Aesthetic Chess Puzzle Generation

Deep Mehta
New York University
New York, USA
dmehta@nyu.edu

Jayesh Gaur
New York University
New York, USA
jjg9777@nyu.edu

Cunshu Ni
New York University
New York, USA
cn2139@nyu.edu

***Abstract*—Chess is one of the well-studied OTB (over-the-board) puzzles because of its simple rule-set and finite states. With DeepMind taking over the SOTA results after training itself with self-learning in 2017, there has been little progress in the domain. While we built very strong chess engines to play the game, the use of AI to build Chess Puzzles has always been an overlooked topic. Most puzzles available today are filtered through the database of all played games by looking for particular patterns. The current SOTA chess puzzle generator builds puzzles from scratch, but only works for the "Mate in X moves" category. Moreover, it was worked upon before the rise of use of high computational algorithms in the domain of games - which comes with limitations that we look to overcome. We propose a novel solution where we leverage the available computational resources as well as incorporate the Aesthetic heuristics to create an algorithm that generates "beautiful puzzles". We define heuristics as a set of mathematical formulae that quantify a given chess position into numeric value. Using these, we attempt to generate puzzles by exploring the reverse search state of a given end position on the chess board**

I. Introduction

Chess has been one of the board games to receive immense attention from a research perspective to test algorithms against. The victory of IBM's Deep Blue against the former chess champion, Gary Kasparov, changed the perspective of how the game was played [1]. A move that surprised everyone was a Knight sacrifice by Deep Blue for a compensation of a pawn and positional advantage [2]. Until now, sacrifices were rarely seen in professional chess plays unless there was a clear intention. After this event, tremendous progress was seen in the use of computers for chess and its analysis.

In 2017, DeepMind announced its new engine, *AlphaZero*, that used Neural Networks combined with MCTS to further improve the difficulty of chess engines [3]. While the previous methods involved heuristics formulated and tested by masters, experts and previous databases, *AlphaZero* made use of self-learning. The only input to the agent was the basic rules of chess, and the training was done through self-learning by playing over 44M games against itself. AlphaZero defeated Stockfish - the best chess engine in 2017 - by a score of 28W-0L-72D [4]. Since 1997, no human has been able to beat a computer chess engine.

However rich the advancement in state-of-the-art chess engines is, there is an area of chess that is overlooked by these engines - Chess Puzzles. A chess puzzle refers to a particular state in chess that consists of a solution that gives a strong advantage to one of the players. Solving chess puzzles is considered an inseparable part of Chess practice. As we will find further, a good chess puzzle can be evaluated based on a few heuristics. If we can come further to such an extent with chess engines, why is it so difficult to build chess puzzles?

To understand this, we need to go through what makes a puzzle. For the basics, a puzzle needs to have:

1. Focus on a particular theme for analysis of the interaction between chess pieces
2. Provides advantage to the solver over the opponent
3. A single solution regardless of what the opponent plays
4. No unnecessary moves - the solution has to be in the minimum moves possible

Azlan Iqbal summarizes all the puzzle essentials into 21 points in their work [5]. Such puzzles are either manually composed by chess composers or searched through a huge database of played games following particular patterns. What makes it so hard for a computer to do the same is that it is considered as a form of art, and with any other art model, AI is far from achieving human-like behaviour. Whether chess qualifies as a form of art or not is out of scope for our goal, but a lot of research is done over the topic [6].

II. Related Work

While a chess puzzle can be defined by the above-mentioned objectives, a puzzle also has an Aesthetic aspect, or in the chess world, "beauty of chess". For a computer program, every move is a move, regardless of how ugly it may be for humans. This limits generation of a quality puzzle with the use of computers. Beauty or aesthetics is a highly subjective heuristic and involves subjective responses that are affected by personal tastes. However, we attempt to encapsulate them in a few objectives, based on the common patterns observed throughout the puzzles [5] [7]:

1. Violate basic heuristics - sacrifice, silent key moves, disguise
2. Emphasize on the most powerful piece
3. Play moves that prepare other pieces for a follow-up

An Aesthetic puzzle is not about the randomness or difficulty of the puzzle, but the set of moves that are not so natural under regular positions. Moreover, an Aesthetically pleasing puzzle may not occur in real gameplay.

