Highlights

Re-evaluating Metamorphic Testing of Chess Engines: A Replication Study

Axel Martin, Djamel Eddine Khelladi, Théo Matricon, Mathieu Acher

- Our work reproduced the original study and also replicated it while varying three factors, namely: 1) the depth value; 2) the dataset through the inclusion of realistic positions; and 3) the version of Stockfish.
- The metamorphic relations are not as effective as in the original article, especially on realistic chess positions and increase of depth.
- We raise awareness of the sensitivity of depth: metamorphic relations may only be violated at specific depths, and there is a depth threshold beyond which the testing method becomes ineffective.
- Through a rigorous and in-depth analysis of the source code, we found why Stockfish can exhibit discrepancies on transformed positions and why at certain low depths, metamorphic relations are not effective. Our overall conclusion is that it is not a bug, but a feature of the exploration process of modern chess engines.

Re-evaluating Metamorphic Testing of Chess Engines: A Replication Study

Axel Martin[a], Djamel Eddine Khelladi[a], Théo Matricon[b], Mathieu Acher[a,c]

[a] *Univ Rennes, CNRS, Inria, IRISA, Rennes, France*
[b] *University of Bordeaux, CNRS, LaBRI, Bordeaux, France*
[c] *Institut Universitaire de France (IUF), Rennes, France*

Abstract

Context: This study aims to confirm, replicate and extend the findings of a previous article entitled *"Metamorphic Testing of Chess Engines"* that reported inconsistencies in the analyses provided by *Stockfish*, the most widely used chess engine, for transformed chess positions that are fundamentally identical. Initial findings, under conditions strictly identical to those of the original study, corroborate the reported inconsistencies.
Objective: However, the original article considers a specific dataset (including randomly generated chess positions, end-games, or checkmate problems) and very low analysis depth (10 plies[1], corresponding to 5 moves). These decisions pose threats that limit generalizability of the results, but also their practical usefulness both for chess players and maintainers of Stockfish. Thus, we replicate the original study.
Methods: We consider this time (1) positions derived from actual chess games, (2) analyses at appropriate and larger depths, and (3) different versions of Stockfish. We conduct novel experiments on thousands of positions, employing significantly deeper searches.
Results: The replication results show that the Stockfish chess engines demonstrate significantly greater consistency in its evaluations. The metamorphic relations are not as effective as in the original article, especially on realistic chess positions. We also demonstrate that, for any given position, there exists a depth threshold beyond which further increases in depth do not result in any evaluation differences for the studied metamorphic relations.

[1] A ply refers to a single turn taken by one player in a game. Two plies, one from each player, together constitute a complete move.

We perform an in-depth analysis to identify and clarify the implementation reasons behind Stockfish's inconsistencies when dealing with transformed positions.
Conclusion: A first concrete result is thus that metamorphic testing of chess engines is not yet an effective technique for finding faults of Stockfish. Another result is the lessons learned through this replication effort: metamorphic relations must be verified in the context of the domain's specificities; without such contextual validation, they may lead to misleading or irrelevant conclusions; changes in parameters and input dataset can drastically alter the effectiveness of a testing method.

Keywords: Reproducibility, Replicability, Metamorphic Testing, Chess Engines

1. Introduction

A test oracle determines whether a test execution reveals a fault, by typically comparing the observed program output to the expected output [4]. This is not always practical, for example when a program's input-output relation is complex and difficult to capture formally [23]. Chess engines enter in this category and are an excellent instance of such programs. Given a chess position as input, what should be the evaluation score (the output) that is supposed to quantify the advantage of white or black pieces? The ground truth is most of the time not known: chess is far from being resolved while top chess players are unable to formulate an accurate assessment and struggle to find the best moves. In practice, chess players are largely inferior and rely on chess engines to get hopefully reliable evaluations – no need to say they cannot play the role of testers that know the expected result.

Metamorphic testing [28, 5] has been useful in addressing the oracle problem, which arises when testers lack a specification to decide if the observed outputs are expected. Instead of deciding on individual outputs, metamorphic analyzes a set of inputs and outputs to check for violations of certain relations, called metamorphic relations, and has proven to be versatile in testing complex systems across various fields. Metamorphic testing is thus an appealing candidate to test chess engines. The article entitled *"Metamorphic Testing of Chess Engines"*, published at IST journal in 2023 [22], proposes a metamorphic testing approach to test chess engines. The key underlying idea is to define metamorphic relations that state that the evaluation of two equivalent positions should be the same. For example, considering a position and rotating all the pieces with respect to the central axis, then both positions should have the same evaluation. Méndez et al. reported inconsistencies in the analyses provided by *Stockfish*, the most widely used chess engine, for transformed chess positions that are fundamentally identical.

This study aims to confirm, replicate and extend the findings of Méndez et al. [22]. Our motivation was that the original article considers a specific dataset (including randomly generated chess positions, end-games, or checkmate problems) and very low analysis depth (10 plies, corresponding to 5 moves). These decisions pose threats that limit generalizability of the results, but also their practical usefulness, both for users and maintainers of Stockfish. Based on our discussions and domain knowledge, we hypothesize that the specificities of the dataset and the low depth of the analysis may be open to question. There is also a need to understand why Stockfish can exhibit discrepancies on transformed positions, something that was not addressed in the original article. In a nutshell, we aim to re-evaluate the effectiveness of metamorphic testing of chess engine. In general, there is this question whether metamorphic relations, though theoretically appealing and reasonable, are effective in practice for testing complex programs and reveal actual faults that would justify an effort to fix them.

Thus, we carefully design a replicability study considering this time (1) positions derived from actual chess games, (2) analyses at appropriate and larger depths, and (3) different versions of Stockfish. The replication results show that the Stockfish chess engines demonstrate significantly greater consistency in its evaluations. The metamorphic relations are not as effective as in the original article, especially on realistic chess positions. We also demonstrate that, for any given position, there exists a depth threshold beyond which further increases in depth do not result in any evaluation differences for the studied metamorphic relations. We further conduct an in-depth analysis to find and explain why Stockfish can exhibit discrepancies on transformed positions. The move ordering of legal moves during the search process is identified as a key factor that can lead to evaluation differences. This factor explains most of the discrepancies observed in our experiments. It also explains why, at a certain depth, the evaluation differences disappear.

The contributions of this paper are as follows:

- Under conditions strictly identical to those of the original study, we confirm the reported inconsistencies in the analysis provided by Stockfish for transformed chess positions that are fundamentally identical. Hence, despite non-significant differences, we were able to reproduce the original results.

- We design a replicability study and make explicit our hypothesis about chess positions of the dataset, Stockfish depth, and version based on the review of the original study as well as discussions, observations, and knowledge of the domains of chess and chess engine.

- We execute the replicability study and provide a detailed account of the results, systematically analyzing the sensitivity of the proposed metamorphic testing to the dataset, the depth of the analysis, and the version of Stockfish. Our findings reveal that the Stockfish chess engine demonstrates significantly greater consistency in its evaluations when operating on realistic chess positions and at higher depths.

- We find why Stockfish can exhibit discrepancies on transformed positions – it is a feature of the Stockfish implementation, not a bug. It also explains most of the discrepancies and why at certain depths, metamorphic relations are not effective.

Overall, metamorphic testing of chess engines has been shown to be less effective than expected. We provide such evidence, and explain the underlying reasons. The practical usefulness of the approach is thus currently limited, both for chess players and for maintainers of Stockfish.

2. Background

This section provides a generic understanding of chess engines and how they function, along with the metamorphic relations used in the original paper and related work.

2.1. Chess engines

Chess engines are software programs that analyze chess positions in the chess board and propose the best following moves. A board (and position) is represented as a single string in the standard Forsyth– Edwards Notation (FEN). The main goal of a chess engine analysis is to decide which player has an advantage and evaluate how big the advantage is. Using this analysis, they can decide on the next best moves to play. State-of-the-art chess engines are strong players that can consistently beat even the best human players.

Chess engines usually evaluate a given position with a value in centipawns (cp), which is a unit of measurement in chess engine analysis that quantify the advantages or disadvantages of the position. A positive score means that White has an advantage and a negative score means that Black has an advantage.

Definition 1. An *evaluation* $e \in \mathbb{Z} \cup \{(w, b) \times \mathbb{N}\}$ represents the advantage one player has over the other. If e is a positive number, then we say White has a lead over Black of e centipawns; if e is a negative number, then Black has a lead over White of $-e$ centipawns. If $e = 0$ then we say that the position is even. Finally, if $e = (w, x)$ (respectively $e = (b, x)$) we say that White (respectively Black) has checkmate in x moves. An evaluation is performed given a tuple (p, depth) where p is an FEN string with the board information and $\mathit{depth} \in \mathbb{N}$ is the analysis depth.

One of the most popular open source state-of-the-art chess engines is Stockfish [31]. It has consistently won most of the editions of the Top Chess Engine Championship (TCEC) [1]. It is the chosen subject in the original study by Mendez *et al.* [22]. Stockfish is a powerful open-source chess engine renowned for its strength and efficiency. It analyzes chess positions and makes closed to optimal moves [31, 1] – even though we ignore optimal moves in general and chess is not resolved. Stockfish's strength lies in its ability to assess positions deeply. Indeed, one of the most important values in chess engines is the depth. It is a value of chess engine analysis that indicates the number of half moves (i.e., a move made by one side only) the engine looks ahead. A higher depth value usually means better analysis results as it evaluates a larger number of possible moves. This also allows navigating complex tactical and strategic scenarios with good precision.

Figure 1 shows two chess boards evaluated by Stockfish of a game by Magnus Carlsen grand master at *depth = 20.* We will further explain their relationship in the next section. In Figure 1, Stockfish returns an evaluation of *+0.66* for the first board and of *-2.17* for the second board. It respectively interprets as a slight advantage for white and a decisive advantage for black.

