "PV Splitting is designed for the case when alpha beta pruning is used along with minimax search. In PV Splitting, starting with the child nodes of the root node, the leftmost child node's sub tree is searched first and then the remaining child nodes are then searched. This is done recursively. That is when searching the leftmost sub tree, the leftmost child node is expanded into its children first. Among its children, the leftmost child's sub tree is again searched first before searching the remaining children in parallel."

PV-SPLITTING

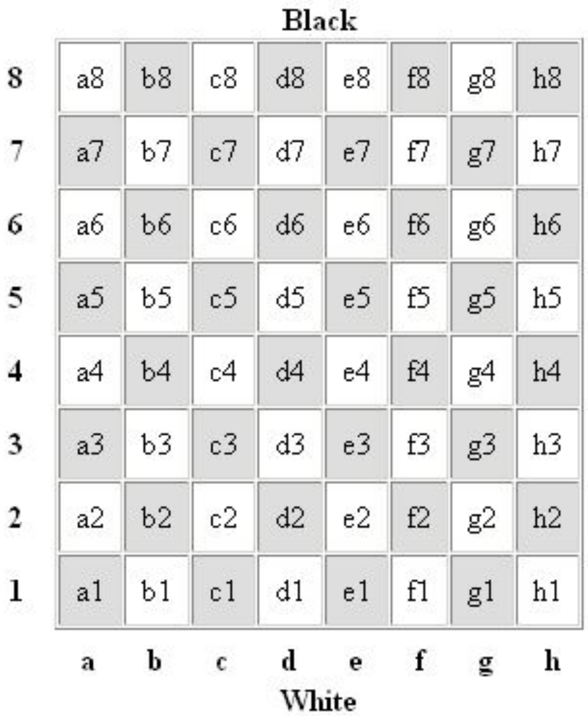
- In the picture along side we can visualize the procedure. First all the children of s11 are searched before other children of s0 are searched. Similarly s21 is searched before any other children of s11. Once we get the bounds by the leftmost child, we can search the other nodes in parallel.





CHESS ENGINE IMPLEMENTATION

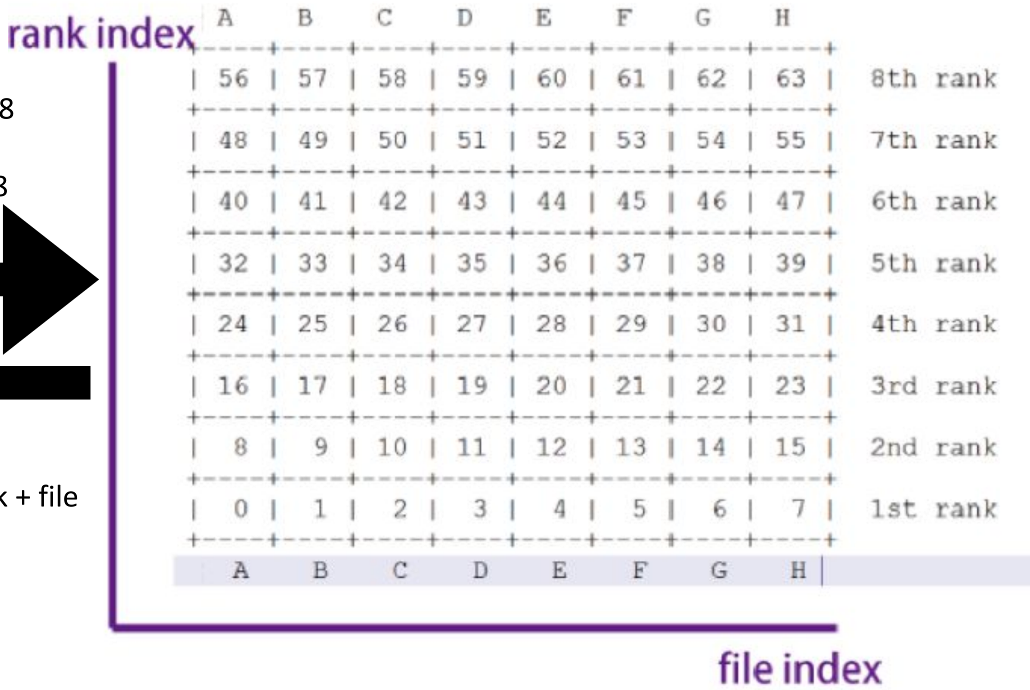
ChessBoard:



