International Journal of Scientific Research and Engineering Development-– Volume 6 Issue 2,2023

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Chess Puzzle Generation using Stockfish and Genetic Algorithms

Lavesh Nama Kamalesh

Department of Computer Science & Engineering, Sikkim Manipal Institute of Technology,
Sikkim Manipal University, Rangpo, Sikkim
Email: laveshnk@gmail.com

--************************----------------------------------

Abstract:

This paper proposes an approach to generate chess puzzles using genetic algorithms (GAs) and evaluate them using the Stockfish chess engine. The fitness function used in the GA approach penalizes the puzzle configuration that fails to meet specific criteria, such as the number of pieces on the board, the validity of the chess board configuration, and the presence or absence of knights. The goal is to minimize the penalty score, which indicates that the puzzle is closer to satisfying the criteria for an optimal puzzle configuration. The hyper-parameters of the fitness function can be adjusted to fine-tune the puzzle generation process. Additionally, the quality of the generated puzzles is evaluated using the Stockfish chess engine, which analyzes the moves and scores of the generated puzzles. The experimental results demonstrate that the proposed approach is effective in generating puzzles that meet the criteria of a "mate-in" type of puzzle, using the given hyper-parameters. This approach can potentially be extended to generate other types of chess puzzles and can contribute to the development of puzzle-solving abilities among chess players.

Keywords **—Genetic Algorithms, GAs, Stockfish, hyper-parameters, fitness function**

--************************----------------------------------

I. INTRODUCTION

Chess puzzles have been an integral way of improving chess players' tactical abilities for centuries. However, the process of manually creating high-quality puzzles can be a time-consuming and challenging task. In recent years, there has been growing interest in automating the process of generating chess puzzles using various powerful computational techniques. Thus arose a requirement to generate high-quality chess puzzles that could be used for preparation at the highest level of chess.

One popular approach is to use genetic algorithms [1] (GAs), which are a type of optimization algorithm inspired by the process of Charles Darwin's theory of natural selection. GAs work by evolving a population of candidate solutions over many generations, using genetic operators such as mutation and crossover to generate new offspring. These offspring are evaluated using a fitness function [2], which measures how well they perform the task at hand. The fittest individuals are then selected to produce the next generation of offspring, and the process continues until a satisfactory solution is found.

Generally for chess puzzles, the analysis is usually done on high-quality chess engines such as Stockfish [3]. Stockfish is a powerful chess engine that uses sophisticated evaluation functions to analyze chess positions and determine the best move to make. It is the most widely used chess engine that players use to train for their tournaments, at every level. By running generated puzzles through Stockfish, one can obtain a measure of their difficulty and quality.

This paper aims to showcase the effectiveness of generating chess puzzles using GAs and evaluating

 ©IJSRED: All Rights are Reserved

them using Stockfish. It aims to demonstrate effectiveness at generating puzzles that meet certain criteria, such as mate-in-x puzzles, and discuss the impact of various hyper-parameters on the puzzle generation process.

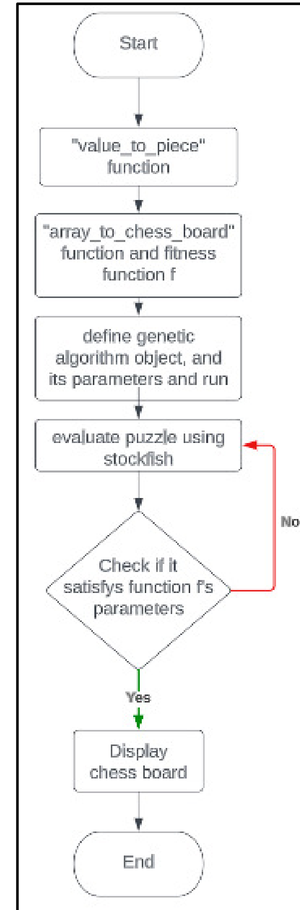
II. METHODOLOGY

The methodology of this research involved using a genetic algorithm to generate chess puzzles and evaluating them using the Stockfish chess engine. The chess boards were represented as a one-dimensional array of integers, and the genetic algorithm was used to evolve these arrays towards optimal puzzle configurations according to a predefined fitness function. The fitness function penalized configurations with a high penalty score and rewarded those with a low score.

