Time Management in a Chess Game through Machine Learning

Guga Burduli and Jie Wu
Department of Computer and Information Sciences
Temple University, Philadelphia, PA, USA

***Abstract*—Chess includes two significant factors: playing good moves and managing your time optimally. Time, especially in blitz games, is just as essential to the game as making good moves. Nowadays, several incredible engines are already developed, more than enough to defeat all the best human chess players. For studying how to make good moves, these engines are crucially useful. Professional chess players are using them in addition to coaches to prepare for the matches or to examine the mistakes in their played games. However, managing time still is a huge challenge. There are no basic rules for managing time. A lot of factors influence the decision about how much time should be spent in a particular position. For computers, it is easier because they calculate much faster and they have all the theoretical knowledge. However, even grandmaster chess human players are struggling with time trouble. In this article, we describe how the data was collected from an online chess platform and show methods of how time can be managed based on different features. In this regard, we will use two different models: using a customized neural network and using a proposed segmented least square approximation method. In both of the models, we will use our collected data.**

***Index Terms*—time management, neural network, segmented least square approximation, chess, collecting the data**

I. Introduction

Time management in chess is one of the most critical problems, especially in the case of sudden death time control. Sudden death time control is the simplest form of time control; each player has a fixed amount of time for the whole game. If the remaining time equals zero, the player loses the game immediately. The challenging part is determining how much time should be spent in a particular position. Spending more time gives us more accurate results; however, we have limited time until the end of the game, and we do not know how much more moves will be till the end of the game. Even grandmaster (GM) players struggle to manage time effectively and often reach the time-trouble.

A lot of factors influence the decision about how much time should be spent on a particular move, such as remaining time, the complexity of the position, the expected number of moves until the end of the game, the opponent's remaining time, player and their opponent's strength, even remaining physical power of the player [1] [2]. In addition, sometimes the assessment of the position is crucial as well; if a player is winning, in most cases, it is easier to make the decision faster than when the player is losing. There are no general rules for time management; however, if a player has only one candidate move in the position, thinking is wasting time, so they are making a move immediately. In general, the process is the following: players choose several candidate moves from all legal moves in the position and calculate chosen ones in depth. The depth depends on the allocated time at this move, and the move's accuracy depends on the depth.

Fig. 1: Chess board with the clock

We already have a lot of knowledge of how to use the help of the computer to play good moves. Direct engine with some theoretical knowledge such as "Stockfish" [3], or based on self-learning knowledge with reinforcement learning such as "Alphazero" [4]. Both engines are already strong enough to defeat any human. However, can we have the same success in the case of time management?! There are different strategies for time allocation, and a lot of research is made in this regard. The paper [5] describes statistical methods for time management. They collected the data, and according to statistical results, they made the linear model of time allocation, which depended on the expected amount of moves till the end of the game. There are time management systems for tree-search based systems such as Monte Carlo Tree Search algorithms [6] [7] [8]. In addition, there are some stochastic optimal time management strategies mentioned in the paper [9]. In the paper, [7] time allocation is divided into two parts: for each search before it is started and each move search while it is running. These methods are optimized for computer use. However, humans still struggle to manage time during the game; blunders made in time trouble are the most common during the human match.

The importance of time management clearly showed during the match between "Stockfish" and "Alphazero" [10]. On

December 6, 2017, another crucial milestone was reached, when the best engine was defeated by the AI-based engine [11]. “Alphazero” won 28 and tied 72 - from 100 played games versus “Stockfish 8”. The second match between them was conducted after one year, in 2018. The results of the match showed that with the same amount of time, “Alphazero” had a huge advantage. Even with the $1/10$ time “Alphazero” still was dominant. “Stockfish 8” begun outscore “Alphazero” when the odds reached 30-to-1 [12].

In this paper, we address four challenges: (1) Collect the data on the human chess games by spending time from the online website ’chess.com’. (2) Segmented Least Square Approximation for data points with a fixed amount of lines. (3) Neural Networks with different features and architecture for predicting the thinking time for each move. (4) Simulations with different time management strategies and different types of engines. The following summarizes our contributions:

1) We collected matches from the online chess platform “chess.com”, parsed each match, and got important features from each position.
2) We changed the Segmented Least Square Approximation algorithm, and instead of cost value for each line, we optimized an algorithm for the fixed number of the lines using dynamic programming.
3) We propose how to predict the thinking time with a Neural Network, what should be the features of the position, and what architecture of the Neural Network gives better results.
4) We make the simulations of the games with different time management systems for the segmented least square approximation model and the neural network’s model. Simulations were made with different engines for different data, and results were compared with each other.

This paper is organized as follows. Section II describes the background and important information about chess and time controls. Section III introduces how the data was collected and preprocessed. Sections IV and V introduces two different algorithms for time management, with the neural network and with segmented least square approximation appropriately. Section VI then combines the results for all methods mentioned above. In section VII, we are shortly describing relevant works. Section VIII describes the future work of the paper and what can be improved more. In the last Section IX we are summing up the paper.

II. Background

This section will discuss chess history: how computer chess was improving and what features are most important for chess.

A. Chess History with Computers

Chess is a game that requires a lot of creativity and sophisticated decisions. It was frequently compared to activities like poetry, writing, and painting as examples of tasks that can only be done by humans [13]. While writing poetry has remained very difficult for computers to this day, we have a lot of improvement in building chess-playing computers [3]

Fig. 2: Kasparov vs. Deep Blue 1996.

[4]. The first-time computer beat a chess master was in 1978, Chess 4.7, won one game out of 6 versus International Master David Levy [14], and ever since, it has been improving a lot. In 1997, IBM’s Deep Blue defeated the reigning World Chess Champion, Garry Kasparov, under standard tournament rules, for the first time in chess history [15] [16] [17]. Nowadays, computer hardware (”Stockfish”) and AI research advanced state-of-art chess-playing computers (”Alphazero”) to the point where even the best humans today have no realistic chance of defeating modern chess engines.

