Estimating Chess Elo with Bayesian Inference and Entropy-Minimization Puzzle Selection

Itai Reingold-Nutman

November 2024

Abstract

This paper describes the details and development of the project PuzzleRating, which estimates chess Elo ratings based on performance in chess puzzles. The Elo estimation is updated via Bayesian Inference after each puzzle, and puzzle selection is inspired by entropy-minimization.

1 Introduction

Motivating Question All chess players are familiar with the Elo System[1]. The conventional way to obtain an Elo rating is to play games; the more you win, the more your rating grows; the more you lose, the more your rating decreases. However, a curious question arises: Can chess puzzles also be used to offer specific insight into a player's skill and Elo strength? If so, the applications could be powerful.

Namely, it would allow coaches to assign puzzles to their students on key specific themes they wish to test for (like the tactic of a "fork", or the positional concept of "blockade"). This fun puzzle-solving exercise could generate Elo ratings for each theme, revealing the student's strengths and weaknesses to the coach. In this project, I demonstrate a proof of concept for this "categorization" of chess puzzles; I group puzzles into three themes (openings, middlegames, and endgames), and in the end provide a general Elo prediction, along with a prediction for each of those themes, highlighting the stage of the game the user is strongest and weakest.

2 Implementation

Data For the data, 51,210 puzzles are taken from the lichess.org database. These 51,210 puzzles each contain an FEN string representation of the starting position, the rating of the puzzle, a string representation of the solution, and a theme for the puzzle. The theme is used to classify the puzzles into openings, middlegames, and endgames for the "categorization" reason explained in the introduction.

Bayesian Inference The core of the problem relies on Bayesian Inference. After each puzzle, we update our Elo prediction for the user based on this new information. To be more specific, all users begin with the initial distribution in Figure 1, based on the distribution of all players on lichess.org (Figure 2). In other words, without knowing any information about the user, we assume that their Elo can be represented by the general player distribution. Then, we begin to learn new information, in the form of their puzzle performance. Based on this new information, we wish to update every possible Elo rating, which we can do via Bayesian Inference:

$$P(\text{Elo} \mid \text{Puzzle Performance}) = \frac{P(\text{Puzzle Performance} \mid \text{Elo}) \cdot P(\text{Elo})}{P(\text{Puzzle Performance})}$$

[1]In the Elo rating system, a number is assigned to each player, which allows for the comparison of relative chess strength. In this project, all Elo ratings are based on the lichess.org site. For reference, the best players on lichess.org have an Elo rating of roughly 3000, while the average player on lichess.org is roughly rated 1500.

We can find the $P(\text{Elo})$ by our current prior distribution. We can find the denominator term through normalization. But how can we find $P(\text{Puzzle Performance} \mid \text{Elo})$? This is where I make a natural assumption, that the probability that a player rated A solves a puzzle rated B is the same as the probability that a player rated A beats a player rated B in a chess game. This assumption is explained in detail in Appendix A, and allows me to continuously – after each puzzle – apply Bayesian inference across all Elo ratings to obtain an updated belief distribution in the user's Elo rating. We do this until the user chooses to stop, and then we present the entire distribution, along with the distribution's expectation, which represents their most likely rating.

3 Simulations: Puzzle Selection

An interesting question in the project was deciding which puzzle to present to the user next. Eventually, all methods of selecting puzzles lead to accurate results, but the challenge is minimizing the time until we reach those results. After all, we don't want the user to spend unnecessary time solving puzzles. My initial intuition is that an Entropy-based approach could get accurate results most quickly, as it minimizes uncertainty and achieves narrower distributions. I had a lot of fun testing this hypothesis, and comparing various potential methods. Simulations for these methods are presented below:

The Baseline The first method of selecting puzzles is random choice. This can be considered the "baseline" method as it offers a benchmark by which I can test the more sophisticated and ambitious options.

See Figure 3 in the Appendix for the baseline simulation[2] results.