So far, we have only seen one advancement in the domain of chess puzzle composing computer program - Chesthetica [8]. It looks at the Aesthetic heuristics of the game and uses an

https://github.com/jayeshgaur/Chess-Puzzle-Generator approach called Digital Synaptic Neural Substrate (DSNS) [9]. DSNS allows the AI to leverage computational aesthetics. It generates multiple positions using the puzzle essentials which are then evaluated against the theme and these heuristics - while many assume DSNS to be a Neural-Network based approach, it is not. It combines data from unrelated domains into strings to enable regeneration of creative elements [8]. However, Chesthetica comes with a few limitations:

- Uses traditional approaches: During the time Chesthetica was being developed - 2006, Deep Learning approaches were uncommon in the game domain.
- Supports one type of puzzle: These puzzles are targeted for a specific kind of puzzle - Mate in X moves, where X is between 2 to 5. It does not consider puzzles where the goal of the puzzle may be to win material or improve the position immensely.
- White has a clear advantage: One common observation with Chesthetica puzzles is that White always has a material advantage over black.
- Puzzles have low stakes: Most puzzles involve a position where White is winning. A good puzzle is where one wrong move from white can turn the game in black's favor, which frequently occurs in real games.

Thus, while there is no definitive definition for a great puzzle, it can be written as a perfect combination of Composition (difficulty) and Aesthetics (beauty).

III. Backtracking

The fundamental part of our approach for generating puzzles revolves around reversing chess moves from a particular state. A single backtracking or a reverse move refers to all the possible moves that could be played in state S_{t-1} to reach the state S_t - where x in S_x refers to the move at depth / move number x. For generating puzzles, we start with any given game state such that one player is at a definite advantage and find the best reverse moves for each player in turns from that state, ideally within the depth of 10 (5 moves for each player). While it is easy to backtrack chess moves when you have the history in the form of PGN of any particular game, choosing the best reverse move is a little complex. We tackled this problem by implementing our own tree-search algorithm first to generate all possible backtracking moves.

The chess backtracking moves can be classified into three main categories:

A. Legal

Every forward move - non-capture, non-pawn pieces - Knight, Bishop, Rook, Queen and King, will also be a legitimate reverse move. To generate these, we utilize existing chess engines that generate all possible forward moves and assume their reverse to have been executed in the past turn.

