Generating chess puzzles using genetic algorithm

Faculty of Mathematics
University of Belgrade

Anđelija Vasiljević
mi20222@alas.matf.bg.ac.rs

Table of Contents

1 Introduction....................3
1.1 Description....................3
2 Implementation....................3
2.1 Representation....................4
2.2 Tournament selection....................4
2.3 One-point crossover....................4
2.3 Mutation....................4
2.4 Fitness function....................5
2.4.1 Number of pieces on the board....................5
2.4.2 Validity of the board and constraints for number of specific pieces....................5
2.4.3 Classification of the puzzles....................6
2.4.4 Adding Stockfish....................6
3 Evaluation and Results....................7
3.1 Classification Model and GA Results....................7
3.1.1 Comparing The Models....................8
3.1.2 Quality....................13
4 Conclusion....................17
5 References....................18

1 Introduction

Chess puzzles have long been admired for their ability to test and improve a player's strategic and tactical skills. These puzzles, consisting of specific chess positions, require finding the optimal moves to achieve a desired outcome, such as checkmate or gaining a significant advantage. While chess puzzles are traditionally made by analysing played chess games,the use of artificial intelligence (AI) techniques, such as genetic algorithms (GAs), to automate the puzzle generation process. The idea was proposed by Andrew Israel on the Propelauth website. This suggestion by Andrew Israel presents an exciting opportunity to explore the application of GAs in creating chess puzzles.

1.1 Description

The project focuses on generating a specific category of chess puzzles: “Mate in 3” puzzles using genetic algorithm. These puzzles require players to find a sequence of moves that leads to checkmate within three moves. By automating the puzzle generation process, we can overcome the limitations of manual methods, ensuring a larger pool of puzzles with varying degrees of difficulty. This automated approach offers chess players a dynamic puzzle collection, enabling them to continually challenge themselves, refine their skills, and deepen their understanding of the game.

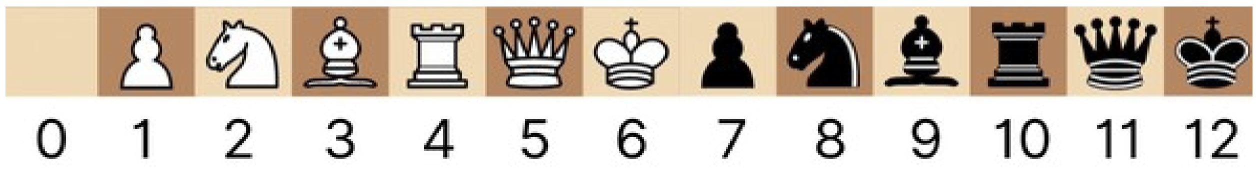
2 Implementation

Genetic algorithms (GAs) provide a powerful optimization technique inspired by the principles of natural selection and genetics. This algorithmic approach mimics the process of evolution by iteratively evolving a population of candidate solutions towards an optimal solution. The implementation of the genetic algorithm for puzzle generation involved several key components and steps:

- Representation of the chess board
- Tournament selection
- One-point crossover
- Mutation
- Fitness function

2.1 Representation

Chess positions were represented using an array of 64 integers – one for each chess position. The integers are between 0 and 12, either an empty square or a specific chess piece. By representing it this way we can make it suitable for the genetic algorithm to use and find new solutions.



2.2 Tournament selection

Tournament selection selects a group of individuals randomly from the population. The fitness of the selected individuals is compared and the best individual from this group is selected and returned by the operator. For crossover with two parents, tournament selection is done twice, once for the selection of each parent.

2.3 One-point crossover

A one-point crossover operator was developed that randomly selects a crossover point, and the arrays after that point are swapped between the two parents. It is important to note that the one-point crossover operator is applicable to problems with a fixed-length representation, such as the chess puzzle generation problem in this project.

2.3 Mutation

Mutation is a genetic operator used to introduce random changes in the genetic material of individuals. It helps in maintaining diversity within the population and prevents the algorithm from getting stuck in local optima. In the chess puzzle generation problem, mutation can be applied to the genetic representation of an individual, which is an array of integers representing the chess positions. The mutation operator randomly selects a gene (position) in the array and modifies it to a different value within the allowable range.

2.4 Fitness function

The fitness function plays the most crucial role in the GA as it evaluates the quality of each individual in the population. Here, in the context of chess puzzles, the ultimate goal was to get a puzzle that has a mate in 3 for white. But before that, adding other constraints of different kinds is necessary to achieve the best possible puzzle.

2.4.1 Number of pieces on the board

Firstly, a constraint was added for the number of pieces that the board should have. The function uses the python-chess library to determine the number of pieces on the board. Then the desired number of pieces can be specified and the GA is rewarded for generating puzzles with that number of pieces. By doing that the GA can create different types of puzzles.