B. Important Features for Chess

Before going into details about time management in chess, we should determine some of the chess specifications. As we mentioned in the introduction, a lot of factors determine the thinking period in the position. For humans, the most important factor is the remaining time of the game: If you have seconds on your clock, the position does not matter anymore; everyone depends on their intuition and plays as fast as possible. However, if a position is already winning (e.g., if you have an extra queen in the endgame), even seconds are enough to finish the game with a checkmate. In this regard, the second most important feature is the complexity of the position: is it winning or losing already, or how much material is remaining, how many possible moves exist, and how many moves are expected till the end of the game. This part is the most challenging one to determine because sometimes even the same material means absolutely different complexity of the position. The third specification that is worth considering is the players’ strength; different levels of the players manage time differently. The best players spent a little time in the openings and some theoretical endgames because they already knew the positions clearly. Most of the time by the international masters and grandmasters is spent during the middlegame when positions are complex and mostly new for them. So, they need to make a game plan and consider a lot of different moves in depth. Amateur players do not have so much knowledge in opening or theoretical endgames, so they manage time most likely the same from the beginning of the game.

Another considerable factor in managing time is color. The white pieces have one extra tempo; they are starting the game, so they have a little advantage at the start of the game. Players with black pieces are trying to equalize the position first and then try to win the game.

C. Computer Way of Playing Chess

Humans consider all those factors mentioned above to determine the time spent on a particular move. However, it is interesting to note that the way computers play chess is very different from how humans play. In the case of Deep Blue (1997), it relied on the brute force to explore as many moves as possible, even some of them would be automatically thrown away by any skilled human [18]. In a sense, the way humans play chess is much more computationally efficient - using Garry Kasparov vs. Deep Blue as an example, Kasparov could not have been searching more than 3-5 positions per second, while Deep Blue, a supercomputer with 480 custom' chess processors', searched about 200 million positions per second to play at approximately equal strength (Deep Blue won the 6-game match with two wins, three draws, and one loss) [13]. The main power during playing chess is the effective and precise assessment of the position. After AI became strong, computers even increased the efficiency of calculation. Using multiple deep artificial neural networks trained in a temporal-difference reinforcement learning framework, there are already statically evaluating positions - estimating how good a position is without looking further [13] :

- Deciding which branches are most "interesting" in any given position and should be searched further, as well as which branches to discard.
- Ordering moves - determining which moves to search before others, which significantly affects the efficiency of searches.

Artificial neural networks are used as a substitute for "intuition"; however, they are more precise and reliable [13]. Humans with more experience have better intuition and hence have a better sense of assessment of the position. Assessment of the position is the key to fast calculation. If you know which positions are better or worse, you filter the possible moves faster and get more effective results. So, when the computer has artificial intelligence for the "intuition", which is trained on a self-played game, it gives the power of the best assessment and optimal search.

III. Data Collection and Preparation

The section describes how data was collected and preprocessed for the models. We will see how different kind of features were extracted and saved from the game.

A. Source

There is a huge database where a lot of chess games are saved, known as "Chessbase" [19]. Unfortunately, they are holding only moves, not the time spent on every move. On the other hand, there are a massive amount of online games where times are saved as well. There are a lot of online platforms where you can play chess, such as "lichess.org", "chess24.com" or "chess.com" [20] [21] [22]. The most popular online platform to play chess is "chess.com". Figure 3 shows the visualization of the website. About 427,580 people use chess.com to play chess online, including grandmasters such as Fabiano Caruana, Wesley So, Magnus Carlsen, or Hikaru Nakamura. Everyone can be registered on the website and play different types of chess games. Most popular time controls are one-minute games, so called "bullets" and three or five minutes games known as "blitz". Data used in this article comes from that platform. "Chess.com 1.7.6" is a python wrapper for Chess.com API, which provides public data from the chess.com website [23]. The data comes in JSON format, and we have a string of the different specifications of the game, including player's ratings, game score, moves with spent time, etc. Using that API, we can get all the games for the given username of the player. However, until these data are useful, it needs preprocessing.

Algorithm 1 Data Collection

Require: Receives username of the player
1: Get All Games for the given username with API
2: Create empty list of lists: L
3: **for** each game **do**
4: Play Game as Python-Chess game
5: Create list for one move data: K
6: **for** each move **do**
7: Calculate: total material, material difference, expected moves till the end of the game, legal move amount
8: Save: spent time, remaining time
9: Save static features: Rating of the players, Result of the game, Reason of the result, Color, Move.
10: Append everything in K
11: add K into L
12: Create dataframe with L
13: **return** dataframe

B. Dynamic Features

During the preprocessing phase for every player's username, we get all the games played by that particular user. The parsing algorithm can fetch all-time control games. However, in our case, we choose sudden death time control for one, three, or five minutes for each player during the whole match. Then, each game given in string format was replayed as a python-chess type game [24]. And for each played move in the game, we calculated features of the position:

- Was it white's move or black's
- Which move it was
- What was the remaining time after the move
- How many legal moves existed in the position
- What was the total material
- What was the difference of material
- What was expected moves till the end of the game
- How much time was spent on that move

TABLE I: Dataframe for the first 3 moves of the game

WR	BR	TC	NM	Col	Move	TR	NLM	Mat	MD	EM	R	LR	ST	Sum
1347	1142	180	1	1	e4	180.0	20	78	0	67.5	1	checkmate	0.0	0.0
1347	1142	180	1	0	e6	180.0	20	78	0	67.5	1	checkmate	0.0	0.0
1347	1142	180	2	1	d4	178.5	30	78	0	67.5	1	checkmate	1.5	1.5
1347	1142	180	2	0	c6	179.3	30	78	0	67.5	1	checkmate	0.7	0.7
1347	1142	180	3	1	c4	176.5	38	78	0	67.5	1	checkmate	2.0	3.5
1347	1142	180	3	0	Qb6	178.0	31	78	0	67.5	1	checkmate	1.3	2.0

Pseudocode 1 shows the whole process. Let's define what the features mentioned above are. Legal moves mean how many different actions the player has at the current configuration. In some positions, there is only one legal move; in that case, players are not thinking at all. Sometimes there are several legal moves, but it is obvious that only one is better than the others, and we do not need a lot of time to make that move as well. However, there are key positions where players have two or more good candidate moves; in that case, the calculation is time demanding. Hence, this feature should be important because, in general, chess players calculate faster when there are few choices of legal moves.