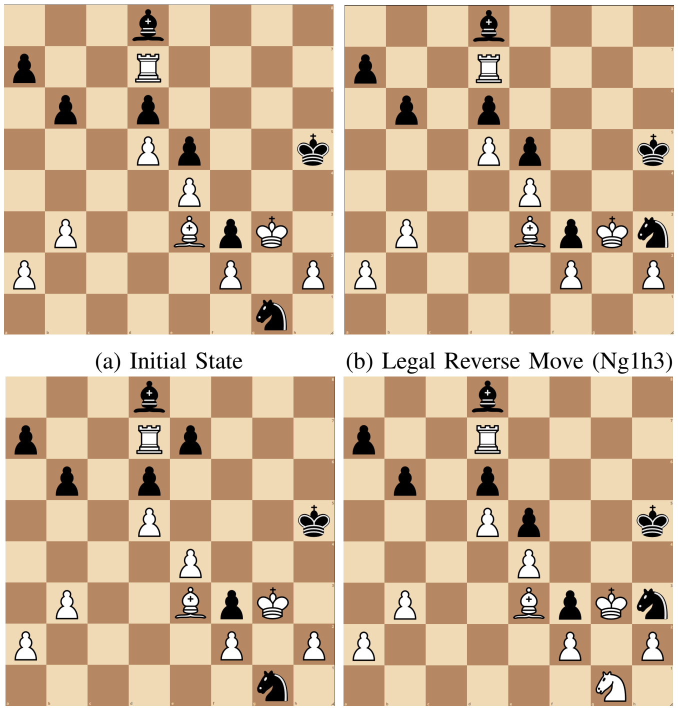
B. Pawn

Pawns can fall back one or two steps, depending on their position on the board. They can also step back diagonally, considering they captured a piece when they moved. We achieved this by picturing the chess board as a 8x8 matrix, with numbered cells in sequence and then identifying a numerical pattern of pawn movements. For example: a single step back would mean subtracting 8 from it's current position.

C. Uncapture Moves

These are essentially reverse of a move where a piece is captured. All of "Legal" moves and diagonal reverse moves for Pawns can also be uncapture moves. We generate the Legal and Pawn moves first, play the move and then add missing pieces from the board on the initial position of the moved piece.

Additionally, we add a validator to ensure all the backtrack moves are legal by playing the corresponding forward move in the previous state. This is to take care of indirect illegal moves - discover checks, pinned pieces, etc. For the purpose of this project, we do not consider castling, pawn promotions and En-Passant moves for backtracking as they are usually uncommon in Chess Puzzles.


(a) Initial State (b) Legal Reverse Move (Ng1h3)

(c) Pawn Reverse Move (e5e7) (d) Uncapture Move (Ng1h3)

Fig. 1: Backtrack Moves

IV. Implementation

Chess engines like StockFish [10] leverage the computation power to rapidly evaluate multiple search states. On the other hand, solely relying on Aesthetics limits the scope of the puzzles. We can leverage the best of both worlds to build a reverse chess puzzle builder.

A. *Heuristics*

To begin with, we first have to decide how we decide the Aesthetics. For the same, we select the most common themes as heuristics along with the classical evaluations.

1) Classical position evaluation - We use Stockfish's open-source evaluation to find the zero-sum advantage [10]
2) One heuristic each for: sacrifice, fork, pin, material disadvantage [7]

Heuristics form the base for the algorithm to quantify beauty of a chess puzzle into numeric metric in order to evaluate and identify good puzzles. We use these heuristics to retrieve puzzles of varying themes and further optimize the puzzle by reducing the irrelevant pieces and moves. As we develop further, we can modify these heuristics - add new heuristics, change weights, create pre-defined themes, etc. to adapt to diverse themes.

1) **Sacrifice:** Sacrifice refers to giving up on material to gain an advantage in the position - generally a mate or positional compensation. We calculate this based on the difference in material between the initial and final state of the puzzles.

$$\frac{(w_1 - w_2) - (b_1 - b_2)}{m}, m \subset \{9.5, 15.13, 20.76...\}$$

Here, w_x, b_x refers to material of white and black respectively on move x, where $x = 0$ for initial position of the puzzle. m is the sacrifice constant that increases with an increase in the number of moves. It indicates the maximum sacrifice possible in the given moves. In 2a, after sacrificing the queen on *Qg8+*, we follow *Rg8*, *Nf7#*. THus, we give up the queen for a smothered mate.

2) **Pin:** A pin themed puzzle is where you attack an opponent's piece that is protecting another piece of a higher value. A special case of pinning is *absolute pin* where the pinned piece is defending the king. In such scenario, moving the pinned piece is an illegal move and thus is a stronger pin.

$$\frac{1}{f_p}\frac{v(f_{p_1}) + v(f_{p_2})}{f_p}$$

f_p is the pin constant, $v(f_{p_1})$ is the value of the pinned piece, $v(f_{p_2})$ is the value of the higher-valued piece and f_p is the value of the pinning piece - opponent. In 2b, we see an example of absolute pin where black Knight is pinned by white's rook.

3) **Fork:** A fork is a position in chess where one of the piece of the player with current turn is attacking opponent's more than 1 pieces of higher value than itself. We commonly see forks using Knights because of their mobility. However, they're possible with all the pieces.