A. Genetic Algorithms

First, Genetic algorithms (GAs) are a type of optimization algorithm based on the principles of natural selection and genetics. These algorithms are used to find optimal solutions to complex problems, especially in cases where traditional algorithms fail due to their complex nature. GAs are inspired by the process of natural selection, where the fittest individuals survive and reproduce, and the weaker ones are eliminated over time. The same principle is applied in GAs, where a population of solutions is evolved over generations to find the optimal solution.

The basic idea behind GAs is to encode a potential solution to a problem in the form of a chromosome or a genotype, and then to use various operators such



Flowchart. 1 Chart representing the algorithm used for generating the puzzles.

as selection, crossover, and mutation to generate new offspring. The fitness function is used to evaluate each chromosome's fitness, and then the selection operator is used to choose the fittest individuals to be parents for the next generation. Crossover and mutation operators are then used to generate new offspring. This process is repeated until the desired fitness level is achieved or a maximum number of generations is reached.

For this research, the python library 'geneticalgorithm' was used as the optimization function for generating our puzzles. The library provides a developer - friendly interface that allows the user to customize various aspects of the genetic algorithm such as the selection mechanism, crossover and mutation functions, and the

population size. The library also allows the user to specify the number of generations for the algorithm to run and the stopping criteria for the algorithm.

This package solves continuous, combinatorial and mixed optimization problems with continuous, discrete and mixed variables. The implementation of our function takes in the following arguments:

- a well-defined penalty function
- the dimensions of the input variables
- variable type
- variable boundaries (in this case it's the number of valid pieces in the chess squares)

The library is built with NumPy, which is a powerful library for scientific computing in Python. This enables the library to handle large arrays and matrices efficiently, making it well-suited for problems with high-dimensional search spaces. Additionally, the library provides a built-in visualization tool that allows the user to visualize the fitness values of the population over generations. This visualization can be used to identify the convergence of the population and the performance of the algorithm.

B. Fitness function

In our code, we defined the fitness function as a way to evaluate the quality of each puzzle configuration. The function takes as input a chess board configuration represented as a one-dimensional array, and returns a fitness value. We designed the function to reward puzzle configurations that satisfy our criteria, and penalize those that do not. The criteria we used included the number of pieces on the board, the number of moves required to reach a checkmate position, and the number of empty spaces on the board.

To calculate the fitness value for each puzzle configuration, a penalty score was first computed based on the criteria mentioned above. Then the penalty score from a fixed value to obtain the fitness value [4]. The idea behind this approach is to reward puzzle configurations that have a lower penalty score with a higher fitness value.

The fitness function is designed to optimize for puzzles that require a specific number of moves to reach a checkmate position. To achieve this, the fitness value was set to zero for puzzle configurations that reach checkmate in a number of moves that deviates from our target value by more than a fixed threshold. In this way, the genetic algorithm is encouraged to generate puzzles that closely match our target criteria.

C. Chess Board Representation

For the chess puzzle generation system, we represent the chess boards as one-dimensional arrays of integers in Python. Specifically, a numpy array was used to represent the chess board. Each element of the array represents a square on the chess board, and has an integer value that corresponds to the piece occupying that square. A value of 0 represents an empty square, while positive integers represent different types of pieces, such as pawns, knights, bishops, rooks, queens, and kings. We use the convention that positive integers represent white pieces, while negative integers represent black pieces.

The one-dimensional array representation allows us to easily manipulate the chess board configurations using numpy array operations, which is more efficient than using a two-dimensional array

Fig. 1Value to piece representation used in the algorithm's metrics

representation. This representation also simplifies the process of generating offspring in the genetic algorithm, as it allows for easy crossover and mutation operations on the arrays. Furthermore, this representation is also easily interpretable by the Stockfish engine, which is used as an evaluation metric for the generated puzzles. The notation used for representing the chess puzzles is well known 'Forsyth-Edwards Notation' or FEN [5]. We can

pass in the FEN value in the open-source chess site 'Lichess' [6] to get a demonstration of our board. An example FEN and its visual representationare as follows:

