UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Kristiine Saarmann

Exploration vs Exploitation of Chess Openings Learner with Metropolis-Hastings Algorithm

Master's Thesis (30 ECTS)

Supervisor: Kallol Roy, PhD

Tartu 2022

Exploration vs Exploitation of Chess Openings Learner with Metropolis Hastings Algorithm

Abstract:
Opening lines are an integral part of ensuring a win, or at least a draw in chess. However, there are no well known openings learners that are available for the public that respond dynamically, thus allowing players to learn multiple openings at the same time. The problem with implementing such an algorithm is the fact that most chess databases use a notation that only expresses the movement of each step, not the board state as a whole.

It has been reported that the frequencies of the opening moves follow a power-law with an exponent increase linearly with game depth. This thesis presents an innovative approach that can compare the states in different opening lines and dynamically choose which one to respond with. To be able to implement such a response, it is necessary to determine how to pick which state to transition to. We have proposed a weighted Metropolis Hastings algorithm to decide which move should be played by the computer. The move is jointly decided as a weighted sum of the Metropolis Hastings output and parameters set by the learner. The algorithm allows for the modification of the parameters used, as well as the weight (importance) of them. Our method gives the flexibility of exploration (from Metropolis-Hastings) vs exploitation (from the player parameters) in the chess openings and adapts the game accordingly.

We have converted the opening lines represented in the string form to a list of matrices. This matrix encodes the board state of the game, with each entry of the matrix encoding the presence or the absence of a piece, the type of the piece, and the color. This transformation allows us to find matching states in the opening lines, even at a different move order. We have reported the distribution of chess board states under different number of iterations, parameters and weights, depending on the preferred playing style. We believe this is the first kind of study of chess openings from weighted stochastic approach and we are not aware of similar study to the best of our knowledge. Our method has the applicability for all chess learners and players, both new and experienced. In future we want to extend our method to include more complex variations, such as playing in positions with colors reversed, with one piece missing/displaced.

Keywords:
Acceptance/Rejection Sampling, Monte-Carlo Methods, Markov Chain, Metropolis-Hastings, Chess, Chess Opening Moves

CERCS:P175

Male avangute õppimisprogramm kasutades uurimise vs ekspluateerimise meetodit ning Metropolis Hastings algoritmi

Lühikokkuvõte:

Avangud on males keskse tähtsusega. Avangute oskus on oluline mängu võitmiseks või vähemalt viigi saavutamiseks. Hetkel ei eksisteeri avangute õppimiseks programme, mis oleksid avalikult kättesaadavad ning vastaksid dünaamiliselt kasutaja käikudele laiemalt kui ühe avangu ühe alaliini ulatuses. Säärase progammi implementeerimise teeb keerukaks juba see, et male mängude ning avangute ülesmärkimiseks kasutatav süsteem märgib ära ainult käigu, mitte malelaua seisu.

Varasematest uurimustest on teada, et avamisliigutuste sagedused järgivad võimsusseadust, kus eksponent kasvab lineaarselt mängu käikude sügavusega. Käesolev magistritöö esitleb innovatiivset lähenemist avangute õppimisele. See hõlmab endas malelaua olekute võrdlemist erinevate avangute lõikes ning dünaamilist vastukäigu valimist. Selleks, et kirjeldatud vastukäigu valikut implementeerida, on tarvis kindlaks teha, millist käiku kõigist võimalikest valida. Meie kirjeldatud mudel kasutab kaalutud Metropolis Hastings algoritmi. Algoritm kasutab otsuse langetamiseks kombinatsiooni Metropolis Hastings algorismist ning parametriseeritud sisendväärtusi, mille saab määrata kasutaja. Parameetrite hulka kuuluvad nii binaarse väärtusega muutujad kui ka väärtuste kaalud. Kirjeldatud mudeli tugevuseks on kombinatsioon uurimismeetodist (Metropolis Hastingsilt) ning eksploitatsioost (kasutaja sisenditest dünaamilisest kasutamisest).

Avangute andmebaasi põhjal oleme konverteerinud sõne kujul kirjeldatud avangud maatriksite listiks. Iga maatriks sisaldab endas kirjeldust malelaua hetkeseisust, kusjuures iga maatriksi element kirjeldab, kas vastaval ruudul on nupp, ning kui on, siis mis tüüpi ning värvi see on. Kirjeldatud info transformatsioon võimaldab meil võrrelda avanguid ka siis, kui käigud on tehtud muus järjekorras kui teoorias kirjeldatud. Antud töö sisaldab reportaaži käikude tõenäosuse jaotuses erineva arvu iteratsioonide puhul, erinevate parameetrite ning kaalutud väärtuste kasutamisel. Meile teadaolevalt on käesolev töö esimene, kus on kasutatud kaalutud stohhastilist lähenemist male avangutel. Loodud meetod ja programm on kasutatav nii malega alustajatele kui ka edasijõudnutele. Tulevikus on plaanis programmi muuta veelgi komplekssemaks. Selleks saaks avangutele lisada vahetatud värvidega positsioone, ühe-nupulise erinevusega positsioone ning teisi avangute variatsioone.

Võtmesõnad: Vastuvõtu/tagasilükkamise valim, Monte-Carlo meetodid, Markovi ahelad, Metropolis-Hastings, male, male avangud

CERCS:P175

Contents

1 Acknowledgements

I would like to sincerely thank my supervisor, Kallol Roy. His constant support, effort in helping my and time spent helped me immensely in writing the thesis and programming this work. I am especially grateful that he was willing to let me work on my own topic, something that was not actually in his area of expertise. Yet, he put in the effort to meet in the middle for which I am forever thankful for.

I would also like to thank the Erasmus+ program of the European Union. Their scholarship program allowed me to get an invaluable experience in studying in not one, but two foreign countries, and under a completely different system and curriculum. For anyone considering applying for the Erasmus+ studies program, I could not recommend it highly enough.

2 Introduction

Decision making to pick one of the possible lines among multiple alternatives is critical for winning in chess. Switching among the opening lines is a very complex process, given the number of factors (e.g playing style) that influence the choice. The moves in chess are of sequential in nature, hence the players select their next move among a finite set of possible choices prescribed by movement rules. The total number of possible movement combinations even in the opening stage is huge and many of them are not explored in practice. A database of the movement sequences considered the best so far has been explored in the **Encyclopedia of Chess Openings** [Wik21]. With the availability of open-source chess opening databases, the optimal moves for each player can be generated.

We have represented the chess opening moves as a directed graph where each chess board state is encoded as nodes and each edge corresponds to a legal move. Each chess opening move is represented as a Markov State-Transition on the directed graph and the probability of the transition is computed from the player's preferences, given as input parameters with some weights. The chess database gives the weighted measure (a.k.a popularity) from the frequency measures. The joint probability distribution of chess opening move of depth n is given by:

$$\begin{aligned} P(X_1 \cap X_2 \cap \cdots X_n) &= P(X_n | X_{n-1} \cap X_{n-2} \cdots X_1) P(X_{n-1} \cap X_{n-2} \cdots X_1) \\ &\vdots \\ &= P(X_n | X_{n-1} \cdots X_1) P(X_{n-1} | X_{n-2} \cap X_{n-1} \cdots X_1) \cdots P(X_1) \end{aligned} \tag{1}$$

where the X_i is the opening move at time i and will sample (with some probability distribution) from the universal set of board state space $S = \{S_0, S_1 S_2 \cdots S_M\}$. Each state $S_i, \quad \forall 1 \leq i \leq M$ encodes the information of whether the square is occupied or not, and the color and the type of the piece on it in the first case. For our algorithm, the first board state X_0 is fixed and will choose from set S with probability 1. This board state represents the game board before any pieces have been moved. The next board state X_1 given X_0 is computed as $P(X_1|X_0)$. The joint probability distributions of opening moves are difficult to compute and can be approximated from the data. Popularity distributions of chess openings are studied by Bernd Blasius and Ralf Tonjes [BT09] and found to follow Zipf's law because of the self-similar nature of the game tree of chess.

