Chess Evaluation at Variable Search Depth

Devin Borchard
dmb1035@usnh.edu
The University of New Hampshire

ABSTRACT

There are many proven highly effective chess AIs out today that use complex machine learning and deep learning algorithms, they are capable of referencing databases with millions of historical game positions to inform their decisions. I want to simplify the problem down to what base strategies the AI should implement when doing static position evaluation, and what search depths are needed to be effective with these strategies.

1 BACKGROUND

Before approaching the problem, we have to understand some aspects of the alpha beta pruning algorithm, and how we can fit those aspects to chess specifically. Search depth is the term for how many layers deep an algorithm will evaluate ahead when deciding what move to make. For this implementation, a single player's turn is a single depth in the search. The branching factor of chess is too large to run a search tree to a finished game state, so pruning is necessary to make faster decisions.

For the pruning algorithm to make good decisions, the chess board state after each move needs to be given evaluations. These evaluation scores are what is used to decide if a side has a greater advantage.

1.1 Strategies

- **Material**, where the position of each player is evaluated based on the difference of the total values of the pieces the players have on the board. I'll be using the classic chess piece values of 9, 5, 3, 3, and 1 for queens, rooks, knights, bishops, and pawns, respectively.

- **Mobility**, where pieces are given more value if they have more moves available, for instance, a bishop that is trapped in a corner is much less valuable than a bishop that's able to move and attack from one corner of the board to the other.

- **Checks**, most winning combinations in chess are achieved by checking the king or pinning pieces. These types of moves force the opponent into a limited set of actions making planning easier.

2 APPROACH

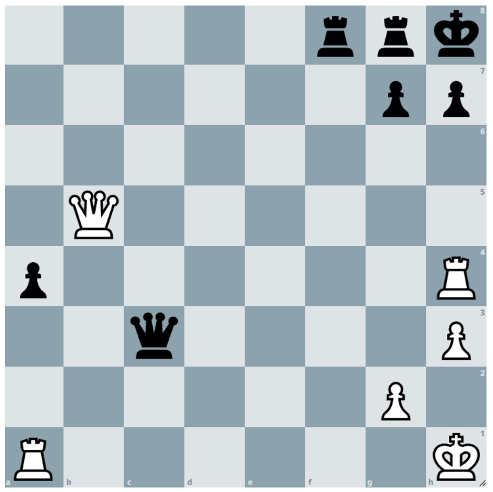

Figure 1: Example Puzzle (Mate in 2)
FEN: 5rrk/6pp/8/1Q6/p6R/2q4P/6P1/R6K

The Alpha Beta Pruning search algorithm was used for the AI decision making using the various static evaluators. The different evaluators were given different board states at different depths and were scored based on their performance. The board states were gathered from the website lichess.org. Lichess has puzzle tools where users can select popular puzzle themes to play.

Lichess and many other chess tools use the chess notation FEN (Forsyth Edward Notation), to represent board position. After scraping various FENs from the puzzle tool they are ready to be fed into the algorithm for evaluation. The FEN was converted into a two dimensional eight by eight array to represent the board. Integer values were assigned to pieces where negative values were black pieces. This array representing the board, a boolean to represent who turn it is, the evaluation, and some pointers to connect previous and next states was all that was used for state representation. A state representation for the position in Figure 1 would look as follows:

```
State{
    vector<vector<int>> board ={
      { 0, 0, 0, 0, 0,-4,-4,-6},
      { 0, 0, 0, 0, 0, 0,-1,-1},
      { 0, 0, 0, 0, 0, 0, 0, 0},
      { 0, 5, 0, 0, 0, 0, 0, 0},
      {-1, 0, 0, 0, 0, 0, 0, 4},
      { 0, 0,-5, 0, 0, 0, 0, 1},
      { 0, 0, 0, 0, 0, 0, 1, 0},
      { 4, 0, 0, 0, 0, 0, 0, 6},
    };
    bool turn = true;
    double eval = -1.0;

    State* last_state;
    vector<State*> next_moves;
};
```

2.1 Evaluators

The material evaluator was the minimum required for the AI to perform reasonably and was used as a base for the other evaluators to be built on. The evaluations were calculated as follows:

- **Material**= $WhiteMaterial - blackMaterial$
- **Mobility**= $Material + (whiteMoves - blackMoves)/C1$
- **Checks**= $Material + or - C2$ (if in check)

In the current implementation C1 = 2 and C2 = 50. These numbers were chosen by hand after some manual evaluation.

To determine the correct move for a given position a Stockfish API was used. The API accepts a FEN as a POST body, and returns the top move in the position. If the move from the algorithm matches the move given by stock fish the test passes and the evaluator is given a score of 1. It scores 0 otherwise. The tests were run on each evaluator at search depths from one to three. The test scores were averaged to get the average performance of each evaluator at different search depths. A depth of four was also tried, but many of the larger board states would take too long to run, and the tests that finished with a depth of four gave nearly the same results as depth three, so the following results and discussion will be focused on the results gathered from running the tests with depths one to three.

3 RESULTS

Based on the results, depth was the most important factor for making the correct move. As depth increases the probability of the algorithm succeeding increases.

An important note. Many of the puzzles generated by lichess were a 'Mate in 2' style puzzle, or a puzzle where the correct combination of moves can be achieved in only a few moves, such as the example in Figure 1. As a result, a depth of three was often enough to find the state with an evaluation much higher than other states. This also explains the significantly better performance of the Checks evaluator at depth 1. Many of these styles of puzzles start with a check. Although the Checks evaluator excelled at those puzzles, its decline in performance at higher depths is evidence of its poor performance on any other type of puzzle.