Intuitively, there are reasons to suspect random sampling might not be the best: For instance, imagine a user is believed to be rated 700, and the algorithm randomly gives it a 2500 puzzle. The vast likelihood is that the user will fail the puzzle, but this reveals almost no information, as solving the puzzle would be extremely unlikely, even for a 1700-rated player.[3] The point is that random sampling would eventually lead to a confident distribution, but it could take far longer than a more targeted approach. It would be wonderful to present a puzzle whose result would maximize how certain we are in our distribution. But how? Entropy!

Entropy-Based Selection Entropy is a measure of uncertainty for a random variable. For our purposes, the random variable, R, is the Elo rating of the user. We are unsure exactly what the rating is, but based on their puzzle history, we can develop a belief distribution for the values that R can take on and the probability for each of those values. Ideally, we wish to present the user with a final distribution that is narrow, so they can feel confident in their Elo rating. To fulfill this goal we minimize Entropy.

Namely, for any distribution R, the entropy of R is given by:

$$H(R) = -\sum_{r \in \mathcal{R}} P(r) \log P(r)$$

Lower entropy indicates higher certainty about the user's rating. Conversely, higher entropy corresponds to greater uncertainty.

How does this algorithm work? For each puzzle, I consider the current distribution R. Then, I cycle through the rating of every possible puzzle. For each, I update R and produce two belief distributions: one for the user solving the puzzle, and one representing the user failing the puzzle (found via Bayesian Inference). I calculate the entropy for both distributions and weigh them by how likely they are to occur (based on the likelihood assumption from Appendix A) to obtain the expected entropy. I select the puzzle with the smallest expected entropy. I should note that this method is not being used in its purest form. Instead, while I do heavily prioritize minimizing entropy, I add some randomness in the

[2] A note about simulation methodology: For each simulation, I perform multiple "runs", where each consists of sampling a random Elo rating from the distribution in Figure 1 and attempting x puzzles. For each puzzle, the probability of success is given by the likelihood assumption explained in Appendix A. Then, for each x, I calculate the median error across all runs. The median is used to reduce the impact of outliers, which are fairly common in these simulations. Note that each median calculation (for each x) is always based on at least 100 data points.

[3] A secondary consideration, from the perspective of the solver, is that the user may get frustrated by overly difficult puzzles and bored by overly simple puzzles.

selection process so users can play many times with different puzzles. Indeed, as expected, Figure 4 shows that this Entropy-based approach yields accurate results far faster than random selection. For instance, obtaining a median error smaller than 40 Elo points requires 100 random puzzles but only 40 Entropy-selected puzzles.

Other Selection Methods I played around with other selection methods to see if we could achieve better results. In one simulation, I tried a "Greatest Change" approach, where I chose the puzzle that is expected to change the expectation the most. The intuition is that this would make the most progress quickly. As can be seen in Figure 5, though, this was not an improvement from the baseline. A second simulation, testing a "Plus-Minus" approach rotated between giving the users a puzzle rated +X and -X compared to their current expectation. The idea is that this gives the user ample opportunity to increase and decrease their rating. This method was better than the baseline (see Figure 6). In fact, as Figure 7 shows, as X decreases, this method seems to approach the accuracy of the Entropy-based approach. From this, it seemed pertinent to see what happens if X is 0. That is an "Expected" method, that simply gives the user a puzzle rated their current expectation. As can be seen in Figure 8, this in fact did as well as entropy-based selection, especially after some initial volatility. As it turns out, by puzzle 20, the rating that minimizes entropy tends to be the expectation (see Figure 9), which explains the lack of difference between the two selection methods.

Therefore, for our current implementation, instead of calculating the rating that minimizes entropy, which is computationally expensive, we use the user's Elo expectation to approximate this rating. As an interesting aside, the computationally intensive Entropy-based approach required grouping Elo ratings into buckets of 25 to reduce user lag. Thus, I was pleased that the boost in computational efficiency from expectation-driven selection allowed me to reduce the bucket size to 1. I assumed this would lead to more accurate results given the increase in precision. However, the surprising simulation in Figure 10 showed that while there is a very significant increase in accuracy for changes in large bucket sizes, the difference for small bucket sizes (1 - 100) seems fairly negligible. Nonetheless, the current implementation remains with individual Elo ratings (bucket size 1) for good measure!