Chess puzzles can be categorized based on the stage of the game they represent:

- Opening puzzles: These puzzles focus on the initial moves of the game and usually involve strategic considerations and piece development. They aim to test the player's understanding of opening principles and their ability to make sound opening moves.
- Middlegame puzzles: Middlegame puzzles occur after the opening phase and before the endgame. They often involve complex positions with multiple pieces on the board. These puzzles challenge players to find the best strategic plans, tactical combinations, and positional maneuvers.
- Endgame puzzles: Endgame puzzles focus on positions with a limited number of pieces remaining on the board, typically involving kings and a few other pieces. These puzzles require precise calculation and understanding of endgame principles to find the winning or drawing moves.

The primary focus was on middlegame and endgame puzzles. The opening puzzles being the most challenging for generating because of the structure that they have and also high number of pieces. It's important to note that the specific number of pieces in a chess puzzle can vary but for the purpose of this project, a range of number of pieces for every category was considered.

2.4.2 Validity of the board and constraints for number of specific pieces

Adding a constraint for the validity of the board was the next crucial step. A valid chess puzzle should have a single king per side, legal moves and a legal position, all according to the rules of chess. Luckily python-chess has a function that checks the validity of the board. If the puzzle was found to be invalid it would be given a high penalty.

To guide the GA further constraints were added for the number of pieces every side should have. For example, one king per side, two knights per side, two bishops per side and so on. By doing this we can get puzzles that have specific attributes. Changing these attributes, we can get all sorts of interesting puzzles like, having as many knights as possible. In the end a penalty is given when there are more pieces of a certain kind. In general all of these numbers can be tailored to give a desired puzzle of choice.

2.4.3 Classification of the puzzles

In order to get the best puzzles, a classification model was added. The purpose of it being to classify if the puzzle is real or fake. For the real database Lichess database of chess puzzle was used and for the fake “Yet another chess problem database”.

Real puzzles are puzzles derived from played games, where engines use that game to analyse it and potentially find positions that can be a chess puzzles. On the other hand, ‘fake’ puzzles that the model used so called chess compositions. The definition of chess compositions, used from the YACPDB site: “Puzzles set by somebody using chess pieces on a chess board, that present the solver with a particular task to be achieved.” Also more context from the person who holds the database: “The composed chess problems are indeed non-realistic in the sense that these positions are extremely unlikely to arise during a normal chess game.” Knowing this we can now say that the real puzzles are made from played games and that the fake ones are not. Very important note to add that was also given by the founder of the database: “There are aesthetic criteria that are followed in composition (of course there are exceptions, but they are relatively rare)

- The position must be legal (that is theoretically reachable from the initial array by the series of the legal moves)

- It's desirable that the single box of piece is used (that is no promoted units) - this is not so strict, but still applies

You will find that most of orthodox threemovers in YACPDB obey to these two rules.”

Meaning that the compositions don’t have invalid positions and also can use only the 32 pieces that are given in the beginning of the chess game(16 pawns, 4 rooks, 4 knights, 4 bishops, 2 queens, and 2 kings), but not so strict to this. To address the validity of puzzles and the use of specific pieces, the constraints outlined in section 2.4.2 cover these aspects. The invalid positions won't be given to the model because if its invaild it will give it a penalty and the function will return it.

After the model is finished we can add it to the fitness function and it can evaluate the generated puzzles and classify them as real or fake. If the puzzle is classified as fake the fitness function will return a high penalty. The accuracy of the models that were trained will be discussed in the chapters ahead.

2.4.4 Adding Stockfish

For the final part of the fitness function, adding Stockfish, a powerful chess engine, was necessary. The main goal is to get puzzles that have mate in 3. Using the stockfish engine we can achieve that. The following steps were added:

- If the game is over, meaning there are no available moves – return a high penalty. Similary, if there is only one move that can be played – also return a high penalty because those puzzles are uninteresting.
- We want specifically puzzles where white can mate in 3 moves. To do this a penalty is applied to cases where white does not have a forced mate. The fitness function aims to favor puzzles where White has a clear path to mate in 3 moves. Also add a penalty for the distance away from mate in 3, the farther the puzzle is from mate in 3, the higher the penalty.

By adding Stockfish, the fitness function is complete and can now be used to generate puzzles.