This thesis introduces a new approach to chess openings that uses a database of openings to compute the probability of transitions among the board states and a tune-able player policy. Thus our game comprises of novel combination of learning from openings statistics (from the chess database) and flexible self-play attributes (preferred style).

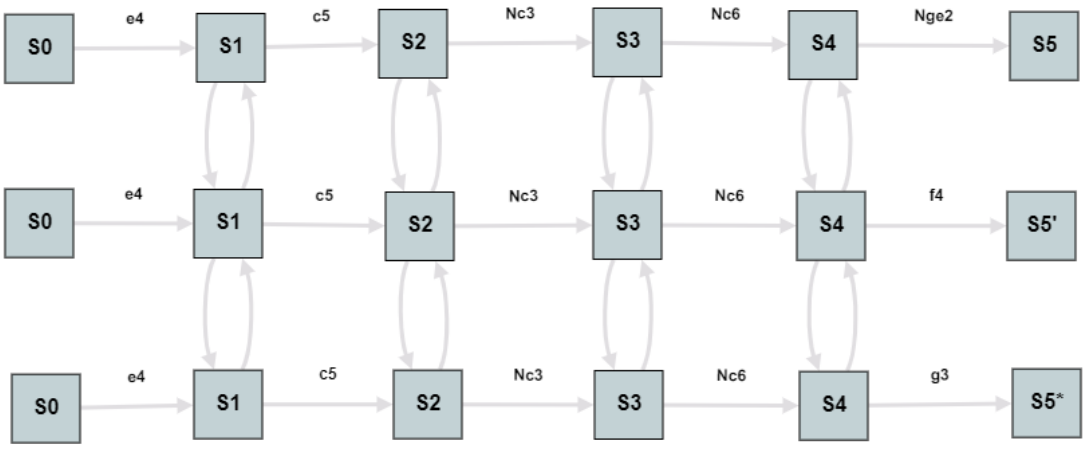

Figure 1. From up to down: Sicilian Opening/Chameleon Variation, Sicilian Opening/-Grand Prix attack, Sicilian Opening/Closed Variation.

The player can tune their policy using the five weighted parameters for generating the response moves by the computer. For an optimum chess opening strategy, there needs to be right combination of knowledge of human experts from the database (exploitation) and individual game style (exploration). The logic of switching between the opening lines is visualized on figure 1. The graph compares the three variations of the Sicilian Opening, each equal for the first five states and varying only at the last move. Our algorithm can switch between the equal states at any given point, with a probability calculated with a combination of exploitation and exploration.

There is a huge thrust of learning the correct mix of this exploration vs exploitation strategy in [SHM$^+$16, SSS$^+$17]. The thesis builds a computational pipeline of predicting the legal opening moves from legal opening move spaces with Monte Carlo roll-outs. An opening move is selected/rejected according to acceptance/rejection method (acceptance probability) A given by [SM09, RK16]:

$$A = \min(1, \frac{P(X_{t+1})P(X_t|X_{t+1})}{P(X_t)P(X_{t+1}|X_t)}) \tag{2}$$

where with probability A accept the next state X_{t+1}. Otherwise the next state is set as the current state $X_{t+1} = X_t$. The thesis has major four major contributions:

1. Investigating the chess opening problem first time using the dual combination of Metropolis-Hastings method and flexible playing strategy.

2. Usability for new and innovating methodology for learning chess openings.

3. Increasing the domain knowledge for the chess openings.

4. Building the open source code for the chess community.

The third section of this thesis gives an overview of chess rules, legal movements, and opening theory. The fourth section covers our proposed model for transforming the opening lines' into matrices, introduces the theory behind state transitions and gives an in-depth overview of our proposed algorithm. This section also covers the structure of the implementation in Python. The fifth section of this thesis analyzes the effects of the usage of different number of iterations run, a variance in the usage of weights and input parameters, as well as the effects of the sub-samples on the final probability distribution. The sixth section of this thesis covers the conclusions from the work done and touches on the future work planned.

2.1 Literature Survey

The first comprehensive source for chess openings is *Modern Chess Openings* by Nick de Firmian [DF08]. Recent work by Ian Cero and John Michael Falligant investigates the application of generalized matching law (GML) with Queen's Gambit over $71,000$ archived games [CF20]. Gambits result in a short-term loss of chess pieces, but have the potential of long-term positional advantages. Their research has shown that chess openings involving the Queen's Gambit exhibits orderly matchings predicted by the GML. Other important work in this area is of Theoretical Openings using sequence knowledge by Philippe Chassy and Fernand Gobet. They have investigated the role of long term memory for the players' performance. They have analyzed 76,562 games from a large chess database and measured the extent to which players deviate from their previous knowledge of chess moves ("openings"). Related work studies topological characteristics of a network of chess players, and study the degree of distribution and correlations in the chess community [CG11]. An emergent rich-club structure, composing by the chess experts comes out through the interactions.

3 Chess

This section introduces the background about chess needed to understand chess openings and the algorithm implemented. This includes the legal moves for each piece, the general game rules and the aim of the opening theory. The rules introduced in this thesis are based on Lasker's Manual of Chess Book [Las13] and the official FIDE handbook [Fid18].

3.1 Rules

Chess is a two-person strategy game that has been popular for centuries. It is played on an 8 times 8 board with 32 pieces - 16 per player. At the start of the game, the second row of the board (called the second rank) of each player's position is filled with pawns, while the first rank is filled with two rooks, two knights, two bishops, one king and a queen.

To win, one of the players has to checkmate an opponent in time. In some cases, especially in beginner's and intermediate chess with short time control, players also win by making their opponent run out of time. The fastest chess - up to 3 minutes per player - is called bullet chess. 3 to 10 minutes per player is called blitz, while classical chess is considered to last from 10 minutes hours, depending on the time format fixed.

The game can end in one of the two situations - one player wins and the other loses, or the players draw. To win by checkmate means to create a situation where the opponent has no more legal moves while the king is also in check. A stalemate is a situation where one of the players has no legal moves but is not in check. This is considered a draw by the rules. In tournaments, the players can also agree to a draw verbally or by repeating the same move pattern three times. Threefold repetition, dead position, fifty-move rule, and draw on time or by agreement are some of the other methods for drawing.

3.2 Movement

Pawn

Pawns can only move forwards, never backward. The pawn's first move can, but does not have to be two squares forwards. Every other step can be one square forwards, granted that there are no pieces blocking the square.

Pawns can only capture diagonally. Hence, to capture an opponent's piece, it has to be on the square diagonal to the pawn. Unlike for checkers, capturing is not compulsory.

Pawns have a unique capture, called *en passant*. *En passant* capture can only be used in one situation - when the player's pawn is in the fifth rank, and the opponent advances its pawn by two squares, also to the fifth rank in one of the adjacent squares. In this case, the player can capture the pawn by moving diagonally behind the pawn, ending its turn in the sixth rank.

Pawns, as the only piece to do so, can promote. When a pawn reaches the last square of the board (the eighth rank), either by movement forwards or by capturing diagonally, it can promote to one of the four pieces - a knight, a rook, a bishop, or a queen.

Figure 2. Starting with two steps (white), one step (black) and capturing diagonally (black to white) visualized.

Rook

Rooks can move in a straight line, either horizontally or vertically. The movement can be done either until the last square of the board or the square before another piece.

Rooks can capture at the end of their movement. Provided that a player's own piece is not blocking the way, the opponent's piece residing 1-7 squares vertically or diagonally from the piece can be captured, ending the player's turn on the square where the opponent's piece used to stand.

Rooks are involved in a special movement called castling. This is further explained under the 'king' section.

Figure 3. Legal movement shown for the rook on a8. Gray dots denote possible movements.