4 Conclusion

I thoroughly enjoyed this project. It illustrated that puzzles can be used to calculate Elo ratings, which offers coaches a new tool to gain knowledge about their students and pinpoint strengths and weaknesses. In the same way that my current implementation can compare Opening, Middlegame, and Endgame strength, this can easily be extended to account for various other themes and aspects of chess. It is true that with more themes, more puzzles need to be solved. But, for example, if a coach/student wished to compare 5 specific themes in chess, and we allow for a median error of 75 points, it would take 60 puzzles. This could be done in 1 hour of work and will leave the coach with a great sense of the material to work on next!

Moving Forward This project showed signs of initial success in implementing Bayesian Inference and Entropy to calculate Elo ratings by puzzle performances. The next steps could involve research in various interesting avenues. Firstly, experiments should evaluate the accuracy of the current implementation for real chess players, as opposed to the simulations presented here. Further tests could address and perhaps revise the likelihood assumption made (see Appendix A). Another area of research can improve the puzzle Elo rating assignment. For instance, perhaps puzzles should have a time limit. Puzzles with longer time limits should be harder (for the same rating) than those with shorter time limits. Added functionality could allow students/coaches to create more thorough diagnostics, with more control over themes. Additionally, combining more information, like the user's game or puzzle history could create a player profile that leads to more accurate and personalized Elo predictions. With additional work, this method for determining Elo ratings, strengths, and weaknesses could seriously change and improve how players develop effective study plans.

Appendix

A The Likelihood Assumption

As it turns out, there is a handy formula in chess for calculating the probabilistic result for a match. If player 1 has Elo X, and player 2 has Elo Y, then the probability that player 1 will beat player 2 is given by:[4]

$$\frac{1}{10^{\frac{Y-X}{400}}+1}$$

The assumption I make is that the probability that a player rated X solves a puzzle rated Y is the same as the probability that a player rated X beats a player rated Y in a game.

Is this assumption entirely accurate? Perhaps, though there could be several reasons why it may be easier or harder to solve the puzzle rather than win the game. To that end, future research could explore this assumption in more depth and possibly refine the formula.

An encouraging piece of information is that lichess.org, and likely other websites, make the same assumption for their own algorithmic needs. After completing my project, I spoke to a lichess.org developer about these topics. Unlike my project, "Lichess doesn't predict elo from puzzles." Instead, users have a real and separate puzzle rating. "Doing a puzzle is treated like playing a game against it and the elo is adjusted as usual." Similarly, for my implementation, doing a puzzle is treated like playing a game against it, and the same probability calculation is used. In other words, both lichess.org and my implementation make the same central assumption, though we do so for different reasons with different goals in mind.

B Experimental Graphs

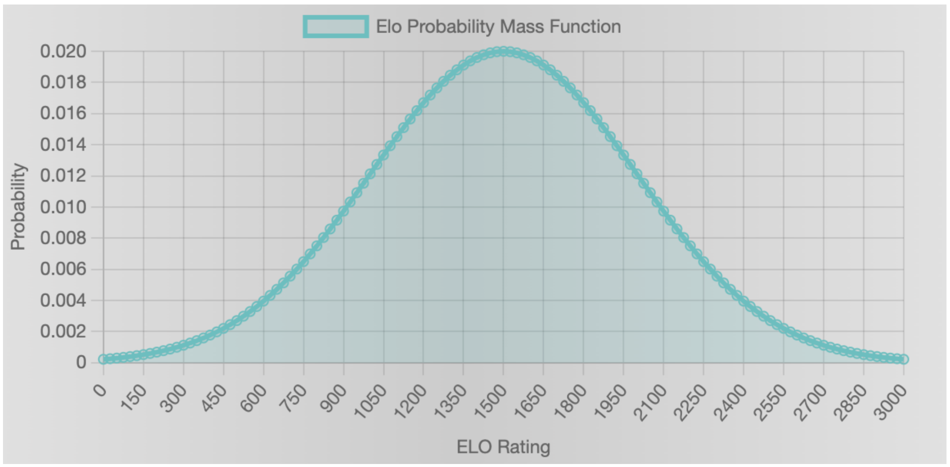

Figure 1: The initial distribution used for the prior. A Gaussian centered at 1500 with a standard deviation of 500, resembling Figure 2.