3 Evaluation and Results

3.1 Classification Model and GA Results

Logistic regression

The initial model that was used is Logistic regression. Here's how the model was trained and evaluated:

1. Data preparation:
    - Two sets of data were used: one for real puzzles and one for fake puzzles.
    - Since both sets use FENs (Forsyth-Edwards Notation, standard notation for chess boards), it needed to be converted to feature arrays. Using a function which converted the FENs representation of each puzzle into a numerical feature array. The arrays that we get are the same as the ones we use for representation of the board for the GA. Also one more feature was added that represented number of pieces on the board.
    - For each set, the feature arrays were generated from the corresponding FENs, and the target labels were assigned (1 for real puzzles, 0 for fake puzzles). Then they were concatenated to form the final training data (X) and target labels (y).
2. Train-Test split:
    - The training data and labels were split into training and testing sets using the train_test_split() function from scikit-learn.
    - The training set (X_train, y_train) was used for model training, and the testing set (X_test, y_test) was used for evaluating the model's performance.
3. Logistic regression model:

- An instance of the LogisticRegression class was created and the model was trained using the training data and labels by calling the fit function on (X_train,y_train)

4. Models evaluation:

    - The trained model was used to predict the labels for the testing data by calling predict on (X_test), resulting in y_pred.

Several metrics were calculated to evaluate the model's performance, including accuracy, precision, recall, and F1 score.

The model that was made used 238.968 of puzzles from the real dataset, where that number consists of 88.968 mate in 3 and 150.000 random puzzles. The fake dataset took 236.749 puzzles, where around half are from the YACPDB and the other half collected during generating of the puzzles. It achieved the following preformance:

- Accuracy: 0.8673
- Precision: 0.8766
- Recall: 0.8560
- F1 score: 0.8661

The accuracy shows that the model's predictions are correct approximately 86.73% of the time.

Decision Tree Classifier

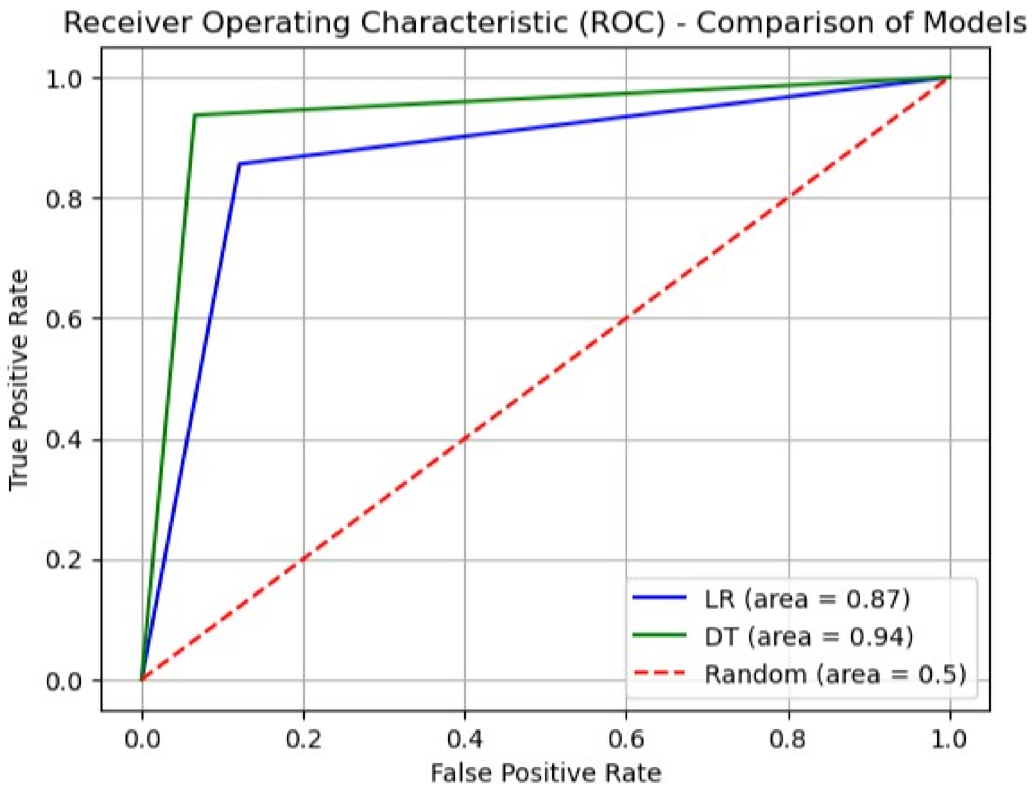
This classifier used the same data preparation and train-test split as the logistic regression model did. The difference is only on the model used. The max depth for the decision tree that was used is 71, which had proven to give the best results. It achieved the following:

- Accuracy: 0.9356
- Precision: 0.9347
- Recall: 0.9370
- F1 score: 0.9359

The scores for this model are much better and seem more promising.

3.1.1 Comparing The Models

To compare the two models we can use the ROC curve. In a binary classification problem, the ROC curve helps evaluate the model's ability to distinguish between the positive and negative classes.



We can see that the decision tree has a better ROC score of 0.94 meaning that it does a better job at classification. Now they can be added to the GA.