One of the critical things in chess is material, which means how many pieces you have. The "cheapest" piece in chess is Pawn, which is 1 point. However, each player has 8 Pawns in the beginning. After this, there are Knights and Bishops, which are equal to 3 points each; every player has two Knights and two Bishops before the start of the game. Next once are Rooks with 5 points each, and there are two Rooks for each player. Last but not least is Queen, the most powerful piece on the chess board. Queen is equal to 9 Points and is one for each player. And there is the King, which is priceless. Losing King (which is checkmate) is the end of the game, no matter how much other material you have remained. To sum up, every player has 39 points without King at the beginning of the match. We have two types of material features; one shows the difference between the white and black pieces materials, second shows the total remaining material. For example, on the Figure 3 white have one rook, one queen and six pawns, which means that total material is $Mat = 1*5+1*9+6*1 = 20$. Black have two rooks instead of one, but two pawns instead of six, so total material is $Mat = 2*5+1*9+2*1 = 21$. Hence, material difference is $MD = 20-21 = -1$. In general, getting a material advantage gives a higher chance of winning the most cases during playing chess. However, it is important to mention that sometimes people are sacrificing material for some compensation or attack.

Another interesting feature is expected moves till the end of the game, which is calculated based on the material with a linear function, and we are using the formula mentioned in the article [5]. Managing time can be more effective when you know approximately how much more moves should be played in a match.

C. Fixed Features

In addition to the position's dynamic features, there are fixed (same for every move, but different with the game)

Fig. 3: Visualisation of online platform "chess.com"

specifications of the match:

- Rating of the first player
- Rating of the second player
- Time Control
- Result (Win, Lose, Draw)
- Reason for the result (Can be agreement, resigning, or losing by time)

Ratings of the players show their strength; Amateur level rating is typically before 1500 rating, then comes beginner level from 1500-2000, from 2000-2400 it is FIDE master level, then comes international masters between 2400-2700, 2700+ ratings are already grandmasters. Different levels of the players manage time differently. For instance, grandmasters do not lose time in openings or theoretical endgames, while the beginners and even most likely FIDE master chess players are burning a lot of time in openings. The time control means how much time have each player until the end of the game. We have another fixed feature which is the reason for the result, why the game was a draw, or why someone lost the game. The reasons for the draw result can be:

- Agreement of the players.
- 50 move without taking a piece or pawn move.
- 3 times repetition of the position.
- A player is out of time, however opponent does not have enough material for checkmate.

The reasons for the loosing can be:

- Checkmate.
- Losing by time (case when the opponent has enough pieces for checkmate).
- Resignation, the player can stop the clock anytime and resign.

After all, features are calculated, we create the dataframe with all the data I. The only column not mentioned above is the "Sum", which appropriately describes the cumulative sum of the spending times for black and white pieces. This feature is used in Section V, about Segmented Least Square Approximation. The data is saved move by move for every game. We create different dataframes for the different levels of the players.

D. Other Usage of the Data

Chess games data is used in many different fields as well. In the article, [25] the collected data was used to get statistical analyses on a range of topics about humans, including skill development over the lifetime, birth cohort effects, effects of activity and inactivity on the skill, and gender differences. Hence, our created dataframes can be used in different variations of problems related to chess.

IV. Neural Network

For time management, we developed two different methods. The first one is the time prediction with a neural network. In the previous chapter, we described how data was collected. Based on the method, we collected data for different players playing one, three, or five minutes games without increment (sudden death time control). For training the Neural Network, various features were used from the dataframe shown in I. This chapter will describe the architecture of the neural network, the basic model training, and the feature selection process.

A. Architecture

Since the prediction should be time spent per move and it is a continuous function, we used Neural Network for the regression problem [26]. Figure 4 describes the basic model of Sequential Neural Network, with one input layer, one output layer, and five dense hidden layers. The batch size for input data is 1024; hence "None" value changes according to input data and batch size. For the basic model, we had eight features; because of this, the second dimension of the input layer is eight. The first two hidden layers have 512 neurons. Then we have two layers with 256 neurons each, one with 128 neurons and the output layer with one neuron. Each layer except the output has "ReLU" as an activation function. One major benefit of the "ReLU" activation function is the reduced likelihood of the gradient vanishing. The gradient of "sigmoids" becomes increasingly small as the absolute value of x increases. The constant gradient of "ReLUs" results in faster learning. Since our prediction is a continuous variable, the last layer has a linear activation function [27].

For testing purposes, we changed the amount of the layer as well as the amount of the neurons in each layer. Regarding the number of neurons in each layer, we changed the first hidden layer size, and hence, all the following hidden layer sizes were changed. If the first two hidden layers' size was n, we get the third and fourth layers' size as $n/2$ and the fifth $n/4$.

The results for a different amount of neurons in each layer are shown in Table II. "WNeuron" is the number of neurons

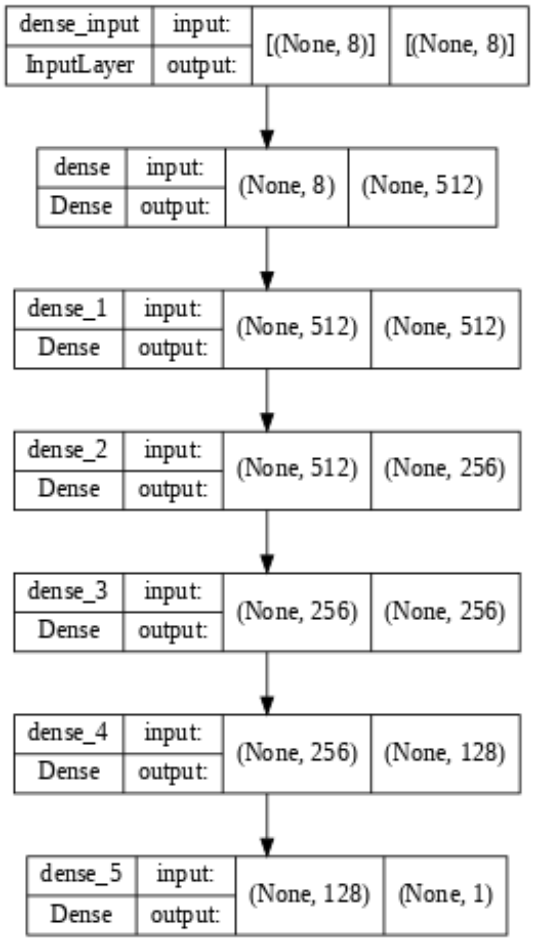


Fig. 4: The proposed neural network architecture

TABLE II: Result for different amount of Neurons

WNeuron	BNeuron	Total	White	Draw	Black
512	512	10	2	8	0
256	512	10	3	6	1
128	512	10	3	6	1
64	512	10	1	7	2