[4]This is a widely accepted formula, used by FIDE, and mentioned in various websites. I should note that the formula technically refers to a game with three possible outcomes, and as such represents an "expected score" equal to the probability of winning + 1/2 probability of drawing. Since, for my implementation, there are no draws, I use this formula as an approximation for the probability of winning, which is how the formula is often interpreted and used (see bottom note in this website – the website table is derived from the formula). But, my implementation would work with any formula for "win" probability, which could be a source of future research and refinement.

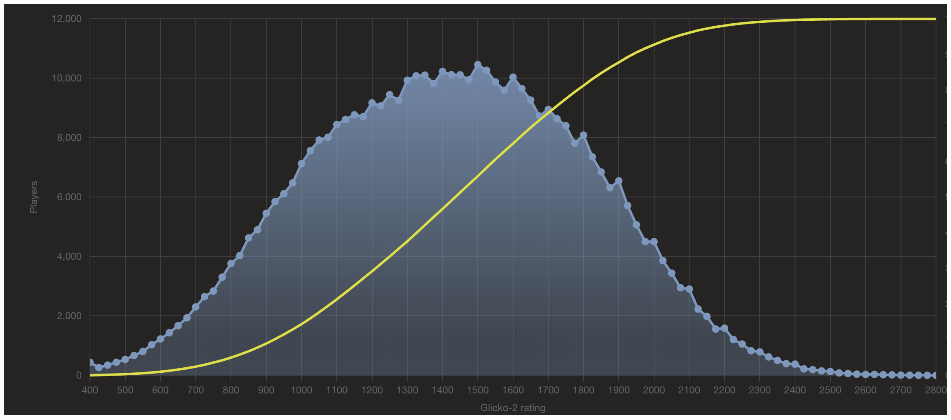

Figure 2: The rating distribution of lichess.org players (Specifically, rapid rating for the week of November 25th).

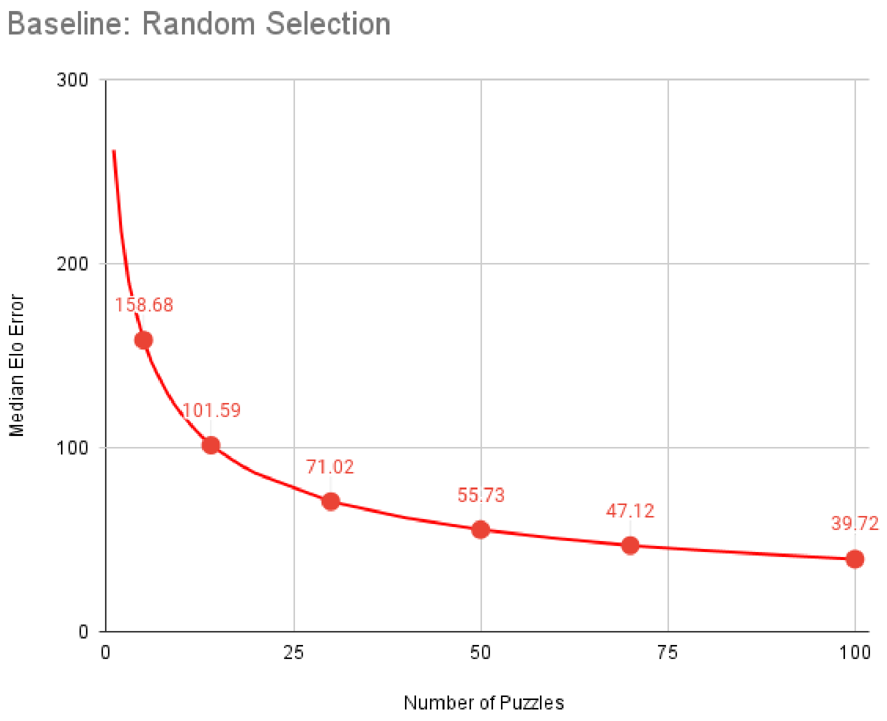

Figure 3: Baseline results for random puzzle selection (red).