Knight

The knight's movement is the most complex out of all the pieces. Knights can move to any unoccupied square in an "L" shape from the starting square (one square vertically and two horizontally or two squares vertically and one horizontally). The knight is also the only piece allowed to move over other pieces in its path. To capture a piece, the knight has to finish its turn in a square occupied by an opponent's piece.

Figure 4. Legal movement shown for the knight on c3. Gray dots denote possible movements, red dots possible captures.

Bishop

Bishops can move diagonally and anti-diagonally, both forward and backward. Both players have two bishops at the beginning of the game - one on a white square, the other on a black square. The pieces can move to the end of the board following white and black squares, respectively, or to the square diagonally before a piece. Bishops, similarly to rooks, capture at the end of their move. Provided that their own piece is not blocking the movement, they can capture the first opponent's piece on one of the diagonals, ending their turn on that square.

Figure 5. Legal movement shown for the bishop on f5. Gray dots denote possible movements, red dots possible captures.

Queen

Queens are considered the most powerful pieces on the board. They can move in the same manner as a rook and a bishop combined - horizontally, vertically, and diagonally - until the last square of the board or until the last square before an occupied square. To capture a piece, a queen would have to finish its movement in the same square as the opponent's piece, provided it is the first piece in its movement path.

Figure 6. Legal movement shown for the queen on d4. Gray dots denote possible movements, red dots possible captures.

King

The king, being the most important piece of the game, is also one of the least potent ones regarding its movement capabilities. During the player's turn, the king can move one square in any direction. The exception to that rule, and the rules of the movements of all the other pieces, is that the king can never end the movement in check. That is, the king cannot move to a square that is under fire by an enemy's piece. Additionally, the player cannot move a piece covering the king from being in check. Such restriction of movement is called a pin.

The king can capture the enemy's pieces with regards to the rules mentioned above, finishing its turn on the square previously covered by the enemy piece. Additionally, the king is forced to capture a piece if it is under check and capturing the piece is the only movement where it does not end its turn under check. Similarly, other pieces are forced to capture an enemy's piece or move in front of the line of fire to cover the king if no

other legal movement is possible. If the king is under fire and the player has no legal moves, the game ends with the opponent winning with a checkmate.

To ensure the king's safety, a maneuver called castling can be used. Castling involves moving both the king and the rook simultaneously - the king two squares towards the rook and the rook either two or three squares over the king, depending on whether the king is castled "queenside" or "kingside."

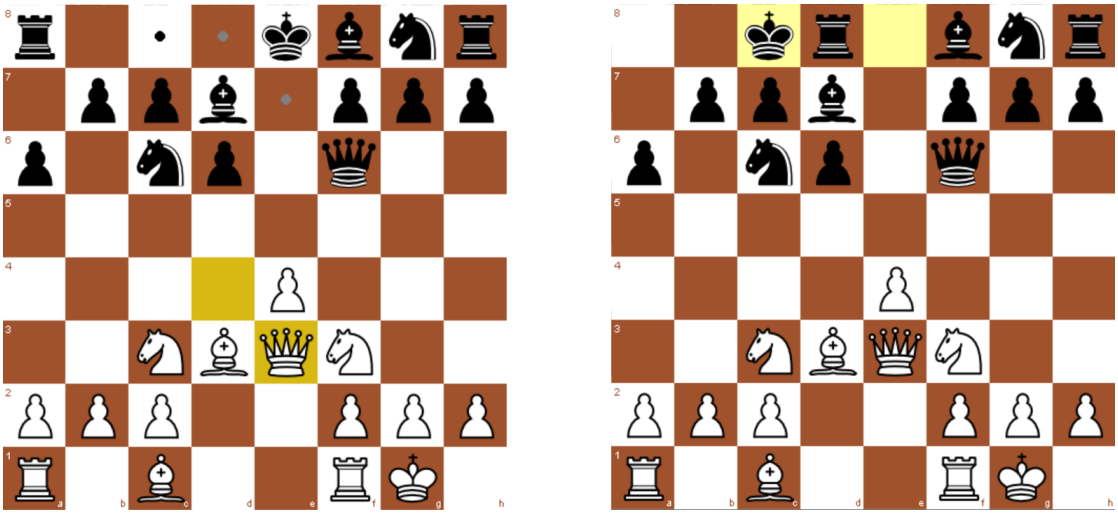

Figure 7. The black king and rook before and after castling queenside.

3.2.1 Openings

The first part of the game is considered the opening. Depending on the opening line, the theory can range from 10 moves up 40 moves in some cases. There is no standardized list of openings, although hundreds of them exist, many of them having a more than one variation (or up to tens of variations).

Opening theory is an integral part of the studies for a chess player. Often times, the opening will determine the course of the game. The main aims of the opening theory are the following:

1. Each player wants to control the center of the board, whether it be with pawns or with pieces.

2. The players want to ensure the king's safety. Most often, castling is used for this purpose.

3. A good pawn structure is integral for gaining a winning position. While some players prefer a more closed position (i.e. more pawns on the board), other prefer

a more open one. Another aim is to minimize the number of pawns that can be easily attacked, while giving pawn weaknesses to the opponent - isolated pawns, doubled or tripled pawns, backwards pawns, etc.

4. The players want to develop their pieces to the squares where they control the maximum number of squares while also being guarded by other pieces or pawns.

5. Lastly, the players want to play or transposition to the opening lines that they know better than their opponents and are comfortable with. Additionally, introducing novelty lines, often analyzed by a computer is a well-known way to get the opponent out of their preparation.

In our algorithm, we have used the database from the Encyclopedia of Chess Openings from the University of Siena [Tea21]. This database includes 1944 best opening lines, compiled from the games played by masters over decades.

4 Proposed Model

This section gives an overview of the proposed model. That includes the transformation of the openings data into comparable states, the explanation of state transitions and the algorithm implemented. The last part of this section gives a brief overview of the Python implementation for this algorithm.

4.1 Opening Lines

The standard notation for recording chess data is the portable game notation (PGN for short), used both for games and openings. This notation includes the data about the game or the opening line in square brackets, followed by the description of the moves. A thorough overview of this notation can be found in the guide for the PGN format [Sta22].

The moves are represented in string format with four possible variations:

1. A piece is moved to a new square, denoted by the abbreviation of the piece, followed by the name of the square. The abbreviation for pawns is an empty string, while B stands for bishop, R for rook, N for knight, K for king and Q for queen. In the example of Figure 8 below, moves one through three and four to five describe movements without capture.

2. In the case of capture, the move is described by the abbreviation of the piece, followed by x and the square where the piece finished the capture. In the case of pawns, instead of the abbreviation of the piece, the name of the column where the pawn captured from is used. In the Nimzo-Indian line, the seventh move by black is a capture with a pawn.

3. In the case of castling, O-O or O-O-O is used as notation, representing castling king side or queen side, respectively. In the Nimzo-Indian example, the fourth move of the black player was castling.

4. Pawn promotion is represented by the name of the square where the pawn promoted to, followed by the abbreviation of the piece it promoted to: Q, R, B or K.

Additionally, "+" is added to the end of the move if the move ended in a check. "#" is added if the last move of the game was a checkmate.

Since no information about the board is held in the PGN format apart from the end square of the last move, a notation was needed to be able to compare the board states across different opening lines. For that, I have implemented a function to transform each board state into an 8x8 matrix.