rank = squareNum / 8

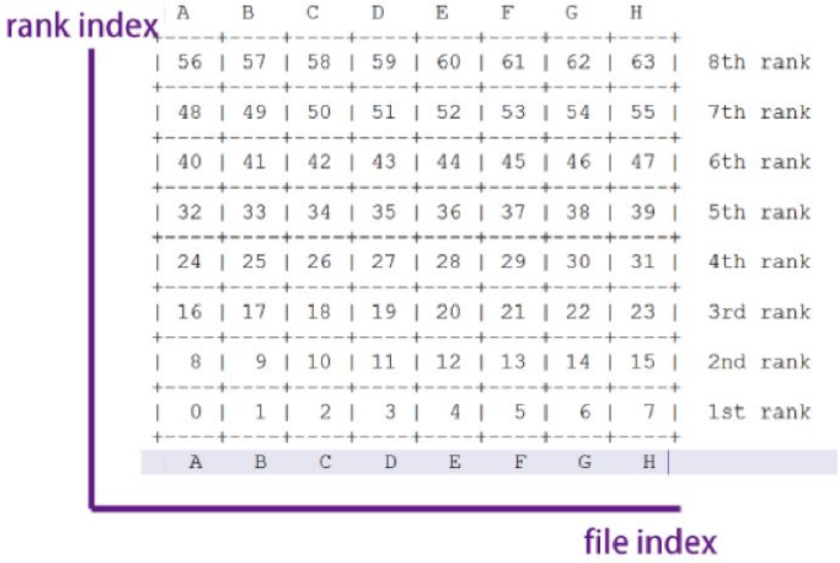
file = squareNum % 8

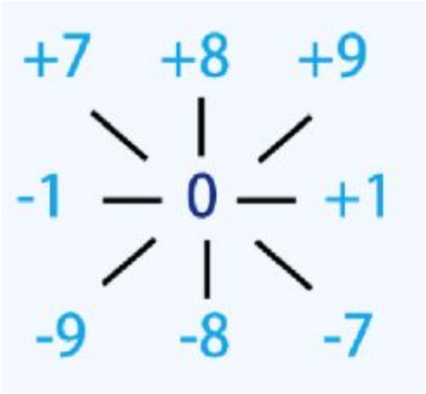
squareNum = 8*rank + file



CHESS ENGINE IMPLEMENTATION

Directions:





CHESS ENGINE IMPLEMENTATION

- Piece movement is implemented as the possible block values at which a piece can move according to the rules of chess. For example knight can move two blocks and then cuts across 90 degrees. The code to the right implements it by using the block numbers in a chess board representation as show in previous slides.

```
virtual bool checkMoveValidity(int initial, int final, int* board) {
    int finalRow = final / 8;
    int finalCol = final % 8;
    int initialRow = initial / 8;
    int initialCol = initial % 8;

    // move along row and then one cross column
    if ((finalRow >= 0 && finalRow <= 7) &&
        (abs(finalRow - initialRow) == 2)) {
        if ((finalCol >= 0 && finalCol <= 7) &&
            (abs(finalCol - initialCol) == 1)) {
            return true;
        }
    }

    // move along col and then one cross row
    if ((finalCol >= 0 && finalCol <= 7) &&
        (abs(finalCol - initialCol) == 2)) {
        if ((finalRow >= 0 && finalRow <= 7) &&
            (abs(finalRow - initialRow) == 1)) {
            return true;
        }
    }

    return false;
}
```

