Fuzzing Chess Engines

Andrew Song
University of Virginia
Charlottesville, USA
ajs4aw@virginia.edu

Abstract—**This paper describes the approach and findings of a project to fuzz three different chess engines, which are programs that can find the 'best' move that should be played in a given chess position. Specifically, the chess engines analyzed were Stockfish, RubiChess, and Halogen. Creating a fuzzer to generate large quantities of chess positions, these positions were then input into the chess engines to analyze how the engines' suggested best moves differ and to search for patterns in these differences. Ultimately, some trends were found in how the engines differ in their output, and the paper concludes with a few ideas about how related works may be able to build on this work.**

I. Introduction

Chess, one of the most popular board games all around the world, has continued to grow in popularity throughout the past couple decades. At the same time, chess engines have also become much more developed and sophisticated during this period. Chess engines serve many functions, but perhaps its most important function, and the one I will explore in my project, is to suggest the 'best move' in any given position. This functionality is a key part of how chess players improve their ability and has altered how players prepare for tournaments and competitions. Currently, there exist a number of different chess engines, some of which take different approaches to how they analyze games.

Given this, in my project, I aim to solve the problem of: How and why do different chess engines differ in their analysis of a chess position? Chess engine analysis is one of the most important aspects of competitive chess, and given the multitude of chess engines that have been created today, it can be important for chess players to understand the differences between engines and why these differences might happen to be able to distinguish between engines.

In this project, I examine the differences between three chess engines in specific: Stockfish, RubiChess, and Halogen. These engines were chosen for being open-source, and also for their strong rankings according to the CCRL (Computer Chess Rating Lists). All three engines were ranked around the top 25, with Stockfish ranked at 1, Rubi at 5, and Halogen at 26.

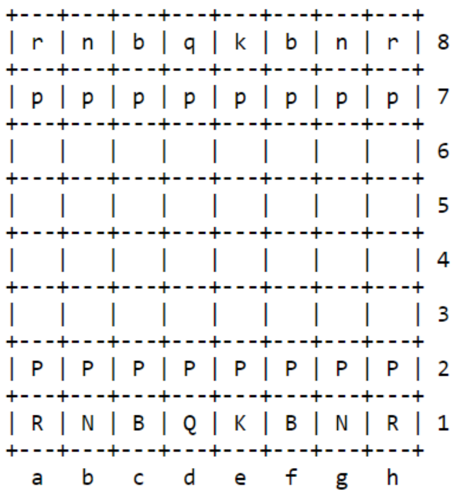
While there have been works exploring the differences between chess engines, such as an analysis of the popular engines, Fritz and Junior, these works use existing input positions that come from games, like the World Chess Championship matches, or some other existing test collection set [1]. This differs from my method of randomly generating chess positions from a given seed input to use as input for the engines. Additionally, while fuzzing has been used to generate engine input to chess engines [2], the goal of that work was to fuzz the Stockfish engine specifically to find buggy behavior in the engine. On the other hand, my work does not concern buggy behavior in the chess engines, but rather, collects the specific moves of multiple engines and analyzes them.

As mentioned, fuzzing is a fairly unique approach for this type of project since most works make use of game-found positions. While using positions that come from actual games inherently have more practical value because they examine highly realistic positions, they also are limiting in the sense that, when following a logical progression of the game, especially in high level games, there are usually only a small handful of "good" or "winning" moves that can be played. Thus, it is more difficult to find differences in engine outputs. On the other hand, the benefit of fuzzing is that a large number of positions can be generated quickly, which not only naturally increases the chances of finding a difference in engine outputs, but also generates positions that are more random and generally allow for a larger number of possible "good" moves. In these random positions, we are more likely to find patterns in how the engines 'think' and what kinds of decisions they prefer because they may consistently select a certain type of move among many others as opposed to a few others. Also, by taking into account differences as feedback for the fuzzer, making incremental changes to positions that cause engine differences can make it more clear what exactly is causing the engine to make its decisions. For example, for some position causing engine differences, we can make incremental changes to that position to examine if and how the engines differ again, while seeing clearly what changed in the position to cause that. Additionally, random positions simply have more variety. For example, a fuzzed position may lead to having 4 queens on the board, which is something that is very rarely seen in actual games. This kind of variety introduces a lot of new potential ways that engines may differ in their thinking.