2.2. Metamorphic testing of chess engines

This section presents the four metamorphic relations introduced by Méndez et al. [22] *et al.* to test the chess engines. As motivated by the original study, although the chess engines can be fantastic players, it is hard to ensure that their code is fault free because it is very difficult to test them.

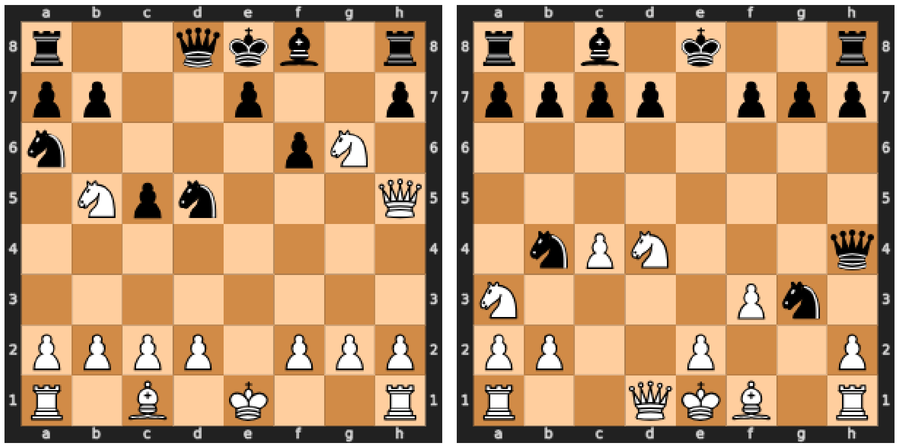


Figure 1: Original position and symmetry mirror. Evaluation of SF15 is +0.66 and -2.17 at depth=20

In particular, they face the oracle problem: if the chess engine plays better than a tester, how can the tester claims that a given evaluation of the next move is wrong or not the best one in general. Thus, Méndez et al. [22] *et al.* used metamorphic testing to test chess engines with the following metamorphic relations:

- *Equivalent* relation: It aims to detect situations where two equivalent positions have different evaluations by the chess engine. Herein, two symmetries are considered, namely:

  1. the symmetry on the axis where all pieces are rotated w.r.t. the central axis sim_axis, and
  2. the symmetry on the diagonal sim_diag where all pieces are rotated w.r.t. the right diagonal a1h8. Note that this later symmetry only applies if no pawns are in the chess board.

  The second and sixth boards in Figure 2 show an example of the two symmetries applied on the first and fifth boards. Therefore, two symmetric positions are equivalent if any move on one board can also be executed in its symmetric board.

- *Mirror* relation – sim_mirror: It also aims to detect situations where two equivalent positions have different evaluations by the chess engine. Herein, a third type of symmetry is applied, which is where black and white pieces are exchanged with a board rotation. The third board in Figure 2 shows an example of this symmetry applied to the first board. It is basically a mirroring of the position that is symmetric to another vertically, but with the different colors and with the opposite player to move. For example, a position with black/white to move is mirrored to a position with white/black to move.

- *Better* relation – better: This aims to assess the evaluation of positions where a piece is replaced by a better one. This aims to detect whether having a better position yields at least the same evaluation. For example, replacing a rook or a bishop by a queen can only improve the evaluation of the original evaluation, as the queen can perform more moves than the original pieces. The fourth board in Figure 2 shows an example of this relation applied to the first board.

- *First* relation – first: This aims to assess equivalent positions after the best suggested move by the chess engine. It aims to detect whether the evaluation of two related positions with the above symmetries is preserved after the best suggested move. In fact, the evaluation of a position gives the advantage of a player at a given position. After performing the best move suggested by Stockfish, such advantage should be mostly preserved.

In the remainder of this article, we consistently use specific terms like 'Equivalent relation,' 'Mirror relation,' 'Better relation,' and 'First relation.' These terms are intentionally chosen to align with the original study that forms the basis of our work. Although these terms might appear to occupy different hierarchical levels – where 'Mirror relation' indicates types of symmetry and 'Better relation' suggests a purposeful change in piece value – this choice of nomenclature is deliberate. Each term is selected to represent the unique characteristics of the chess positions they describe. 'Mirror relation' illustrates symmetry transformations, 'Better relation' highlights upgrades in piece value, and 'First relation' evaluates the outcomes of recommended moves. We adhere to this terminology to maintain consistency with the original study, ensuring clarity and ease of understanding for both seasoned experts and those new to chess engine testing.

In Figure 1, the metamorphic relation sim_mirror with a mirror symmetry is applied on the right board. One would assume a similar evaluation by Stockfish, but at *depth* = *20*, the chess engine gives different results. It increases the advantage from slight to decisive advantage (of the previously white and now black).

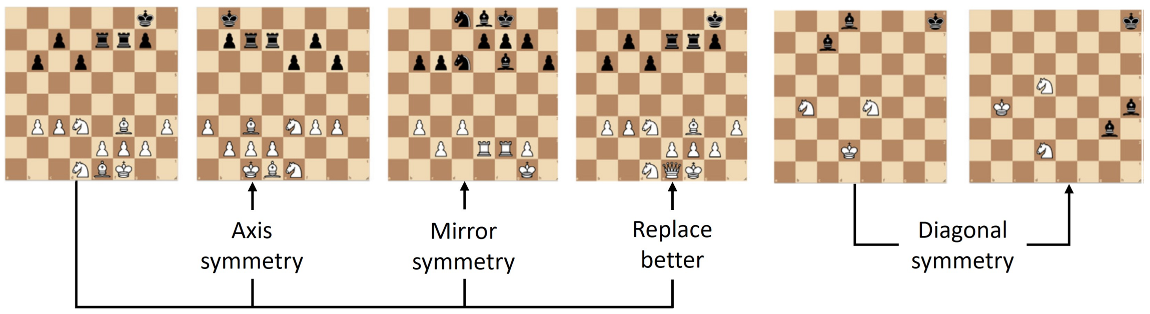


Figure 2: Example of the metamorphic relations.

Thresholds. A distinguishing feature with respect to usual metamorphic testing approaches is the use of thresholds that allow deviations from the expected result. Expecting the same centipawns was too strong. Specifically, the authors considered two thresholds associated with two measures of the deviation.

The first one allows small deviations between evaluations in *relative* terms while the second one, in case the first one fails, allows small deviations in *absolute* terms. Intuitively, there is not much difference between having an advantage of 1000 centipawns and an advantage of 1100 because in both cases the player has a clear advantage and will win the game. In contrast, having an advantage of 50 centipawns versus an advantage of 150 is a completely different situation: an advantage of 50 indicates that the player has a slight advantage while 150 pawns is a comfortable advantage for competent players. The best way to interpret these thresholds is twofold. On the one hand, the looser the threshold, the more critical the faults detected, as they generate a bigger difference between the analysis of positions that should be equivalent. On the other hand, the tighter the threshold, the less critical the faults detected, as they produce smaller differences in the analysis [22].

Definition 2 (Relative and Absolute Distance)**.** Let $x, y \in \mathbb{Z}$ and $d \in \mathbb{R}$. We define the following predicates:

- $\text{dist}_{\text{rel}}(x, y, d)$ holds iff $\frac{|x-y|}{|x|+|y|} \leq d$.
- $\text{dist}_{\text{abs}}(x, y, d)$ holds iff $|x - y| \leq d$.

It should be noted that

$$\max_{x,y\in\mathbb{Z}}\left\{\frac{|x-y|}{|x|+|y|}\right\}=1$$

and this value is reached only if x and y have opposite signs.

Definition 3 (Metamorphic relation)**.** A *metamorphic relation* for chess engines computing at depth *depth* is a relation

$$R(p_1, p_2, e_1, e_2, \delta, \varepsilon)$$

over:

- two inputs (positions p_1 and p_2), p_1 being the original position and p_2 being the transformed position,
- their corresponding outputs (evaluations $e_1 = \text{StockFish}(p_1, \textit{depth})$ and $e_2 = \text{StockFish}(p_2, \textit{depth})$),
- and two thresholds δ and ε to parameterize the acceptable relative and absolute evaluation difference

A typical expectation is that $\text{dist}_{\text{rel}}(e_1, e_2, \delta) \vee \text{dist}_{\text{abs}}(e_1, e_2, \varepsilon)$ holds. There are some of course some specificities depending on the transformed position (and we refer the reader to the original article for more details). However, the overall idea remains there should not have evaluation differences under certain thresholds.

In order to display the results of our experiments, we use so-called heatmaps in which the y-axis displays different values for the threshold δ used as part of dist_{rel}while the x-axis displays different values for the threshold ε used as part of dist_{abs}(see Definition 2). For example, in Figure 3, for $\delta = 0.05$ and $\varepsilon = 10$, the percentage is 30.93. That is, there are 30.93% of positions' evaluation that differ from either 10 centipawns (0.1) in absolute terms or with a ratio of 0.05 between evaluation of the original position and the transformed position. 10 centipawns is a small difference, and the ratio is also very low. Hence, it is typical of negligible difference and has little to no practical impact for users. Another example in Figure 3: 10.48% of positions (value at the bottom-right and corresponding to $\delta = 0.5$ and $\varepsilon = 100$) violate the MR property.

3. Motivation and Design

In this section, we elaborate on our motivations to replicate the work of Méndez et al. [22].

3.1. Review of the original study

To instrument and experiment with the approach briefly summarized in previous section, the original article considers:

- a dataset constituted of randomly generated chess positions, end-games, checkmate problems, and realistic positions;
- a depth of 10 plies, corresponding to 5 moves;
- Stockfish version 15 (and other chess engines), with Stockfish being central ;

We now review these three constituents, as well as some results and conclusions of the study.