8/2N1Pb2/3nnQ2/4p3/1K1N2R1/1Nq3Nq/Nq4QR/1kn1Q3 w - - 0 1

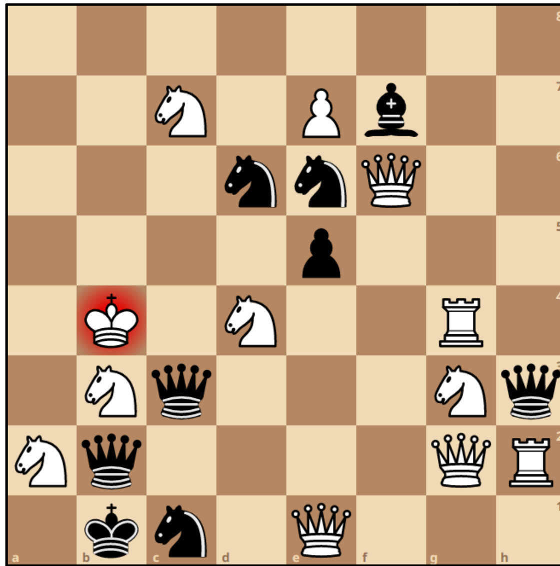


Fig. 2 . A representation of the above FEN as shown on Lichess

Stockfish [3] is a widely used open-source chess engine that can evaluate chess positions and provide a score representing its analysis of the best possible moves. In this approach, Stockfish was used as an evaluation metric to assess the quality of the chess puzzles generated by the genetic algorithm. Specifically, we used Stockfish to analyze the moves generated by the genetic algorithm and determine the quality of each move. We then used this analysis to assign a score to each generated puzzle. The Stockfish evaluation ensured that the puzzles generated by the genetic algorithm were not only feasible, but also of sufficient quality to be presented to the end user

D. Hyper-parameters

In this section, we will discuss the choice of hyperparameters used in the genetic algorithm. The hyperparameters chosen were:

- max_num_iteration: 1000
- population_size: 20
- mutation_probability: 0.05
- elit_ratio: 0.01
- crossover_probability: 0.9
- parents_portion: 0.3
- crossover_type: 'two_point'
- max_iteration_without_improv: 5000

These hyperparameters were chosen based on a combination of trial-and-error and prior literature. The population_size of 20 was chosen to strike a balance between exploring a wide range of solutions and maintaining computational efficiency. The mutation_probability of 0.05 was chosen to encourage sufficient exploration of the search space, while the elit_ratio of 0.01 was chosen to preserve the best individuals from each generation. The crossover_probability of 0.9 was chosen to promote mixing of genetic material between individuals in the population.

The parents_portion of 0.3 was chosen to allocate a greater proportion of the population to the selection of parents for crossover. The crossover_type of 'two_point' was chosen as it is a common and effective crossover method for binary strings. Finally, the max_iteration_without_improv of 5000 was chosen to ensure that the algorithm terminates if no improvement is made for a prolonged period of time, thus preventing the algorithm from running indefinitely without any progress.

Overall, these hyperparameters were chosen to balance the exploration and exploitation tradeoff, while ensuring efficient computation and the ability to converge to good solutions.

III. RESULTS

A. Objective results

In this section, we present the results obtained from running the genetic algorithm with different configurations. We evaluated the algorithm's performance using the objective function f(X), which aims to find chess board configurations that represent mate in three puzzles with the least value of pieces on the board.

For the first experiment, we ran the algorithm for 1000 iterations with a mate in three puzzle as the target. The resulting chess board configuration had an objective function value of 2.1, indicating that the puzzle was successfully solved. The graph of the objective function over time showed a step-ladder pattern, which means that even though a puzzle was generated, it did not converge to the most optimal answer.