Overall, this work serves as a foundational project in the relatively niche topic/field that is competitive chess. There have not been a lot of close and careful studies of how chess engines operate, and even less have taken the approach of using randomly generated positions like this project does. This project lays out some groundwork for future chess engine studies like this one while also providing some potentially interesting findings about 3 chess engines in particular (Stockfish, RubiChess, and Halogen) that will have to be further, and more carefully, studied to be substantiated.

II. Approach

Chess engines take, as part of their input, a 6-part string representation of the chess board and pieces containing information on the physical layout of the pieces, the active color whose turn it is, any potential en-passant moves, etc. The first part of the string, the representation of the physical pieces, was of greatest interest for fuzzing purposes. As an example, the starting state of the chess board can be represented by the following string: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR. In this representation, each row is separated by a forward slash, and the first "row" of the string corresponds to row 8 of the chess board. A "row" of the string can be read left to right to determine how the pieces fall on the board. Finally, numbers represent the number of spaces without pieces on them, and lowercase is for black and uppercase is for white. Thus, by this representation, row 8 of the chess board would have "r" (black rook), "n" (black knight), "b" (black bishop), and so on, from left to right, as seen in Figure 1.



```
+---+---+---+---+---+---+---+---+
| r | n | b | q | k | b | n | r | 8
+---+---+---+---+---+---+---+---+
| p | p | p | p | p | p | p | p | 7
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   | 6
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   | 5
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   | 4
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   | 3
+---+---+---+---+---+---+---+---+
| P | P | P | P | P | P | P | P | 2
+---+---+---+---+---+---+---+---+
| R | N | B | Q | K | B | N | R | 1
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h
```

Fig. 1. Starting chess board visualization

Given this, my approach to solve the problem presented in Section 1 was to fuzz the chess engines of interest by making incremental changes to the physical position representation strings, making it fast and easy to generate a large number of chess positions for the engines to analyze. These incremental changes consist of single move, or single piece, differences. In total, 5 different types of changes were implemented: 1) Changing a piece on the board to a different piece, e.g. a bishop to a knight, 2) Changing the color of a piece, 3) Adding a piece somewhere on the board, 4) Removing some piece on the board, and 5) Moving some piece on the board from one square to another. As an example, if we took the following position string: "4B3/2K5/3P4/2k5/8/7b/8/8" and added a black pawn onto the sixth row, it could result in the following string: "4B3/2K5/3P4/2k5/8/1p5b/8/8". This is visualized in Figure 2.

Additionally, the fuzzer contained logic to make sure that the chess positions were valid from the standpoints of 1)

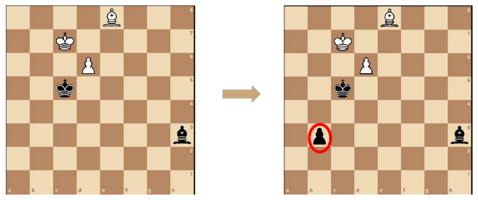


Fig. 2. Add a piece

Staying within the confines of the board, and 2) Recognizing the invariants of any given chess position, such as no side having less than or more than 1 king, and not having any pawns on the first or last rows of the board.

From there, the fuzzer continued in a process shown in Figure 3. Once the physical position string was generated (Labeled as "String" in Figure 3), the complete input was generated for each engine. This "complete input" consisted of the whole 6-part string representing the overall, complete state of the chess position as well as other commands that allowed each engine to run properly, such as "go depth 20" which told the engines to find the best move, looking ahead 20 moves (20 was an arbitrarily chosen number that is generally considered reasonable depth for an engine). Each chess engine generally follows similar input patterns, but they differ enough such that three different inputs (all meaning the same thing) had to be created for each engine. These are represented by "State t1", "State t2", and "State t3" in Figure 3, where t1 is Stockfish, t2 is Rubi, and t3 is Halogen. Because there is little documentation on chess engines in general, it was tricky to find the correct input format for them to run properly. Below is an example of the contents of an input file to Stockfish.

```
position fen 8/6KP/8/4k3/3q4/8/8/8 w - - 0 1
go depth 20
ucinewgame
```

The first line includes the aforementioned 6-part position sting and establishes the position that we want Stockfish to analyze. The first part is the physical position string, 'w' indicates that it is white's move, the first '-' indicates that no side currently has the ability to castle, the second '-' indicates that no side can make an en passant move, '0' is the number of half-moves since the last capture (used to track draws in the fity-move rule), and '1' is the number of full moves. The second line tells the engine to search for the best move as described earlier, and the third line essentially waits for the engine to finish finding a move before exiting.

