Multi-threaded Chess AI

Jonathan Cirillo
Undergrad, Dept. Computer Science
University of Central Florida
Orlando, FL, USA
Cirillojon@knights.Ucf.edu

Jesse Johnson
Undergrad, Dept. Computer Science
University of Central Florida
Orlando, FL, USA
j-jesse23@knights.ucf.edu

Lester Miller
Undergrad, Dept. Computer Science
University of Central Florida
Orlando, FL, USA
lmiller44@knights.ucf.edu

Marco Peric
Undergrad, Dept. Computer Science
University of Central Florida
Orlando, FL, USA
mperic@knights.ucf.edu

Jonathan Velez
Undergrad, Dept. Computer Science
University of Central Florida
Orlando, FL, USA
johnvelez01@knights.ucf.edu

***Abstract*—For complex problems whose solutions tend to be time-consuming, one of the ways to improve runtime is through parallelization. Even for artificial intelligence (AI) algorithms that play complex games like chess, we can break down the processing time by parallelizing its main algorithm. This paper will be used to explore this potential in performance. We will use a chess algorithm, Minimax, that takes the brute force approach and implement an Alpha-Beta pruning algorithm with it. By itself, the Alpha Beta algorithm increases the processing speed of the searching algorithm but we plan to make it faster. Using java, we successfully implemented the parallelization of the Alpha Beta pruning. We then further compare the performance of the parallelized version to the sequential version, as well as comparing both parts to the base comparison of Minimax.**

***Index Terms*—Alpha Beta pruning, Minimax, Parallel algorithms**

I. Introduction

Since its inception, chess has long been considered a challenging and complex game that requires players to memorize and predict their opponent's next moves. With the advancement in technology and the potential to incorporate Artificial Intelligence (AI) to play chess, there has been a significant improvement in chess AI. So much so that AI has long surpassed human capabilities when the IBM computer Deep Blue beat the chess grandmaster in 1997 [1]. There are now plenty of algorithms that can beat humans in a game of chess, but we find that the same limitations that affect chess players also affect these algorithms: the number of moves that a player can look ahead is limited by resources and time.

Identify applicable funding agency here. If none, delete this.

For simple games such as tic-tac-toe or checkers, an algorithm could determine the winning moves after a player's first or second move. But what makes chess such a complex game is the almost limitless amount of moves that a player can take. After just 3 moves from each player the pieces can have over nine million possible variations [2]. To address this challenge we attempt to parallelize a well-known chess search algorithm called Alpha Beta pruning. We will then demonstrate its effectiveness by measuring and comparing its performance with the sequential Alpha Beta pruning and the base algorithm that it comes from, the Minimax algorithm.

A. Minimax

B. Alpha Beta Pruning

The algorithm begins by evaluating the root node of the tree, which represents the current state of the chess game. From this root node, the algorithm will explore all possible moves that can be made by the current chess position. The alpha and beta values to keep track of the best scores found so far for maximizing and minimizing each play. Initially, alpha is set to negative infinity, which means that any position score greater than negative infinity will become the new alpha value. Beta is set to positive infinity, which means that any position score less than positive infinity will become the new beta value.

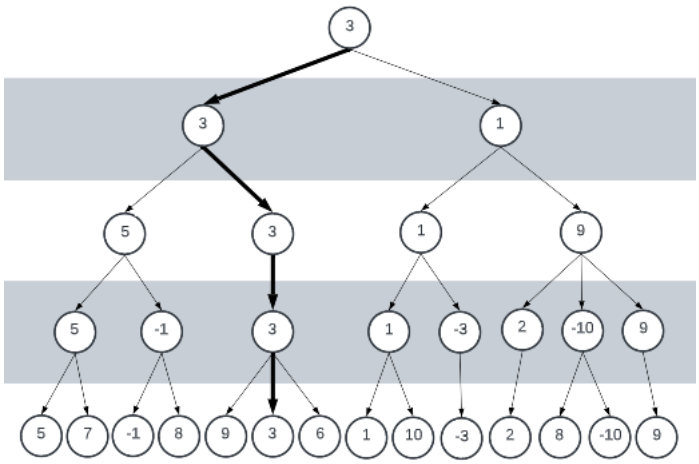

Fig. 1. Example of a Minimax algorithm.