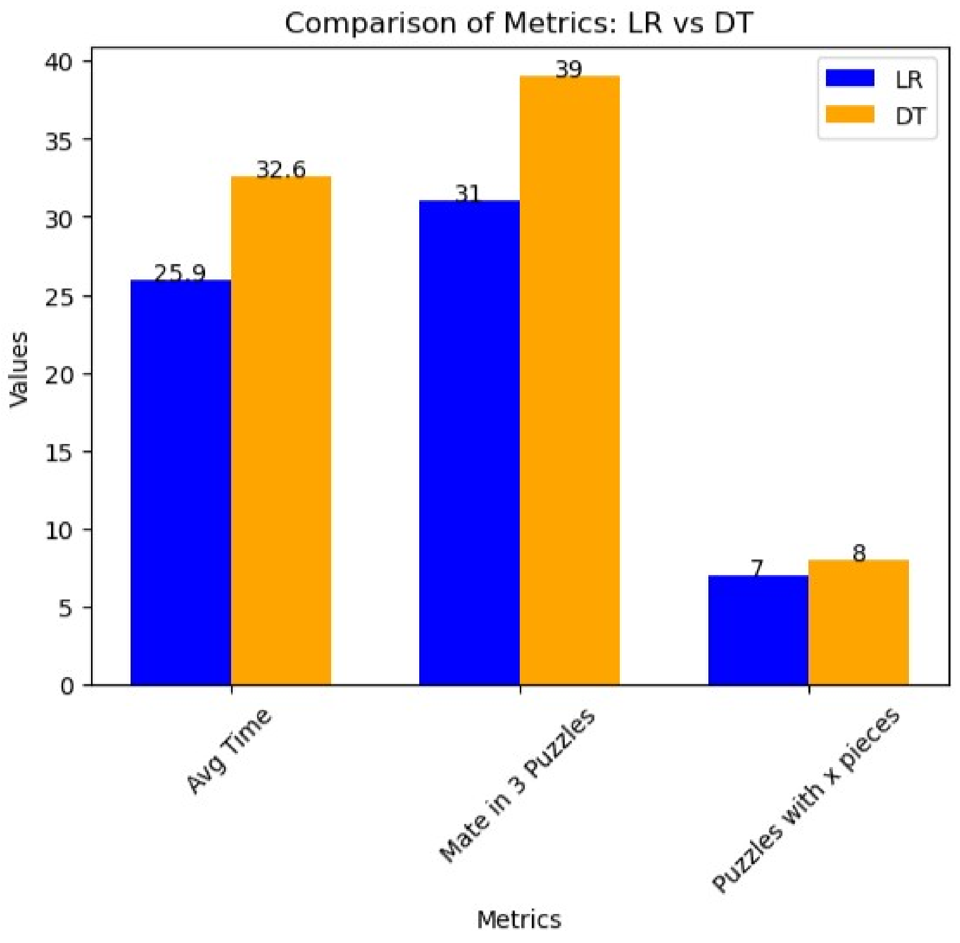
First test:
Lets run two GAs one with DT(Decision tree) and the other one with LR(Logistic regression). Both GAs will use these parameters:

- Population size - 20,
- Number of iterations – 2000,
- Tournament size – 5,
- Crossover probability – 0.9,
- Mutation probability – 0.05,
- Elitism size – 2,
- Number of iterations with no improvement – 500.

For each configuration of pieces on the chessboard, 10 puzzles were generated using each model, resulting in a total of 220 puzzles—110 puzzles per model.

The results:

Number of pieces(x)	**Average fitness function**		**Average time in seconds**		**Number of invalid boards**		**Number of mate in 3 puzzles**		**Number of puzzles that have x number of pieces and are mate in 3**	
	LR	**DT**	**LR**	**DT**	**LR**	**DT**	**LR**	**DT**	**LR**	**DT**
5	7.74	8.49	49.72	47.27	0	0	8	8	0	0
7	9.43	7.15	40.62	77.72	0	0	3	8	0	0
9	8.67	8.79	45.15	47.51	0	0	4	6	0	0
11	8.08	6.55	51.45	50.22	0	1	6	6	1	3
13	10.05	9.40	29.03	39.50	1	2	3	4	0	0
15	9.26	6.47	25.15	35.24	2	0	2	2	1	2
17	13.80	10.68	8.61	35.83	3	3	2	3	2	1
19	12.15	16.79	13.90	6.80	3	6	2	1	2	1
21	14.25	13.85	13.96	11.51	4	3	0	1	0	1
23	19.04	19.04	3.46	3.57	8	8	1	0	1	0
25	18.04	19.04	3.88	3.46	6	8	0	0	0	0
mean/sum	11.86	11.48	25.9	32.6	27	31	31	39	7	8



