Can Explainability Metrics Improve Genetic Programming? Lessons from 2048

Lauren Paul*
DFKI — Cyber-Physical Systems
Bremen, Germany
lauren.paul@dfki.de

Christina Plump†
DFKI — Cyber-Physical Systems
Bremen, Germany
christina.plump@dfki.de

Bernhard J. Berger
Hamburg University of Technology
Hamburg, Germany
bernhard.berger@tuhh.de

Rolf Drechsler‡
University of Bremen
Bremen, Germany
drechsler@uni-bremen.de

Abstract

Genetic programming (GP) produces solutions that, while often less performant than neural networks, are uniquely amenable to analysis. In domains like game-playing, this explainability could reveal why certain policies succeed or fail—but such insights are rarely leveraged to improve the algorithms themselves. Here, we analyze a GP algorithm for 2048, testing whether structural, behavioral, or semantic metrics of evolved policies correlate with performance. While structural metrics showed no predictive power and behavioral features yielded ambiguous results, we identified a semantic feature that correlated with policy quality. Using this insight, we designed a mutation operator that improved performance. Though modest, this improvement suggests that explainability metrics can guide operator design, even when broader explanatory goals remain unmet. More broadly, our work highlights a potential advantage of GP: its solutions may be analyzable in ways that opaque methods like neural networks are not.

CCS Concepts

• **Mathematics of computing → Evolutionary algorithms**.

Keywords

explainability, genetic programming, structural analysis, behavioral analysis, game policies

ACM Reference Format:
Lauren Paul, Christina Plump, Bernhard J. Berger, and Rolf Drechsler. 2026. Can Explainability Metrics Improve Genetic Programming? Lessons from 2048. In *Genetic and Evolutionary Computation Conference (GECCO Companion '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3795101.3814717

1 Introduction

The explainability and interpretability of AI systems have become critical areas of research in recent years. As AI becomes more integral to daily life and the global economy—with projections estimating that AI could increase global GDP by 7% over the next decade [16]—the need to understand how these systems operate has grown correspondingly. Despite significant advancements, many AI systems remain opaque: while their components and construction are well understood, the mechanisms underlying their decision-making processes are often unclear. This lack of transparency is particularly problematic in high-stakes domains such as medicine, law, and education, where trust in AI is essential for its adoption and effective use. Explainability and interpretability are necessary not only to evaluate prediction quality and uncover biases but also to justify and improve decisions made with the support of AI models [9, 10, 14].

The growing recognition of these challenges has driven substantial research in explainable AI (xAI). A key objective in xAI is to distinguish between—and improve—interpretability (the ability to understand input-output relationships) and explainability (a broader understanding of a model's internal logic and decision-making process) [7, 9]. Additionally, most xAI research focuses on post-hoc explanations of trained models, rather than leveraging explainability to enhance the models themselves.

In this work, we investigate whether explainability metrics can inform the design of improved algorithms. We focus on genetic programming (GP) for the game 2048, a domain where GP's inherent interpretability enables detailed analysis of evolved policies. Unlike neural networks, which usually achieve higher performance but resist explanation, GP allows us to examine both the final policies and the optimization process that produces them. Specifically, we build on EvoAl2048 [2], a winning entry from the 2024 *Interpretable Control Policies* competition at GECCO[1], which provides a transparent structure for analyzing policy evolution.

Our primary objective is to understand the relationship between policy characteristics and performance during optimization. To this end, we:

(1) develop and analyze structural, semantic, and behavioral metrics of policies, evaluating their predictive power with respect to policy performance; and
(2) design a novel mutation operator based on the insights gained from this analysis, demonstrating how explainability can directly inform algorithmic improvement.

While our results are specific to 2048 and GP, they suggest that explainability metrics may offer a viable path to algorithmic enhancement in domains where solutions are otherwise opaque. The remainder of this paper is structured as follows: Section 2 provides the necessary background, Section 3 describes our approach, Section 4 presents our results, and Section 5 concludes with a discussion of our findings and future work.

*Also with University of Bremen.
†Also with University of Bremen.
‡Also with DFKI — Cyber-Physical Systems.