The function iterates over all of the squares on the board and denotes the square with 0 if it is empty. In the case of a white piece standing on the square, the type of the piece

```
[Site "E51"]
[White "Nimzo-Indian"]
[Black "4.e3, Ragozin Variation "]

1. d4 Nf6 2. c4 e6 3. Nc3 Bb4 4. e3 O-O 5. Nf3 d5 6. Bd3 Nc6 7. O-O dxc4
```

Figure 8. An example of a PGN-represented opening line.

is marked by an integer between 1 and 6 (a pawn, a rook, a knight, a bishop, a queen, a king, respectively). In the case of black pieces, the notation stays the same, with the difference that the piece number is preceded by 1. In figure 9, the examples of the first and the last board state of the Nimzo-Indian opening are shown.

```
[[ 2  1  0  0  0  0 11 12]
 [ 3  1  0  0  0  0 11 13]
 [ 4  1  0  0  0  0 11 14]
 [ 5  1  0  0  0  0 11 15]
 [ 6  1  0  0  0  0 11 16]
 [ 4  1  0  0  0  0 11 14]
 [ 3  1  0  0  0  0 11 13]
 [ 2  1  0  0  0  0 11 12]]
```

```
[[ 2  1  0  0  0  0 11 12]
 [ 0  1  0 14  0  0 11  0]
 [ 4  0  3 11  0 13 11 14]
 [ 5  0  4  1  0  0  0 15]
 [ 0  0  1  0  0 11  0  0]
 [ 2  1  3  0  0 13 11 12]
 [ 6  1  0  0  0  0 11 16]
 [ 0  1  0  0  0  0 11  0]]
```

Figure 9. The first board state on the left, the last board state on the right.Nimzo-Indian opening in 4.e3, Ragozin Variation.

4.2 State Transitions

In the algorithm implemented, an important step is choosing which opening line to play. To do so, the algorithm iterates over all of the possible lines to find matching board states. For the player to switch lines, there are two possible options: either the algorithm finds another line with a board state equal to the current state, or a state close to the current state by a difference of 2-4 squares. A maximum of two-square difference is needed to transition from one opening line to another using a move, a capture or a pawn promotion. Three-square difference in board states occurs if the transitioning move is *en passant* which affects the state of three squares. A four-square difference between two board states is allowed if the transitioning move is castling, either short or long.

Figure 10. First two moves of the Vienna game opening and the Bishop's Opening/Greco gambit

Figure 11. Third through fifth move of the Vienna game opening

Figure 12. Third through fifth move of the Bishop's Opening/Greco gambit

In the example shown in figures 10, 11 and 12, the Vienna game opening and the Bishop's opening start out using the same moves - e4 and e5. Therefore, after the second move, the algorithm is able arbitrarily to choose either to continue with the Vienna game opening by playing Nc3, or continue with the Bishop's opening by playing Bc4. Depending on the number of opening lines in the database, at any given moment there

can be tens or hundreds of transitions to choose from.

On occasion, two different opening lines might not have two matching board states, yet have two similar board states that allow the player to get from one opening line to another by at most one move. In chess, this is called transpositioning and it can be a powerful move to take the opponent out of their theory preparation. An example of this is transpositioning from the Vienna opening to the Bishop's Opening. Some lines of the Bishop's opening, such as the Greco gambit, arrive to a board state after the fourth move that is close to the fifth state of the Vienna game opening by one move. In this case, the player can choose to continue the Greco gambit, as shown in figure 12. However, the player can also choose to transposition to the Vienna opening by playing the move Nc3, which is not a move from the opening theory, yet takes the player to another valid state in a different opening.

4.3 Algorithm

A Markov chain is a mathematical system that simulates the transitions from one state S_t to another S_{t+1} according to some probability rules. In our work we encode each chess board configuration at time t as X_t. Our random variable X_t can sample any member of the state space $S = \{S_0, S_1 \cdots S_N\}$. The opening moves makes the board state move from S_t to S_{t+1} with a transition probability of $p_{t,t+1}$. Our board state-transitions with Markov chain are illustrated as as a graph shown on figure 13, where the board states are encoded as vertices and the transition probabilities as weighted edges of the graph.

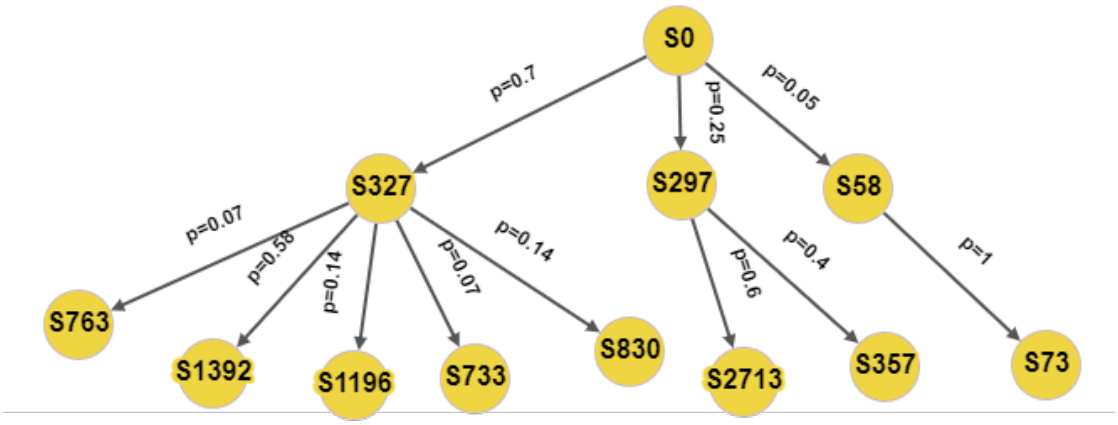

Figure 13. State transition of chess board states using Markov Chain

The graph was constructed for the first two opening moves, iterating over 1944 lines and 3617 board states. The probabilities and choices given are the result of the first 20

iterations of running the model. As shown later on, with the growth in the number of iterations, the graph will include more low-probability outliers for each state.

The transition matrix for our board state is shown below. Each row encodes the transition probabilities, with most entries including 0s as there are no legal transitions between the two states.

$$\begin{bmatrix} p_{1.1} & p_{1.2} & \cdots & p_{1.3616} & p_{1.3617} \\ p_{2.1} & \cdots & \cdots & \cdots & p_{2.3617} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{3616.1} & \cdots & \cdots & \cdots & p_{3616.3617} \\ p_{3617.1} & p_{3617.2} & \cdots & p_{3617.3616} & p_{3617.3617} \end{bmatrix}$$

We define the legal set $N_{S_t} = \{S_i\}$ of state S_t as a set of possible state transitions from S_i to S_t. The game starts from a fixed initial state S_0 and we compute the transition probability from state S_i.

Our algorithm combines the Metropolis Hastings algorithm with the weighted input parameters. The pseudo-code can be found in figure 14. The algorithm runs for a set number of iterations, at each run sampling one opening line. To do so, the algorithm finds all of the possible states it can transition to from the current state. Next, it chooses one of them uniformly at random. Then, the acceptance probability is calculated by dividing the probability for the chosen move by the probability of the last move having been made. At this point, we are adding the parameter values to the acceptance criteria by introducing a (possible) increase to the acceptance value. The increase is introduced by comparing the parameter values from the input to the values attributed to the potential move. Where those two match, a weight corresponding to that particular parameter is added to the acceptance value. Lastly, a random value is generated and compared to the acceptance value. If A is bigger than the random value, the proposed new state is set as the new working state. If not, the algorithm queries a new value from the possible moves until it is accepted as the next state. Once the algorithm reaches the end of an opening line, this line is added to the list of samples and the process is repeated until the number of iterations needed is reached.

```
# Number of opening lines sampled
for _ in range(number_of_iterations):
    S[0] = first_board_state
    i = 1
    # Running through the database until we have reached the last
       board state
    # of the current opening line being sampled
    while True:
        current_S = S[i-1]
        for opening in openings_database:
            for board_state in opening:
                # If the current state is in the openings database
                # & it is not the last state in line, we select it
                if current_S == board_state and board_state not last:
                    possible_moves.add(board_state)
        if possible_moves:
            proposed_S = possible_moves[random_index]
            A = probability_of_this_move /
                probability_of_the_last_move
            # Acceptance/rejection sampling
            # Add value where the input parameters are 1
            # & the corresponding parameter in the opening line is 1
            added_value = [weights * input_parameters where True]
            A = A + sum(added_value)
            if random_value < A:
                S[i] = proposed_S
                i += 1
            else:
                S[i] = current_S
        else:
            list_of_samples.add(S)
```

Figure 14. Metropolis Hastings with parametrization

4.4 Implementation

In the section below, a brief overview of the different classes contained in the project is given. The explanation serves as supporting material to the code, found on GitHub [Saa21].