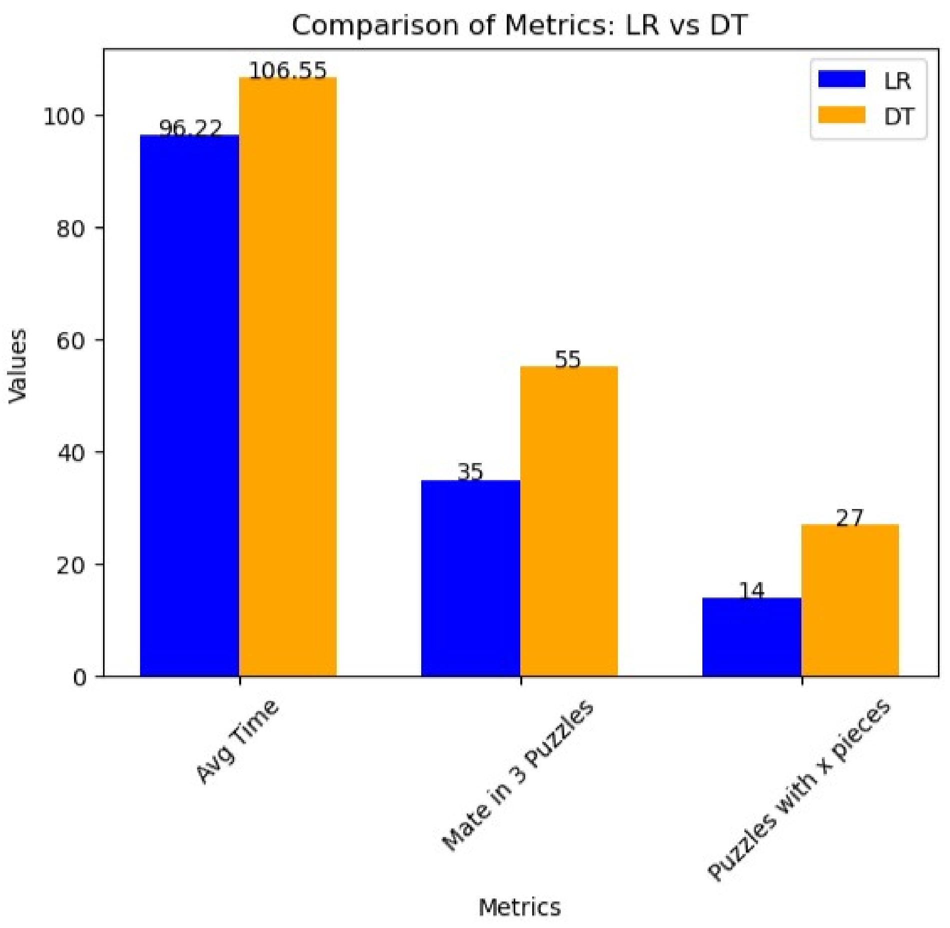
In the first test, the average time taken by both models was similar, with the DT model taking about 6 seconds more than the LR model. However, when considering the number of mate in 3 puzzles, the DT model outperformed the LR model by finding 8 more puzzles. Similarly, in the comparison of puzzles with a specific number of pieces, the DT model found 8 puzzles compared to the LR model's 7. Based on these results, it can be concluded that in the first test, the DT model performed slightly better overall. The slight difference of 6 seconds in average time is insignificant when considering the significant advantage of finding 8 more puzzles.

Second test:
Now lets double the parameters to give the GA a better shot at finding more puzzles.
New parameters:

- Population size - 40,
- Number of iterations – 4000,
- Tournament size – 10,
- Crossover probability – 0.9,
- Mutation probability – 0.05,
- Elitism size – 4,
- Number of iterations with no improvement – 1000.

The results:

Number of pieces(x)	Average fitness function		Average time in seconds		Number of invalid boards		Number of mate in 3 puzzles		Number of puzzles that have x number of pieces	
	LR	**DT**	**LR**	**DT**	**LR**	**DT**	**LR**	**DT**	**LR**	**DT**
5	5.25	6.35	190.24	172.55	0	0	8	9	0	0
7	6.56	6.05	125.20	121.07	0	0	6	10	0	0
9	4.62	2.96	150.11	151.08	0	0	8	9	4	5
11	6.22	3.60	117.07	145.09	0	0	5	8	3	4
13	3.82	5.51	118.01	108.02	0	1	5	4	4	3
15	7.67	1.76	94.08	142.27	1	0	1	5	1	5
17	13.71	10.16	39.37	73.64	2	5	0	4	0	4
19	10.35	11.20	74.01	70.15	2	2	1	2	1	2
21	10.20	10.65	77.04	126.27	0	3	1	2	1	2
23	14.33	12.49	21.79	47.89	1	3	0	2	0	2
25	14.64	19.04	51.46	14.05	4	8	0	0	0	0
mean/sum	8.85	8.16	96.22	106.55	10	22	35	55	14	27



In the second test, the average time taken by both models remained similar, with the DT model taking approximately 10 seconds more than the LR model. However, there was a significant difference in the number of mate in 3 puzzles found by each model. The DT model found 55 puzzles, while the LR model found 35 puzzles, resulting in a difference of 20 puzzles in favor of the DT model. The number of puzzles with a specific number of pieces increased for both models, but the increase was more significant for the DT model with 27 puzzles compared to the LR model's 14 puzzles. Based on these results, it can be concluded that in the second test, the DT model performed even better than in the first test. Despite taking slightly more time, the DT model found a significantly higher number of mate in 3 puzzles and puzzles with specific piece numbers, demonstrating its superiority over the LR model.

