Finding checkmates in one and two move chess puzzles

Aditya Srivastava
adityaks@stanford.edu

Ahmad Nazeri
ahmad.nazeri@stanford.edu

***Abstract*—The goal of this project was to develop a chess engine that focused on finding checkmates across different chess puzzles. By leveraging the minimax algorithm, we were able to correctly identify the sequence of moves that would result in checkmate. But as we increased the number of moves we looked, using the minimax algorithm became really slow and inefficient. Thus, we decided to leverage alpha-beta pruning which showed significant improvements.**

***Index Terms*—chess engine, minimax, alpha-beta pruning**

I. Introduction

Solving Chess puzzles have always been intellectually stimulating and we attempted to apply the knowledge gained during CS221 to solve chess puzzles. We developed an engine that would find the optimal solution given the FEN string as input and output being the subset of moves and compare them with the optimal set of steps provided by Human experts or known powerful chess engines a.k.a Oracles. The game of chess is considered an intellectual game and a game that requires a lot of intelligence. Developing an engine that can play chess effectively is a step towards developing artificial intelligence.

II. Related Works

A. Programming a Computer for Playing Chess by Claude Shannon

Programming a Computer for Playing Chess is one of the first papers that discusses the topic of developing a chess engine. Shannon states that each position of the game is either a win for white, a draw, or a win for black. And if we can play the game, we can identify the exact sequence of wins to get a win or at least a draw. The issue becomes that the possibilities of chess is endless which becomes more difficult to solve it. Thus, he recommends that we use an approximate *evaluation function* and minimax algorithm in order to more practically solve the problem at hand. The evaluation function provided in the paper is a little different than the one we leveraged. Since, we were interested in identifying checkmates, we weighted the moves that resulted in a check and checkmate more heavily and didn't take into consideration doubled, backwards, and isolated pawns.

B. Balrog - An AI Chess Engine by Arijit Banerjee and Aparna Krishnan

This paper is from CS221 students that wanted to develop a chess engine that would create a UCI protocal which made it easier to leverage existing GUIs. They utilized both minimax and alpha-beta pruning in order to identify the best moves at each position. The really interesting part about this paper is the aspect of move ordering and the quiescence search. Since we were focused on short puzzles, we thought that it wouldn't have been valuable to implement these features. In hindsight, it might be valuable to implement, at least, move ordering; especially, if we are leveraging alpha-beta pruning.

C. IBM's Deep Blue

IBM's Deep Blue became one of the first successful engines that beat Garry Kasparov, the world champion at that time. The event really gave inspiration to the technologists and the world that computers can be intelligent. The implementation of IBM's Deep Blue and our chess engine varies drastically. One key difference is that they utilized a database of moves in order to improve the process time. And of course, the compute resources they had available was significantly more than the resources we had for this project. Thus, because of the limitations, IBM's Deep Blue was more around inspiration rather than a resource that we can directly leverage in our project.

III. Dataset

For this project, we leveraged two datasets: **Wtharvey** and **Ed Collins**. We used the datasets and separated the set of puzzles as use cases which have mate in two, three or four steps to evaluate how many of these use cases our engine solves and how much time is taken. The main challenges we faced was getting the input data in the right format and making sure that we have large amounts of data in order to successfully test our chess engine. The input data is taken from the internet and had many inconsistencies in the PGN. Sometimes, the header didn't have information about FEN String and other times it was missing information about the number of steps or list of moves from Oracle. we cleaned up such games from our list of input data points to our solution.

In order to make it easier to consume the dataset, we developed a script that would clean the data. The cleansing of the data output the results into a table with columns **board, color, number_of_moves_to_mate, move_sequence**.

A. WT Harvey Dataset

The wtharvey dataset comes from http://wtharvey.com and contains data for checkmate in 2, 3, and 4 moves. The dataset

contains 220 puzzles, 488 puzzles, and 461 puzzles for mate in 2, 3, and 4 respectively.

B. Ed Collins Dataset

Ed Collins dataset had 5400 chess puzzles out of which 4400 puzzles remained after cleaning. We also generated three other sets of puzzles from the Wtharvey data-set of 222, 490 and 463 puzzles for mate in 2,3 and 4 steps for the given color.

IV. Baseline

The baseline is the moves suggested by the oracles and all our metrics compared against the optimal output. For all the puzzles, the datasets include the best sequence of moves which will we will leverage as the oracle. These sequence of moves will be considered as the baseline for our project.

V. Main approach

We used Minimax with Alpha-Beta pruning as chess engine to determine the optimal moves. The engine takes an input board and the color of the mover. We leveraged the python-chess library as framework for move generation and validations and passed FENs as inputs to build the Board. For this project, we developed the following main components.

A. Game database reader

The reader would read the puzzle from clean dataset, parse it and load into memory. Validate Oracle's suggested number of steps using a standard engine for data quality issues.