Dataset. Randomly generated chess positions are interesting input test cases. However, only relying on them can be problematic, since the realism of positions is important for (1) users that want to analyze real-world games or prepare known openings; (2) contributors of Stockfish that are driven towards delivering state-of-the-art performance in terms of real-world playing. Using unrealistic positions for testing might divert the development focus away from this goal. It is crucial to also use positions from actual games to ensure the chess engine's performance is optimized for real-world scenarios. Furthermore, there is a risk that the metamorphic testing method mainly works for this kind of random input – and less on realistic one. At least, it is an hypothesis about the generality and sensitivity of the approach that we aim to verify or refute. End-games are also of interest, but the existence of pre-computed syzygy-tablebases [32] for positions with 7 pieces limits the interest. Hence, end-games with at least more than 7 pieces should be prioritized. Checkmate problems and puzzles are relevant, but finding them requires large computation depth. From this perspective, depth=10 might be too low to find a checkmate in *e.g.,* 18 moves. Overall, we found that the dataset of positions can be better balanced and diversified for assessing the effectiveness of the method on other inputs.

Depth. The choice of a low depth (10) is the first thing that strikes us as intriguing. We fully understand that searching to a greater depth requires more computational resources, especially when analyzing thousands of positions, however, 10 is too low. It corresponds to 5 moves, something club level chess players can handle. Furthermore, as confirmed by practitioners (see subsection 3.2), people are now using higher depths thanks to the commoditization of powerful hardware and the innovation made in chess engines (Stockfish). A low depth also can cause the engine to miss tactical opportunities or threats that require a deeper search to be found. A depth of 10 is only the starting point of the search space and a comprehensive search typically requires a much greater depth. It is even unclear what the Elo rating of Stockfish at depth=10 can be. Specifically to the original study, there is a general question on the sensitivity of the method regarding depth. Hence, it is worth considering higher values and study the impact of depth values on the effectiveness.

Version. The original article considers a specific Stockfish version (15). It is a threat to validity, and we can envision to assess the effectiveness changes (if any) when changing the version. Exploring different versions of the chess engine can well be combined with varying the search depth, and the use of different datasets to evaluate the generalizability of the original results. By varying these factors, a better understanding of how they interact and influence the testing method can be gained.

Causality. Méndez et al. [22] uncover a huge amount of violation of metamorphic relations. As recognized by the authors, it was unclear why such discrepancies occur. There are some speculations and hypothesis, but nothing concrete. In fact, whether metamorphic relations have caught actual faults that deserve to be considered or fixed is a general question in metamorphic testing [9]. Looking at the Stockfish community, we did not find activity related to the possible issues raised by the original study. An exception is this short discussion `https://github.com/official-stockfish/Stockfish/discussions/4610`, but to the best of our knowledge, there was no contribution or deep exchanges. Hence, an axis of research is to investigate the reason why discrepancies across transformed positions occur.

3.2. Chess players perspective

To gain further insights about chess engines, we conducted a semi-structured interview with n=4 chess players. Two are grand-masters, one is a FIDE Master, and one is a woman FIDE Master. All of them have experience and interest in livestreaming, coaching, preparation of top players or themselves, or analysis of chess games.

We asked a first question about the adequacy of setting the search depth to 10 in a chess engine. For any usage, all responded that it is not enough. One respondent stated that *"the only usefulness of chess engines is to analyze and arrive at a certain truth, so a module with a depth of 10 often gives inaccurate results."*

We then asked about search depth usually employed as part of their activities. One respondent said *"I have access to a good computer that can reach a depth of 30 in just a few seconds... it's reassuring. good machines give you confidence and above all, save time"*. Two respondents said they rely on chess engines hosted in platforms like Lichess or chess.com. One reported that *"when using the chess.com module, it's Stockfish with a depth of 18. Overall, it's not bad, but it can definitely make mistakes, so it's not sufficient if one wants to do precise work."* while another concurred "*Rarely below 25, except for watching online when we activate the modules on the platforms."*. Respondents tend to align with the need to use high depth *"For analysis, I use my computer, and depending on the positions, I try to go at least to depth 30 to be sure. And for all the preparation work, you need even more powerful computers, and I would say that 30 is really the minimum"*.

As a final topic, discussions were around possible fluctuations in the evaluations provided by the Stockfish chess engine during analysis. Specifically, from what difference in centipawns do they consider this change to be significant? All brought a nuanced answer and perspective on this issue: *"It really depends on the position, when it's flat, for example in Italian games, getting a +0.3 is good, so it can be significant.[...] In Gruenfeld I would say that 0.4-0.5 starts to be really critical or even risky."* or *"The flatter it is, the more concerning the changes in evaluation are. When it's chaotic, it's complicated. One must assess the human ability to find moves beyond the computer's evaluation, so even losses of 500 cp can be acceptable."*.

3.3. Replication

We are using the following definitions of reproducibility and replicability as presented in the National Academies of Science, Engineering and Medicine report "Reproducibility and Replication in Science" [11] (overview). *Reproducibility* is obtaining consistent results using the same input data, computational steps, methods, and code, and conditions of analysis. This definition is synonymous with "computational reproducibility." *Replicability* is obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data. Two studies may be considered to have replicated if they obtain consistent results given the level of uncertainty inherent in the system under study.

We fall in the two categories. We first aim to reproduce original results, a pre-condition before considering deviating and making variations. We then want to replicate the original study considering this time variations of datasets, depths, and versions:

- (1) a different dataset (mainly positions derived from actual chess games and a subset of the original dataset)
- (2) analyses at appropriate and larger depths. As argued, 10 is too low. Hence, we choose to explore larger values (depth=20) and infinite depth for a sample.
- (3) different versions of Stockfish (SF15 and SF16)

4. Reproduction

In the following, we first report on our attempts to reproduce the fault discovery findings of Méndez et al. with the original implementation[2].

We reused all the code used to perform the experiments, in particular the implementation of MRs and the datasets, available in `https://github.com/MMH1997/MT_ChessEngines`. The notebooks and procedures run out of the box. The sole effort was to set the path to the Stockfish binary. There are different versions of Stockfish, and we stick to version 15 as originally used. There are also different variants of Stockfish, targeting different operating systems, possibly with some optimizations w.r.t. CPU instructions. We selected a variant of Stockfish matching the name of the binary.

Once familiar with the code and offered facilities, we had to develop some programs and scripts: (1) procedures to filter out illegal[3] positions (as also

[2]Reproduction results can be verified at the beginning of the notebook, available in the replication package [34]

[3]The dataset used in [22] contains randomly generated positions. Some are illegal for subtle reasons *e.g.,* impossible double checks or situation where both kings are checked. In such cases, Stockfish either crashes or still performs an evaluation, but on impossible positions. We developed some procedures to detect invalid FEN based on a library such as `https://github.com/niklasf/chessops/` or through the catching of Stockfish errors.

reported and done in the original study); (2) a way to fully instrument the execution of Stockfish on the transformed positions; (3) a way to analyze the results and depicts heatmaps similar to Figure 3. Overall, we reproduced the experiments with the original dataset, depth=10, and SF15.

Figure 3 compiles the percentage of applications of the MR sim_mirror that failed. A first observation is that running (more than) twice our experiments lead to the exact same heatmaps and results. Stockfish in a single thread is indeed deterministic, and so the evaluations.

We then compared the heatmap included in the original paper and corresponding to our setup. The percentages were very similar with little differences. The small differences can be explained by a different variant of Stockfish running on a different platform. Another source of explanation is related to the thresholds used that can be sensitive to small centipawn differences. Even a difference of a few centipawns on some positions can affect some percentages here and there. As an example, we re-analyzed Carlsen vs Nepo, WCC 2021, game 6 with sim_mirror $\delta = 0.25$ and $\varepsilon = 25$ as thresholds. We did not find 4 MR violations as in the original article, but only 1 MR violation. Looking at centipawns evaluation, the numbers were very close but not exactly similar, making relative and absolute differences slightly different due to the thresholds used. The use of Stockfish 16 (instead of Stockfish 15) lead to 0 MR violation, with similar observations[4].

Overall, we found the discrepancies negligible and for all metamorphic relations (sim_mirror, sim_axis, better, first) we obtained results consistent with the original study. We also concur with the observations of the original study *e.g.,* better leads to very few violations and is the less effective metamorphic relation.

The original article is reproducible. Though there are small raw differences of numbers, the overall observations and trends remain consistent. Using the different metamorphic relations, we confirm the reported violations for Stockfish at depth=10, using the original dataset.