3.1.2 Quality

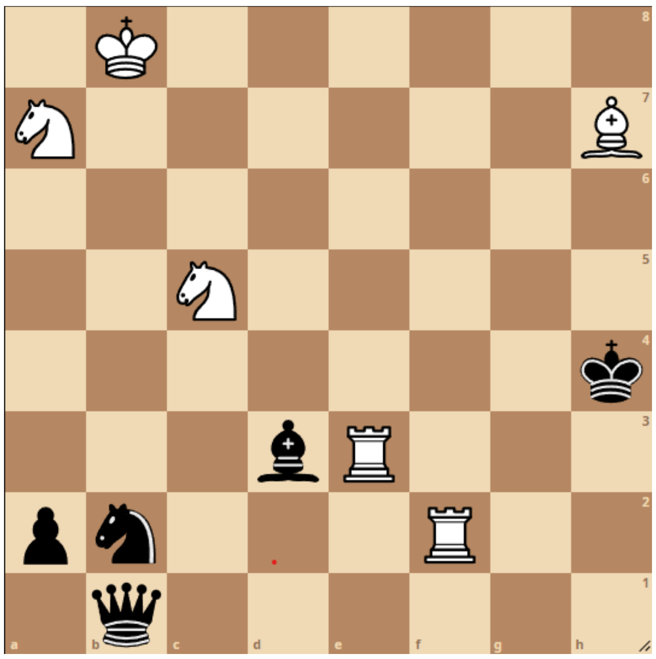
The quality of the generated puzzles was done by human judgment as it remains the most reliable method. In addition to that certain criteria were considered :

- How difficult it was to see the winning combination,
- The overall look of the puzzle,
- Favored when white side has fewer pieces as it requires more precise moves to achieve checkmate.

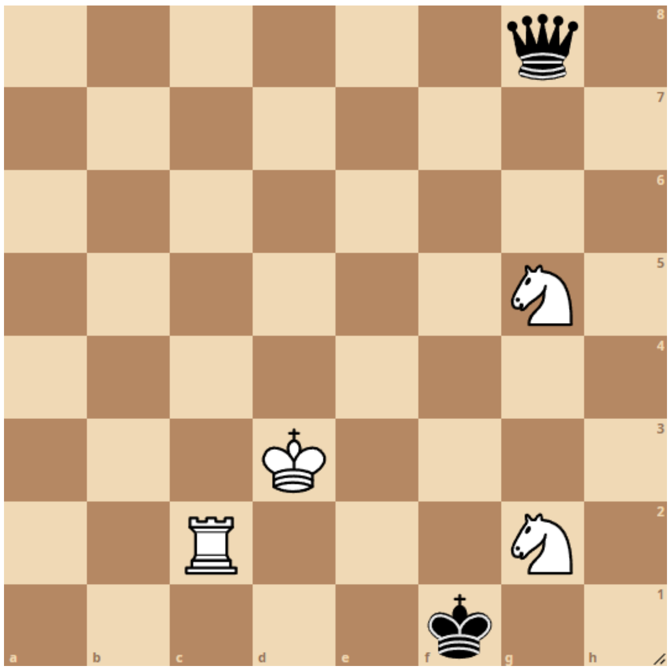
In addition to the criteria, the puzzles that were picked were personal preference. The showcased puzzles were chosen to demonstrate the creativity and variety that can be achieved through the puzzle generation process. They represent a combination of different difficulty levels, strategic elements, and aesthetic qualities.

LR puzzles

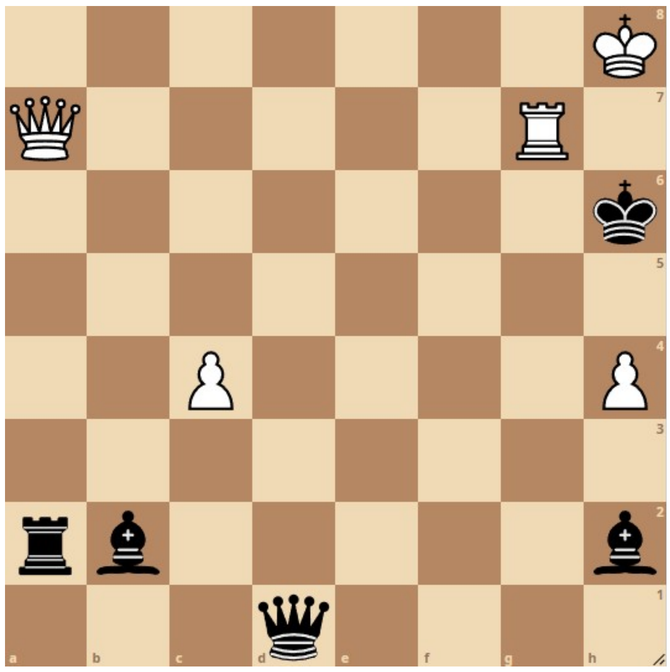
The puzzles generated by the LR model exhibited less difficulty and limited variety.
Here are some examples that were interesting:



Here black is one move away from promoting but white has a mating position.
Winning moves: 1. Rf4+ Kh5
2. Rh3+ Kg5
3. Ne6#



A material disadvantage for white but a winning combination with the knight and the rook.
Winning moves: 1. Ne3+ Kg1
2. Nf3+ Kh1
3. Rh2#



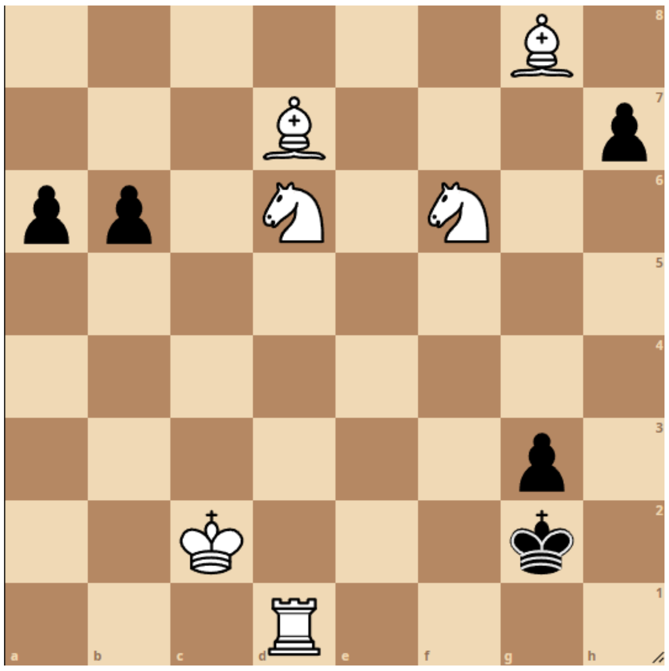
Here only one move wins and all the other are losing.

Winning moves: 1. Qe3+ Bf4
2. Qxf4+ Kh5
3. Qg5#

DT puzzles

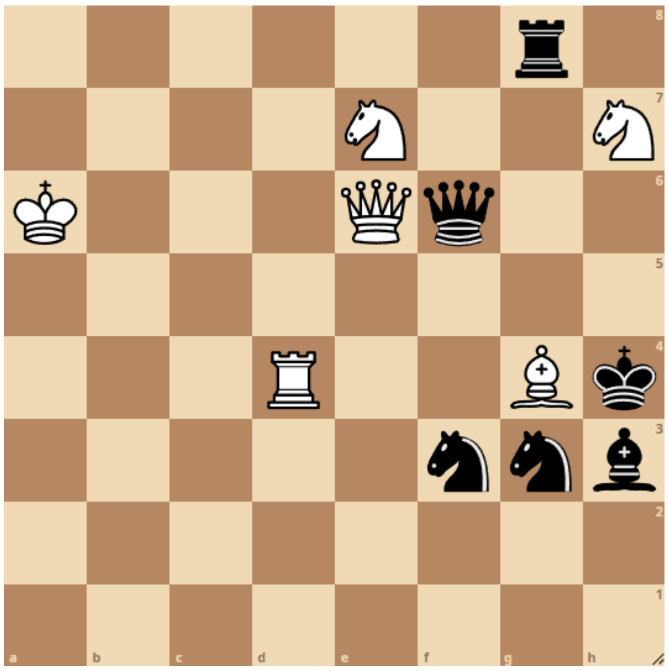
The DT model generated more puzzles so there was more to choose from. Overall they looked better and were challenging. One particularly exciting discovery was that two of the puzzles, when analyzed using Lichess, were found to be positions that had been played in actual games. This shows the capability of the DT model to generate puzzles that closely resemble real-game scenarios.