4.4.1 Chess

The part of the code for my implementation of chess can be found in the sub-folders of BuildingBlocks, Pictures and main.py.

BuildingBlocks contains three sub-folders and a number of Python files. The sub-folder Classes contains three classes - Game, Settings and Square. The class Game

consists of variables needed to store game data: board states, moves, the progress, but also temporary data - whose turn it is, if there is a pawn promotion in progress, which squares are under fire by both pieces, and others. The class Settings includes all of the necessary settings for the visual controls. That is, it includes the variables for setting the colors for showing the possible moves, captures, castling, but also the colors of the tiles, along with some other variables. The class Square includes all of the variables connected to the specific square: the co-ordinates and related data, the tile color, if there is a piece on the square and data about related temporary data. The class also includes eight visualization-related functions used for displaying the game and changing the parameters for viewing it.

Another sub-folder of BuildingBlocks is Pieces. This folder contains 6 classes - one for each piece type. The files are: Bishop, King, Knight, Pawn, Queen, and Rook. Each class contains variables holding information on the piece - the name, color, co-ordinates, abbreviation for the piece, and in the case of the king and rooks, if they have been moved. The classes additionally contain a toString method, functions for finding possible moves and captures. In the case of pawns, the functions also include the check for a possible *en passant* or promotion. In the case of the king, the functions also include a check for castling.

The file CheckOrMate includes all of the necessary functions for checking if a piece is under attack, if the king is in check, stalemate or checkmated. The file Initialize includes the functions necessary for initializing all of the Square elements on the board, and the Piece elements of those Squares. Move.py includes functions for moving, capturing, castling, promoting or capturing via an *en passant*. MoveLogic.py includes functions for handling two types of events: a mouse click and a mouse drag. In the case of a mouse click, it is necessary to handle all of the possible options - whether the player's own piece or the opponent's piece was clicked, if a piece was moved or captured by this activity, if the player clicked on the piece to visualize the possible movements, etc. The function also takes care of updating the board state, the game data on moves, board states, and temporary data. The function for dragging a piece works under similar principles, with different handling needed for determining the starting square of the movement, and no possibility for visualizing the possible movements. The file Screen contains functions necessary for updating the game visualized. ShowPossibleMoves includes functions used for visualizing the possible moves, captures, promotions and castling for any one square that has been clicked on by the player.

The sub-folder Pictures contains all of the necessary pictures for The the chess pieces on the board.

The class main.py contains the main controls for the chess and the openings learner, implemented with the help of a Python library Pyglet. The class includes all of the necessary initializations, as well as event handling for mouse presses, drawings and game window related events.

4.4.2 Openings Learner

The section of the code handling the transformation of the data on openings, the Metropolis-Hastings algorithm and the heuristics consists of four Python files and a folder containing the data on openings.

The file called ReadLines contains functions to transform the data from a pgn file into a text file with data that can be used by the algorithm later on. StringToMatrix implements functions to transform the data from the now-cleaned files containing opening lines into matrix form for each state. Helpers.py implements functions to retrieve the first board state (with no moves made yet), as well as all of the board states from the data enumerated. The class MetropolisHastings includes all of the functions necessary to run the algorithm with different parameters, number of iterations, and plot the results.

5 Results and Discussion

In this section, we are introducing the results produced by the implementation of our proposed algorithm. We have shown, the effect of variance of number of iterations, the weights, player parameter values and the openings data affects on the probability distribution of the state transitions.

5.1 Variance in Iterations

To test out the effect of running the Metropolis Hastings algorithm for a different number of iterations, we have chosen a fixed value for the parameters and the weights. To see the effect of both the algorithm and the heuristics over a changing number of iterations, we are using $[1, 0, 1, 0, 1]$ as input values for the parameters and $[0.2, 0.2, 0.05, 0.05, 0.05]$ as the weights. The input value of 1 means the parameter is used and 0 otherwise. So our proposed algorithm will prefer choosing lines with gambits with a 0.2 higher acceptance parameter, as well as lines starting with $c4$ or $d4$ by a preference of 0.05 points. The algorithm will not favor $e4$ lines, nor any of the less standard openings, such as lines starting with $g4$, $d3$, or $Nf3$. The sub-sample of the opening lines used for this comparison included all of the lines that had at least 7 moves played by each player in the theory.

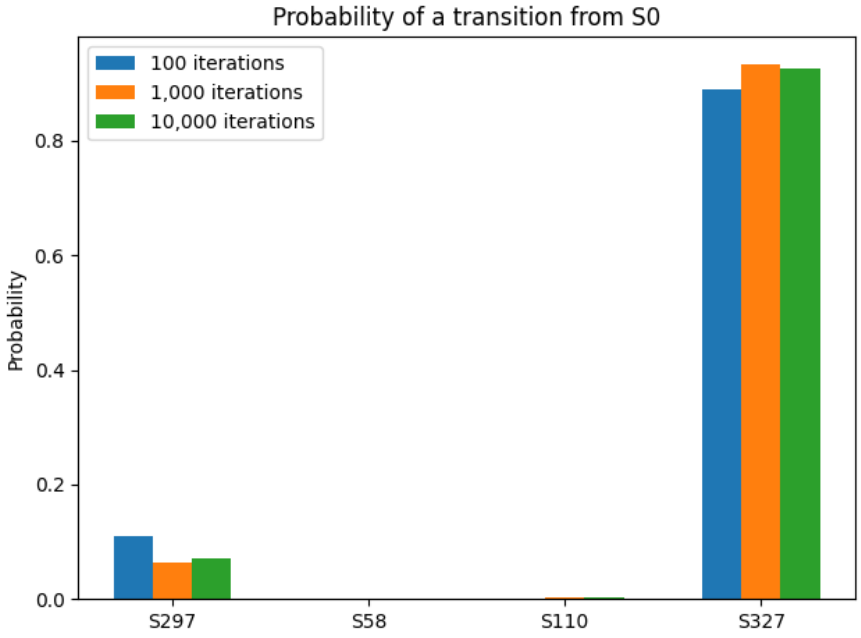

Figure 15. Probability distribution for the first move

Figure 15 displays the probability distribution for the first move. $S0$, the board state where no pieces have been moved yet, is highly likely to be followed by the state 327 or 297, regardless whether the number of iterations is low or high. Notably, in the case of higher number of iterations - already visible at 1000 iterations - the number of less common states visited rises. Whereas state 110 has been the second board state with a probability of 0.002 and 0.0026 for 1000 iterations and $10,000$ iterations, respectively, the state 58 was only visited once, and, surprisingly, under a lower number of iterations - 1000 iterations.