[4]Reproduction results can be verified at the beginning of the notebook while the specific game Carlsen vs Nepo is at the end of the notebook, available in the replication package [34]

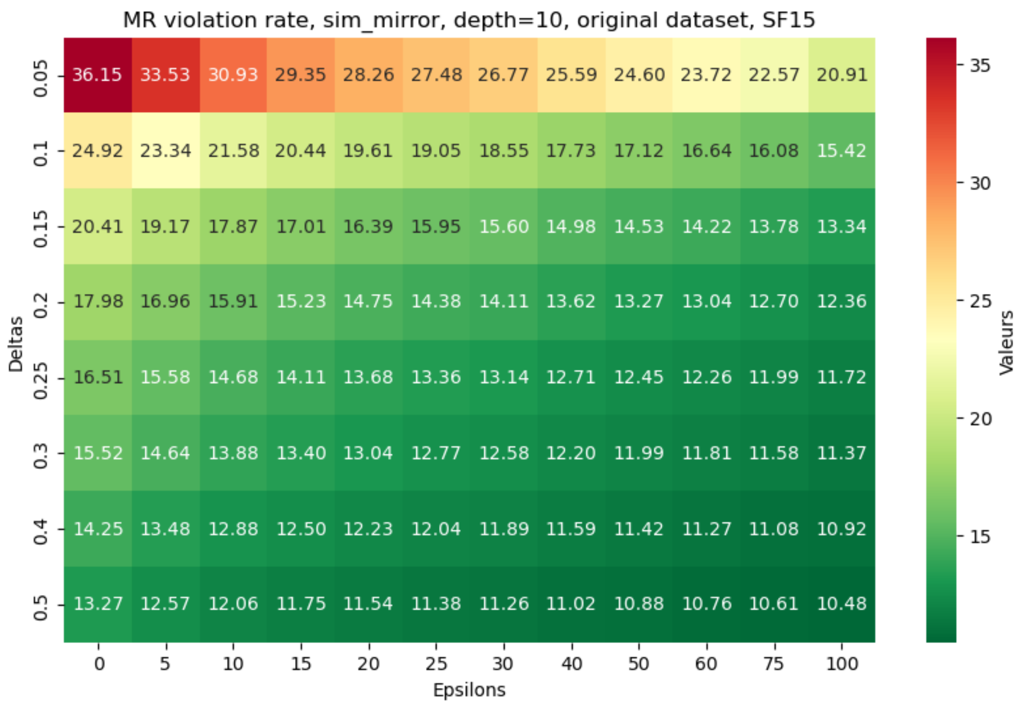


Figure 3: Reproduction of original results with depth=10, SF15, and the original dataset

5. Replicability Study

The reproduction under strict conditions of the original study has shown that metamorphic testing is indeed effective to reveal discrepancies across evaluations of Stockfish on positions that are syntactically different but semantically similar. However, we question and raise some doubts in Section 3 about some key aspects of the study. We now replicate and deviate from the original experiment. Specifically, we explore the possible effects of three factors (depth, dataset, version) and possibly their combinations.

5.1. Effect of depth

We consider the following experimental setup: the same original dataset, the same version (SF15), but this time we increase the depth. Instead of depth=10, we use depth=15 and depth=20 on all positions and all metamorphic relations. We also use larger depths than 20 (possibly infinite) for a sample of positions[5].

Our first observation is that increasing the depth (depth=15 or depth=20) does not alter the effectiveness of metamorphic testing. Percentages of MR violations are even increasing for sim_mirror (resp. best_move): 6.9% on average (σ2.15) (resp. 5.9%, σ1.9). For sim_axis (resp. sim_diag), the effectiveness slightly decreases on average -1.9% ($\sigma = 2.2$) (resp. -1.6%, $\sigma = 2.67$). Hence, the results invalidate our original hypothesis that increasing the depth should mitigate evaluation discrepancies. It is not the case, at least for the original dataset.

An important inquiry comes from the plot of the evaluation evolution w.r.t. depth. Figure 4 shows the evolution for the positions of Figure 1 and for depth until 30. In this example, the evolution of the two lines is similar across depth, and the metamorphic relation is verified at depth 2, 3, 5, 9, 12, 13, 14, 15, 16, 17, 18, 19, 21, 23, 25, 26, 27, 28, 29, 30 for $\delta = 0.25$ and $\varepsilon = 25$ as thresholds[6]. We also note that there could be stronger discrepancies, *e.g.,* at depth=6 or at depth=20. There is some stability after depth 25 and the metamorphic relation then still holds. Another example is given in Figure 5. The position involves a score excessively high (*i.e.,*-1000 centipawns is a

[5] Results at different depths can be verified at the beginning and middle of the notebook, available in the replication package [34]

[6] These specific threshold values of δ and ε are often used in [22], since they represent medium values and a good tradeoff.

decisive advantage). In any case, there are fluctuations with some depths for which absolute discrepancies are severe while for some other depth, there is some stability.

There are several comments and general observations to make. The first is that discrepancies can occur at certain specific depth (*e.g.,* depth=20) and then disappear or re-appear for subsequent depths. It shows the sensitivity of the metamorphic testing to the depth factor. Metamorphic relation can be violated depending on the choice of the depth or some discrepancies can be ignored. Unfortunately, there is no depth value that optimizes the effectiveness of the method. It is specific to a position and a relation to check, hence complicating the task of testers.

Second, we found that there always exist a depth threshold, beyond which further increases in depth do not result in any violation of metamorphic relation. We have verified this property on a sample of 200 realistic positions of the original dataset (excluding endgames with 7 pieces that are anyway resolved with tablebase) and using unlimited depth and 15 minutes for timeout. Intuitively, increasing the depth leads to some stability in the evaluation, as the search space is more and more explored and covered (see also Section 6 for more details about the Stockfish implementation).

Third, when a discrepancy or inconsistency is found at a particular depth, but then no further discrepancies are found, it raises questions about the usefulness and reliability of the testing method. The primary goal of a chess engine is not to be perfectly accurate at any specific depth, especially at lower depths. Lower depths represent shorter-term predictions, which may not always align with the best long-term strategy. Instead, the engine aims to provide the best overall evaluation of the position once the search is complete, which typically occurs at higher depths.

Using the original dataset, we found that metamorphic relation violations occur only at specific depths, with a threshold beyond which the testing method loses its effectiveness. A depth of 10 may lead to false violations, with higher depths providing more accurate results.

5.2. Effect of dataset

As argued in Section 3, the original dataset contains randomly generated positions, checkmates, and end-games that may limit the applicability of

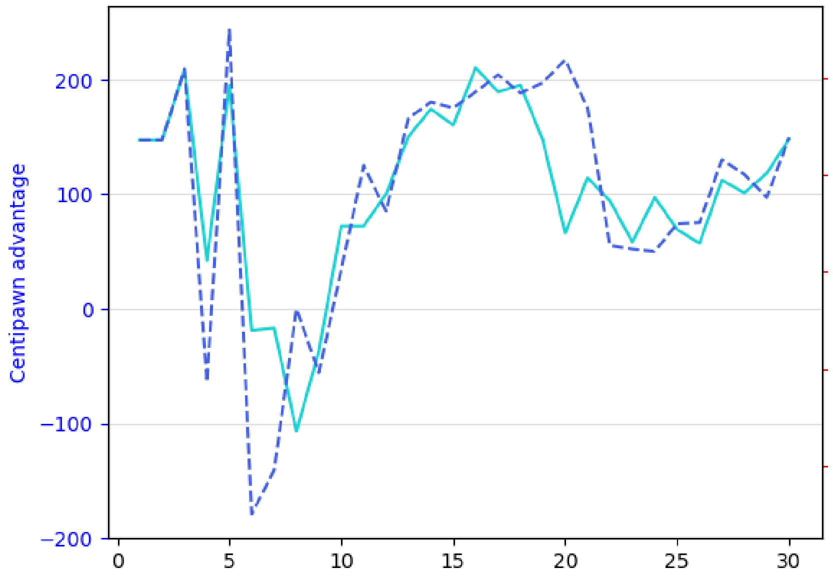


Figure 4: Evolution of the evaluation w.r.t. depth. The dashed line corresponds to the original position of Figure 1, while the other one is about the transformed position.

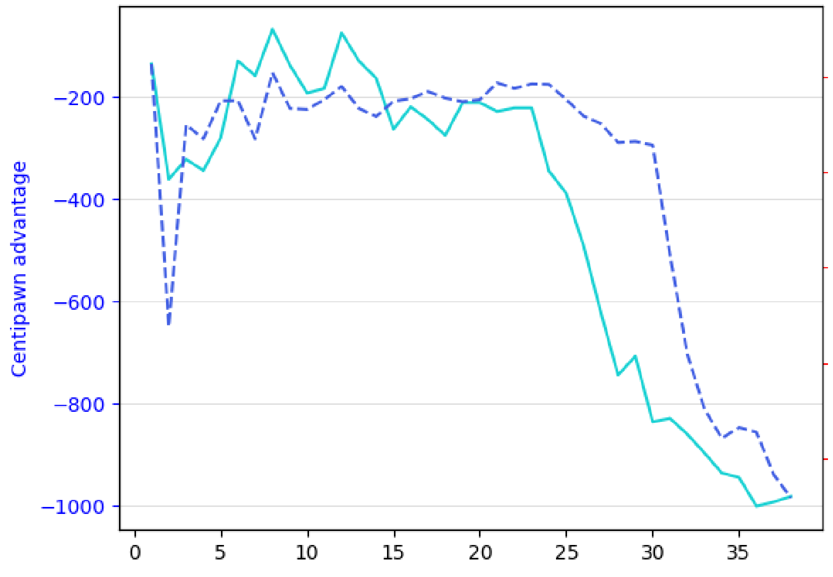


Figure 5: Evolution of the evaluation with SF=15 (sim_mirror)

the testing method. How about realistic positions that come from existing games?

Original dataset with real positions. The original dataset also contains a subset of real-world games. Hence, we use this opportunity to filter out the original dataset and only keeps realistic positions[7]. Figure 6 depicts the results for depth=10 and sim_mirror. Though it is not the same data, we can compare with the results of Figure 3. We can observe that the focus on realistic positions has positive effects, with an increase of percentage for small values of ε and δ (top-left of the heatmap). There are also negative effects, with low percentage (closed to 1 instead of 10) for large values of ε and δ (bottom-right of the heatmap). The general observation is that on average the percentages are closed to 0.01 (with strong standard deviation 9.9), suggesting a neutral balance. However, there is also a shift from the right-hand side to the left-hand side: metamorphic testing is less effective for large values of ε (absolute value in centipawn).