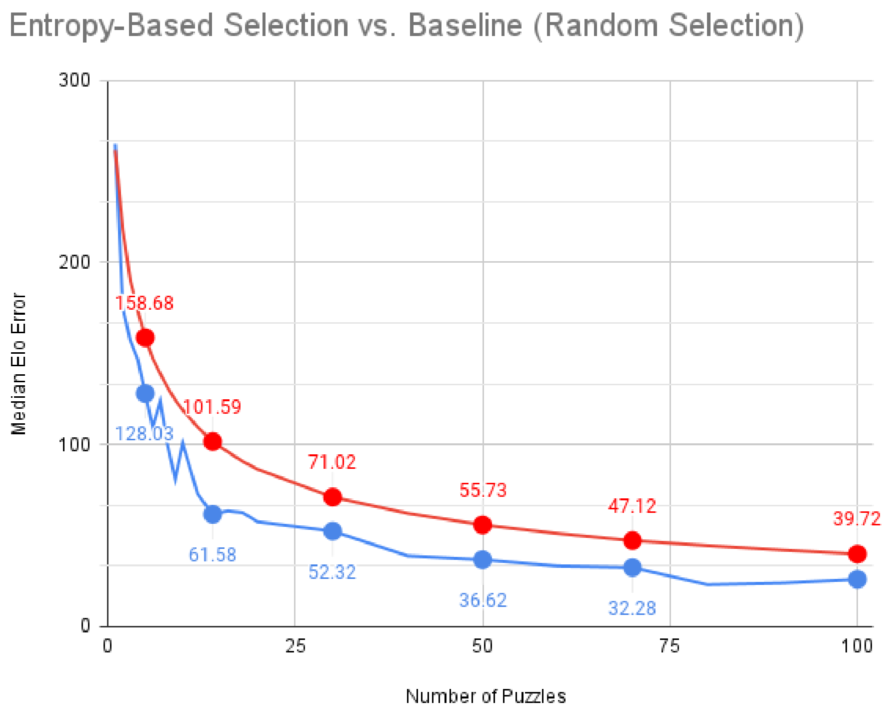

Figure 4: Comparison of entropy-based puzzle selection (blue) to random selection baseline (red).

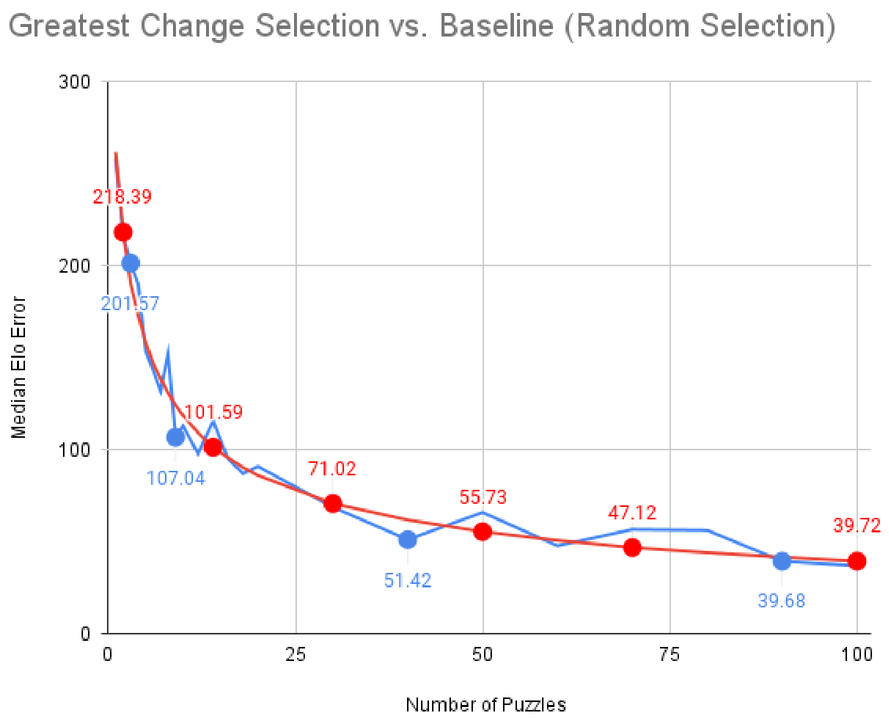

Figure 5: Comparison of greatest change puzzle selection (blue) to random selection baseline (red).