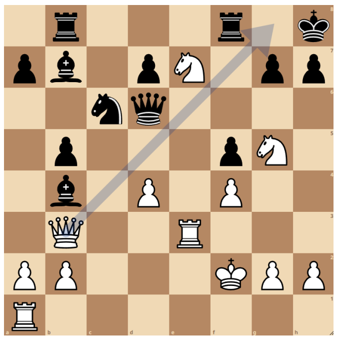
$$\frac{1}{f_f}(\frac{\sum_{i=1}^{n} v(f_{p_n})}{v(f_k)} + n), f_f = 41.52$$

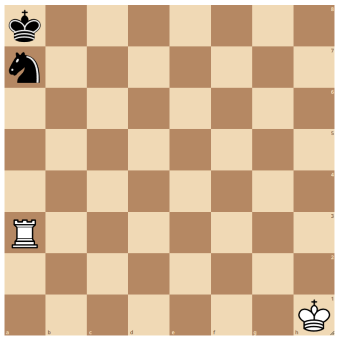
f_f is the fork constant, $v(f_{p_n})$ is the value of the n^{th} forked piece and $_v(f_k)$ is the value of the forking piece. It is the maximum possible value of the 7 pieces forked by a knight. Theoretically, it is possible to fork 8 pieces, but since knight has to move from one of the squares, it will be empty. In 2c, the Knight shows a great example of fork. Here, it is attacking black's King, Queen and Rook at the same time.

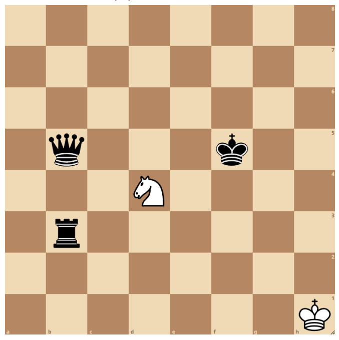
4) **Material Disadvantage:** Chess is not only about having more pieces than the opponent, but also the positional advantage. One major aspect of puzzles is having a winning line with less material than the opponent - you may have only 1 rook against opponents 1 queen and 2 rooks, but you can have a forcing mate in 1 move!

$$\frac{(b_1 - w_1)}{m}, m = 41.52$$

Here, m is the material disadvantage constant. We see this in figure 2d where white has much less material than black, but has a mate in 1 move.


(a) Sacrifice


(b) Absolute Pin


(c) Fork

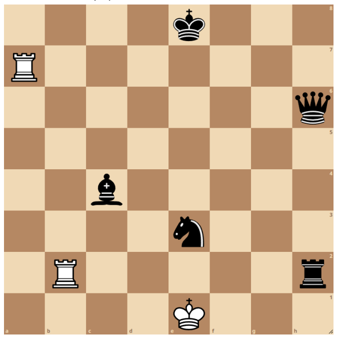

(d) Material Disadvantage

Fig. 2: Heuristics Examples

Weights:

A major hyper-parameter in the heuristics is the use of weights for each heuristic. The puzzle search can be seen as a maximization problem of these heuristics such that:

$$\max H = \sum_{i \subset n} W_i \cdot H_i$$

where, H_n is an array of each heuristic value calculated as per the given formulae.

In the aforementioned heuristics, all the constants are the weights used in this work. We can tweak these based on our