Invalid inputs, such as a position where the two kings are on two squares adjacent to each other, were recognized by the engines and ignored. Further, any positions that were a checkmate were also ignored since there was no best move to be suggested by the engines.

After running the chess engines on their respective inputs, the engines' outputs were collected and analyzed, as shown in the "Compare" box in Figure 2. If any of the engines differed in their suggested best moves, the chess position for which the engines differed was used as feedback to my fuzzer to use as a base position to fuzz again. An example Rubi output file is shown below:

```
...
info depth 1 seldepth 1 multipv 1 score cp -450 nodes 36
nps 18000 hashfull 0 tbhits 0 time 2 pv d2b1 a1b1
...
info depth 20 seldepth 3 multipv 1 score cp -430 nodes 74
nps 37000 hashfull 0 tbhits 0 time 2 pv d2b1 a1b1
bestmove d2b1 ponder a1b1
```

We can see that the engine outputs its ”thinking” process from depth 1 to depth 20. The last line contains the best move that the engine suggests for the position and is the primary line of interest when comparing the engines for the fuzzer. In this case, Rubi suggests the best move in this case would be to move the piece on d2 to the square b1. Also of note, on the depth 20 line, the ”score cp -430” indicates that the suggested move would result in black having an advantage of 430 centipawns, or 4.3 pawns in technical value, according to Rubi’s engine.

Once the fuzzing process was complete and lists of positions that resulted in engine differences were created, a separate program analyzed these files to further examine the output and collect positions that caused ’significant’ engine differences. From there, I looked at the positions to determine patterns.

One limitation to this approach is that extremely lopsided positions are fairly easily generated. Extremely lopsided positions generally do not offer much value because engine differences are typically insignificant, even if they do arise. Furthermore, with pieces often being scattered around the board in very unorthodox manners as a result of random generation, it can be difficult to recognize patterns in the positions where engines give different outputs.

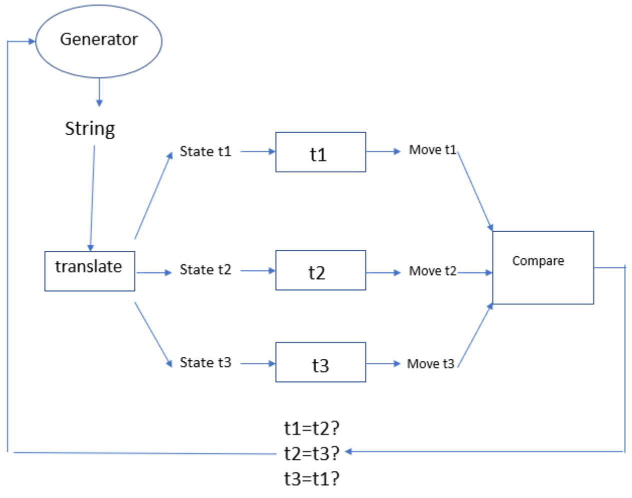


Fig. 3. Fuzzing process

III. Evaluation

The evaluation process began with addressing the research question of: How often do engines provide different outputs? To measure this, I first collected data on the chess positions until there were a total of 200 different outputs between any pair of engines, whether it was Stockfish and Rubi, Rubi and Halogen, or Stockfish and Halogen, and recorded how many times each engine pair differed from each other, averaged across 4 trials. The distribution of the number of differences between each pair of engines are shown in Figure 4.

Engines	Difference Count
Stockfish, Rubi	53
Rubi, Halogen,	85
Stockfish, Halogen	65

Fig. 4. Distribution of Engine Differences in 200 Differences

From these results, it seems to be the case that Halogen, which was ranked considerably lower than the other two engines, generally has the most differences with other engines. Further, Stockfish and Rubi, which are both top 5 engines, had the least amount of differences. However, with so few trials, these numbers cannot allow meaningful interpretation but rather serve as baselines to get an idea of how many differences we can expect between engines, in general, and to see clearly that the engines do differ frequently.