B. Puzzle Solver Engine

For each puzzle, we utilized alpha-beta pruning algorithm (can utilize minimax as well) to determine the sequence of moves. The engine used as state of **(best_move_value, best_move, best_boards)** where best_move_value is heuristic function's value, best_move is the best move at each position and best_boards is the sequence of moves. The heuristic function we used is based on Claude Shannon's heuristic function. Ours is focused around checkmate (a value of 1000000), check (a value of 10000), and insufficient pieces (a value of -10). Otherwise, we took into consideration the difference between player's and opponent's piece and each piece has a value of the following based on type. A pawn is worth 1 point, both knight and bishop are worth 3 points, rook is worth 5 points, queen is worth 9 points, and king is worth 0 points.

C. Evaluation of result

Generate the proposed metrics based on the output of various engines.

D. Display

Display the metrics and moves suggested by the engine. We also leveraged the python chess library's display to present the start and end position for a more intuitive output.

TABLE I
Puzzle Results

Mate in	# Puzzles	# Correct	# Incorrect	Avg Time/puzzle (s)
1[a]	304	272	32	0.006
2[a]	3410	2738	672	0.504
2[b]	220	220	0	1.501

[a]from the Ed Collins dataset. [b]from the WT Harvey dataset.

VI. Results Analysis

As a result of the limitations on resources, we decided to run puzzles for only mate in one and mate in two. While utilizing the clean data, we captured the following results: The table shows that there is a **17%** incorrect rate to the engine. This largely has to do with us leveraging the alpha-beta pruning algorithm. Since we are not sorting moves; when we prune branches it causes us to avoid evaluating some of better moves that might be beneficial later on. As utilized by the Balrog paper, it might be beneficial to implement "move ordering" in order to identify the best moves at each position.

A. Example

The following is an example of one of the puzzles. First we pick a game which has the following information.

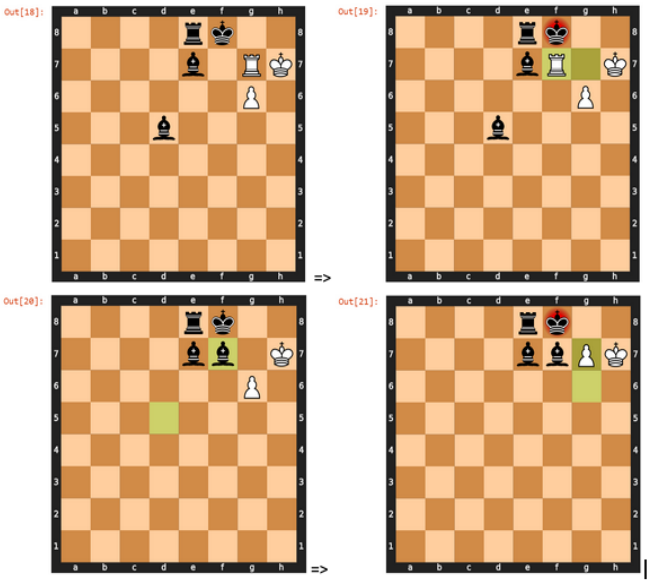
[White "Mate in two"]

[Black "White to move"]

[Result "*"]

[FEN "4rk2/4b1RK/6P1/3b4/8/8/8/8 w - - 0 1"]

List of moves from Oracle: 1. Rf7+ Bxf7 2. g7 white wins as white . The Start and End board positions for the solution generated by the code is (Jupyter notebook version):



The image above shows the sequence of moves that gets predicted by our chess engine.

VII. Future Work

As we continue to work on this problem, there are a few things that we have discussed that could be beneficial to make our chess engine more efficient.

First, it might be beneficial to add a move ordering module that would allow us to determine the order of moves the engine will search. This could improve our search when we are leveraging alpha-beta pruning.

Second, we want to implement our engine to leverage multiple processors. We would love to implement multi-threaded search which would speed it up by a factor of number of processors present. Also given sufficient memory, we could store evaluation of sub-trees which would speed up computation at the cost of memory. In addition, having a database of end game moves will definitely improve the processing time and make the engine more efficient in terms of time.

Third, while we were focused on solving puzzles; it might be an interesting project to develop this into a full chess engine that would allow a user to play against it. This would require more time outside of the classroom and during our free time.

A complex game like chess has so many variations and covering all such corner cases would require a lot more use cases and time than we could devote to this project however we were happy to implement what we learnt in the class of CS221.

VIII. Code

The code is publicly available on GitHub. It can be viewed at **https://github.com/ahmadnazeri/cs221_project**.

References

[1] Shannon, C. E. (1988). Programming a Computer for Playing Chess. Computer Chess Compendium, 213.

[2] Banerjee, A., amp; Krishnan, A. (n.d.). Balrog - An AI Chess Engine.