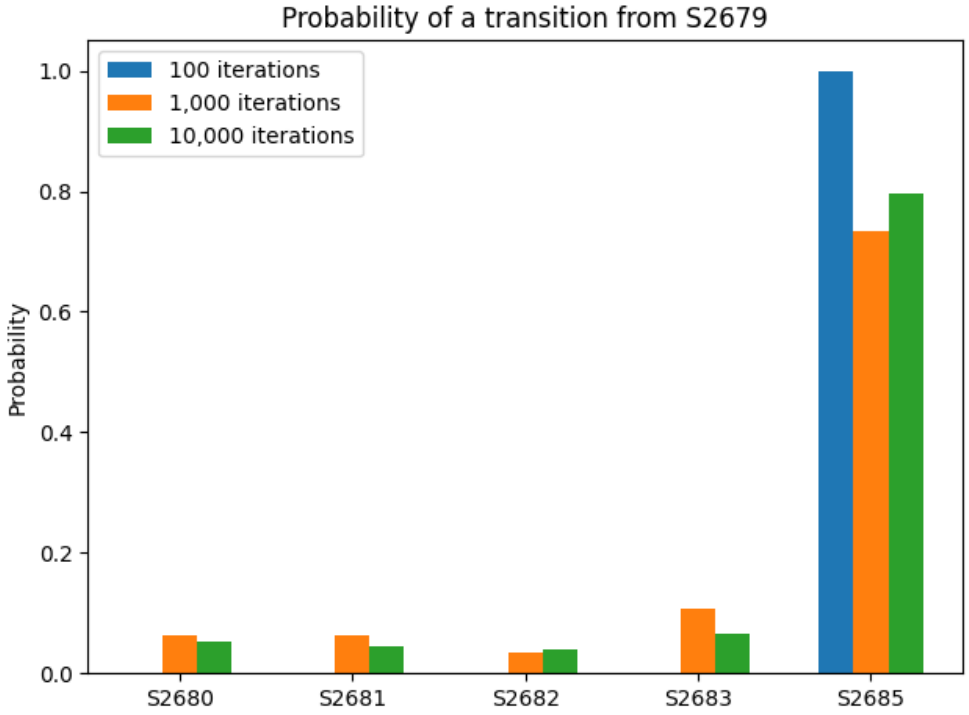

Figure 16. Probability distribution from state 2679

Similar hypothesis hold for most of the state transitions when comparing the difference in probabilities between different iteration values. There are significant differences in the number of state transitions for each turn between 100 iterations and $1000+$ iterations. Yet, the difference in the number of possible transitions at each turn do not change significantly, regardless of whether the number of iterations was 1000 or $100,000$. Though, the number of iterations affected the probability of each transition being made, with the probabilities being relatively close in the case on 1000 and $10,000$ iterations, as shown in figure 16.

5.2 Variance in Weights

To test out the effect of different weights, we have fixed the number of iterations to 2000, the sub-sample to all of the lines that have at least 8 moves defined for both players, and the input values to $[1, 1, 1, 1, 1]$. That is, at each iteration and at each combination of weights, all of the input parameters are being used. In the comparison graphs, we are looking at five cases:

1. Using equal weights on each parameter, set to $[0.1, 0.1, 0.1, 0.1, 0.1]$, to get a data distribution for non-tilted weights.
2. Using weights with an emphasis on $c4$ openings first, and gambits second. The weights were set to $[0.2, 0.05, 0.3, 0.05, 0.05]$.
3. Using weights with a strong emphasis on $e4$ openings, with weights set to $[0.05, 0.05, 0.05, 0.05, 0.6]$.
4. Using weights with a very strong emphasis on $d4$ openings and a strong emphasis on gambits, with weights set to $[0.4, 0.05, 0.05, 0.8, 0.05]$.
5. The same use case the fourth case, but with lowered weights: $[0.2, 0.05, 0.05, 0.4, 0.05]$.

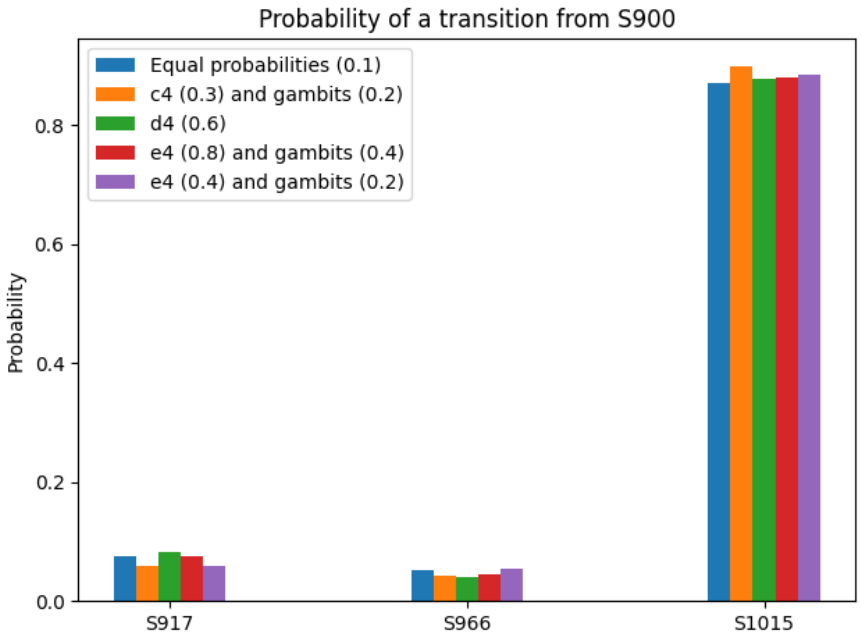

Figure 17. Probability distribution from state 900 with different weights

In general, the effects of changing the weights could be divided into two groups: states where increased or decreased weights changed the probability distribution significantly, and states where the uneven weights played virtually no role in the eventual distribution. An example of the second case can be seen in Figure 17. A small difference can be seen in the distribution of the $2nd$ case (prioritizing $c4$ lines and gambits), but the effect of it is not significant. These types of probability distributions are for the most part the result of the openings database containing a lot of variations of one opening with the first n moves matching (for some n).

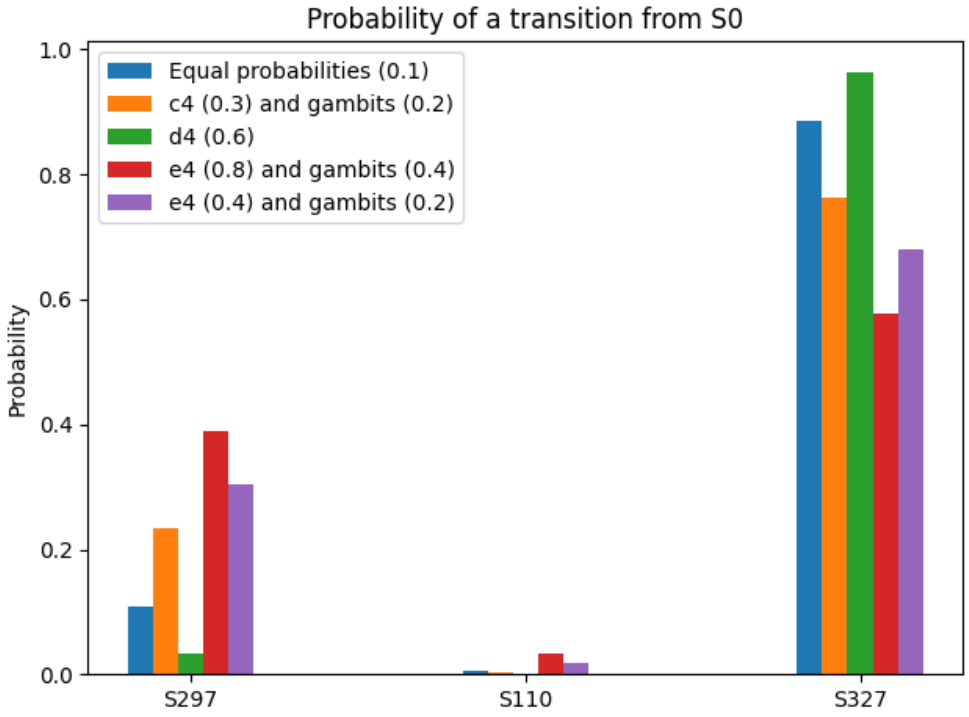

Figure 18. Probability distribution from state 0 with different weights

A more interesting situation is depicted on Figure 18, showing the probability distribution for the first move of the game. The added weights have a direct impact on the first move. Notably, the probability distribution is not dependent solely on the increase in the weight, but also highly dependent on the proportion of the number of specific states in the openings database. Due to this, the probability distribution for $d4$ openings prioritized, even though with a higher weight, is closer to the equal probability distribution than the one for $c4$ and gambits prioritized with a lower value. In the case of a state transition from $S0$, the probabilities for the same properties prioritized ($e4$ lines and gambits), fall under a similar distribution, even though the difference in the weight is two-fold.