GECCO Companion '26, San Jose, Costa Rica

ACM ISBN 979-8-4007-2488-6/2026/07
https://doi.org/10.1145/3795101.3814717

[1]Description at https://gecco-2024.sigevo.org/Competitions.html#id_Interpretable%20Control%20Competition

2 Background

To ensure self-containment, we briefly revisit the work of Berger et al. [2]. First, we outline the rules of the game 2048; then, we describe Berger et al.'s [2] policy design and structure. Finally, we summarize their optimization process.

2.1 The rules of 2048

The game 2048 was developed by Gabriele Cirulli[2] in 2014. The game board is a 4×4-grid and the game pieces are tiles which populate the cells of the grid. Tiles have a value 2^k for some $k \in \mathbb{N},\ 1 \leq k \leq 11$, i.e., powers of two ranging from 2 to 2048. When the game begins, the board is empty except for two randomly placed tiles of value two. In each *step* of the game, the player chooses between one of four directions in which to rotate the board — `up`, `down`, `left` or `right`. Rotating the board causes the tiles to move across the board in the direction of rotation, as if by gravity. If two tiles with the same value are sliding in the same direction and *collide*, they merge into a single tile whose value is the sum of the original tiles' values. Note that if three tiles of the same value slide in the same direction and collide, only the *bottom* two (with respect to direction of rotation) will merge. If four tiles slide in the same direction and collide, the result will be two merged tiles of equal value (twice the value of the original tiles). After the rotation, a new tile with a value of either two or four appears in an empty grid cell. The value of this randomly generated tile is chosen from a default-probability distribution of $\mathbb{P}\{\mathcal{T} = 2\} = 0.8$ and $\mathbb{P}\{\mathcal{T} = 4\} = 0.8$. The goal of the game is to merge tiles until a tile with value 2048 is created. If the grid fills and no moves remain that alter the board, the game ends.

Although the primary objective of the game is to reach a tile with the value 2048, the game also tracks a running score. This score is calculated by summing the values of tiles merged at each *step*, as shown in Figure 1. Since the tile values upon which the score is based grow exponentially, the total score tends to grow exponentially as well; i.e., the longer one plays, the faster the score increases.

With the game rules established, we now describe the policy structure used by Berger et al. [2] to model decision-making in 2048.

2.2 The structure of a policy

Berger et al. [2] use a predefined policy structure to find an optimal and interpretable policy for the game 2048. To that end, they consider the board and the move action space. They use Boolean (and Integer) properties to describe the board's current state. Cascading conditions built with Boolean expressions containing Boolean properties or arithmetic comparisons (=, <, >) of Integer properties as literals then suggest the next move. Thus, they consider the Markov probability of the game and reduce the board state space from all board states to a property space. The following definitions yield the necessary information to understand the policy structure.

Definition 2.1 (State Space). The *state space* $\mathcal{S}$ is the set of all possible 4×4 board configurations in the game 2048. Each board is a matrix whose entries are elements of the tile value set

$$\mathcal{T} = \{\, x \in \mathbb{N}_0 \mid x = 0 \text{ or } x = 2^k \text{ for some } k \in \mathbb{N},\ 1 \leq k \leq 11 \,\},$$

where the value $x = 0$ represents an empty tile.

Definition 2.2 (Property Space and Conditions). The *property space* $\mathcal{P}$ contains descriptions of the board state and is Boolean-based and Integer-based. Formally, $\mathcal{P} = \mathcal{P}^{\mathbb{B}} \cup \mathcal{P}^{\mathbb{Z}}$ contains functions mapping the state to either a Boolean value ($\mathcal{P}^{\mathbb{B}} \ni P^{\mathbb{B}} : \mathcal{S} \to \mathbb{B}$) or an Integer value $\mathcal{P}^{\mathbb{Z}} \ni P^{\mathbb{Z}} : \mathcal{S} \to \mathbb{Z}$. The set of atomic propositions then consists of Boolean properties $P^{\mathbb{B}}$ and arithmetic comparisons (<, >, =) of Integer properties $P^{\mathbb{Z}}$. Based on these Boolean atomic propositions, Boolean expressions $\varphi : \mathcal{S} \to \mathbb{B}$ define conditions on the current board state, which can either be fulfilled or not.

Please note that the property space can be chosen freely. The original work uses a defined set of properties, however, they can be extended or delimited at liberty (by an adaption of the configured property model).

