Automating the *Automate Chess*

Yeshwanth Venkatesha

April 2021

1 Introduction

Chess is one of the most popular board game around the world with a rich history [3]. The popularity of online multiplayer chess has bolstered the interest in variants of chess, especially given the COVID-19 shutdown, there is a renewed interest in online multiplayer games. With free chess servers such as chess.com and lichess.org, the popularity of online chess has grown among enthusiasts and professionals alike. For instance, chess.com alone has a total of 15 million registered users and an active community of 100,000. The game is enjoyed worldwide not only in its standard version but in numerous variants [4]. The variants can be the result of modifying the game in various ways such as changing the board layout, changing the number of pieces, introducing new pieces, varying the number of players, etc. Chess is not something new in the Artificial Intelligence community. It is the most widely studied game in AI dating back to 1956 [8]. Owing to the advances in computer hardware and new machine learning techniques, computer chess has reached superhuman performance [1, 7, 5]. While this means that modern chess engines are practically unbeatable, they can be used to develop new variants of chess and online chess makes the use of engines seamless.

A variant named *Automate Chess* is recently popularized online on chess.com. In this variant, the players take turns to place pieces on the board, adhering to certain predefined rules and once all the pieces are placed on the board, a standard chess engine such as Stockfish [7] will play out till the end of the game to decide the winner. In this paper, we aim to statistically analyse the game to check if it has some obvious strategies that are unbeatable. For example, there is a popular belief that an 8 pawn, 9 knight setup always results in a win. The key question we ask is—Is there a dominant combination of pieces that can beat all other combinations? Further, we propose a Monte-Carlo Tree Search (MCTS) based agent to play this variant.

2 Automate Chess

In the standard variant of *Automate Chess* as on chess.com, each player will be given a budget of 35 points. Each pawn costs 1 point, knights and bishops cost

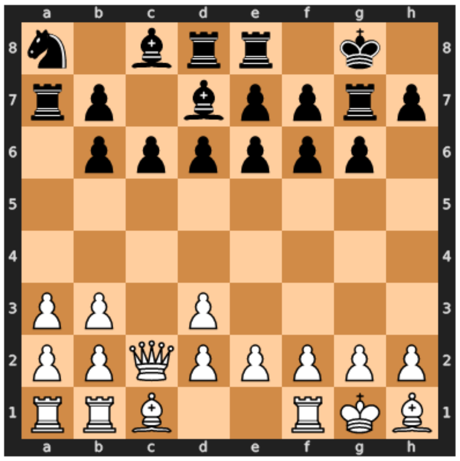


Figure 1: A position attained after the players place their pieces on the board

3 points, a rook costs 4 points and a queen costs 7 points. The players start with an empty board and take turns to move. White shall play the first move. In each move a player has to place one of the pieces by spending the allocated points. The first six moves of each player has to be pawns and a maximum of 10 pawns can be placed by each player. Pawns can only be placed in 2nd and 3rd ranks for white and in 7th and 6th ranks for black. Other pieces can only be placed in 1st and 2nd rank for white and 8th and 7th for black. The king is placed in the end only when no other legal move is available to the player. Once all the pieces are placed, Stockfish plays the game for both the players till the end to determine the winner. Standard rules of chess such as pawn promotion, en passant, draw by repetition and 50 move rule applies to the game. The only exception is that castling is not allowed. An example position attained after the players finish their turn is shown in Fig. 1.

3 Statistical Analysis

In this section we analyse if *Automate Chess* has a dominant combination of pieces.