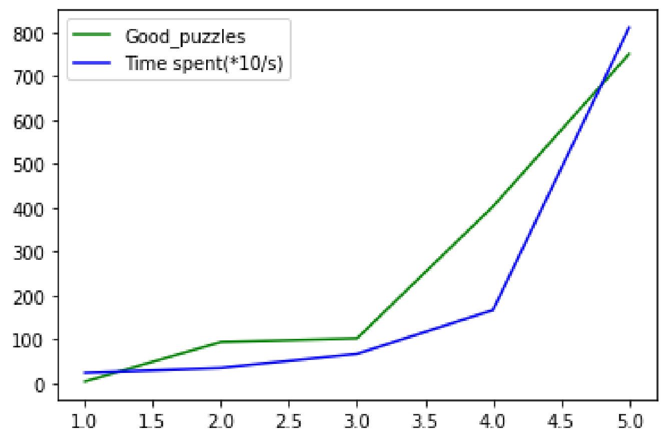


Fig. 3: Relation between depth and time spent

requirements to generate a different set of puzzles. A naive example would be increasing the weights for fork such that if there is any fork themed puzzle in the search state, it will have the highest heuristic value compared to other puzzles. This intentional bias can be used to generate different themed puzzles from a single start state. As for the initial setting of these heuristics, we normalized the weights by running them over existing database puzzles [11].

Current Heuristic Weights:

- Sacrifice: 1
- Pin Constant: 3.57
- Fork Constant: 5.88
- Material Disadvantage: 1

We find Sacrifice and material disadvantage to be fairly balanced and hence did not assign weights to them. Fine-tuning these weights can result in changing the diversity of the puzzles generated.

For the values of the individual pieces. We follow the weights assigned by AlphaZero [3]. Pawn = 1, Knight = 3.05, Bishop = 3.33, Rook = 5.63, Queen = 9.5.

For the exploration part, we plan to use a search space tree, however, in reverse order. i.e. We will start with a random position such that one player is at an advantage and find the best "undo" moves within the depth of $2N$ (N moves each player) with the best classical evaluation [10]. Generally, $6 \leq N \leq 10$ for most puzzles. Every depth that we explore costs us an exponential increase in time - thus it is a necessary trade-off between computation time and number of moves in the puzzle. Using these search states, we can generate a set of possible legal lines (set of moves), which when played in order, makes a puzzle. Once we get a set of such lines, we can trim out the irrelevant moves and filter out the good puzzles using the puzzle objectives mentioned above.

Thus, to summarize the process:

- Begin with an agent that knows the rules of chess, but with additional rules: Player can "un-capture" an opponent piece that is not on the board*, and a pawn can move backward.
- Start with a random position** such that one color has a better classical evaluation according to stockfish
- Generate all possible states constrained by the number of moves and evaluate (filter) them using the heuristics
- With the filtered puzzles, optimize the weights for each heuristic allowing the agent to exploit the knowledge of the previous positions

* In a general chess engine, this will explode the search space. However, these puzzles are limited to only a few moves allowing to explore such states.

** A random position allows to explore a wide variety of puzzle themes. If we want "mate in X" puzzles, we start with a position where one color is already in a checkmate.

Additionally, we can make use of the puzzle databases [12] [13] [14] to search for interesting heuristics that can be further used to optimize the puzzle generation.

B. *Filtering*

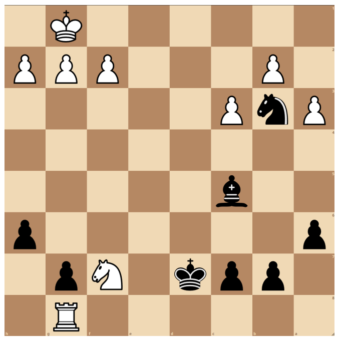
By backtracking for each depth, we have an average of 30 moves per backtrack, i.e. the branching factor is of 30. Although we trim out the moves that do not lead to a puzzle at every depth - a move that leads to another end state than what we start with, we're still left with more than half of the generated moves. This leads to an explosion in the search states and consumes a lot of computational time. Most of the bad puzzles can be detected in the early stages and We can minimize them by filtering them out.

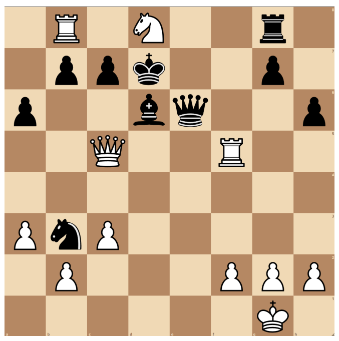
However, even this leads to way too many puzzles - most of them being redundant uncaptures. Our goal is to generate a set of different themed puzzles with high diversity in the type of moves. To achieve this, we identify the types of puzzles from the generated set of puzzles and select top-n using heuristic values and stockfish evaluations.