in the first layer for the white pieces model, and "BNeuron" is the same for black. We changed the number of neurons only for the white to compare the results with each other. For each case, ten games were played using the engine "Stockfish" [3]. "White", "Draw", and "Black" shows the number of games for the white win, draws, and black wins appropriate. We can see that the results are the best in the case of $n = 512$. White did not lose a game. In the case of the $n = 256$ and $n = 128$, the results were similar, white won one game more than for $n = 512$ but lost one game as well; Finally, the result for the $n = 64$ was the worst, black won four games in that case. Since $n = 512$ case still needs the same time for prediction as it is in the case of $n = 256$ in our final model, we decide to maintain 512 neurons in the first two layers. Similar tests were made for the number of hidden layers. We tested the model for three, four, and five hidden layers. The model with five layers had better results, and because there was no significant

difference in prediction times, we maintained a neural network with five hidden layers. It is important to mention that model should not be too complicated to maintain a low prediction time.

B. The Basic Model

Players with white pieces have extra tempo because they are starting. Hence, players with black pieces are trying first to equalize position and then play for a win. Because of that small difference, white has a little advantage at the start of the game. Players choose different strategies to play with different colors, so time management is also different. In this regard, we trained models according to colors, one model for the white pieces and the other for the black pieces. The dataframe shown in I was split into two parts according to the color of the pieces. In addition, for white, we use the data of the games where white won the game. Since white pieces have a little advantage, mostly draws are favorable for black pieces. So, for the black side's model, we used games that were drawn or won by black pieces.

After filtering the dataframe I, we get eight features: White's and Black's rating, number of the move, time remaining until the end of the game, amount of the legal moves in the position, total remaining material, the difference of white's and black's materials, and the expected move until the end of the game. The first model was trained based on all the features mentioned above. Both sides were using the newest version of "Stockfish" as a chess engine [3]. Regarding the time, one side was managed by a neural network and the other side with linear time control mentioned in [5]. In addition, we tested the earlier versions of the "Stockfish", which are comparably weaker versions. The difference between the results for the neural network and linear time controls was more notable in the case of weaker engines played itself than in the case of stronger engines. Strong engines are already so optimized that the time optimization problem is less important. The weaker versions of the engines are more like humans. So, the neural network trained on human data was working a little bit better. We will see detailed results in Section VI.

C. Feature Selection

We trained additional models without each feature mentioned above to calculate the importance of the features. The results of the models were compared versus the original one, which was trained for all the features. From the results, we can tell that amount of the legal moves, and the number of the move were the most important features. These two features are essential once for humans as well. The amount of legal moves determines how many candidate moves you have to consider and calculate. More candidate moves means more work to do, hence more time to spend. The move number shows approximately how much more moves will be until the end of the game and how much time you will need for the future moves.

Improvement of the model was only when we removed the remaining time from the features. Because the data used to train the model was based on human games, the results are not surprising. In general remaining time is one of the most important features for humans. However, humans are too concerned with time trouble and making impulsive moves when they have little time. Also, when remaining time is not very low, humans are still considering not spending a lot of time on one particular move to avoid future time trouble. Computers can be more precise over time-trouble. We had two close features, which were total material and the difference between white and black pieces material. The detailed results for other features as well will be mentioned in Section VI.

In addition to the features mentioned above, really interesting features for the model can be an assessment of the position, have a queen on the board or not, how many pawns are remaining in the position, etc. Positions without the queens are mostly less complicated than with queens, or positions with eight pawns each are definitely harder to play than positions with two pawns each. Hence, a lot of different models can be trained with different combinations of features. Also, it is important to mention that the data used for training was human data with different levels of chess knowledge. Another interesting model would be if we would collect the same type of data for computers and train our model with this data.

Algorithm 2 Segmented-Least-Square-Approximation

Require: Received Times T, Moves M and the amount of lines k.

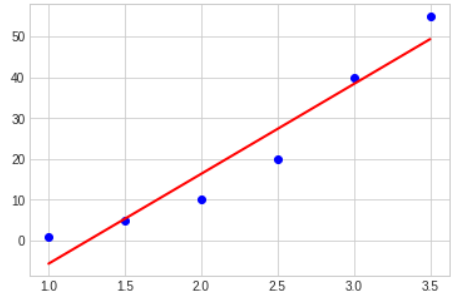
1: **for** all pairs $i < j$ **do**
2: Calculate the least square error $e_{i,j}$ for the segment $p_i...p_j$
3: Create array OPT for errors $k \times n$, for each amount of line and for each segment.
4: Fill first row of the error array with appropriate least square errors calculated above.
5: Fill other rows with ∞
6: **for** i between 1 to k **do**
7: **for** j between 0 to n **do**
7: $\min_{0 \le t < j} (OPT_{i,j}, OPT_{i-1,t} + err_{t,j})$
8: **return** OPT

V. Segmented Least Square Approximation

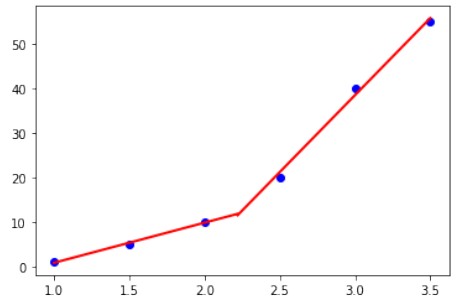
The section describes the dynamic programming algorithm for segmented least square approximation. We will discuss the general idea and then the algorithm.