To answer the research question in another way, I collected the number of positions in which the engine outputs differed from each other out of 100 valid chess position inputs. These numbers varied based on the seed input, which consisted of the start position, a position in an opening stage, a position in a midgame state, and a position in an endgame stage. In specific, the number of differences between Stockfish and Rubi, Rubi and Halogen, and Stockfish and Halogen were all counted, for each of the four seeds. Across four trials, the results are shown in Figures 5-8 below.

Engines	Difference Count
Stockfish, Rubi	36
Rubi, Halogen,	42
Stockfish, Halogen	35

Fig. 5. Seed - Starting Position

Engines	Difference Count
Stockfish, Rubi	39
Rubi, Halogen,	52
Stockfish, Halogen	45

Fig. 6. Seed - Opening Position

Across all of these, the differences between engines was a bit less pronounced. However, there still were slightly more

Engines	Difference Count
Stockfish, Rubi	36
Rubi, Halogen,	40
Stockfish, Halogen	36

Fig. 7. Seed - Midgame Position

Engines	Difference Count
Stockfish, Rubi	35
Rubi, Halogen,	41
Stockfish, Halogen	41

Fig. 8. Seed - Endgame Position

differences between Halogen and the other engines (particularly with Rubi) than other pairs. Additionally, the seed seemed to impact the number of differences, supporting the idea that engines vary more in the opening stage of the game when the theoretical number of "good moves" is higher because positions are more volatile.

The next research question that I aimed to answer upon gathering these quantitative measurements was: How do the engine outputs truly differ from each other? This process contained both quantitative and qualitative analysis. First, I only wanted to examine chess positions where I believed the difference in engine outputs was actually "significant" because, otherwise, the engines may not be truly different in their analysis of a position. Thus, I analyzed the engine output differences and counted how many of the differences were actually "significant".

To detail the process of determining which differences between two engine outputs were significant, the concept of score, which was briefly mentioned in the Approach section when looking at a sample Rubi output, becomes important. For any given move in a position, a chess engine can output a score that will result from that move. For example, if the engine is given a position X, and a move to "search" (in other words, play that move), the engine may output a score of 30 cp, which will mean that, as a result of playing that move, the white side will have an advantage of 0.3 points (30 centipawns). Thus, I began by taking all the positions where there were engine differences and running the chess engine on the other's best move to find the score for that move and compare it against the score for its own best move. For example, if some position X resulted in Rubi outputting a best move of d4d5 and Halogen outputting a best move of c2c3, I evaluated Rubi's score for c2c3 and Halogen's score for d4d5 and then compared those scores against Rubi's and Halogen's scores for their bestmove output. Then, if the score difference in any of the two engines met a certain threshold, the difference was deemed significant. The threshold used was a 100 cp difference, which is technically the value of a single pawn. This was an arbitrary threshold, as a pawn advantage can be a big difference in most cases.

The results counting the number of significant differences are shown in Figure 9, averaged across 4 runs where the total differences were the total number of differences between each engine pair in 100 valid chess positions.

Engine	Total Diffs.	Significant Diffs.
Stockfish, Rubi	31	16
Rubi, Halogen,	38	5
Stockfish, Halogen	37	10

Fig. 9. Significant Differences in Engines

Interestingly, when it came to significant differences, Rubi and Halogen had the least amount relative to other pairs even though its differences were the highest amount. Further, Stockfish and Rubi, having the least amount of total differences, had the most amount of significant differences. Trying to find reasoning for this, as well as trying to find patterns in how the engines differed from each other, became a process of directly analyzing the positions that were causing significant differences in engine output.

Through this process, the reasoning for the higher significant differences between Stockfish and the other engines became apparent: Stockfish was generally much better at detecting checkmates. For example, in Figure 10, while Stockfish scored position as having "mate -7", meaning that Stockfish recognized a checkmate in at least 7 moves for black for the given best move, while Rubi assigned a -5387 centipawn value for the score, thus, recognizing a clear advantage for black but not recognizing the checkmate possibility. However, it was almost always the case that the engines did assign a huge score value to their best move anyways, meaning that these were positions in which there was a clear, big advantage for one side. Thus, practically, there may not have been significant differences in these analyses, but a significant score difference does exist.