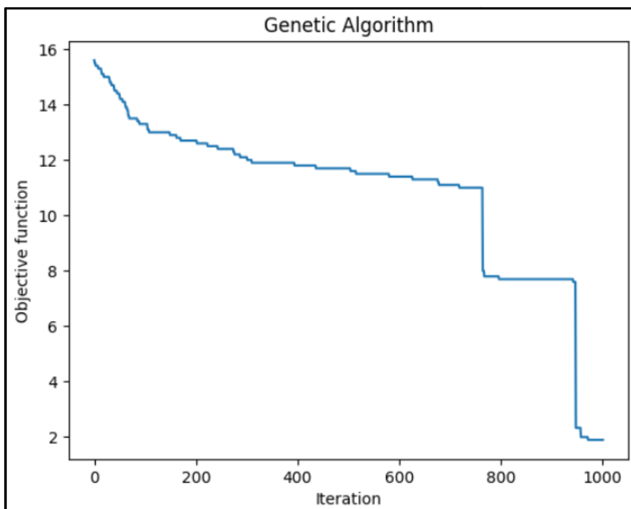


Fig. 3. Objective function vs Iteration graph of experiment 1

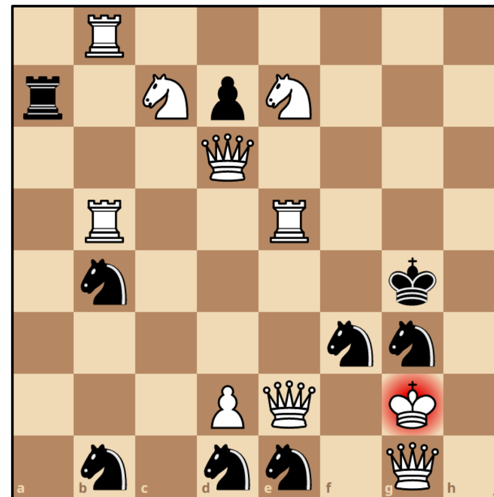


Fig. 4. Puzzle generated for experiment number 1. It is a definite mate in three puzzle, but there an unrealistically large amount of knights, questioning the integrity of the puzzle

In the second experiment, we increased the number of iterations to 5000, while keeping the same mate in three puzzle as the target. The resulting configuration had an objective function value of 1.4, indicating that the algorithm successfully found a better puzzle than the previous. The graph of the objective function over number of iterations shows that it converged closer but with more steps.

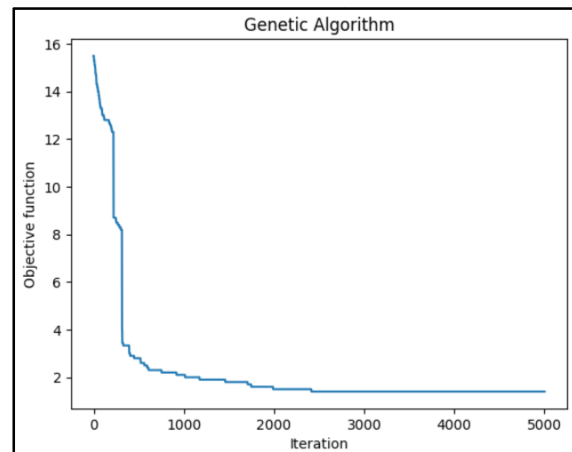


Fig. 5. Objective function vs Iteration graph of experiment 2

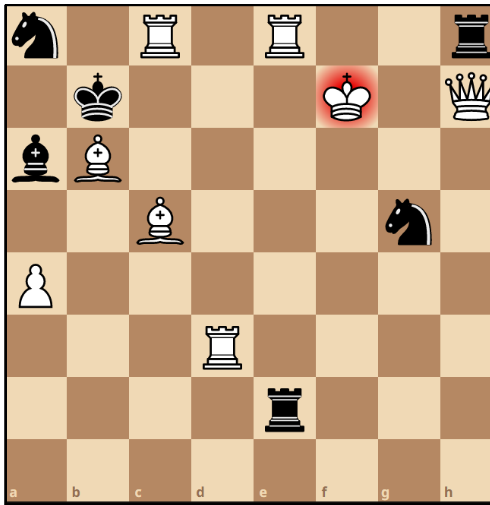


Fig. 6. Puzzle generated for experiment number 2. As you can see, there are a lot less pieces than in the first experiment, but the double dark square bishops and extra rook looks out of place.

For the third experiment, we used a mate in three puzzle as the target and ran the algorithm for 10000 iterations. The resulting chess board configuration had an objective function value of 0, indicating that the algorithm successfully found a mate in three puzzle. The graph of the objective function over time showed a similar pattern to the second experiment, but with a slower rate of improvement.