CHESS ENGINE IMPLEMENTATION

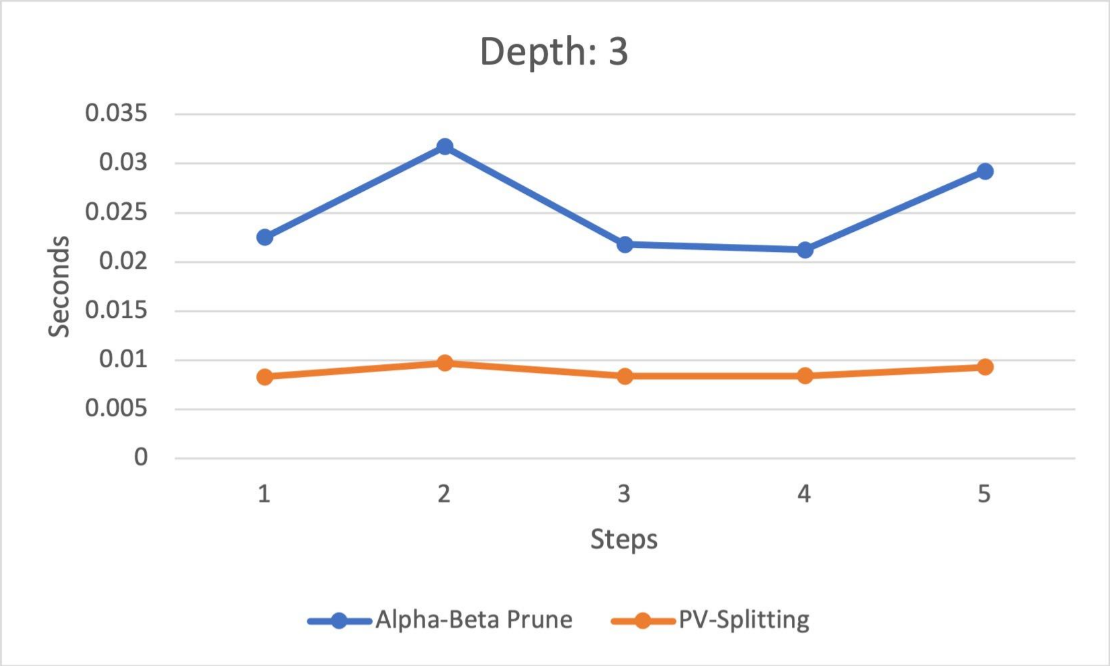
- We use a static evaluation function to get a score of the game in a particular move. The score is determined by the state of the chess board.
- “The static evaluation function returns a score for the side to move from the given position. A score is calculated for both sides and the function returns the score for the side on the move minus the score for the side not on the move.”
- For more: https://www.dailychess.com/rival/programming/evaluation.php

GLIB

- We use the GLib library to manage the thread pools. The big idea is to first get the principal and then execute the other siblings of the principal nodes in a thread pool. GLib provides a framework to do so without worrying about the hassle of keeping track of threads and bunching the together and of passing the data to the different threads.
- For more: https://docs.gtk.org/glib/struct.ThreadPool.html

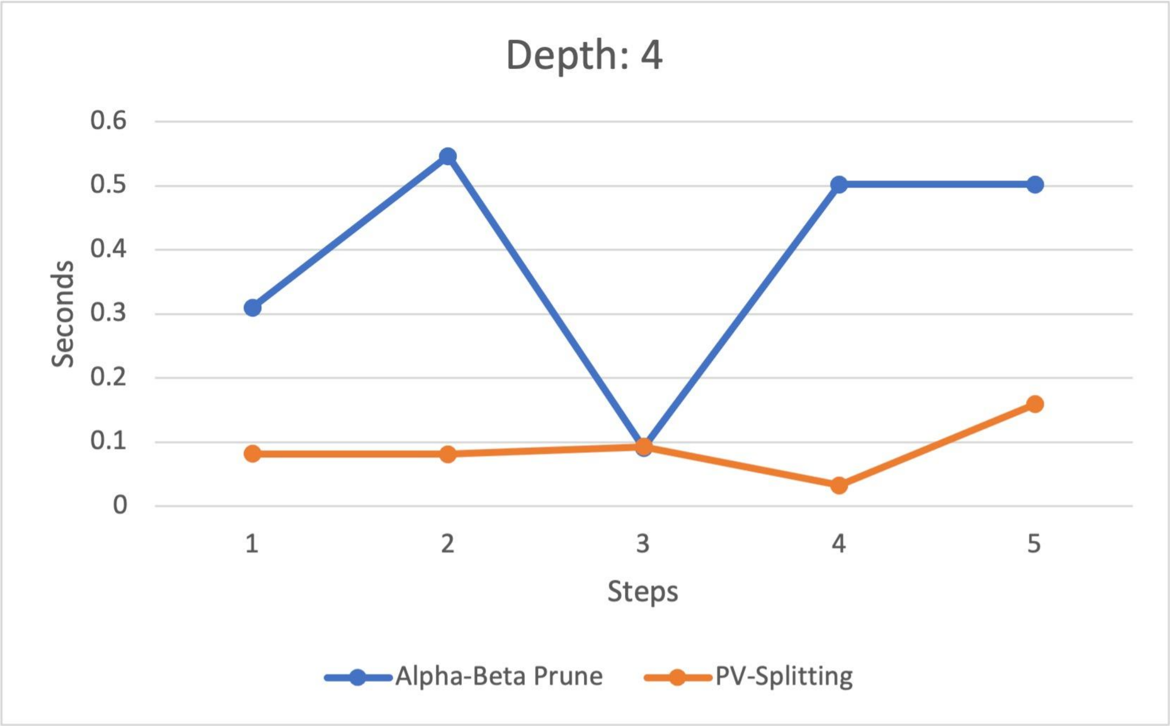
RESULT

- Comparison between Alpha-Beta Pruning and PV-Splitting with nThreads = 5 and maxDepth = 3