Definition 2.3 (Action Space). The *action space* $\mathcal{A}$ is the set of all possible actions which an agent or policy has to choose from.

$$\mathcal{A} = \{\texttt{up}, \texttt{right}, \texttt{down}, \texttt{left}\}.$$

Definition 2.4 (Move). A *move* applies an action $a \in \mathcal{A}$ to a board state $s \in \mathcal{S}$, resulting in a new board state. Formally, a move is a function

$$m : \mathcal{S} \times \mathcal{A} \to \mathcal{S}.$$

With these four definitions, we are ready to define policies.

Definition 2.5 (Policy). A policy is a piecewise function mapping a board state to an action $\pi : \mathcal{S} \to \mathcal{A}$ via properties of the board. The function always has the form

$$\pi(s) = \begin{cases} \texttt{up} & \text{if } \phi_{up}(s) = \top \\ \texttt{right} & \text{if } \phi_{right}(s) = \top \\ \texttt{down} & \text{if } \phi_{down}(s) = \top \\ \texttt{left} & \text{if } \phi_{left}(s) = \top \\ a_{\text{default}} & \text{if } \bigwedge_{a \in \mathcal{A}} \neg\varphi_a(s) \end{cases},$$

where $\phi_a(s) = \varphi_a(s) \wedge \bigwedge_{a' \prec a} \neg\varphi_{a'}(s)$, and $a' \prec a$ denotes actions a' listed prior to a. This formal workaround is used to capture the behavior of `if-else-if`-chains, where subsequent `else-if` conditions are only evaluated when preceding ones have been evaluated to $\bot$.

In Figure 2, an example of a full policy and its structure is shown. Several aspects should be noted here: First, the order of the entries is neither fixed nor are entries disjunct or complete. There may be several entries for one action (cf. entry 2 and default for duplicate actions `down`, and `left` for missing actions). The determinism of a policy in light of multiple entries for one action is given through the order of the entries (`if-else-if`-chain). Properties of the board are presented with p_i for $i \in 1, \ldots, 6$, where p_3, p_4 are Integer-based properties, hence, they need an arithmetic comparison to be included in the final condition. $\varphi_{right}(s) = p_6$ denotes the general property for move `right`, but $\phi_{right}(s) = p_6 \wedge \neg\varphi_{up}(s) \wedge \neg\varphi_{down}(s)$ constitutes the actual condition to be evaluated (taken the `if-else-if`-chain into account).

[2] https://github.com/gabrielecirulli/2048

(a) An arbitrary state of the 2048 board.

(b) The subsequent state of the board after a move in direction down.

Figure 1: Exemplified application of a move in direction down. This move would add a score of $32 + 4 = 36$ to the total score.

Figure 2: Example of 2048 Policy Structure

```
entry 1:
  action: up
  condition: (p1 ∧ p2) ∨ (p3 > p4)
entry 2:
  action: down
  condition: p1 ∨ (p5 ∧ p2) ∨ p5
entry 3:
  action: right
  condition: p6
default: down
```

Berger et al. [2] introduce several sets of properties that can be changed at will. The following list shows the three main types of properties used in this analysis.

- `scoreGain(`a_1`)`: outputs the difference in score resulting from move $m(s, a_1)$.
- `scoreGains(`a_1`, `a_2`)`: outputs the difference in score resulting from moves $m(m(s, a_1), a_2)$.
- `canMoveInDirection(`a_1`)`: outputs true if $s \neq m(s, a_1)$, otherwise false.

It is important to note that these properties do not include any specific knowledge about the game besides the minimal rule set. Berger et al. state this to be due to making the approach more general and exploiting the GP structure rather than expert knowledge on the game. If the focus were, however, more on optimality for the specific use case than on generalisability, specialised properties like (`highestTileInCorner`) can easily be introduced.