A. General Idea

For humans understanding the neural network's prediction process is hard because it could not be described with functions. Humans can not use computers and neural networks during the game, so we decide to make an easier prediction version, such as linear functions. As it was already mentioned in the article [5], we can have a linear function for time prediction based on the expected number of the moves till the end of the game. According to the same paper, the expected number of the moves itself depends on the total material.



(a) Figure for $k = 1$



(b) Figure for $k = 2$

Fig. 5: Simple example of approximations

However, the expected number of the moves is not really precise. We used our collected human data to see how different people spent time during their games and created the model according to this. It would be very logical in chess if we split the game into three parts: Opening, Middlegame, and Endgame. However, it is hard to determine how many moves are included in each part of the game. Openings mostly are from 15-20 moves, then comes middlegame, and when the material is reduced, the time comes for the endgame. The most crucial part is the middlegame, where players spend most of their time. The reason is that they already have some knowledge of the openings and the theoretical endgames and middlegame are mostly new for them.

B. Algorithm

Our idea was to approximate the main function, total spending time versus the number of a move, with several linear functions. In this regard, we invented a new dynamic programming algorithm. There is already known dynamic programming algorithm for segmented least square approximation, where we have line cost. In combination with the cost and the least square error, the number of lines is chosen by the model to approximate the main function [28]. In our case, the main function is the moves (M) dependence on the total time spent for thinking (T). Each data point is the average total time spent after the i^{th} move from a player during the several games. Our modified dynamic programming algorithm receives an argument amount of linear functions as k, and the algorithm gives the k linear functions, which approximates the main function most closely. The algorithm

TABLE III: Number of points vs. Number of lines

L-P	**1**	**2**	**3**	**4**	**5**	**6**
1	173.3	173.3	173.3	173.3	173.3	173.3
2	80	17.5	4.3	9.8	105.3	173.3

works as follows: firstly, we calculate all least square errors between every consecutive subset of the points. For the n points we have $\frac{n\times(n-1)}{2}$ subset. We consider which points should be on that new line to minimize the total least square error for each new line. That can be done using the dynamic programming algorithm described in pseudocode 2. We are using $k \times n$ size array to save minimum least square errors given $[n_1, ..., n_t]$ subset and $[1, ..., k]$ line.

One other interesting part is the results between the different approximations. Results showed that the difference between three, four, and five-line approximations are almost unnoticed. In contrast, the difference between them and one line approximation is comparably high. We will see the detailed results in Section VI. The method's main advantage is that humans can easily remember three or four linear functions and use them to determine their time management strategy during the game.

C. Approximation Example

Let us consider one toy example to understand how the algorithm works. For a simple example, Figure 5(a) shows linear time approximation only with one line for six points. In that case, with one line minimum least square error is 173.3. Figure 5(b) shows the best approximation with two lines. In the algorithm mentioned above, we are starting with one line error, and filling the first row of the Table III with the error. It means that we can not split the data points with only one line, and the minimum error will be the same. When the number of lines increases by one, now we can split the data into two parts at each point. We consider every possible split and choose the one with the best total least square error. If we split the data with the third point, it means that from two lines, the first line will be for the first three points and the second line will be for the other three points, and the total least square error will be the some of the two errors with first-line and with the second line. In that case, the total error was 4.3, which, as we see in Table III is the minimum. Dynamic programming helps us fill that kind of table efficiently, without a brute force algorithm.

VI. Experimental Results

The section describes the experimental results for a different kind of neural network. In addition, we will discuss the segmented least square approximation model for a different amount of linear functions and compare their results with each other.

A. Neural Network Variations

Regarding the neural network, we changed a lot of parameters such as the number of neurons, amount of hidden layers, features, and data. The results for changing the amount of the neurons are already given in Section IV. In the case of different amounts of hidden layers, we trained neural networks

TABLE IV: Result for different amount of Layers

WLayer	BLayer	Total	White	Draw	Black
5	5	10	3	7	0
4	5	10	3	6	1
3	5	10	2	8	0

TABLE V: Result for different Feature Selection

WModel	BModel	Total	White	Draw	Black
Original	Original	10	3	6	1
WRating	Original	10	2	8	0
BRating	Original	10	1	9	0
NofMove	Original	10	1	7	2
Time	Original	10	5	5	0
LegMoves	Original	10	1	7	2
Material	Original	10	1	9	0
MatDiff	Original	10	0	10	0
ExpMoves	Original	10	1	8	1

for three, four, and five hidden layers. Table IV shows the results, where we can see that neural networks with five hidden layers have a little bit better results. In the experiment, both sides were played with the engine "Stockfish" [3]. Both sides were managing time with neural networks, however, with different numbers of hidden layers. In each case, ten games were played. "White", "Draw", and "Black" shows the amount of the white winning, draw, and black winning games accordingly.

The most interesting part was the results for different feature selections. As we already mentioned in Section IV, we trained neural networks for different feature configurations. The main model was with all the eight features we described in Section IV; however other models have seven features, all except the one feature we chose. Table V shows the results for different models played with each other. The latest version of "Stockfish" was used in this experiment from both sides. With white pieces, time management was controlled by the models with seven features, while with black pieces, time management was controlled with the model with all the eight features. The experiment was conducted to determine the importance of each feature for the model. In Table V, "WModel" describes the feature which was removed from all the features. In the first case, the original model trained for white pieces and trained for black pieces independently were played with each other. Then we removed each feature one by one and checked the models without each feature. We can see that only removing the remaining time feature had better results. As we mentioned in Section IV, it is because the data which was used for training a neural network is human data. The most important features were the number of the move and the amount of the legal moves. Without those features, models had much worse results than the original ones.