Here are some examples:

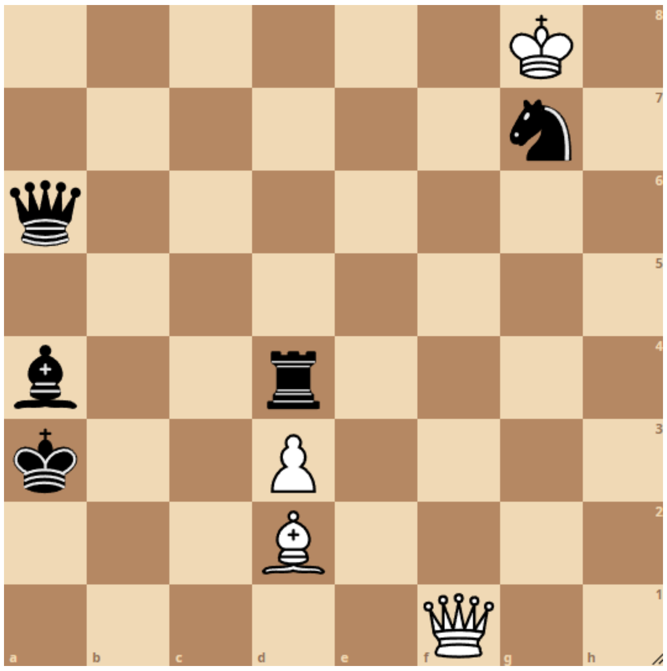


Black is here clearly worse but the mate combination is tricky to find in just 3 moves.

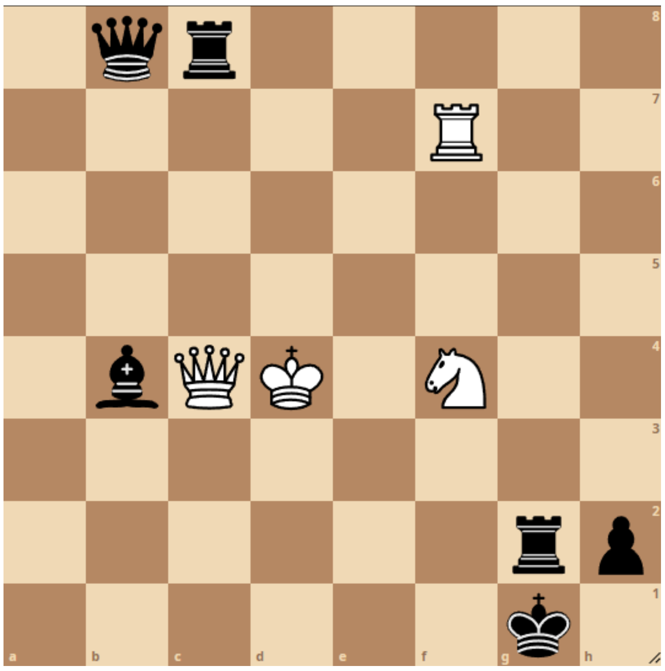
Winning moves: 1. Bc6+ Kf2
2. Kg4+ Ke2
3. Bc4#



This one is interesting because it resemblances a smothered mate.
Winning moves: 1. Qxf6+ Ng5
2. Qh6+ Nh5
3. Nf5#



In this position, white has a material disadvantage. However white can still deliver mate.
Winning moves: 1. Qa1+ Kb3
2. Qb1+ Ka3
3. Bc1#



Whites queen is under attack but we get another smothered mate.
Winning moves: 1. Nh3+ Kh1
2. Qf1+ Rg1
3. Nf2#

4 Conclusion

In conclusion, this project demonstrated the effectiveness of using a genetic algorithm in combination with Stockfish and classification models to generate good puzzles. The results showed that the GA approach performed well in generating puzzles with a targeted number of pieces ranging from 7 to 17. The puzzles made within this range showcased the capability of the algorithm to create engaging and challenging puzzles withing a short time. And the DT model showed to be the better model overall.

It is important to acknowledge the limitations too. The GA approach showed limitations when dealing with puzzles that had a high number of pieces (17 and higher). In cases that it did find those type of puzzles the quality of them was low. These limitations suggest that further improvement is needed to address those challenges.

When it comes to changes and improvement there are several aspects that can be different. Here we focused on mate in 3 but the limitations on what type of puzzle we want to find are endless. One thing that can change is exploring different classification models that may lead to better puzzles. Collecting more data to give the models a better chance at learning the differences. And lastly, finding the solution to generating puzzles with high number of pieces.

By addressing the limitations and implementing the suggested improvements, the puzzle generation process can be further advanced, ultimately providing better puzzles.

5 References

1. Andrew Israel, "Generating Chess Puzzles with Genetic Algorithms," Propelauth, [https://www.propelauth.com/post/generating-chess-puzzles-with-genetic-algorithms]

2. Computational Intelligence - An Introduction, Andries Engelbrecht, John Willey & Sons, 2007.

3. Stockfish: An Open Source Chess Engine. [https://stockfishchess.org/]

4. Lichess database: Puzzles. [https://database.lichess.org/]

5. YACPDB('Yet another chess problem base'). Mate in 3 puzzles. [https://www.yacpdb.org/]