2.3 Optimizing a policy

Berger et al. [2] use a genetic programming approach to optimize the policy. The policy model introduced above provides the program structure. Mutation operators modify literals (e.g., adjusting values) and structural components (e.g., adding, removing, or reordering entries or conditions), while recombination exchanges compatible subexpressions or policy elements between individuals. Selection is based on a performance-evaluation priority list. For each candidate, several test runs of the policy are performed, yielding statistical measures for the highest tile per run: `mean`, `median`, `max`, `min`, and `sd` of the highest tile, resp., and the score per run (`max(score)`, `mean(score)`, `sd(score)`). Developers can select an ordering of these measures to compare them and obtain a fitness value. In this work, we used the final configuration of Berger et al., as shown in Figure 3.

Figure 3: Comparison of perfomance measures

```
comparator := 'hierarchical-comparator' {
  order := [
    data 'median(highest-tile)',
    data 'average(highest-tile)',
    data 'max(highest-tile)',
    data 'min(highest-tile)',
    data 'average(total-score)'
  ];
};
```

While Berger et al. [2] demonstrate the interpretability of their approach, key questions remain unanswered: (1) What structural, semantic, or behavioral features correlate with policy performance? (2) Can these insights improve the genetic operators? In this work, we address these questions.

3 Methodology

Our primary objective is to analyze the optimization process of genetic programming (GP) for 2048 policies, with the goal of understanding how policy similarity relates to performance and leveraging these insights to design improved operators. To this end, we address two research questions:

Research Question 1: What metrics can describe similarities between policies, and do these similarities correlate with similar policy performance?

Research Question 2: Can explanations of policy behavior inform the design of operators that enhance the algorithm's performance?

The following subsections address these questions in turn.

3.1 Similarity measures for policies

To address Research Question 1, we define three categories of similarity metrics for policies: structural, semantic, and behavioral. Structural metrics analyze the syntax of policies with modest semantic information on literals, semantic metrics evaluate structural dependencies with dynamic information of game evaluations, and behavioral metrics analyze fully evaluated runtime decisions during game-play.

3.1.1 Structural Similarity. Structural similarity metrics quantify differences in the syntax trees of policies. We consider *structure* to include information about the specific literals and conditions a policy contains, as well as their sequential arrangement into entries with associated actions. We investigate whether similar policy structure incurs similar policy performance. Establishing such a relationship would be particularly advantageous, as it would allow us to leverage a suite of well-established tree similarity metrics for policy analysis. In turn, this would lead to an improvable operator structure: The structural elements of a good policy should barely be changed, while poorly performing policies should undergo larger structural changes.

In the following, we chose to focus on two well-known tree similarity metrics, Tree Edit Distance (TED) and Maximum Common Subgraph (MCSG).

Tree Edit Distance. Computation of TED-values between pairs of policies was performed with a slightly modified version of the APTED implementation from Pawlik and Augusten [11, 12]. The pre-packaged APTED implementation uses a unit cost function (cost 1 for `insert`, `delete` and `relabel` operations). We implemented a custom cost function for the `relabel` operation.

In the unit cost TED implementation, relabeling is all-or-nothing — identical labels need not be relabeled, and non-identical labels incur a relabeling cost of 1, regardless of how similar the label strings themselves may be. However, in the tree-representation of a policy, labels of leaf nodes correspond to properties, which are composed of literals, Boolean constants, and comparison operators. Because the labels contain important information about a policy's decision-making, we attempt to reflect a more nuanced notion of label similarity in the relabeling cost function. To this end, we implement a custom cost function based on the Levenshtein distance [8] between two labels.

$$lev(L_1, L_2) = \begin{cases} \max(|L_1|, |L_2|) & \text{if } \min(|L_1|, |L_2|) = 0, \\ 1 + \gamma(L_1, L_2) & \text{otherwise} \end{cases}$$

$$\gamma(L_1, L_2) = \begin{cases} lev(tail(L_1), L_2) \\ lev(L_1, tail(L_2)) \\ lev(tail(L_1), tail(L_2)) \end{cases}$$

The intuition is that two comparisons which differ in, say, a single action parameter (small Levenshtein distance), are more similar than two comparisons containing completely different literals (large Levenshtein distance).

Because TED is sensitive to the size of the two trees being compared, we normalize it to obtain a more comparable metric. Following [13], we use the normalized tree edit distance, NTED:

$$\mathrm{NTED}(T, T') = \frac{\mathrm{TED}(T, T')}{|T| + |T'|}$$

Maximum Common Subgraph. Another common metric for tree similarity is the Maximum Common Subgraph [1].