Figure 7 depicts the results for depth=20, sim_mirror, on the new dataset. The effectiveness is largely reduced compared to Figure 3 and Figure 6. The increase of depth (20 instead of 10) has a large negative effect: -7.7% with standard deviation of 5.1. For sim_axis (resp. best_move, we observe similar effect: -6.9 on average ($\sigma = 4.3$) (resp. -7.2, ($\sigma = 5.1$).

Lichess dataset. We gathered a dataset of positions coming from Lichess database [19]. It contains 21,000 realistic positions, much more than the subset of 600 realistic positions from the original dataset. We considered different games starting at move=10, move=20, move=30, and move=40 played by players at different Elo ratings (from amateurs to top grand master). This diversity allows us to determine whether the testing method is effective and general from chess positions' perspective.

For sim_axis and at depth=20, we observed a large decrease of the percentage of metamorphic relations that do not hold. Considering all Elo, the percentage on average is 3.4% ($\sigma = 5.8$) a comparatively very low score. Moreover, for values of ε greater than 30 (right-hand side of the heatmap), the percentage is closed to 0. Similar observations are made for other metamorphic relations.

[7]Results about datasets' changes can be verified at the middle of the notebook (*e.g.*, with the Lichess dataset) available in the replication package [34]

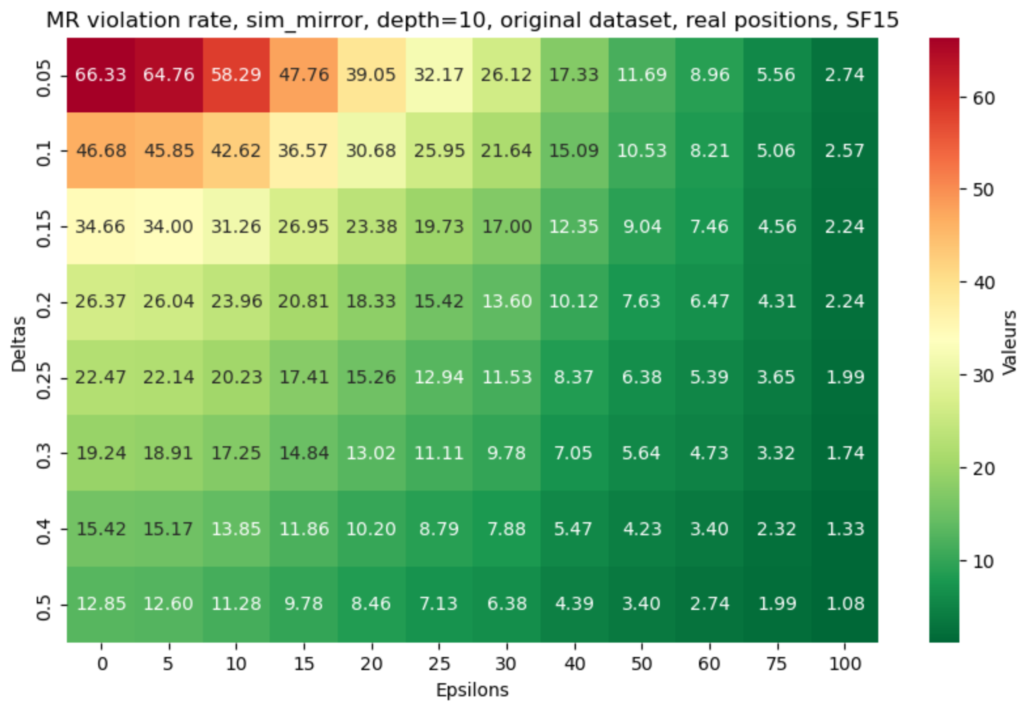
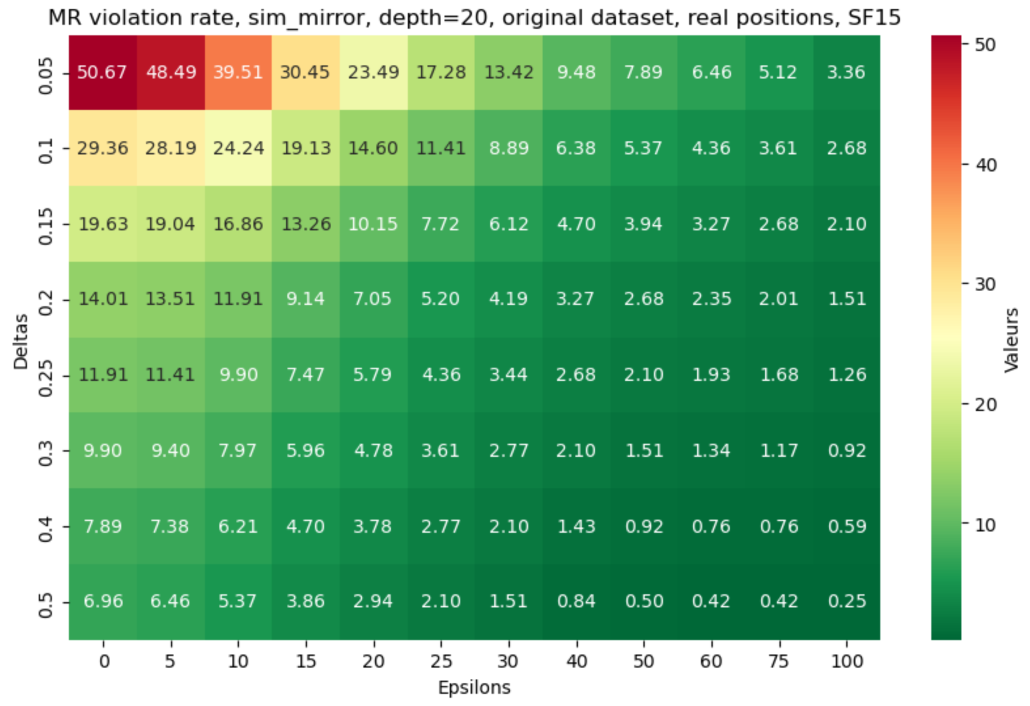


Figure 6: Replicability with depth=10, SF15, and only considering real positions

Controlled experiments on realistic positions (as opposed to randomly generated or puzzle-like positions from the original study) show that metamorphic testing with the current metamorphic relations is less effective. Additionally, increasing the depth to 20 for this type of data further reduces effectiveness, with percentages dropping to 0 across a wide range of threshold values. These findings cast doubt on the applicability of the testing method to realistic positions, suggesting that it may become ineffective as depth increases, warranting a reassessment of its use in such contexts.



Deltas \ Epsilons	0	5	10	15	20	25	30	40	50	60	75	100
0.05	50.67	48.49	39.51	30.45	23.49	17.28	13.42	9.48	7.89	6.46	5.12	3.36
0.1	29.36	28.19	24.24	19.13	14.60	11.41	8.89	6.38	5.37	4.36	3.61	2.68
0.15	19.63	19.04	16.86	13.26	10.15	7.72	6.12	4.70	3.94	3.27	2.68	2.10
0.2	14.01	13.51	11.91	9.14	7.05	5.20	4.19	3.27	2.68	2.35	2.01	1.51
0.25	11.91	11.41	9.90	7.47	5.79	4.36	3.44	2.68	2.10	1.93	1.68	1.26
0.3	9.90	9.40	7.97	5.96	4.78	3.61	2.77	2.10	1.51	1.34	1.17	0.92
0.4	7.89	7.38	6.21	4.70	3.78	2.77	2.10	1.43	0.92	0.76	0.76	0.59
0.5	6.96	6.46	5.37	3.86	2.94	2.10	1.51	0.84	0.50	0.42	0.42	0.25

Figure 7: Replicability with depth=20, SF15, and only considering real positions (*i.e.*, we change both depth and dataset)

5.3. *Effect of version*

Instead of using Stockfish 15, we considered the version 16[8]. A first setup is to experiment on the original dataset, at depth=10 (same as in the original study). Hence, addressing whether the testing is as effective with a new version of Stockfish. The result is that, for all metamorphic relations, there is a decrease in the percentage. This decrease applies to every threshold, but is negligible. For example, for sim_axis (resp. sim_mirror), -0.9% on average (resp. -3.7%). Overall, the sole change of the Stockfish version does not question the results of the original study – using the same experimental settings

A second setup is to experiment this time on the original dataset, at depth=20, and compare the results of Stockfish 16 vs Stockfish 15. The effect is negative and more pronounced (*e.g.,*-5.2% for sim_axis). That is, increasing the depth with Stockfish 16 can further reduce the effectiveness.

We also experimented with different realistic datasets (e.g., real positions, Lichess) and observed similar results: the effectiveness of metamorphic testing is reduced with Stockfish 16 at depth 20 across all metamorphic relations. In other words, changing the version confirms that using realistic positions diminishes the effectiveness of the metamorphic testing method. However, this reduction is not significantly different from what was observed with Stockfish 15.

Finally, we confirm the observations about the importance of depth w.r.t. evaluation. Figure 8 plots the evolution of evaluation for depth up to 35 and for the same FEN and transformation as Figure 5. With Stockfish 16, the discrepancies vary, but the metamorphic relation holds for *e.g.,* medium thresholds. This position also shows that the testing method is sensitive to Stockfish version and depth. If we consider depth=25 and Stockfish 15 (Figure 5) the metamorphic relation does not hold for medium thresholds since the gap is important. However, for depth=25 and the same position (Figure 8), there is no issue.