The Material and Mobility evaluators performed similarly. As depth increases and we see more puzzles less

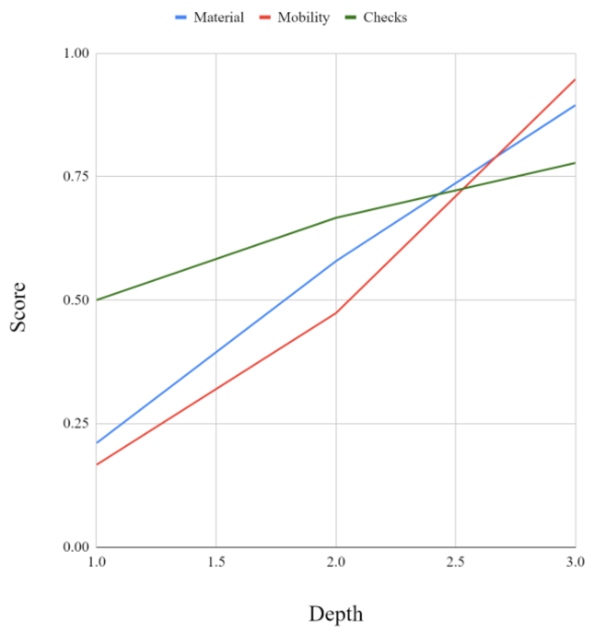

Figure 2: Scores at Different Depths

reliant on the quick gain of material with a smaller combination of moves, the mobility evaluator out performs the material.

It is clear that the best evaluation to use for a given position is dependent on the stage of the game, and some various state attributes. When there were less pieces on the board and the game was closer to the end, mobility was a less important factor and other evaluations performed better. If it is still early in the game, and there are still many pieces on the board, then the mobility evaluator has a higher chance of finding the correct move because there are less winning tactics that gain material available in the position.

The best evaluator would take the stage of the game as a factor and use it to change the calculations for evaluation. After looking into the source code for the stockfish API used as a benchmark, this is the strategy they were using.

4 DISCUSSION

The focus of this project was to see what depths are required for some simple static evaluators to perform on a comparable level to a top chess engine like Stockfish. The best performing evaluator was able to achieve 94.7% accuracy at a depth of three on randomly selected puzzles. While this metric might point to an accurate chess engine, there are many other factors to be considered.

A more accurate testing strategy would be to give the algorithm not only puzzle position, where there is a largely correct solution, but also positions from games where the correct move might just be defending, developing, or gaining space. These types of moves are classified more under positional play, where the current implementation covers more tactical play. A database for these kinds of positions is harder to find than puzzle positions, but is just as important for a successful chess AI.

Another topic of discussion is the constant values used in the evaluators C1 and C2. These values were chosen with minimal testing and better performing values almost certainly exist. With more time it would be easy to try out different values and rerun the tests over and compare results to find an optimal setting for these constants.

5 LIMITATIONS

The evaluators chosen for the current approach leave many limitations. None of the evaluators factor in the advantages that can be gained from good pawn play. Currently the only value pawns provide in any evaluation is their material value of 1. In reality a pawn can provide much more value by defending other pawns to create a pawn chain to hold space, defending other higher value pieces, and creating walls to prevent checks and captures. By expanding the evaluators to account for some of these pawn attributes the algorithm would perform much better in non tactical positions.

Another limitation is a lack of recognition for some chess rules. Because the focus of this project evolved into solving puzzles, some other simple rules of chess were not properly implemented in the evaluation. Castling, promotion, and en passant were not coded. These are all situational moves that are not required for solving puzzles. As a result, if the current AI was to be used to play a game from start to finish it would be handicapped by not recognizing these available moves.

The last limitation I want to mention is how slow the current implementation is. The code for the pruning algorithm was well tested and behaves as expected. But the slow down is from the chess engine. The chess engine was quickly written in c++ for the sake of this project and is not optimized for fast move calculation. As a result each state expansion takes longer than it should and the algorithm is much slower as a result. Integration with a faster more established chess engine, or more time to optimize the one used, would greatly increase search depths achieved in a reasonable time.

6 POSSIBLE EXTENSIONS

The sample size of tested positions was relatively small. It takes some manual effort to gather the FEN strings from Lichess and add them to the testing script. Access to a proper database with historical game positions would make automating the testing process more inclusive and easier, and would greatly increase the accuracy of the results.

The evaluators chosen for the project are simplistic in nature. While this made early testing easier, now that the algorithm is more established some more complex evaluators could be tried at higher depths with the hopes of gaining better results.

Another interesting extension could be the use of a neural network for evaluation. Instead of trying to manually calculate the evaluation from the position with some formula, it could be more effective to pass the position as an input to the neural network and train it against top engines, or historical game data to get a good evaluation. This would require much more training data, but the result would be a much more dynamic evaluator capable of recognizing the state of the game without any manual intervention.

ACKNOWLEDGMENTS

Thank you to Professor Wheeler Ruml and Devin Thomas for teaching me so much this semester.

REFERENCES

- Arman Maesumi Playing Chess with Limited Look Ahead Department of Computer Science University at Austin 2020, https://doi.org/10.48550/arXiv.2007.02130
- Donald E. Knuth, Ronald W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence, Volume 6, Issue 4, 1975, Pages 293-326, ISSN 0004-3702, https://doi.org/10.1016/0004-3702(75)90019-3.