The ISMAGS (Index-Based Subgraph Matching Algorithm) [6] implementation included in the Python network analysis library NetworkX [5] was used to calculate the size of the maximum common induced subgraph between two policy trees.

To calculate the metric proposed by [3], we used the formula:

$$\mathtt{mcsg}_n(T, T') = 1 - \frac{|\mathtt{mcsg}(T, T')|}{|\max(|T|, |T'|)|}$$

this is referred to as the normalized mcsg in the remainder of this work. A higher value indicates greater similarity, a smaller value the opposite.

3.1.2 Semantic similarity. While structural metrics only regard the structure of a policy and the semantic distance of its literals, semantic similarity also investigates semantic congruence of conditions and actions, as well as dynamic information which can be obtained during the optimization process while a policy is being evaluated. We therefore analyzed the variance of present moves, as well as the variance of evaluation of literals (to determine their importance when playing the game).

Move Entropy. We defined a semantic feature we refer to as *move entropy*, which aims to measure the distribution of moves (`up`, `right`, `down`, `left`) a policy makes while playing 2048. The move sequence of policy π in game $i \in 1, \ldots, 50$ is denoted by the tuple $A_\pi^i = (a_1, \ldots, a_k)$. These sequences were then concatenated across all games to form a single aggregated sequence per policy: $A_\pi = A_\pi^1 + A_\pi^2 + \ldots + A_\pi^{50}$.

The distribution of moves in A_π was analyzed using Shannon-Entropy [15] with the following definitions:

$$\mathrm{count}_{A_\pi}(a) := \left|\{\, j \in \{1, \ldots, |A_\pi|\} : A_\pi[j] = a \,\}\right|$$

$$p(a) = \frac{\mathrm{count}_{A_\pi}(a)}{|A_\pi|}$$

$$H(A_\pi) := -\sum_{a \in \mathcal{A}} p(a) \log_2 p(a).$$

As there are four possible outcomes, $0 \leq H(A_\pi) \leq 2$, with $H(A_\pi) = 2$ implying a perfectly uniform distribution over the four move directions and $H(A_\pi) = 0$ implying that a policy moves in a single direction 100% of the time.

We investigate whether similarity in this metric corresponds to similar performance as well as whether move entropy correlates with performance in general.

The rationale behind this metric is the intuition that more than one direction should be necessary for playing a good game (otherwise, it was fairly simple). However, whether a maximum entropy actually correlates to good performance is not clear in advance.

Game experience teaches for 2048 that three directions are sufficient, however, we do not want to include this specific knowledge in our methodological development.

Literal Frequency Evaluation. A literal ℓ is said to *positively contribute* to the policy decision $\pi(s) = a$ on board state s if the following conditions hold:

- ℓ appears in the condition of an entry e in π such that the action $a_e = a$ (i.e., the entry prescribes the action that was taken).
- ℓ occurs within a comparison c_{ij} such that $c_{ij}(s) = \top$ (i.e., it evaluates to true for state s), and all comparisons in the same conjunctive clause containing c_{ij} also evaluate to $\top$ on s.

Given a policy π, we record information about which literals positively contributed to its decisions across 50 evaluation games.

If a literal positively contributes to at least one policy decision in any of the 50 evaluation games, it is *active*. For any active literal ℓ, the following data was recorded:

- *Average number of calls per game*: total number of moves to which ℓ contributed divided by total number of games (50).
- *Average score delta*: the score delta for a single move is difference between the score before and after the move is taken. The score deltas for all moves to which ℓ contributed was summed and divided by the total number of moves to which ℓ contributed.
- *Proportion calls*: total number of moves to which ℓ contributed divided by total number of moves made by π.
- *Proportion score*: The score deltas for all moves to which ℓ contributed was summed and divided by the sum of total scores reached by π.

We chose to record *average number of calls per game* and *proportion calls* under the assumption that an *important* literal would contribute to more decisions than less important ones. However, it could also be the case that a certain direction is chosen only rarely, yet still plays an important role in the policy's performance. This influenced us to additionally record the share of the *score* attributable to each literal.

Again, we want to investigate whether similarity in this metric corresponds to similar performance and whether literal frequency correlates with performance.