[8]Results about Stockfish 16 can be verified at the end of the notebook, available in the replication package [34]

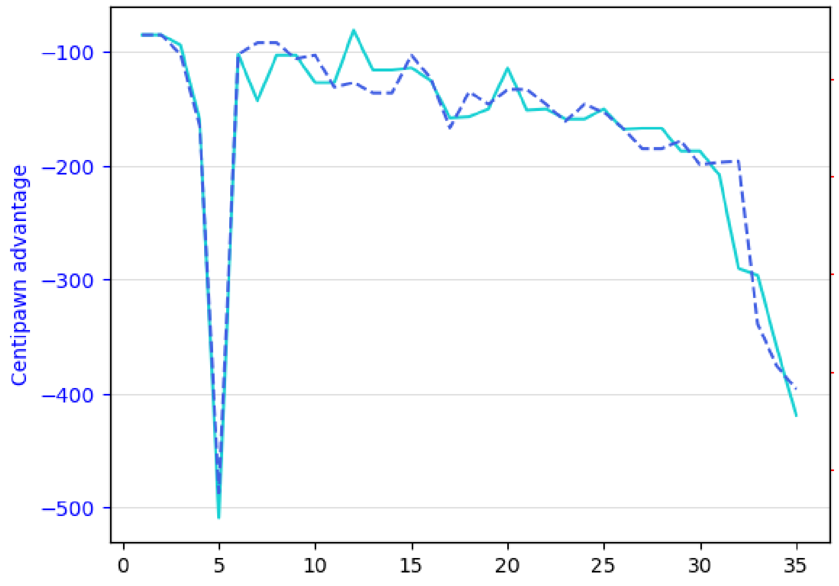


Figure 8: Evolution of the evaluation with SF=16. Same FEN and transformation as Figure 5.

Switching to Stockfish version 16 (from version 15) yields similar key observations: at low depth (10) on the original dataset, MR violations still occur, albeit less frequently. However, when using realistic positions or increasing the depth (*e.g.,* to 20), the effectiveness of the testing method declines significantly, with MR violations for a given position depending heavily on specific depth values.

6. How and Why Transforming Positions Influence Stockfish

So far, we have shown evidence that symmetries may provide some differences in evaluation in Stockfish. Although these discrepancies only occur at certain depth and are less frequent on realistic chess positions, we now would like to find out the reasons *why* Stockfish's evaluations differ. In the original article, the authors hypothesize about possible reasons:

> "Although we do not have concrete evidence, we can still speculate on the potential main cause for the detected failures. Chess engines explore moves in a specific order during the search process. However, if this order is different, for example, between a

> position and its symmetrical position, or if there is any randomness introduced within this order at any point, it could lead to differences in the evaluation due to variations in the lines of play considered."

However, it is crucial to emphasize that they were not certain about them, neither about the specific root cause in the implementation nor whether the issue was a result of a bug or not.

Our basic observation is that Stockfish has to consider all legal moves, however in order to consider those moves they must be in some arbitrary order. We will show that those symmetries change the order of the generated legal moves. Furthermore, we are going to show that in most cases it fully explains the differences in evaluation score. It is also interesting to note that since it explains most differences in evaluations, it also implies that chess specific optimizations have little to no impact on these asymmetries.

In the process of finding behavior differences across positions, we modified the Stockfish source code to trace intermediate steps, compare outcomes, and also edit some parts of the implementation. Then, we executed Stockfish on original positions and transformed positions, at different depth. We initially observed that the first ordering of moves during search is different, for some depths, eventually leading to a different evaluation. Our hypothesis is shown on Figure 9 for the axis symmetry. The order of legal moves is changed even up to symmetry. That is if we applied the reverse symmetry on the symmetric position we would not get the same order of moves as in the original position. This list of moves is then used for search, which is used to evaluate positions.

Protocol:. In order to test our hypothesis about the interplay between legal moves generation and ordering of moves during initial search, we modified the `generate_all` method in `movegen.cpp`. Our idea was to force the same order for the generation of moves, for an original and transformed position. Just before returning the array of legal moves, we sorted them according to an arbitrary order, here in order of increasing 16-bit representations of the moves, this is what we call the "sorted" version. Now, without loss of generality, let us consider the mirror symmetry. We also made the same modification, but we applied the reverse symmetry to go back to the original move when sorting move – let us call this version the "sorted mirror". To be clear that means that if "c3c4" is sorted first in the "sorted" version then "f3f4" would be the first move in the list in the "sorted mirror" version since after applying the symmetry "f3f4" maps to "c3c4".

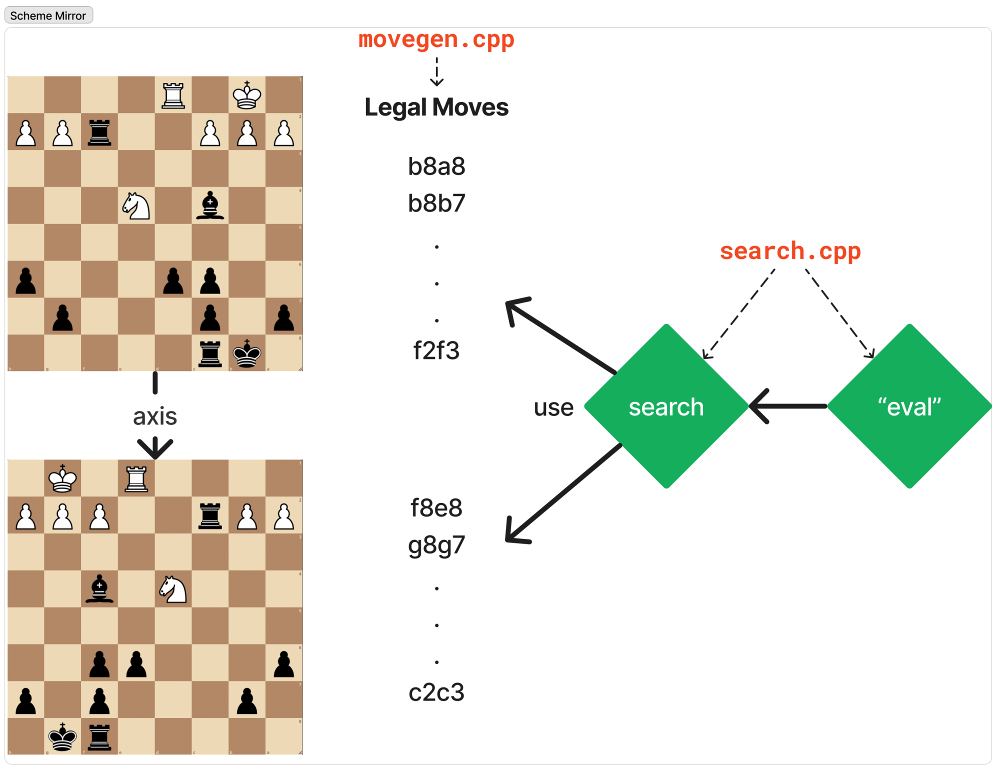


Figure 9: Overview of how legal move order is used for the evaluation of position

Given that we have these two versions, we record the differences of evaluation between the position and its symmetry. We will show the mean, median and standard deviation, we will also consider that move order explains a difference in evaluation when the difference between the evaluations of the "sorted" version on the position and the "sorted symmetry" on the symmetric position is exactly 0 and there was initially a difference in evaluations when we did not sort the moves.

Explanation:. Let us explain why the order of moves changes the evaluation with the help of Figure 10.

First, let us be clear: there is no bug or error in the code; this is a natural property of the algorithm. The process involves exploring different boards at different depths using heuristics to estimate a board's value. As the search continues, the evaluation of a board may be updated based on the

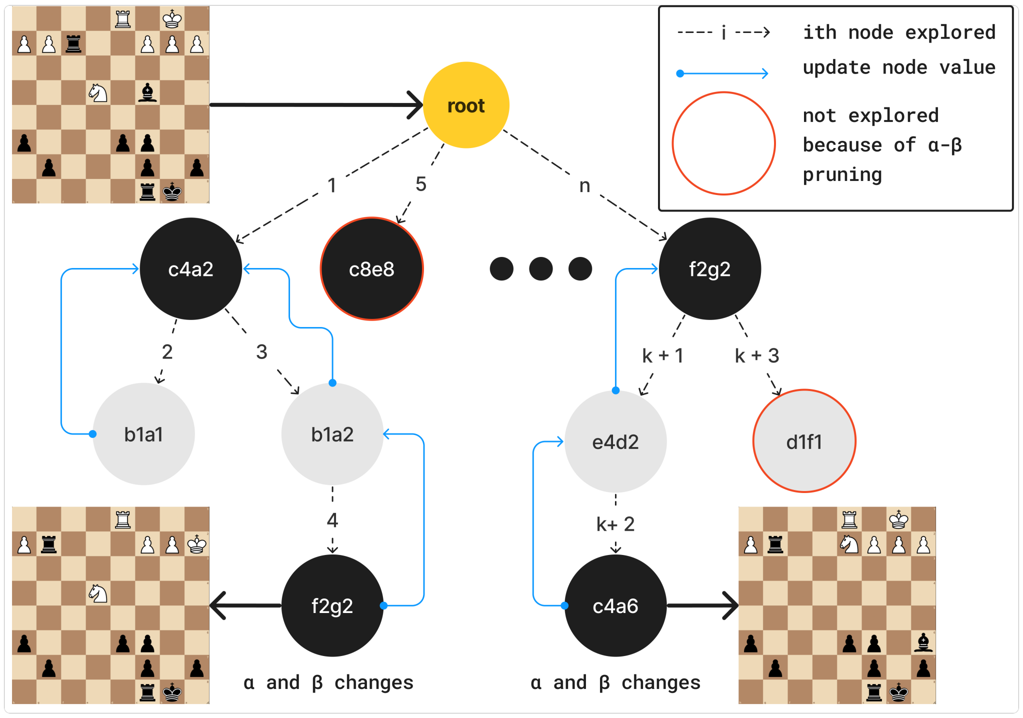


Figure 10: Overview of the position expansion using $\alpha - \beta$ pruning. The order of moves at depth 1 influences the values of $\alpha - \beta$ when evaluating the next moves. Thus, changing the order of moves may change the evaluation.

evaluations of its child nodes, leading to a more accurate assessment. Since heuristics guide the exploration, the order in which legal moves are examined matters. Importantly, altering the move order, as we did, not only changes the exploration sequence but also affects the heuristics used for evaluation. Modifying just one without the other can actually increase the discrepancies in evaluation.