5.3 Variance in Input Parameters

The most exciting results were revealed when comparing the algorithm's work parameter-by-parameter. To do that, we fixed all of the other variables: the number of iterations, the subset of the opening lines, and the weights, to purely compare the probability distribution based on the input parameters. The algorithm was ran for 2100 times - around 3 times as big as was the size of the subset used, 730 lines. The lines were chosen by including all of the openings with at least 5 moves by both players. The weights were chosen high enough for the differences to be notable, set to be 0.4 for each parameter. The iterations compared in the graphs below are the following:

1. Prioritizing $e4$ openings, with input parameters set as $(0, 0, 0, 0, 1)$.
2. Prioritizing $d4$ openings, with input parameters set as $(0, 0, 0, 1, 0)$.
3. Prioritizing $c4$ openings, with input parameters set as $(0, 0, 1, 0, 0)$.
4. Prioritizing non-traditional openings, with input parameters set as $(0, 1, 0, 0, 0)$.
5. Prioritizing gambits, with input parameters set as $(1, 0, 0, 0, 0)$.

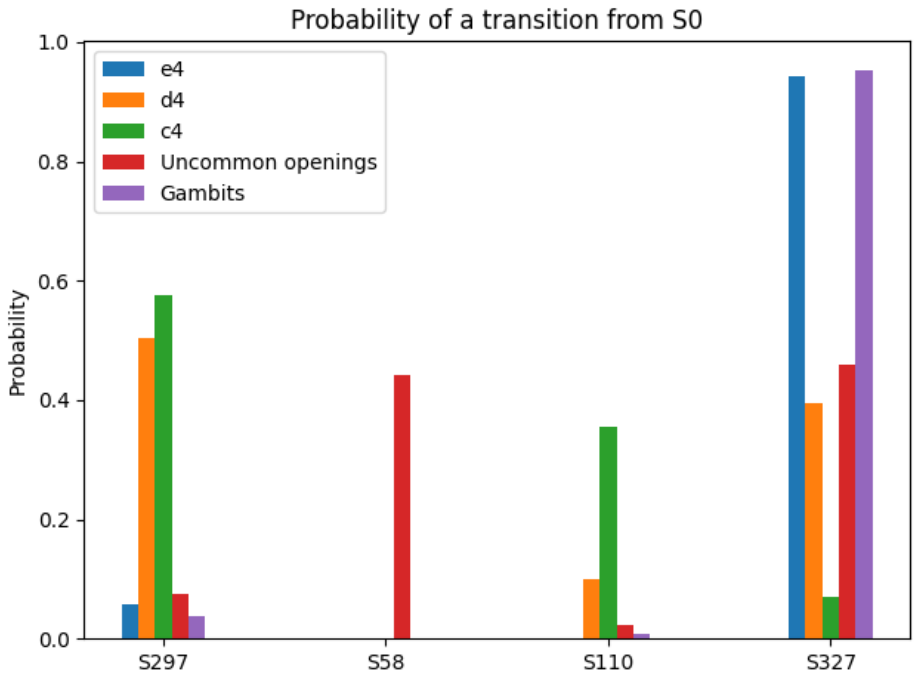

Figure 19. Probability distribution from state 0 with different parameters

In figure 19, we can see the probability distribution for the first move under the five different parameters used. Notably, this graph is very different from the probability distribution shown in figure 15. In that case, the distribution followed largely the probabilities derived from the Metropolis Hastings algorithm, with some priority on gambits (mostly found in $e4$ lines) and $e4$ openings. That configuration did not visit the states 58 and 110 under a lower number of iterations, and only gave them a very low probability under higher number of iterations. As seen in figure 19, with the usage of parametrization and large enough weights, those probabilities can be altered significantly. By using the parameter to prioritize $c4$ openings or uncommon openings, the user would be able to push the probability of starting with the move $c4$ (aka moving to state 58) or with the move $Nf3$ (aka moving to state 110) from near 0% to around 40%. Additionally, those odds can be heightened or lowered based on the weight set for the parameter(s).

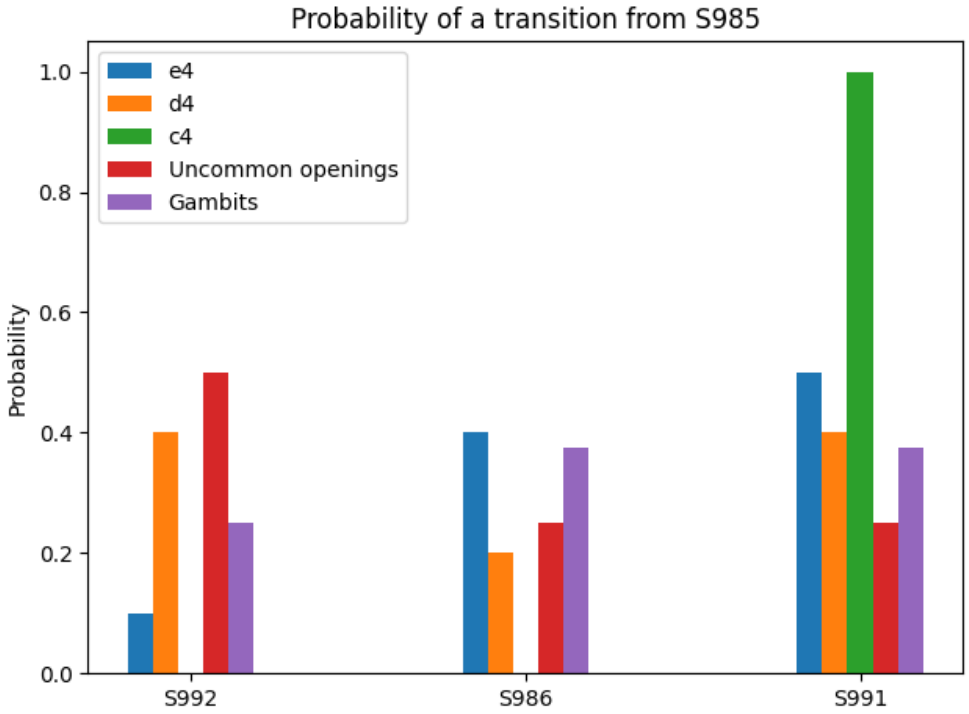

Figure 20. Probability distribution from state 985 with different parameters

Another notable conclusion from the results of testing the different parameters was the difference in the probability distributions between $c4$ openings prioritized and the other four options. As seen in figure 20, the probability distribution only includes one option for $c4$ prioritized, whereas all 3 options can be selected with some probability for the other four. This can be explained by the fact that most of the openings start with $d4$

or $e4$, giving those lines a lot more options for transpositions. When it comes to gambits, they are not associated with a specific opening move, although they also often times follow lines with the two popular starting moves. As to the prioritization of uncommon openings, the explanation can be twofold: either the openings are so different that the later stages can only be reached if the opening moves were decided by the randomness from Metropolis Hastings, not the parameter, or the lines allowed for transpositions to more common lines (such as some $Nf3$ lines transpositioning to $c4$ openings).