An assumption may be that literals that prevent *losing the game* may have more importance for policy performance than literals that make winning more likely.

First Action Agreement. While the last two measures mainly regarded frequency distributions, first action agreement tackles the intuition of semantic consistency. It considers the fact that one would expect policy properties to actually consider the move direction that they guard. Therefore, we define the relationship between a policy's chosen move and the directional input encoded in the literals that positively contributed to that decision as action agreement. In this work, we focus specifically on *first action agreement*, which refers to the alignment between the direction of the executed move and the first input action encoded in each contributing literal. If this first input action is the same as the move direction, the literal is said to be *in agreement*.

To quantify this, we counted the number of positively contributing literals whose first input action matched the move direction for each policy. This yielded an unweighted first action agreement count per policy.

Figure 4: Example Policy for computing *weighted first action agreement*

```
entry 1:
  action: up
  condition: scoreGains(up, down) >
                 scoreGains(down, right)
entry 2:
  action: right
  condition: canMoveInDirection(down)
entry 3:
  action: right
  condition: scoreGain(right)>scoreGain(left)
default: down
```

In addition to this count-based metric, we computed a *weighted first action agreement* score, which accounts for the frequency with which contributing literals were involved in decisions. Specifically, the score sums the proportion of calls for literals exhibiting first action agreement. Importantly, if a literal was negated, its contribution was subtracted rather than added, reflecting its logically inverse role in the decision. We use a small example to illustrate the computation: Assume a policy as given in Figure 4. Additionally, assume that dynamic analysis yielded a relative frequency of calls for the first literals in each entry as shown in Table 1. The first entry in Table 1 agrees with its move direction, i.e., the action is up, and the literal asks for a positive influence of up. The relative frequency now contributes positively to the metric. The contrary holds for the second entry: Action and first literal do not correspond to one another. It is therefore ignored for the metric. The third entry again holds a correspondence, and is therefore added to the metric value, yielding a final value of $0.4 + 0.3 = 0.7$.

Table 1: Example *weighted first action agreement* score for a single policy consisting of 3 active literals.

Literal	Call Proportion	Move Direction
`scoreGains(up, down)`	0.4	up
`canMoveInDirection(down)`	0.3	right
`scoreGain(right)`	0.3	right
Weighted First Action Agreement:		**0.7**

For this metric, we again seek to investigate whether similarity in this metric corresponds to similar performance and whether the weighted first direction agreement correlates with performance.

3.1.3 Behavioral Similarity. As mentioned above, policy performance is a fairly intuitive way of thinking about policy similarity, but it suffers from the fact that a huge variety of dissimilar policies might perform poorly — and in fact multiple dissimilar policies might perform well. To address this, we conceptualized a second candidate definition for policy similarity involving actual policy behavior.

The idea behind "behavioral similarity" is that similar policies are more likely to make the same decision when presented with identical inputs. Recall that the input for a 2048 policy is a board state, which is mapped to an action. Recall also that after each move, a randomly placed tile is generated. By fixing the random seed in the simulation environment, we ensure that, given an identical initial board state and identical sequences of actions, the resulting game trajectories are identical. For two policies, we therefore compare the length of their action sequences from a shared initial state and measure agreement up to the point where their decisions diverge. Move agreement was analyzed using many different initial boards (from early game, mid-game and end game states), which yields a measure of how similarly two policies behave. This metric, however, cannot be soundly evaluated during an optimization run, in contrast to the semantic metrics. It may therefore be helpful for general analysis, but not for online improvement during optimization.

3.2 Improved Operators for Optimization Algorithm

Research Question 2 involves a practical application of the insights gained by answering RQ1. We want to find out whether explainability can be used to enhance optimization algorithms — specifically the GP algorithm used to optimize our 2048 policies.

The *first direction mutator* was developed and implemented to investigate whether insights from explainability can be leveraged to improve the optimization process itself — a practical application of our analysis of structural and semantic features in 2048 policies. Specifically, we sought to operationalize the findings on first direction agreement (see Section 3.1.2) by introducing a mutation operator that modifies the first direction in a comparison to match the action associated with its corresponding entry.

The mutator works by identifying a random entry in the policy to be altered. Recall that every entry has an associated *action* a_e which determines the move which will be output if the entry's condition is fulfilled. Then, a comparison from the entry's condition is chosen at random.