Each node represents a different board. Starting from the root node, all child nodes are generated. Stockfish uses iterative deepening, meaning it progressively selects the most relevant nodes to explore at each iteration. Initially, since all child nodes lack a value, they are chosen in the order they were added. Once a node is selected, it is marked as explored (represented by the dotted arrows), and its child nodes are generated. Their values are estimated, and the best value is propagated back to the parent node (shown by the blue arrows). This process continues as long as Stockfish has time or has reached the maximum depth. A key point is that Stockfish assigns an initial value to nodes using heuristics, but this value is noisy and becomes more accurate as the node and its children are explored. Node selection balances exploration and exploitation, meaning that with more iterations, more paths are explored, and their values are refined. This explains why values converge as depth increases: early choices may be inaccurate, but are corrected over time as deeper exploration leads to better evaluations.

The bias effect of this initial evaluation is amplified with $\alpha - \beta$ pruning. The idea is that $\alpha - \beta$ describes an interval of value in which we are interested, that is nodes whose value is outside this interval are not explored because they are not relevant. When you explore a node you can update $\alpha - \beta$ based on the evaluation of the node, since you have a node that allows you to not do any worse than this node, thus pruning a large part of the search space where you might make worse choices which is irrelevant since you would not choose such a worse path.

Results:. We show the results of our experiments on Table 1 for sim_mirror and similar results with negligible differences have been obtained for sim_axis on Table 2. It is highly likely that these subtle differences depend on some positions that are not equivalent after symmetry, for examples pawns that might get farther away from the opposite side of the board. We count a position as explained by move order if using the same order of move with respect to the symmetry used reduces the differences in evaluation to 0.

We show two different views of the same data, the version where all positions are kept and another where only "stable" positions were kept, that is we kept only the positions whose evaluation did *not* change when the symmetry was applied. This is why 0% of the differences of these positions are explained by move order because the positions that diverged were removed.

First, let us look at only the changes with depth, a pattern emerges when we increase depth, the mean and std increases whereas the median decreases and percentage of move explained by move order decreases. The fact that the median decreases is a synonym of our earlier explanation of the algorithm, in most cases these differences in evaluations are fixed over increasing depths of exploration. In other words, in most scenarios when we increase the depth, in a majority of scenario differences of evaluations tend to disappear. The mean and the increase in std is mainly due to differences in evaluation because of different move orders, as we will explain. In fact, at depth 10 for random positions more than 95% of the differences in evaluation can be fixed by changing the move order, for Lichess position it is still more than 64% of the positions. However, it only accounts for 10% cases for random at depth 20 and at most 5% for Lichess. If we change the threshold of explained by move order to differences of less than 0.4 then at depth 20 it explains around 50% of the differences in both random and Lichess datasets.

While it seems that the move order explain most differences at depth 10, this is not as obvious to depth 20 and furthermore the standard deviation and mean can be seen as still relatively high when considering only stable positions, so we looked into what was happening. We found that on random data in more than 3% of cases, there was a difference of evaluation greater or equal to 100 between the position and its symmetry, in all of these cases the engine found a mate in one position but not in the symmetric one. When we remove these positions, the mean for stable positions goes to 0. For Lichess positions, it occurs in 0.4% of positions, and removing these positions put the mean to 0.03 for stable positions. In both cases, the standard deviation decreases to less than 0.1. So these rare positions explain most of the high mean and high standard deviation, but does the move order help solves these high differences in evaluation? We found that for around 45% for random and 60% for Lichess of these positions, using the same order fixes the discrepancy in evaluation to a difference of less than 0.4. In other words, fixing the move order even up to depth 20 fixes the evaluations differences. We do *not* know how to explain the remaining 2% of random positions and 0.2% of Lichess positions difference up to a negligible amount. A manual review of a sample of these positions tends to suggest positions with high score. Chess engines are not finely tuned for interpreting extremely lopsided positions. When the score is excessively high, it often simply means an imminent mate, and further distinctions in the score become less relevant.

Feature, not bug:. Our analysis explains the source of MR violations, which are caused by changes in the move ordering of legal moves during board rotations. This understanding can also be used to fix these violations by applying a patch to Stockfish to make the move ordering invariant to board rotations. However, enforcing such a fix is unnecessary, as move ordering at shallow depths is arbitrary due to the lack of guiding information. Implementing this patch would result in computational overhead without providing any meaningful performance gains for the engine.

Our controlled, in-depth analysis of the source code revealed that applying board rotations alters the move ordering of legal moves during the search process, explaining the evaluation discrepancies at certain depths. This behavior is a feature, not a bug, inherent to how modern chess engines operate. At a depth of 10, nearly all discrepancies are accounted for, while at a depth of 20, most remaining differences are either negligible or occur in positions with a clear advantage (*e.g.,* large centipawn values or imminent checkmates).

Table 1: Comparative statistics of the differences in evaluation between a position and the mirror position

dataset	positions	depth	mean	median	std	% explained by move order
random	all	10	3.23	0.10	18.15	95.4%
		20	3.66	0	19.4	10.0%
	stable	10	0.05	0	2.31	0%
		20	1.38	0	12.3	0%
Lichess, no castling	all	10	0.46	0.09	5.85	64.45%
		20	0.56	0.06	7.11	4.94%
	stable	10	0.02	0	0.04	0 %
		20	1.37	0.01	12.2	0 %

Table 2: Comparative statistics of the differences in evaluation between a position and the axis position

dataset	positions	depth	mean	median	std	% explained by move order
random	all	10	2.91	0.09	17.12	95.4%
		20	3.63	0	19.3	9.68%
	stable	10	0.02	0	1.57	0%
		20	1.40	0	12.4	0%
Lichess, no castling	all	10	0.43	0.09	5.56	67.76%
		20	0.62	0.06	7.54	5.37%
	stable	10	0.27	0	5.18	0 %
		20	0.92	0.01	9.90	0 %

7. Related work

Metamorphic Testing. Segura *et al.* [28] surveyed metamorphic testing. It can be used to compare different programs for the same input and determine whether some relations are kept from one execution to another [5]. For instance, Donaldson *et al.* [10] applied metamorphic testing to reveal bugs in compilers for OpenGL. Symmetries can reveal to be a key part in detecting potential failures of a system, indeed they tend to be very generic properties that can induce relevant metamorphic relations [38].

The effectiveness of metamorphic testing is highly dependent on the specific metamorphic relations that are used, and designing effective metamorphic relations is thus a critical step when applying metamorphic testing. Defining good metamorphic relations requires knowledge of the problem domain. Chen *et al.* [8] compared the effectiveness of metamorphic relations solely based on the theoretical knowledge of the problem (black–box) versus those derived from the program structure (white–box) using two case studies. They argue that good metamorphic relations should be preferably selected with regard to the algorithm under test following a white–box approach.

Testing and chess engines. There have been several applications of metamorphic testing [39, 7, 16, 25], and chess has attracted some attention. Another notable use of MT for chess was made with a different goal [17, 18], their goal was to ensure that the implementation of the engines were actually playing chess, in other words they checked if the performed moves were legal or not. The goal of their MR was to detect illegal moves, in order to do that they used mutation testing in order to modify part of the code and then checked that their MR detected the faults. Common techniques for analyzing chess engines are made in order to find among a set of chess engines which engine is the best at analyzing the game. By construction, better chess engine is more likely to win a game against a weaker chess engine. Thus, by using tournament techniques [33] or sequential statistical tests [36, 35] one can tackle the question. However, these techniques are of no interest since as in the original work of Méndez et al. the goal is to check the consistency of one, the best, chess engine.

Chess engines, evaluation and depth. The development of chess engines have a long tradition [6, 14, 2, 29], with many applications and use cases. Guid and Bratko [13] leveraged chess engines to determine who is the best chess player in history. Regan and Haworth [24] aim to represent playing strength at chess by the quality of moves played (as stated by computer chess programs), rather than by the results of games. Chess engines can also be used to detect cheaters [3] while cheaters can also use chess engines. There are some attempts to extract learned representation of the chess board out of neural networks to reconstruct many human chess concepts [20]. Specialized chess engines are still proposed. Maia, a customized version of AlphaZero trained on human chess games, aims at predicting decisions made by players at a specific skill level [21]. Some engines specifically target puzzles or fortress [37, 30].

8. Conclusion

We re-evaluated the article "Metamorphic Testing of Chess Engines." that originally reported on evaluation discrepancies of Stockfish, the state-of-the-art chess engine. Our replication stems from our observation that the original paper had multiple threats, and certainly drew incorrect conclusions on the applicability of their testing method, including the alarming claim that Stockfish has numerous faults and issues. We first succeeded to reproduce the results of the original study, thanks to the remarkable availability of source code and datasets.

We then considered potential threats and varied three factors of the experiments:

- the depth value (originally at a very low value, 10);
- the dataset through the inclusion of realistic positions;
- the version of Stockfish;

We analyzed the individual influence of each factor as well as their combinations (*e.g.,* using greater depth on realistic positions with a different version of Stockfish). We conducted novel experiments on thousands of positions, employing significantly deeper searches.

The replication results showed that:

- the metamorphic relations are not as effective as in the original article, especially on realistic chess positions and increase of depth.
- we raised awareness of the sensitivity of depth: metamorphic relations may only be violated at specific depths, and there is a depth threshold beyond which the testing method becomes ineffective.