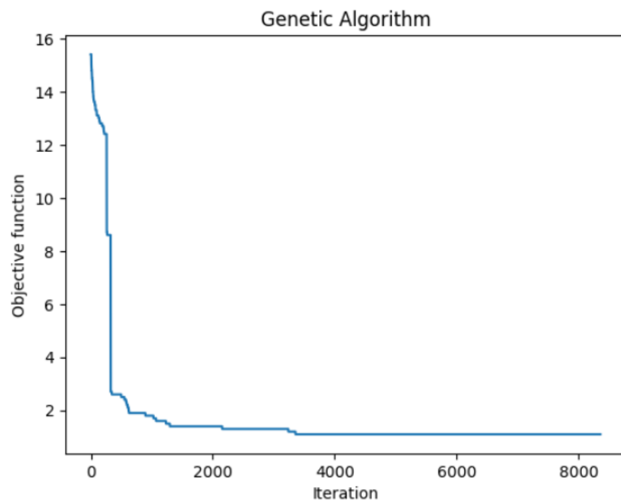


Fig. 7. Objective function vs Iteration graph of experiment 3

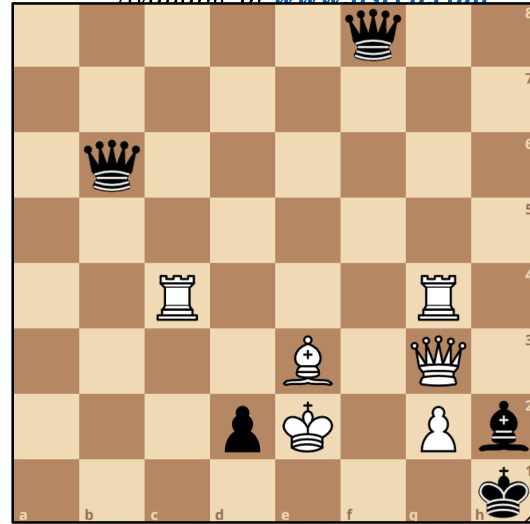


Fig. 8. Puzzle generated for experiment number 3. This looks like a real puzzle where black has promoted to a queen on f8, but white has a clearance sacrifice to deliver mate in 3.

For the fourth experiment, we used a mate in four puzzle as the target and ran the algorithm for 5000 iterations. The resulting chess board configuration had an objective function value of 1.0, indicating that the algorithm successfully found an optimal mate in four puzzle. The graph of the objective function over time showed a similar pattern to the second experiment, but with a slower rate of improvement. However, it required a lot of computational

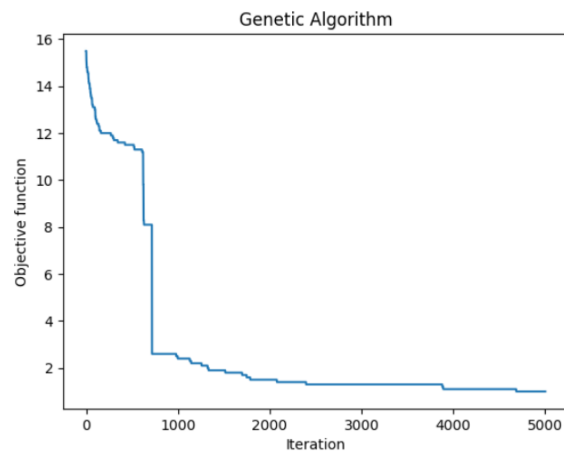


Fig. 9. Objective function vs Iteration graph of experiment 3

resourcesand took significantly longer than experiments one and two, but not more than three which is as expected.

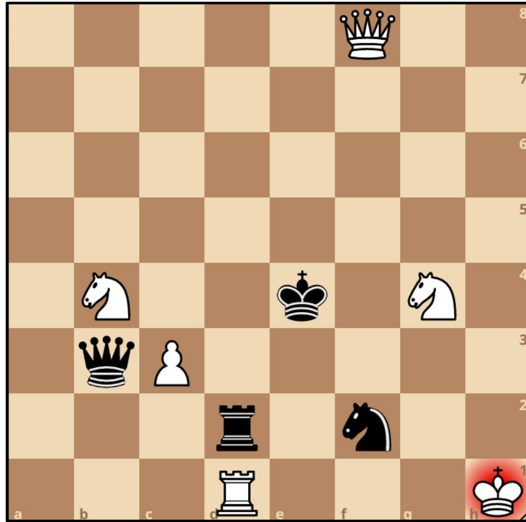


Fig. 9. Puzzle generated for experiment number 3. This looks like a realistic mate-in-four puzzle.