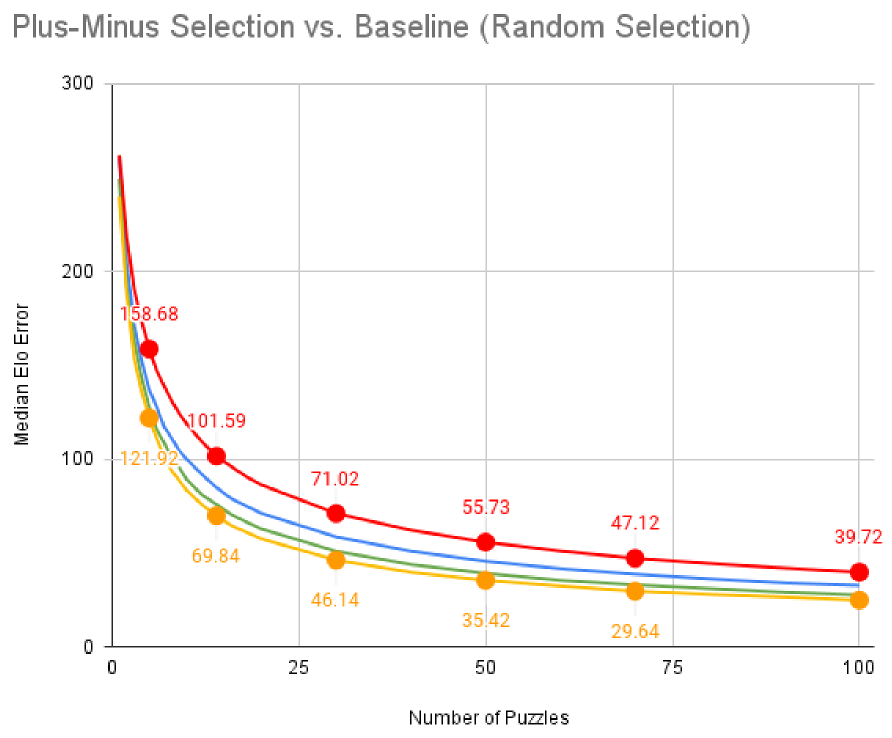

Figure 6: Comparison of random selection baseline (red) to different distances (X) for Plus-Minus method: X = 300 (blue), X = 200 (green), X = 100 (yellow).