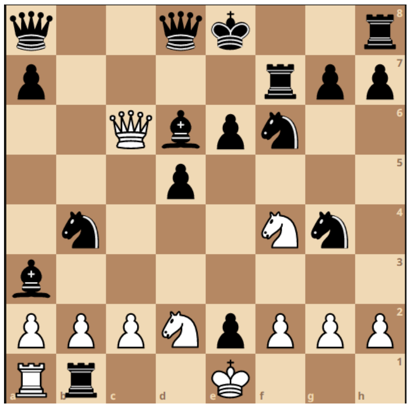


Fig. 10. Position where only Stockfish detected checkmate

Beyond those differences, a lot of significant differences seemed to arise when one engine "cared" less about the actual point values assigned to a given piece. In many cases, one engine would choose a move that would give it a lesser point advantage in terms of the point values of the pieces in favor of a move that may give it some other advantage. For example, in Figure 11, on the left board, Stockfish wants white to capture the bishop on c2 with its queen, while Rubi wants white to capture the knight on d2 with the queen. And on the right board with white to move, Stockfish opted to prevent a pawn promotion with queen to a1 while Rubi chose to capture the black knight with its queen and allow the pawn promotion.

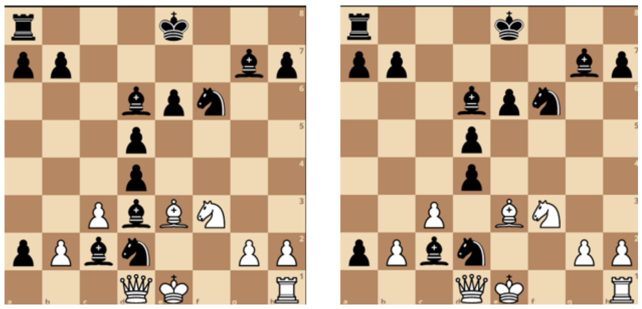


Fig. 11. Positions where engines opts for lesser points

Finally, the other pattern that seemed to be prevalent was that capturing tended to cause significant differences in engines. Many of the differences arose from positions where one side had a multitude of possibilities for capturing a piece. For example, in Figure 12 on the left board, Stockfish suggested capturing the pawn on e2 with the white queen, while Halogen suggested capturing the black knight on f3 with the pawn on g2. On the right board, Rubi suggests capturing the pawn on e2 with the white queen, while Halogen suggests capturing the knight on f3 witht he queen on f6.

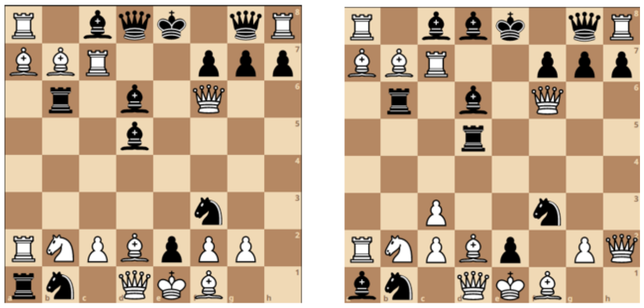


Fig. 12. Positions where engines make different captures

In addition to these patterns, there were a number of other positions that had significant differences but I was not able to piece together trends in either pieces, positions, or behaviors involved in the difference.

The method of evaluation described in this paper are comparable to other evaluation approaches. For example, the work comparing Fritz and Junior [1] also takes engine scores for a number of suggested engine moves and compares them with absolute value of score difference across the two engines to get an idea of the difference.

IV. Conclusion

Ultimately, in this project, several patterns and trends were discovered in the chess positions where different engines will suggest different "best moves". In some cases, it was seen that one engine will behave a certain way, while in others, it was seen that differences arise from certain positions/behaviors, but it was unclear if one engine tends to behave a particular way for these cases.

There exist a number of unknowns for this project, particularly with regards to how substantive and valid these discovered patterns and trends are. Further, as mentioned, for some patterns of chess positions causing different best moves, it is unknown in which direction the differences are happening, or if there is any direction at all. For example, in the capturing positions, it was difficult to tell if Stockfish tended to capture one way while Rubi captured another way.

In addressing the limitations of this work, first, a more nuanced and complex index is required to find positions of significant difference. A pawn difference, as used in this project, can vary in its significance because the difference matters less when positions are already lopsided, for example. Further, analysis of positions would require more data and also a more disciplined approach. Discovering patterns in the positions causing significant engine differences would be something that may require a lot of study on its own.

References

[1] J. Bar-Ilan, and M. Levene, "Comparing Move Choices of Chess Search Engines," June 2005.

[2] S. J. Lewis, "Win by Segfault and Other Notes on Exploiting Chess Engines,"