Finally, our original model was checked versus linear time management. As we already see above, white always have some advantage. In this regard, we first checked the results for linear time management with white versus the same time management with black. The 20 games were played, 11 games were drawn, white won 6 games, while black won 3 games. Table VI shows the result for neural network and linear time

TABLE VI: Result for different Time Management

WModel	BModel	Total	White	Draw	Black
Linear	Linear	20	6	11	3
NN	Linear	20	5	13	2
Linear	NN	20	3	15	2

TABLE VII: Result for different Time Management with weaker engine

WModel	BModel	Total	White	Draw	Black
Linear	Linear	20	7	8	5
NN	Linear	20	7	10	3
Linear	NN	20	5	11	4

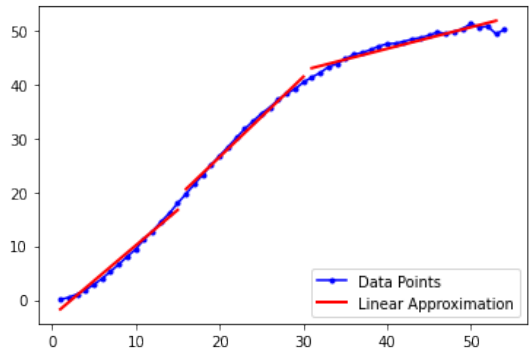


Fig. 6: Result for beginner, $k = 3$

management. As we can see the results are very close in all three configurations. Firstly, it means that neural network has promising results. On the other hand, it is because in these experiments the latest version of the "Stockfish" were used, which is already highly optimized and time management gives a little difference. In this regard, we made the experiments with the first version of the "Stockfish", which is more close to human strength. Table VII shows the result with weaker version of the "Stockfish". We can see that in that case neural network trained on human data has better results.

B. Segmented Least Square Approximation

This section will show the additional results for our least square approximation algorithm. We collected data for three different levels of human players: beginner, FIDE master, and international master. In addition, we will show the results for different linear approximations played with each other.

1) Different Level Player: Figures 6 and 7 describes segmented least square approximations for different level chess players. As we mentioned in Section V, data points are the average total spent time for a particular move. The figures 6 and 7 are for the one minute games. We can see that the first linear function in the beginner's case is from the first till about the fifteenth move, and totally they are spending about 20 seconds for the first 15 moves. A similar situation is regarding the number of moves in the case of the FIDE master player. Figure 7 shows that the first function is from the first till about the seventeenth move. However, totally they spent about 10 seconds for the first seventeenth move, while in beginners' case, it was approximately 20 seconds. The reason

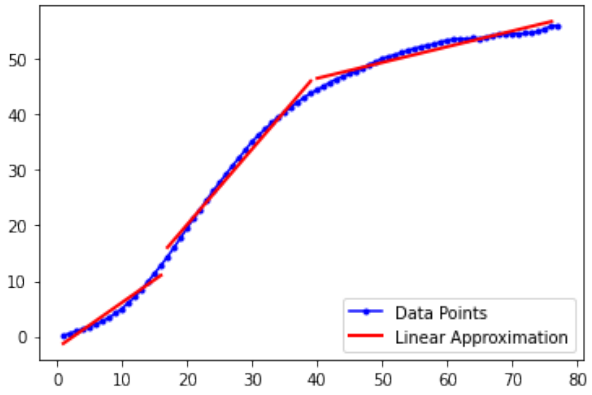


Fig. 7: Result for FIDE master, $k = 3$

TABLE VIII: Result for different Time Management

WLine	BLine	Total	White	Draw	Black
5	4	10	3	6	1
5	3	10	3	6	1
5	1	10	4	6	0
1	5	10	2	6	2
3	5	10	1	8	1
4	5	10	2	7	1

is that FIDE masters have better theoretical knowledge of the openings than beginners. If we carefully see the slopes for each linear function, we can discover that the last linear function has the smallest slope because most of the time is already spent before, and players have to make moves faster. In the case of the FIDE master player, on average, 80 moves were played, while in the case of beginners, it is 60 moves. The reason is that stronger players are playing faster and have the opportunity to make more moves at the same time. In addition, first function slopes are different too; the reason is the same as we already mentioned above better players know the theory of the openings.

2) Different Amount of Linear Functions: Figures 10 and 9 describe the linear approximations with four lines for the beginner and FIDE master chess players. We can see that approximately in both cases, the line from 1 to 17 moves was split into two parts: 1 to 9 and 10 to 17. However, we can see that using three lines for the beginner player had approximately the same error as it was with four lines. While in the case of the FIDE master player, it has a little bit more improvement. We made experiments to see how different amounts of lines were connected to the final results. In this experiment newest version of the "Stockfish" was used from both sides. However, the time control was according to the different amount of linear functions. Table VIII shows the results for a different amount of lines during linear approximations played with each other. Results between 3, 4, or 5 lines were almost identical, while in the case of 5 vs. 1, 5 lines linear approximation has a notable advantage.

3) Poor Comparison: Figure 8 shows the different level players data points on the same plot. We can clearly see that the stronger player saves time in the beginning and plays faster.

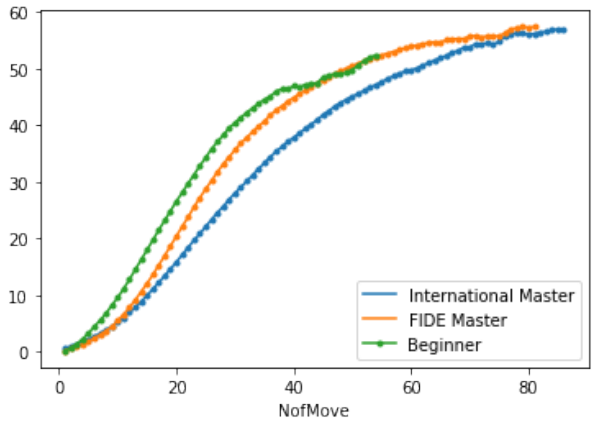


Fig. 8: Different Level Comparison

Hence, the stronger player plays more moves. It is important to mention as well, that the data was collected from "Chess.com" platform. The algorithm of the platform is such that players are playing versus the same level of the player. So, if someone is a beginner most likely their opponents are beginners too, or if someone is a FIDE master, their opponents strength will be mostly FIDE masters' level.