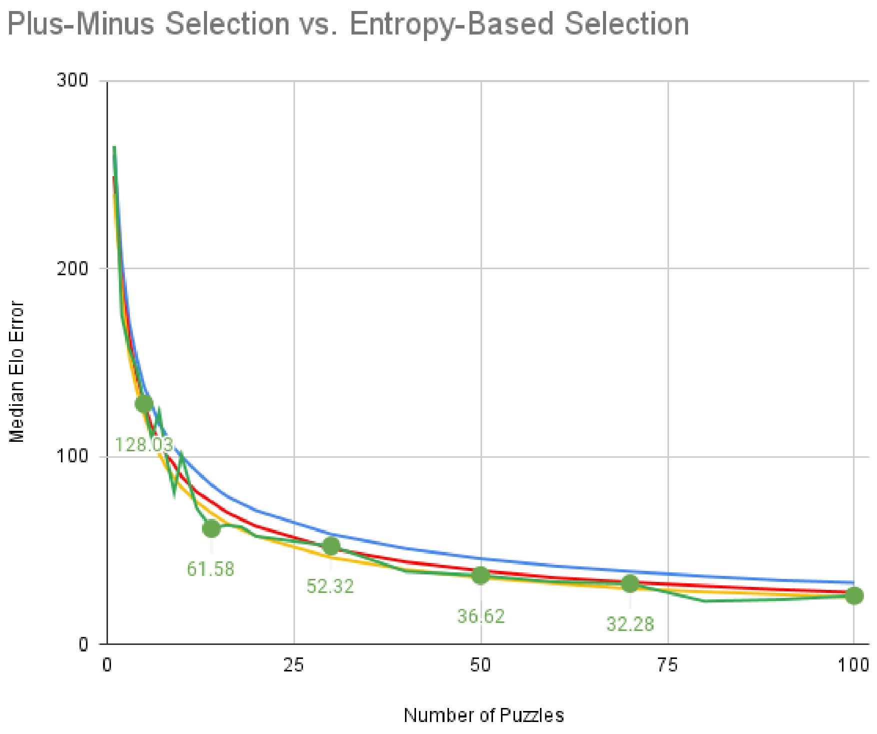

Figure 7: Comparison of entropy-based selection (green) to different distances (X) for Plus-Minus method: X = 300 (blue), X = 200 (red), X = 100 (yellow).

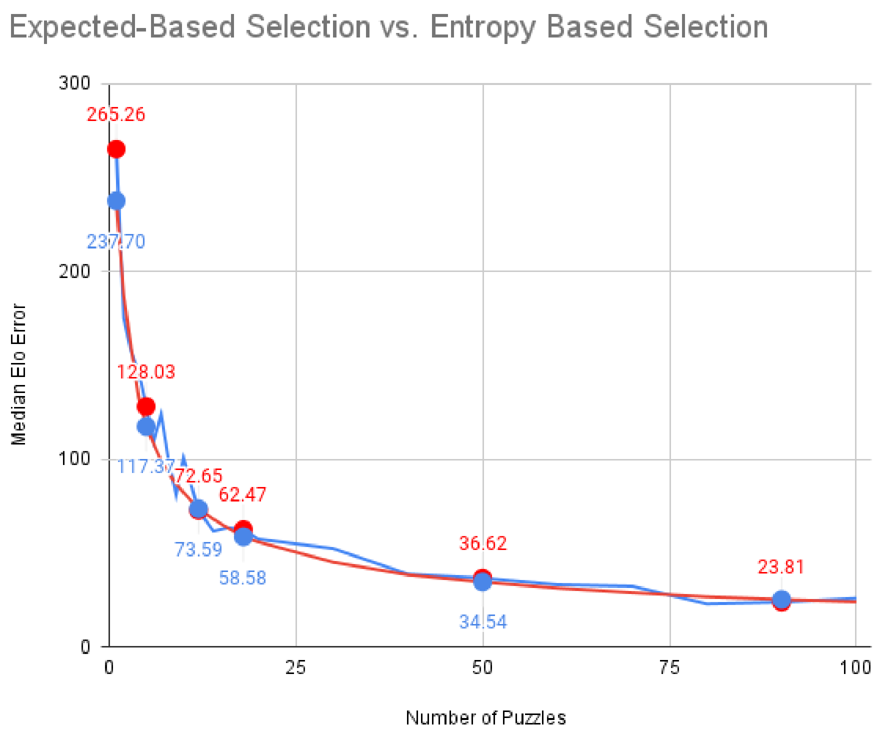

Figure 8: Comparison of entropy-based puzzle selection (blue) to expectation-driven puzzle selection (red).