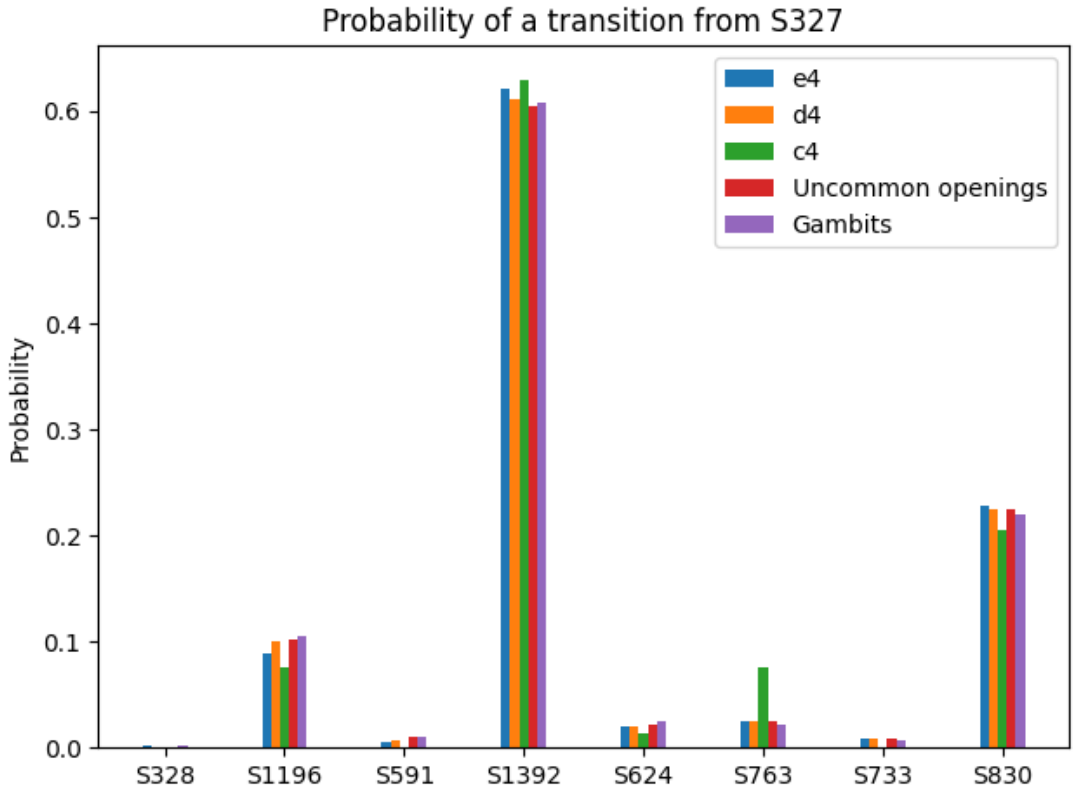

Figure 21. Probability distribution from state 327 with different parameters

In the case of testing different weights, the variance not changing the probability distribution much was a common occurrence. In the case of testing different parameters, the probability distribution only stayed the same in a small handful of cases. One of those cases is visualized in figure 21, showing very limited variance in the probability distributions. Those cases were most likely strongly affected by the input data from the opening database. As State 327 corresponds to the move $e4$ being played as the first move, there are no possibilities of transpositioning to a $d4$, $c4$, or a rarer opening straight away, so the distribution mainly follows the probabilities based on the number of possible

state transitions in the openings database. As mentioned, the number of states where the probability distribution did not change significantly stayed low when comparing the different parameter values.

5.4 Optimized Selection of Parameters

The analysis presented in this section showed that there are four main parameters influencing the probability distribution: the (sub)sample of the openings chosen, the number of iterations run, the weights and the parameters used.

In this thesis, we did not focus on the methodology of the selection of sub-samples. Nevertheless, the algorithm allows for the use of the whole database, as well as a subsample of opening lines based on their length (number of moves), the name of the openings family for white or black, and the opening type. This shows the versatility of the program as the algorithm could be used to focus on learning one specific subgroup of openings, but also teach the lines based on a large selection of openings. We though believe that grid-search or Bayesian optimization algorithm can be used for optimum parameter selction.

Section 4.1 presented the effects of a varied number of iterations. Based on the probability distributions generated by the different number of iterations, the general guidelines for finding the ideal i (number of iterations) value are:

1. Choosing the number of iterations that is equal or close to the number of lines in the sample of openings used will produce a probability distribution that will focus heavily on the most common states. This i value will leave out most of the outlier states.

2. An i value that is three to five times larger than the database used will give a more accurate probability distribution, yet also include most of the less common states.

3. A number of iterations that is ten or more times bigger than the size of the openings sample used will produce the most accurate probability distribution, including most or all of the outlier states. The trade-off in choosing a high number of iterations is the slower run-time, especially when using a large database.

Section 4.2 demonstrated the effects of using different weights for different parameters. Based on tests, our guidelines for choosing the weights to use are the following:

1. Weights from 0 to 0.2 will produce a mild skewing effect on the eventual data.

2. Weight values between 0.2 and 0.5 will produce a moderate effect on the final distribution.

3. Any weight higher than 0.5 will cause the probability distribution to be heavily in favor of the parameter used.

Notably, the effect of the weight is not affected by the underlying sub-sample of the openings used.

In section 4.3, we followed the effects of the selection of parameters. As demonstrated in section 4.2, the eventual effect of each parameter is dependent on the weight used with it. However, we noted the tendency of the third parameter - corresponding to the preference of $c4$ openings - to skew the final probability distribution significantly more than any other parameter.

6 Conclusion and Future Work

In this thesis, we have presented a methodology for effective comparison of states in the opening lines. By turning the data represented in string format into matrices, we have transformed the data on opening lines into comparable information about board states. Using these matrices a database for openings, we have implemented a mixture of the Metropolis Hastings algorithm with parameterized and weighted values, decided by the user, simulating the dynamics of the game.

While testing out the algorithm, we found a number of notable results. First and foremost, the parametrization of the Metropolis-Hastings algorithm allowed us to modify game dynamics and gives various probability distribution of state transitions. Depending on the weights chosen, the user can skew the probability distribution of the board states on a lower or higher manner, as preferred. Additionally, the number of iterations needed for outlier distributions of board states varies linearly (3 times) with the sample size of opening lines and thus run-time needed for determining the distribution is reasonably low.

In the future, we are planning on expanding the scope of the search for transition states using reinforcement machine learning. Often times not included in the opening theory, variations of opening lines with one piece misplaced or missing can be considered valid for following the opening moves of the similar board state. Additionally, the players can sometimes transposition to a board state where they are effectively playing with the colors reversed (i.e the white player playing the black's moves and the black playing the white's.). Iterating over a larger variation of theory can give the learner unique knowledge and better ideas for beating the opponent. However, such expansion will require a lot more sophisticated search with a lot more optimal implementation to not extend the run-time too much.

References

[BT09] Bernd Blasius and Ralf Tönjes. Zipf's law in the popularity distribution of chess openings. *Physical Review Letters*, 103(21), nov 2009.

[CF20] Ian Cero and John Michael Falligant. Application of the generalized matching law to chess openings: A gambit analysis. *Journal of applied behavior analysis*, 53(2):835–845, Apr 2020. 31329274[pmid].

[CG11] Philippe Chassy and Fernand Gobet. Measuring chess experts' single-use sequence knowledge: an archival study of departure from 'theoretical' openings. *PloS one*, 6(11):e26692–e26692, 2011. 22110590[pmid].

[DF08] N. De Firmian. *Modern Chess Openings: MCO-15*. Chess Series. Random House Puzzles & Games, 2008.

[Fid18] *FIDE Laws of Chess*. 2018.

[Las13] E. Lasker. *Lasker's Manual of Chess*. Dover Chess. Dover Publications, 2013.

[RK16] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. Wiley Publishing, 3rd edition, 2016.

[Saa21] Kristiine Saarmann, 2021.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[SM09] R.W. Shonkwiler and F. Mendivil. *Explorations in Monte Carlo Methods*. Undergraduate Texts in Mathematics. Springer New York, 2009.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

[Sta22] Standard: Portable game notation specification and implementation guide, 2022.

[Tea21] The Chess Informant Team. *Encyclopaedia of Chess Openings*. Chess Informant, 2021.

[Wik21] Wikipedia contributors. Encyclopaedia of chess openings — Wikipedia, the free encyclopedia, 2021. [Online; accessed 15-May-2022].

II. Licence

Non-exclusive license to reproduce thesis and make thesis public

I, **Kristiine Saarmann**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive license) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Exploration vs Exploitation of Chess Openings Learner with Metropolis-Hastings Algorithm,
   *(*title of thesis)

   supervised by Kallol Roy.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kristiine Saarmann
13/04/2022