3.1 Experiment Setup

Due to lack of publicly available datasets of real games, we generate a synthetic dataset to evaluate the statistics of the game. The games are simulated exactly like in chess.com with first 6 moves being pawns and 35 point budget to each player. The piece values are unchanged. The moves are picked uniformly at random from all the available legal moves at each position. This is not an ideal dataset as this would involve some positions that human players are unlikely

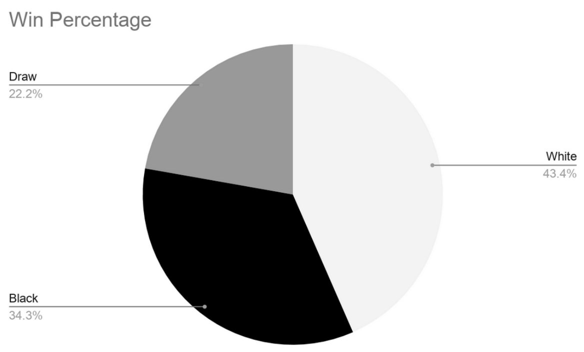


Figure 2: Overall Win Percentage

to play. However, it is a good baseline to evaluate the balance of the game. Disclaimer: The official Stockfish does not support board positions that are unreachable in the standard variant of chess. For example, a position with 10 pawns is never attained in the normal variant hence Stockfish deems it illegal. Such examples are discarded from the current dataset.

3.2 Overall Win Percentage

First we measure the overall win percentage of the game to check what is the tendency of the game ending in draw and is there a first move advantage. If yes, then what is the magnitude of first move advantage. This can be useful to make the game more balanced by introducing another form of advantage for the player moving second. For example, by giving additional points to the player moving second. As shown in Fig. 2, we observe that there is a 9% advantage for the white player which is huge compared to the estimated 2% in standard chess. Also, there is a considerable chance of draw at nearly one in four games.

3.3 Match-Ups

In this section, we examine the top-10 statistically most likely combinations in Fig. 3. Interestingly, the combination of standard variant comes up as the most played combination by both players. This endorses that the standard chess variant is very well balanced. We observe that the combinations with 2 queens consistently outperform the combinations with one queen suggesting that queens working in pairs are more powerful than a single queen supported by other pieces.

We analyse the top-10 match-ups that result in most number of draws in Fig. 4. In general, we observe that having more number of costly pieces (rooks and queens) on both side is resulting in a higher percentage of draws. This can be because of straight-forward exchanges of high value pieces.

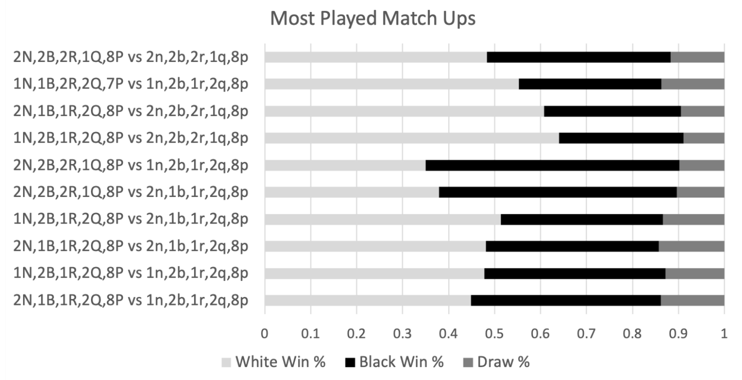


Figure 3: Most played match-ups

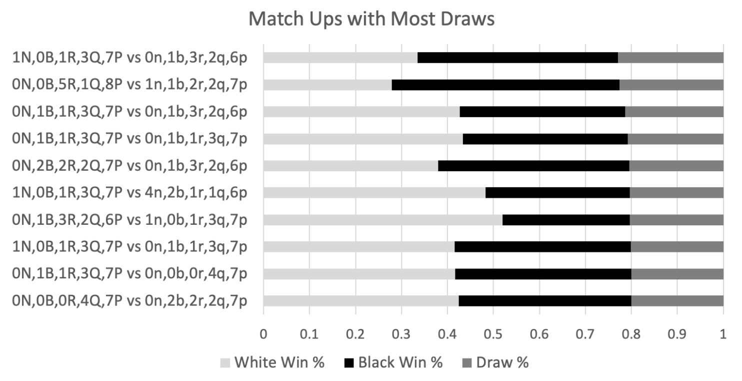
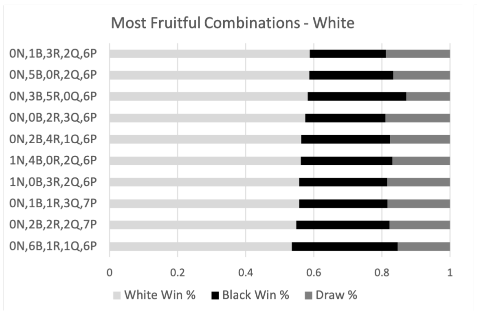