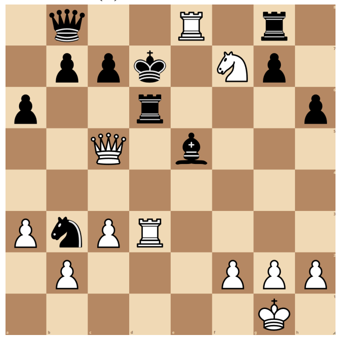
C. *Evaluation - Survey*

A major challenge for evaluating problems involving art is evaluation. Art is subjective in nature and deciding whether it is good or not is quite a task to represent in terms of numeric values. To tackle this, we use the approach similar to the Turing Test. We take a survey with the help of chess players [15] and take their feedback regarding the comparison between the puzzle generated by our algorithm against existing puzzles. The goal of the survey is to determine whether humans can identify the difference between an existing database puzzle against the puzzles generated by our algorithm.

For the purpose of the evaluation, we generate a survey that consists of 10 comparisons of different combinations between our generated puzzles, existing lichess puzzles and puzzles from Chesthetica [16]. We allow the users to select which puzzles they like more. Additionally, if they like both the puzzles - or dislike, they can choose to provide their response accordingly. We also collect the overall feedback on whether they could see the difference between the puzzles - or if they would like to see something differently.


(a) Initial State


(b) Puzzle 1


(c) Puzzle 2

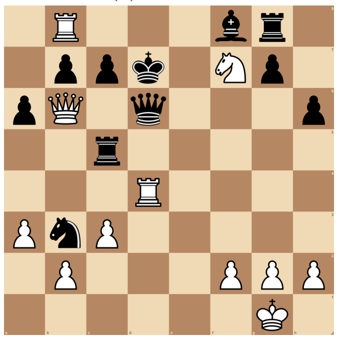
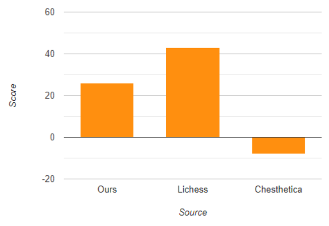
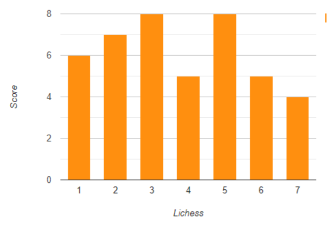

(d) Puzzle 3

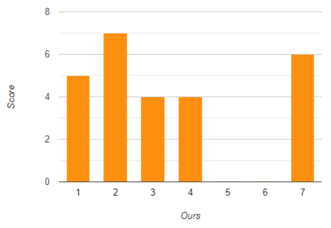
Fig. 4: Top 3 Filtered Puzzles with Initial State (a)



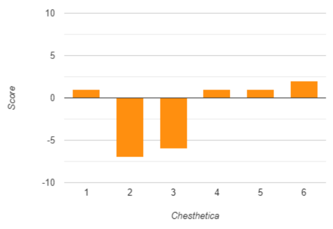
(a) Overall comparison



(b) Puzzle 1



(c) Puzzle 2



(d) Puzzle 3

Fig. 5: Puzzle Source Comparison based on Survey

V. Survey Results

We see the following results:

- **Overall Scores:** We value positive, negative and neutral to each puzzle based on the user responses. Using these, we aggregate the overall scores by summing them (including negative scores). This is shown in 5a We find that although we perform better than Chesthetica, there is a room for improvement in comparison with existing lichess puzzles.