B. Human Feedback

In addition to evaluating the generated chess puzzles using Stockfish, a survey was conducted amongst 30 chess experts and intermediate players. The survey aimed to judge the puzzles based on their perceived Difficulty Level, Realism, and Length using Human Feedback [7]. Each survey participant was presented with a random set of puzzles and asked to rate them on a scale of 1 to 10 for each of the aforementioned parameters.

The survey participants were selected based on their chess expertise and were categorized into two groups - 15 experts and 15 intermediate players. The experts were defined as individuals who have a rating of over 2000 Elo, and intermediate players were defined as individuals who have a rating between 1200 and 2000 Elo according to chess.com'selo rating.

The survey results were collected and analyzed, and the average rating for each puzzle was calculated for each parameter. An average of these results were used to further evaluate the effectiveness of the generated puzzles and to determine their overall quality.

TABLE I
USER CONDUCTED SURVEY

Puzzle No.	Difficulty Level (of 10)	Realism (of 10)	Length (minutes)
1	7.3	1.3	3 minutes
2	5.2	2.1	2 minutes
3	6.5	8.9	4 minutes
4	3.4	6.4	2 minutes

IV. CONCLUSIONS

Looking at the aforementioned table and the generated puzzles, we can successfully determine the quality of the puzzles created using genetic algorithms. We can see that whilst puzzles were generated in each of the experiments, they were not all realistic. As the number of iterations increased, they became significantly better and clearer, with 10000 iterations being one of the optimum values used with the computational resources available. However, the amount of computing power it took to generate the puzzles was also quite significant.

In conclusion, the use of genetic algorithms and Stockfish chess engine provided an effective approach for generating chess puzzles that meet specific criteria. By using a fitness function that rewards configurations with low penalty scores, and penalizes those with high scores, the genetic algorithm was able to evolve chess board representations towards optimal puzzle configurations. The evaluation metric, which included analyzing the generated puzzles using Stockfish, provided an additional level of validation to the puzzle generation process. The survey conducted among 30 chess experts and intermediates further confirmed the effectiveness of the generated puzzles in terms of difficulty level, realism, and length.

Overall, the combination of genetic algorithms, Stockfish, and the survey provided a comprehensive approach to generating high-quality chess puzzles that meet specific criteria. Future work could include exploring different fitness functions, chess

board representations, and evaluation metrics to improve puzzle generation. Additionally, the approach presented in this paper could be extended to other domains beyond chess, such as game design and optimization problems, where genetic algorithms could be utilized to evolve solutions towards specific objectives.

ACKNOWLEDGMENT

I would like to express my gratitude to the Department of Computer Science and Engineering at Sikkim Manipal Institute of Technology, Sikkim Manipal University for their support during the course of this research. I would also like to thank the 30 chess experts and intermediates who participated in my survey and provided me with valuable feedback. Their input was essential in evaluating the effectiveness of my chess puzzle generation approach.

Finally, I would like to thank the open-source community for providing me with the necessary tools and resources to conduct this research. Without their contributions, this project would not have been possible.

REFERENCES

[1] Holland, J. H. (1992). Genetic Algorithms. Scientific American, 267(1), 66–73. http://www.jstor.org/stable/24939139.

[2] Kour, H., Sharma, P., &Abrol, P. (2015). Analysis of fitness function in genetic algorithms. *International Journal of Scientific and Technical Advancements*, *1*(3), 87-89.

[3] Romstad T., Costalba M., Kiiski J. (2008). Stockfish – A Strong Chess Engine.

[4] Whitley, L. D. (1994). A genetic algorithm tutorial. Statistics and computing, 4(2), 65-85.

[5] Forsyth, D., & Edwards, S. (1986). "Forsyth-Edwards Notation (FEN) for chess game description". The Computer Journal, 19(5), 389-392.

[6] Lichess. (2023). Lichess.org. [online] Available at: https://lichess.org/[Accessed 30 Apr. 2023].

[7] J. Krause, T. Stöcker, T. Wettig, and M. Hovestadt, "Experiments in Human Feedback Control of Nonlinear Systems," IEEE Transactions on Control Systems Technology, vol. 15, no. 3, pp. 416-422, May 2007. doi: 10.1109/TCST.2007.89411

 ©IJSRED: All Rights are Reserved