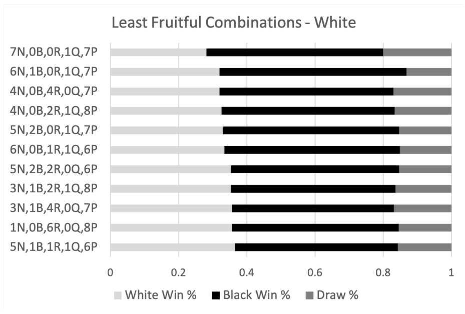
Figure 4: Match-ups ending in draw

3.4 Favourable Combinations

In this section, we examine the top-10 most favorable and least favorable combinations for each player (Fig. 5 for white and Fig. 6 for black). We observe that most of the fruitful combinations involve at least one queen and more than one more often than not. On the other hand, the least fruitful combinations are the ones where more number of knights are used. This finding contradicts the popular belief of more knights resulting in a winning combination. This suggests that queens are undervalued at seven points and knights are overvalued at three points. Additionally, since pieces in chess work remarkably well in pairs a new points system can help with the game balance. For example, increasing the cost for the subsequent pieces of the same kind. Note that since we are not considering the case with more than 10 pawns, it is possible that some of the dominant combinations are being left out.

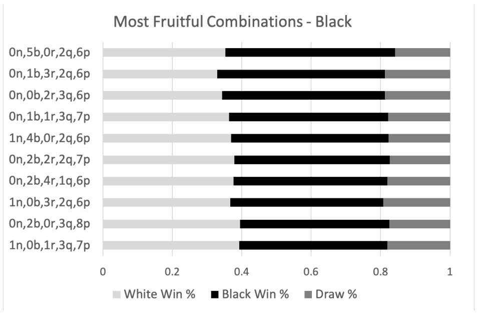


(a) Most fruitful combinations for white

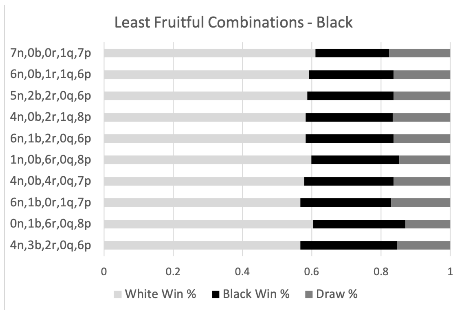


(b) Least fruitful combinations for white

Figure 5: Statistically favorable piece combinations for white



(a) Most fruitful combinations for black



(b) Least fruitful combinations for black

Figure 6: Statistically favorable piece combinations for black

4 MCTS-based Agent

Monte-Carlo Tree Search (MCTS) is a widely used technique in AI engines for board games. It uses the principle of exploration vs exploitation from modern reinforcement learning to classical tree search method. MCTS along neural networks has been applied to develop state of the art AI for playing Go [6]. MCTS aims to find the most promising moves in the game tree by sampling the search space and simulating the game till the end. This sampling and simulation of search space is called as playout or roll-outs. In a pure MCTS based approach, all possible actions at a given state is explored for a fixed number of times and the action is most number of victories is chosen as the preferred action. However, since the game trees generally tend to explode with numerous branches, it is not feasible to explore all the branches with a limited time limit. Moreover, exploring all the actions is often wasteful since in most of the games, there will be only a handful of good moves in a given state. On the other hand, it is possible that the agent can settle on a suboptimal move while there is a better move in the unexplored region of the tree. Hence, in practice

Playouts	Win	Loss	Draw	Avg. Time/move
10	9	8	3	3.81s
20	11	5	4	7.38s
50	11	6	3	18.92s
100	15	3	2	42.62s

Table 1: Results of MCTS based agent with varying strength vs an agent taking random actions in a 20 game tournament.

the tree search is performed with an explore-exploit tradeoff—current promising moves are chosen more often but at the same time other moves are also explored occasionally. This is achieved using the method called Upper Confidence bound for Trees (UCT) [2]. Typical MCTS method consists of four phases:

1. Selection: From the root node traverse the tree by selecting the most promising child node according to the UCT equation:

$$S_i = x_i + C\sqrt{\frac{ln(t)}{n_i}}, \tag{1}$$

   Here, S_i, x_i, and n_i are the value, empirical mean and number of visits of child node i respectively, and t is the total number of visits of the current node. If a new node is reached during this selection process expand it.