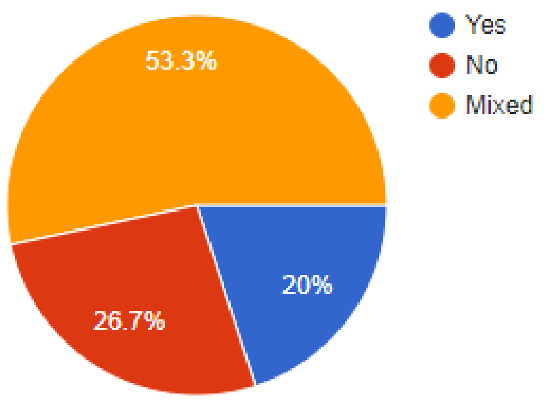


Fig. 6: Did the users feel our puzzles were AI generated?

- **Individual Scores** When we dive further into individual puzzles, Lichess puzzles were consistently voted high. We see that a few of our generated puzzle performed neutral leading to drop in overall comparison. However, we also find that few puzzles from Chesthetica received heavily negative feedback. These scores are out of 15 - although not all users choose to respond to all the questions. We see these results under 5b 5c 5d.
- **AI or not?** At the end of the survey, we ask them if they felt like they were solving AI generated puzzles. As seen in 6, we find that very less users were certain on a ”Yes”, although a lot of users had mixed feelings.
- **Open-ended feedback:** Since the users did not know which puzzles where ours, lichess or chesthetica, this section is based on intuition. A major feedback we received was regarding the sparsity in the positions, which is the trait seen in Chesthetica. Another feedback included a clear advantage in the winning position - another limitation that we discuss about Chesthetica. For overall AI based puzzles, a few users were able to detect the puzzles were artificial and not from real games.

VI. Limitations

A. *Redundancy problem*

A major issue that lies in the heart of the backtrack method is uncapture moves. Since we can uncapture every piece that is not on the board on every move, we end up having multiple moves that mean the same, but with a different piece on the board - which doesn’t change the dynamic of the puzzle; i.e. Having a queen instead of bishop in positions where there is no advantage in using one over the other. Here, our algorithm will process both the moves that leads to redundancy in the further depths.

Another redundancy issue that we suffer from is having a piece at different positions, but on the same line of it's movement. For example, a rook in h file can be anywhere between *h1* to *h8*. While some positions may lead to a different outcome, most of the positions will end up suggesting the same move if the rook is to be moved in the same file. This would be an issue to further address to reduce the redundancy.

With a decrease in redundancy, we can see a direct increase in the performance and allows us to explore deeper into the search space.

B. Limiting early search on heuristic value

Due to the high branching factor, each puzzle on the lower depths of the search space leads to an explosion of searches further down the line. To overcome this, we filter out puzzles that contain no heuristic value. This means that every puzzle must have an aesthetic importance in the initial depths - these make for the last moves of the puzzles. This restricts to explore all the potential puzzles. Improvements can be done around this filtration by adding dynamic thresholds.

VII. Conclusion

Through this project, we attempt to shed some light on the overshadowed domain of Chess and the use of computers in creating chess puzzles. High computation and immense studies over the Aesthetics of chess puzzles allows us to build an agent that can leverage the resources to generate great puzzles. We proposed a novel approach of generating backtrack moves to explore the search state in a reverse order. We incorporate the use of chess engine - Stockfish, and aesthetics - heuristics to find the potential puzzles that are fun to solve. We look at different generated examples of the puzzles and attempt to find improvise them. To evaluate our algorithm, we take a survey filled by chess players that aims to answer the question of whether these puzzles are differentiable from existing puzzles. In the survey, we also include the puzzles generated by the current SOTA technique - Chesthetica. Based on the results of the survey, we find that we make great progress over Chesthetica in terms of generating fun puzzles. However, lichess is still preferred. We also find the variance of the generated puzzles by changing the heuristic weights. This can allow us to have control over the category of puzzles generated while maintaining the element of aesthetics. These puzzles can be analyzed to further explore the beauty of chess and unveil some interesting ideas.

Similar approaches can be applied to other games like Go, Checkers, Shogi, etc.