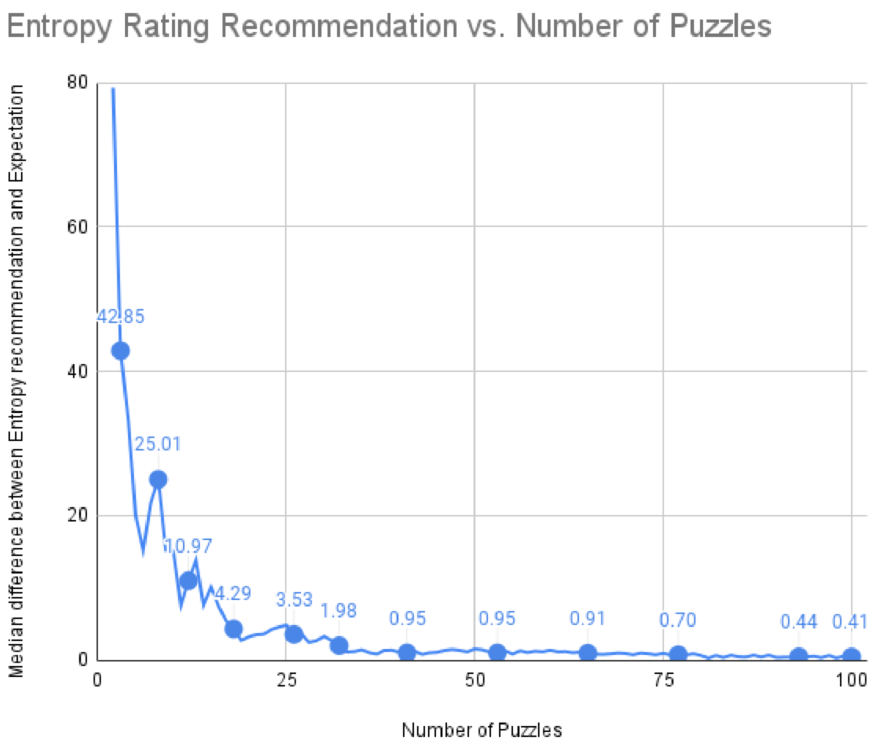

Figure 9: The median difference between the rating recommendation given by Entropy-based puzzle selection, to the expectation of the current distribution.

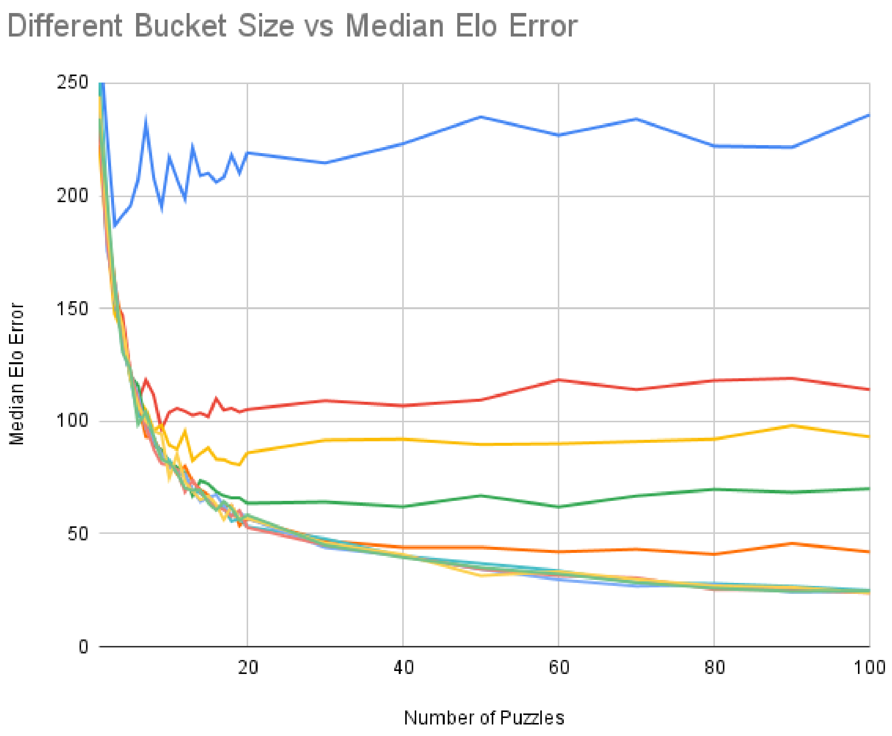

Figure 10: Comparison of bucket sizes. Distinguishable, and in order from top to bottom are sizes 1000 (blue), 500 (red), 400 (yellow), 300 (green), 200 (orange). Indistinguishable/negligible are the mixed bottom graphs representing sizes = 100, 50, 25, 10, and 1.