2. Expansion: In this process, a new node is added to the search tree.

3. Simulation: Once a leaf node is reached, the game is simulated till the end with a *default policy*—typically random simulation.

4. Backpropagation: Once the result is known, update the empirical means of the nodes in the path appropriately.

Since the same game state can be reached by multiple paths, it is helpful to avoid to duplicate nodes by using a cache table, known as transposition table to store the mapping of game state to a tree node.

We introduce an MCTS based agent to play *Automate Chess*. The game state is encoded by the pieces on the board and the player to make the next move. The actions are calculated based on the legal moves available from the given state. The selection of moves is performed based on the UCT method described in Eqn. 1 with parameter $C = 0.5$ by experimentation. Actions are chosen uniformly at random in the *default policy*.

Table 1 summarizes the results of a 20 game tournament with MCTS agent playing against an agent that takes random actions. We observe the MCTS based agent is winning more than 50% of the games even with 20 playouts and the win ratio reaches 75% with 100 playouts. The experiments were conducted on a machine equipped with Apple M1 chip and 16GB of RAM with minimal background processes. As the number of playouts are increased the time per move becomes very high. However, the current implementation is in python and has ample room for improvement. Hence, this can be circumvented with an efficient implementation of move generation and game simulation. Additionally, a parallel variant of MCTS can help speed up the search.

5 Conclusion and Future Work

In this paper we analyse the game of *Automate Chess* to check if there are dominant strategies with respect to choosing the combination of pieces alone. We find that there is no one particular combination that can win all the games for a player. In fact, the most favorable combinations, (0N, 1B, 3R, 2Q, 6P) for white and (0n, 5b, 0r, 2q, 6p) achieve $\approx 60\%$ success. On the contrary to the popular belief of all knight combinations being dominant, we observe that the combinations with more knights are some of the worst performing combinations.

Since this analysis was performed on a simulated dataset, it may not reflect the standard patterns from real games. A similar analysis on a real dataset will be very insightful. From the game design point of view, exploring different policies for assigning points to each of the pieces is a potential direction for future work. For example, increasing the price for the second queen once one queen is placed on the board, etc.

The proposed MCTS based agent performs reasonably well to beat an agent playing moves uniformly at random. However, random agent is a very crude baseline. Exploring other AI techniques such as q-learning and policy networks is also fruitful as it gives alternate engine to compete against. Moreover, since the performance of MCTS depends on the number of playouts at each step, it is important to optimize the execution time of the move generation and simulation which can be one of the primary focus of future work.

Source code: `https://github.com/yeshwanthv5/chess-automate`

References

[1] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. “Deep blue”. In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[2] Levente Kocsis and Csaba Szepesvári. “Bandit based monte-carlo planning”. In: *European conference on machine learning*. Springer. 2006, pp. 282–293.

[3] Harold James Ruthven Murray. *A history of chess*. Clarendon Press, 1913.

[4] David Brine Pritchard and John Derek Beasley. *The classified encyclopedia of chess variants*. J. Beasley, 2007.

[5] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *CoRR* abs/1712.01815 (2017). arXiv: `1712.01815`. URL: `http://arxiv.org/abs/1712.01815`.

[6] David Silver et al. “Mastering the game of go without human knowledge”. In: *nature* 550.7676 (2017), pp. 354–359.

[7] *Stockfish 13*. URL: `https://stockfishchess.org/`.

[8] Wikipedia contributors. *Computer chess — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-May-2021]. 2021. URL: `https://en.wikipedia.org/w/index.php?title=Computer_chess&oldid=1020133918`.