VIII. Future Work

For our puzzle generation, we are utilizing four heuristics: sacrifice, pin, fork, material disadvantage. We could further explore other heuristics like exploiting weakest piece, moving longest distance, under-promoting, specific puzzles- en passant, smothered mate. This would cause a considerable increase in the diversity of puzzles created and enables us to plug-and-generate puzzles on the go without changing the overall structure of the algorithm.

We find that multiple generated puzzles differ in moves by a measure of distance travelled by a piece in a particular move - this is redundant and increases the branching factor. For example, in 4b refer to example figure a puzzle, a Rook moving from *f5* to *f7* is essentially the same as a puzzle that starts with the Rook moving from *f3* to *f7*. While our end-filter trims out most of such scenarios using stockfish and heuristic evaluations, we still find a lot of redundancy. Moreover, not being able to deal with these in the early depths leads to an explosion in branching factor and we spend a lot of computing time on the redundant puzzles. Another way to optimize this filter with experimentation to eliminate redundant states would be a great improvement to the performance and final filteration.

Our generated puzzles tend to heavily favor uncapture moves while backtracking, which is justified considering a move where you capture something instead of moving to that square will be more valuable in numerical sense based on our evaluations. Moreover, for every legal move - with some exceptions, there exist N uncapture moves leading to a bias in the number of uncapture moves. To address this issue, we experiment by introducing weights to heuristics in an attempt to suppress this behavior. While we see improvements in the nature of puzzle moves, the inclination towards uncapture moves was still dominant. There is scope for further work by experimenting around with these weights, especially for key heuristics like Sacrifice which are directly related to capture by penalizing the capture moves.

After making these updates, we can take another detailed review for individual puzzles. This will allow us to evaluate our algorithm further and create solid results.

1) weigh other themes more - or at least equally?
2) add some randomness? Randomly abandon uncapture possibilities for some moves

References

[1] M. Campbell, A. Hoane, and F. hsiung Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001291

[2] H. Editors, "Deep blue defeats garry kasparov in chess match," 1997. [Online]. Available: https://www.history.com/this-day-in-history/deep-blue-defeats-garry-kasparov-in-chess-match

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[4] MikeKlein, "Google's alphazero destroys stockfish in 100-game match," 2017. [Online]. Available: https://www.chess.com/news/view/google-s-alphazero-destroys-stockfish-in-100-game-match

[5] A. Iqbal, "A systematic and discrete view of aesthetics in chess," *Journal of Comparative Literature and Aesthetics*, vol. XXIX, pp. 53–65, 01 2006.

[6] A. Iqbal and M. Yaacob, "Computational aesthetics and chess as an art form," *Journal of Comparative Literature and Aesthetics*, vol. XXVIII, pp. 49–59, 01 2005.

[7] Y. M. Iqbal, Azlan, "Advanced computer recognition of aesthetics in the game of chess," *WSEAS Transactions on Computers*, vol. 7, pp. 497–510, 05 2008.

[8] A. Iqbal, "A discrete computational aesthetics model for a zero-sum perfect information game," Ph.D. dissertation, 09 2008.

[9] A. Iqbal, M. Guid, S. Colton, J. Krivec, S. Azman, and B. Haghighi, “The digital synaptic neural substrate: A new approach to computational creativity,” 2016.

[10] “Stockfish.” [Online]. Available: https://github.com/official-stockfish/Stockfish

[11] “Chess puzzles by gms - historic and modern games.” [Online]. Available: https://wtharvey.com/

[12] “Lichess puzzle database.” [Online]. Available: https://database.lichess.org/

[13] “Yet another chess problem database.” [Online]. Available: https://www.yacpdb.org/

[14] “Chess blunders - endless database of chess puzzles.” [Online]. Available: https://chessblunders.org/

[15] “Survey: Ai-chess puzzle generation.” [Online]. Available: https://forms.gle/BoW9VoPokV4b6Cgq5

[16] “Chesthetica youtube channel.” [Online]. Available: https://www.youtube.com/channel/UCvub4ug2bM9nlx7VbF5nWhw