- through a rigorous and in-depth analysis of the source code, we found why Stockfish can exhibit discrepancies on transformed positions and why at certain low depths, metamorphic relations are not effective. Our overall conclusion is that it is not a bug, but a feature of the exploration process of modern chess engines.

There are two main takeaways. From a chess engine perspective, our re-evaluation shows that Stockfish is not as faulty as originally suggested. In particular, we do not found evidence of realistic positions with evaluations' discrepancies at large depths. From a metamorphic testing perspective, the metamorphic relations should be parameterized by the search depth – ignoring this key parameter of modern chess engines is a fatal methodological flaw and leads to false conclusions.

As of now, metamorphic relations are not an effective technique for finding real faults of Stockfish, but the original attempts can certainly be refined (*e.g.,* with depth in mind), based on our quantitative and qualitative insights. The replication process taught us a valuable lesson: metamorphic relations need to be verified within the unique context of a specific domain. Without this context-specific validation, conclusions can be misleading or irrelevant. Also, changes in parameters and input dataset can drastically alter the effectiveness of a testing method – hence the importance of replicating studies.

Although our findings indicate that the metamorphic relations in our study are inadequately defined and lack the domain knowledge specificity required for effective testing of the state-of-the-art chess engine Stockfish, this limitation should not be seen as indicative of metamorphic testing as a whole. On the one hand, the development of more sophisticated metamorphic relations could enable effective metamorphic testing of chess engines in future research, and we believe that our insights will contribute to the formulation of these new relations. On the other hand, there is potential for applying metamorphic relations to non-standard chess engines, such as those based on transformers (*e.g.,* [26, 12, 15, 27]). Overall, our work highlights the need for further exploration of the original ideas and for refining metamorphic testing approaches.

Data availability. The material of this study is publicly available online: `https://github.com/acherm/chess-MT-Stockfish` and includes all data (Stockfish analysis of all positions at different depth, with different version, and on transformations), instructions to reproduce experiments, scripts used to analyze data, as well as heatmaps and results integrated in notebooks.

Statement. During the preparation of this work, the authors used ChatGPT and Mistral in order to improve the readability and language of the manuscript. After using these services, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Acknowledgements. We thank Martin Monperrus, Helge Spieker, and Arnaud Gotlieb for their insightful discussions about metamorphic testing and feedbacks about the paper.

References

[1] 2024. Top Chess Engine Championship (TCEC). Online Computer Chess Tournament. https://tcec-chess.com/ Accessed: 2024-04-13.

[2] Mathieu Acher and François Esnault. 2016. Large-scale Analysis of Chess Games with Chess Engines: A Preliminary Report. arXiv:1607.04186 [cs.AI] https://arxiv.org/abs/1607.04186

[3] David J Barnes and Julio Hernandez-Castro. 2015. On the limits of engine analysis for cheating detection in chess. *Computers & Security* 48 (2015), 58–73.

[4] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2014. The oracle problem in software testing: A survey. *IEEE transactions on software engineering* 41, 5 (2014), 507–525.

[5] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (May 2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[6] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial intelligence* 134, 1-2 (2002), 57–83.

[7] Tsong Yueh Chen, Jianqiang Feng, and TH Tse. 2002. Metamorphic testing of programs on partial differential equations: a case study. In *Proceedings 26th Annual International Computer Software and Applications*. IEEE, 327–333.

[8] Tsong Yueh Chen, DH Huang, TH Tse, and Zhi Quan Zhou. 2004. Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*. Citeseer, 569–583.

[9] Andrew G. Clark, Michael Foster, Neil Walkinshaw, and Robert M. Hierons. 2023. Metamorphic Testing with Causal Graphs. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 153–164. `https://doi.org/10.1109/ICST57152.2023.00023`

[10] Alastair F. Donaldson and Andrei Lascu. 2016. Metamorphic testing for (graphics) compilers. In *Proceedings of the 1st International Workshop on Metamorphic Testing, MET@ICSE 2016, Austin, Texas, USA, May 16, 2016*. 44–47. `https://doi.org/10.1145/2896971.2896978`

[11] Engineering, Medicine, National Academies of Sciences, Engineering, Medicine, et al. 2019. Reproducibility and replicability in science. (2019).

[12] Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. 2023. ChessGPT: Bridging Policy Learning and Language Modeling. arXiv:2306.09200 [cs.LG] `https://arxiv.org/abs/2306.09200`

[13] Matej Guid and Ivan Bratko. 2006. Computer analysis of world chess champions. *ICGA journal* 29, 2 (2006), 65–73.

[14] David Heath, Derek Allum, and Park Square. 1997. The Historical Development of Computer Chess and its Impact on Artificial Intelligence. *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence* 63 (1997).

[15] Adam Karvonen. 2024. Emergent World Models and Latent Variable Estimation in Chess-Playing Language Models. arXiv:2403.15498 [cs.LG] `https://arxiv.org/abs/2403.15498`

[16] Vu Le, Mehrdad Afshari, and Zhendong Su. 2014. Compiler validation via equivalence modulo inputs. *ACM Sigplan Notices* 49, 6 (2014), 216–226.

[17] Aisha Liaqat and Muddassar Azam Sindhu. 2018. A metamorphic relation based approach for testing a chess game. In *2018 14th International Conference on Emerging Technologies (ICET)*. IEEE, 1–6.

[18] Aisha Liaqat, Muddassar Azam Sindhu, and Ghazanfar Farooq Siddiqui. 2020. Metamorphic testing of an artificially intelligent chess game. *IEEE Access* 8 (2020), 174179–174190.

[19] Lichess.org. 2021. Lichess Database. `https://database.lichess.org/`. Accessed: [Access Date].

[20] Thomas McGrath, Andrei Kapishnikov, Nenad Tomašev, Adam Pearce, Martin Wattenberg, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. 2022. Acquisition of chess knowledge in alphazero. *Proceedings of the National Academy of Sciences* 119, 47 (2022), e2206625119.

[21] Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. 2020. Aligning Superhuman AI with Human Behavior: Chess as a Model System. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1677–1687. `https://doi.org/10.1145/3394486.3403219`

[22] Manuel Méndez, Miguel Benito-Parejo, Alfredo Ibias, and Manuel Núñez. 2023. Metamorphic testing of chess engines. *Information and Software Technology* 162 (2023), 107263. `https://doi.org/10.1016/j.infsof.2023.107263`

[23] Krishna Patel and Robert M. Hierons. 2018. A mapping study on testing non-testable systems. *Softw. Qual. J.* 26, 4 (2018), 1373–1413. `https://doi.org/10.1007/S11219-017-9392-4`

[24] Kenneth Regan and Guy Haworth. 2011. Intrinsic chess ratings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 25. 834–839.

[25] Manuel Rigger and Zhendong Su. 2022. Intramorphic testing: A new approach to the test oracle problem. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 128–136.

[26] Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, Cannada A. Lewis, Joel Veness, and Tim Genewein. 2024. Amortized Planning with Large-Scale Transformers: A Case Study on Chess. arXiv:2402.04494 [cs.LG] `https://arxiv.org/abs/2402.04494`

[27] John Schultz, Jakub Adamek, Matej Jusup, Marc Lanctot, Michael Kaisers, Sarah Perrin, Daniel Hennes, Jeremy Shar, Cannada Lewis, Anian Ruoss, Tom Zahavy, Petar Veličković, Laurel Prince, Satinder Singh, Eric Malmi, and Nenad Tomašev. 2024. Mastering Board Games by External and Internal Planning with Language Models. *arXiv* (4 December 2024).

[28] Sergio Segura, Gordon Fraser, Ana B. Sánchez, and Antonio Ruiz Cortés. 2016. A Survey on Metamorphic Testing. *IEEE Trans. Software Eng.* 42, 9 (2016), 805–824. `https://doi.org/10.1109/TSE.2016.2532875`

[29] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[30] Hedinn Steingrimsson. 2021. Chess fortresses, a causal test for state of the art Symbolic [Neuro] architectures. In *2021 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[31] Stockfish Team. 2024. Stockfish, Powerful Open-Source Chess Engine. `https://stockfishchess.org/` Accessed: 2024-04-13.

[32] Syzygy Tablebase. 2021. Syzygy Tablebase. `https://syzygy-tables.info/`. Accessed: [Access Date].

[33] M Tearth. 2022. A few thoughts about testing chess engines. (2022). `https://tearth.dev/posts/a-few-thoughts-about-testing-chess-engines/`

[34] `https://github.com/acherm/chess-MT-Stockfish`. 2024. Replication package.

[35] Abraham Wald. 1992. Sequential tests of statistical hypotheses. In *Breakthroughs in statistics: Foundations and basic theory.* Springer, 256–298.

[36] Abraham Wald and Jacob Wolfowitz. 1948. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics* (1948), 326–339.

[37] Tom Zahavy, Vivek Veeriah, Shaobo Hou, Kevin Waugh, Matthew Lai, Edouard Leurent, Nenad Tomasev, Lisa Schut, Demis Hassabis, and Satinder Singh. 2023. Diversifying ai: Towards creative chess with alphazero. *arXiv preprint arXiv:2308.09175* (2023).

[38] Zhi Quan Zhou, Liqun Sun, Tsong Yueh Chen, and Dave Towey. 2018. Metamorphic relations for enhancing system understanding and use. *IEEE Transactions on Software Engineering* 46, 10 (2018), 1120–1154.

[39] Zhi Quan Zhou, ShuJia Zhang, Markus Hagenbuchner, TH Tse, Fei-Ching Kuo, and Tsong Yueh Chen. 2012. Automated functional testing of online search services. *Software Testing, Verification and Reliability* 22, 4 (2012), 221–243.