C. Simulation Summary

For testing our models described in Sections IV and V, we used real-time simulations. We wrote code in python, where different versions of the engine "Stockfish" can play with each other. "Stockfish" engine function has the opportunity to receive time for every move. We used this functionality and tested the same versions of "Stockfish" played with each other, with different time management strategies. The time for each player was the same as it was used during the neural network training. For instance, if we trained the neural network on three minutes of human games, the engine also had three minutes to play the match. The results mentioned in this article are real-time data collected by playing with different time management and different kind of engine levels with itself. It is important to mention that this kind of simulation takes a lot of time; however, it is the best for measuring the accuracy of the models.

VII. Related Work

The section shows related works for time management strategies, usage of neural networks in chess, and approximation algorithms.

A. Time Allocation for Computer Chess

Several strategies can be used for time management in chess. The easiest one is to calculate the average amount of moves in one game and divide the total time by that number [5]. However, this strategy depends on the static number and is not the best; if the game continues less than the average amount of moves, part of our time will remain unused; or if the game continues more than the average

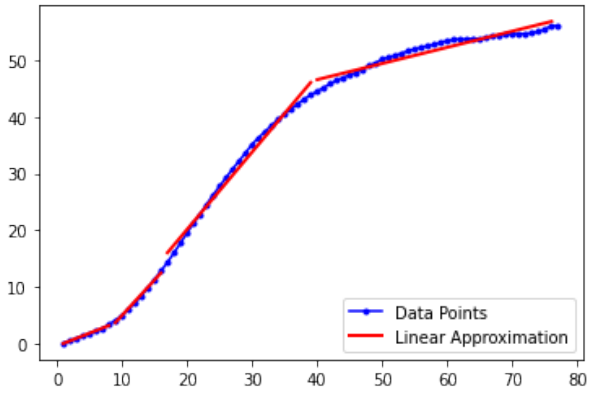


Fig. 9: Result for FIDE master, $k = 4$

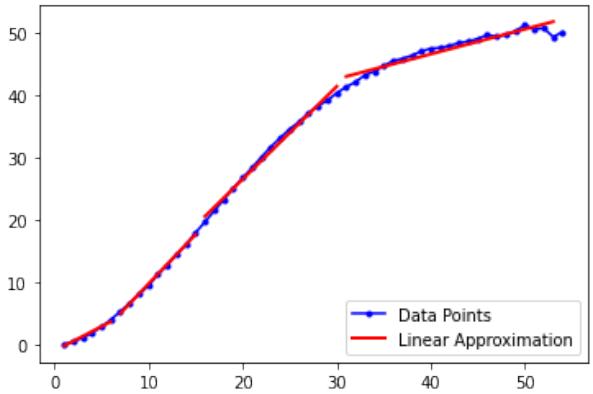


Fig. 10: Result for beginner, $k = 4$

time, we will lose the game by time [29]. So, a simple improved strategy is to have a dynamic amount of the moves till the end of the game. The paper [5] describes statistical methods to get the linear dependence between the amounts of moves till the end and the total material. Since we have the dynamic amount of moves calculated from the material, we can update our time management according to the new formula $t = T/(N(m) - n)$, where t is the allocated time, T - total remaining time, n - number of already played moves, N(m) - the dynamic amount of the moves till the end of the game calculated from total material [5]. Another interesting formula for time management was given by Robert Hyatt [1] [2], which was inspired by the several grandmaster chess players' tournament games. The calculation is following: (1) $nMoves = min(numberOfMovesOutOfBook, 10)$, (2) $factor = 2 - \frac{nMoves}{10}$, (3) $target = \frac{timeLeft}{N}$, (4) $t = factor \times target$, where N is number of moves until next time control, and t is the final allocated time for the move. All the grandmaster players know some opening theory. In that case, at the beginning of the game number of moves out of the book is equal to zero; hence factor is in most cases equal to 1. However, after the opening, the factor will decrease, so players will have less time to think. It is important to mention that chess openings in 1985 were not as improved as they are today. Nowadays, players are not spending a lot of time in openings in most cases.

B. Machine Learning

There are different methods of machine learning used for the problem of allocation of time. Most of the computer games, including chess, rely on full-width game tree search [6]. It started with a brute-force algorithm in the case of Deep Blue vs. Kasparov match [15] [16] and ended with Monte Carlo Tree-Search used by Alphazero [30]. Hence, the time management systems are mostly for tree-search-based systems. In the paper, [6] genetic algorithm was compared to the TD learning. The experiments were conducted for the game of LOA, and the TD learning had superior results. The approach shows that the opponent's strength was an important factor during TD learning. In the early years of the neural network development, there were a lot of different works about time allocation [31] [32]. During that period, machine learning algorithms and computer calculation strength were not strong. On the other hand, modern machine learning algorithms are mostly used for searching for the best move, and in most cases, engines are using simple heuristic time management systems [30].

C. Segmented Least Square Approximation

There are a lot of cases when we can not approximate the given data with one linear function; however, we need to maintain the simplicity of the approximation functions. The algorithm described in chapter [28] is known as the segmented linear approximation algorithm. Using dynamic programming data is approximated with several linear functions. The amount of the linear functions depends on the cost of one line. The amount of used lines depends on the cost of each line. In that case, we do not know how many lines we need to get the minimum error at the start of the problem. The amount of lines is totally dependent on the cost of the one line. If the cost of the line is high, the number of lines is less, and vice versa. There are different versions of the segmented least square approximation, which is mentioned in paper [33]. Segmentation criteria depend on different types of coefficients in that case. Our paper introduces a method to minimize the error with a fixed number of lines, and we do not have any line cost in the algorithm.

VIII. Future Work

Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e., the experiments with real data are usually very time-consuming). Future work concerns a deeper analysis of particular mechanisms and new proposals to try different structures of neural networks. There are some ideas that we would have liked to try during the description and the development of the models described in Sections IV and V. The following ideas could be tested:

(1) Collect the data for computer games with each other and train neural networks on that data; This part is really interesting and important because we saw that for the model trained

on human data remaining time was not an essential feature. Without remaining time, we had a better model; however, this feature should be important in general. The reason was that data was collected by human matches, and the remaining time for humans is mostly the reason of nervousness, and they are making some inaccuracy moves in the position.

(2) Add or remove more features such as: Queens on the board, the number of pawns, and the position assessment; As we mentioned in the previous paragraph, removing the remaining time feature gave us a better model. However, what would be if we added some more features?! There are different kinds of feature combinations that can be tested. It is important to mention that we need the features which are easily calculated from the position and do not take time.