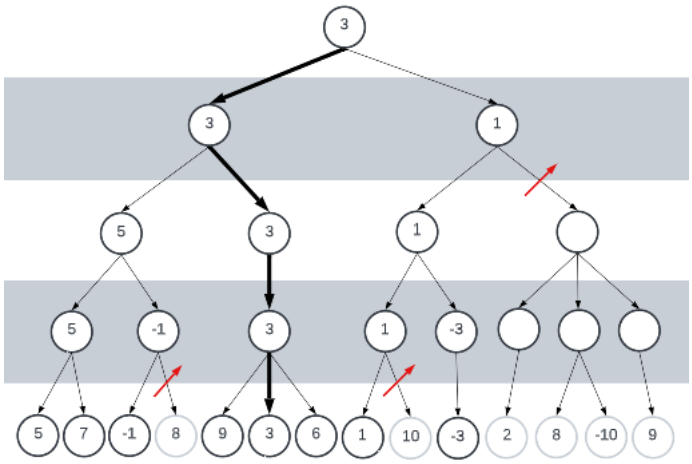

Fig. 2. Example of Alpha Beta pruning of the Minimax algorithm

When the algorithm evaluates each child node, it updates the alpha and beta values based on the scores found so far. During the search, alpha becomes greater than or equal to beta, then the algorithm can prune the current subtree. After all, it knows that any further exploration of this subtree will not contribute to the final result, since the subtree contains moves that are not optimal. When A child node is a maximizing node, then the algorithm updates the alpha value with the maximum score found so far. This means that a move will always choose the results in a score greater than or equal to the current alpha value. At the same time, a child node is given a minimizing node, then the algorithm updates the beta value with the minimum score. In other words, the algorithm aiming to minimize the score will consistently select a move that generates a score that is either equivalent to or lower than the present beta value.

II. Motivation

A. *subsection*

III. Implementation

Our attempt to parallelize the Alpha-Beta pruning algorithm makes use of a number of optimizations that can also be applied to the sequential version. Currently, the algorithm will detect how many processors are available for use, and will then assign a thread to each node. Our AlphaBetaTask class extends the RecursiveTask class, thus allowing it to be divided into smaller subtasks when it's submitted to the ForkJoinPool. It continues to be divided until it can be solved directly, and the solutions are then combined to produce the final result of the task (aka the node that's being evaluated). By having several nodes being evaluated at once, we will ideally be able to improve the overall performance.

Additionally, a method known as Best First Search is being implemented that will cache the values of child nodes that have already been evaluated, thus allowing the algorithm to focus on evaluating branches that appear to better choices. This is an improvement over simply working from one side of the tree to another, and will ideally prune branches of the tree earlier. Each thread will have its own copy of the cache to reduce contention and to prevent race conditions from occurring when multiple threads try to access the cache simultaneously.

A. *Anticipated Encountered Challenges*

At this time we have a working parallel implementation of the Alpha-Beta pruning algorithm, however, it still runs substantially slower than the sequential version. This unfortunately holds true for every size of tree that we have tested. Trees with larger depths, such as 25, encounter heap size errors while the sequential algorithm is able to return a correct value in roughly 30ms. We are continuing to research optimizations to our parallel implementation.

IV. Testing

V. Evaluation

VI. Conclusion

References

[1] G. Joanna, "How IBM's Deep Blue Beat World Champion Chess Player Garry Kasparov" https://spectrum.ieee.org/how-ibms-deep-blue-beat-world-champion-chess-player-garry-kasparov.

[2] “How Many Possible Moves Are There In Chess?” https://www.chessjournal.com/how-many-possible-moves-are-there-in-chess/.