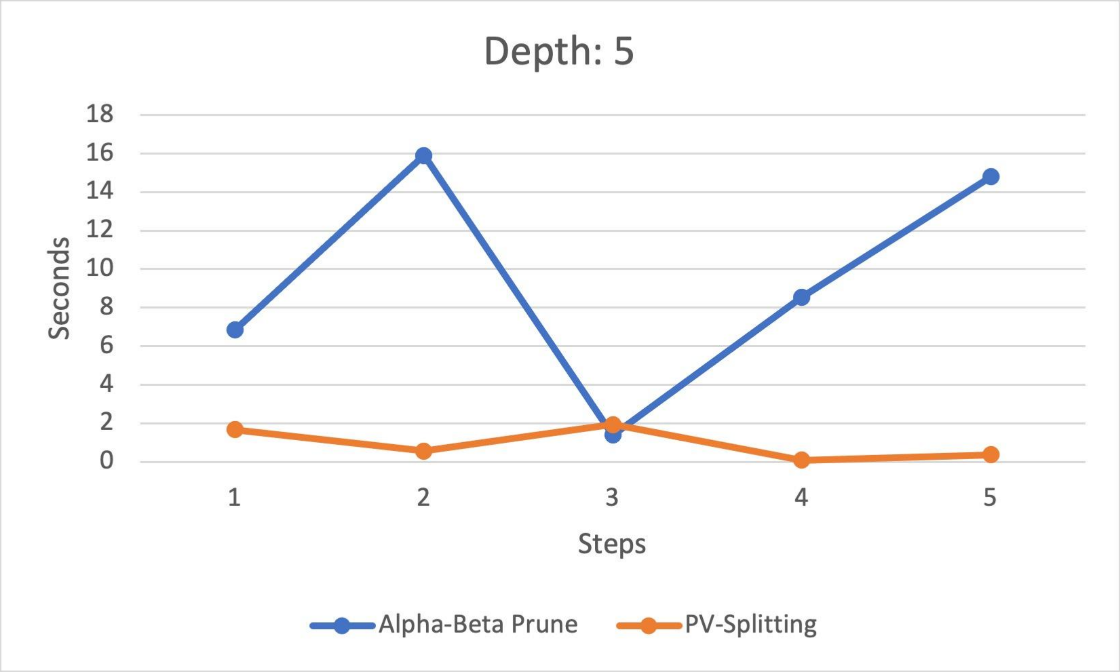
RESULT

- Comparison between Alpha-Beta Pruning and PV-Splitting with nThreads = 5 and maxDepth = 4



RESULT

- Comparison between Alpha-Beta Pruning and PV-Splitting with nThreads = 5 and maxDepth = 5



CONCLUSION

- We can clearly see a speed up is gained when using PV-Splitting. In all three scenarios we saw that we obtained a speed up.
- Using different number of threads produced similar results.
- However, we still need to consider about the scalability of the algorithm. Therefore more in-depth analysis of the algorithm is required.
- Instances where the tree is cut/pruned very early lead to similar time for both serial and parallel algorithms. This is shown by step 3 in the graphs for depth 4 and depth 5.
- Therefore, this is a possible drawback of our algorithm.
- Load balancing will also prove to be a drawback as the game progresses and some subtrees will have smaller width and depth which will lead the threads to go ideal sooner.
- Using depth as 6 leads to a massive increase in time which leads me to believe that further optimization of the program is needed.

MOVING FORWARD

- Optimizing the current code to improve times for greater depths
- Looking at other parallel approaches to implement which will deal with the shortcomings of the current algorithm
- Adding a dynamic evaluation function or improving the current one to get more optimal moves from the AI.

REFERENCES

- https://www.freedesktop.org/software/gstreamer-sdk/data/docs/latest/glib/glib-Thread-Pools.html
- https://www.javatpoint.com/ai-alpha-beta-pruning
- https://www.chessprogramming.org/Main_Page
- https://www.dailychess.com/rival/programming/evaluation.php#:~:text=The%20static%20evaluation%20function%20returns,side%20not%20on%20the%20move
- https://www.chessprogramming.org/Evaluation
- https://docs.gtk.org/glib/struct.ThreadPool.html
- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.218.405&rep=rep1&type=pdf
- https://docs.microsoft.com/en-us/windows/win32/procthread/thread-poolsœ

THANK YOU

Alpha-Beta Pruning & PV-Splitting

Chirayu Garg

https://github.com/garg104/ChessEngine