(3) Our models were trained based on different levels of the human players. It is also interesting what the results would be if we combined the different level players' games together as training data. We already saw that model trained on the low-level players' data was worse in comparison to the models trained on the high-level players' games.

(4) In this article, we considered the sudden death time control; however, there are a lot of different time controls in chess. Interesting once are time controls with additional time, such as "5+2", which means that each player has five minutes till the end of the game, and for every move played, they are receiving two more seconds. In that case, you always have two more seconds to make moves, so the percentage of the games losing over time decreases. In addition, the interesting part is "Rapid" games, in which players have more than ten minutes for the games.

IX. Conclusion

To sum up, we introduce two models for time management. The segmented least square approximation can be very useful for human chess players. They can determine the thinking time by several linear functions, which can be easily remembered before and be useful during the game. The usefulness of neural networks in playing chess is already well known. Our results showed that using the neural network model for time management is also promising. In general, time management in chess is a huge problem, which depends on many dynamic variables. Hence, even choosing the features for the neural network is challenging. Neural Networks are hard to use for humans; however, it is very useful for computer playing chess. Our simulations showed that it has promising results and can be improved more in different aspects. In conclusion, we can say that time management is a highly complicated problem with a lot of variables, and it is very hard to say which strategy is the most optimal.

Acknowledgement

This research was supported in part by NSF grants CPS 2128378, CNS 2107014, CNS 2150152, CNS 1824440, CNS 1828363, and CNS 1757533.

References

[1] R. M. Hyatt, A. E. Gower, and H. L. Nelson, "Using time wisely, revisited," in *Proceedings of the 1985 ACM annual conference on The range of computing: mid-80's perspective: mid-80's perspective*, 1985, p. 271.

[2] R. M. Hyatt, "Using time wisely," *ICGA Journal*, vol. 7, no. 1, pp. 4–9, 1984.

[3] [Online]. Available: https://stockfishchess.org/

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[5] V. Vučković and R. Šolak, "Time management procedure in computer chess," *Facta Universitatis Series: Automatic Control and Robotics*, vol. 8, no. 1, pp. 75–87, 2009.

[6] L. Kocsis, J. Uiterwijk, and J. v. d. Herik, "Learning time allocation using neural networks," in *International Conference on Computers and Games*. Springer, 2000, pp. 170–185.

[7] H. Baier and M. H. Winands, "Time management for monte carlo tree search," *IEEE transactions on computational intelligence and AI in games*, vol. 8, no. 3, pp. 301–314, 2015.

[8] B. Bošanskỳ, V. Lisỳ, M. Lanctot, J. Čermák, and M. H. Winands, "Algorithms for computing strategies in two-player simultaneous move games," *Artificial Intelligence*, vol. 237, pp. 1–40, 2016.

[9] M. Kulldorff, "Optimal control of favorable games with a time limit," *SIAM Journal on Control and Optimization*, vol. 31, no. 1, pp. 52–69, 1993.

[10] M. Sadler and N. Regan, "Game changer," *AlphaZero's Groundbreaking Chess Strategies and the Promise of AI. Alkmaar. The Netherlands. New in Chess*, 2019.

[11] Y. N. Harari, "Why technology favors tyranny," *The Atlantic*, vol. 322, no. 3, pp. 64–73, 2018.

[12] [Online]. Available: https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match

[13] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *arXiv preprint arXiv:1509.01549*, 2015.

[14] M. Newborn, "Chess 4.7 gives levy a run for his money," *The Mathematical Intelligencer*, vol. 1, no. 4, pp. 215–217, 1979.

[15] J. Schaeffer and A. Plaat, "Kasparov versus deep blue: The rematch," *ICGA Journal*, vol. 20, no. 2, pp. 95–101, 1997.

[16] Y. Seirawan, "The kasparov–deep blue games," *ICGA Journal*, vol. 20, no. 2, pp. 102–125, 1997.

[17] D. B. versus Kasparov, "Computer chess."

[18] D. Sieberg, "Kasparov:"intuition versus the brute force of calculation."," *CNN/ACCESS*, 2003.

[19] [Online]. Available: https://database.chessbase.com/

[20] [Online]. Available: https://lichess.org/

[21] [Online]. Available: https://chess24.com/en

[22] [Online]. Available: https://www.chess.com/home/

[23] [Online]. Available: https://pypi.org/project/chess.com/

[24] [Online]. Available: https://python-chess.readthedocs.io/en/latest/

[25] N. Vaci and M. Bilalić, "Chess databases as a research vehicle in psychology: Modeling large data," *Behavior research methods*, vol. 49, no. 4, pp. 1227–1240, 2017.

[26] N. K. Manaswi, "Regression to mlp in keras," in *Deep Learning with Applications Using Python*. Springer, 2018, pp. 69–89.

[27] C. Zhang and P. C. Woodland, "Parameterised sigmoid and relu hidden activation functions for dnn acoustic modelling," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[28] J. Kleinberg and E. Tardos, "6.3 segmented least squares: Multi-way choices," in *Algorithm design*. Pearson Education India, 2006, pp. 261–266.

[29] S. Markovitch and Y. Sella, "Learning of resource allocation strategies for game playing," *Computational Intelligence*, vol. 12, no. 1, pp. 88–105, 1996.

[30] M. C. Fu, "Simulation-based algorithms for markov decision processes: Monte carlo tree search from alphago to alphazero," *Asia-Pacific Journal of Operational Research*, vol. 36, no. 06, p. 1940009, 2019.

[31] M. Gagliolo and J. Schmidhuber, "A neural network model for inter-problem adaptive online time allocation," in *International Conference on Artificial Neural Networks*. Springer, 2005, pp. 7–12.

[32] M. Gagliolo, V. Zhumatiy, and J. Schmidhuber, "Adaptive online time allocation to search algorithms," in *European conference on machine learning*. Springer, 2004, pp. 134–143.

[33] E. Fuchs, T. Gruber, J. Nitschke, and B. Sick, "Online segmentation of time series based on polynomial least-squares approximations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2232–2245, 2010.