Figure 5 shows an exemplary (albeit very simple) application of this mutator. The original policy contains an entry with a mismatch between action and first literal direction (`right` vs. `left`). The mutation operator now adjusts the first literal direction to match the corresponding action (changing `left` to `right`).

3.3 Realization

We used the optimization research tool suite EvoAl [4] to implement our analyses and the new operator. To this end, we implemented a plugin that adds the first-direction agreement mutator to the set of possible operators used by the genetic algorithm initially employed by Berger et al. in [2]. We analyzed the results using Python scripts and the EvoAl release.[3]

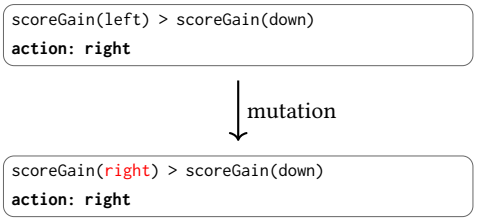


Figure 5: An example of the first direction mutator changing the first direction to match the entry action.

4 Evaluation

In the following, we detail our results on the respective similarity measures and their relation to policy performance. Additionally, we show the results attained by including the First Direction Agreement Mutator in the optimization process. Before presenting these results, we briefly describe the preprocessing steps applied to the policies, which ensure comparable results.

4.1 Setup - Normalising Policies

Before comparing policies, several preprocessing steps were undertaken to bring policies into a comparable form. These steps are outlined in Figure 6. There are two fundamental issues which have to be addressed in order to compare 2048 policies.

(1) **Rotational Symmetry:** Since the game 2048 is rotationally symmetric, there are essentially 4 versions of any policy which differ only in literal and entry directions. In fact, these versions can be generated by rotating all entry and literal directions by 90°.
This makes comparing two arbitrary policies difficult, since they may *appear* to be quite different — for example, containing many differences with respect to literal and action directions — while actually behaving quite similarly. Therefore, *Normalization* of policies was performed in an attempt to bring policies into similar rotations before comparing them.
(2) **Boolean Comparisons:** During optimization, mutation operators alter various components of the Boolean conditions contained in a policy. Boolean comparators can be flipped (> becomes <), and literals may be negated. As a result, separate policies might contain Boolean conditions which are semantically identical but syntactically different (see equations 3.1 and 3.2 in Section 4.1.3). In order to effectively compare multiple policies, *Literalisation* was performed to bring Boolean conditions into a standard form.

4.1.1 Normalization. Normalization was applied to 2048 policies in an attempt to bring them into comparable rotations. While rotating the policies is a simple matter of changing each direction by 90°, 180°, or 270°, *choosing a pivot point*, or how much and in which direction to rotate each policy, is not quite as simple. One could

[3]We make all artifacts available at acceptance of the paper for reproducibility.

Figure 6: Policy-processing pipeline implemented to facilitate comparison of 2048 policies generated during optimization.

assign a fixed direction to an arbitrary entry (e.g., the first or default entries) for each policy; however, there is neither guarantee nor particular reason to believe that an arbitrary entry plays the same role in various policies. For this reason, the method applied in this pipeline selects its pivot point based on a policy's most-frequently-moved-direction, i.e., the direction chosen most frequently by the policy over the course of 50 games. This direction is rotated to down and all other directions are updated accordingly.

4.1.2 Pruning. Many generated policies contain "inactive entries" whose conditions are never fulfilled during simulated game-play. We chose to remove these portions of the policy, reducing it to its relevant entries with the goal of reducing the influence of these inactive and therefore unimportant portions of the policy when comparing it to other policies.

4.1.3 Literalization. When a policy is represented as a tree, the boolean expressions contained in the entries correspond to node labels in the tree representation. Comparing tree structure usually involves comparing node labels. Therefore, the way these Boolean expressions are represented can have a big effect on the output of similarity metrics which compare trees, subtrees, and nodes.

Because of this, it is necessary to perform a second kind of normalization, called *literalization*, on the policy entries. The basic problem is summarized in equations 4.1.3: equivalent Boolean expressions have multiple equivalent representations. The solution is simple. All Boolean expressions are transformed into a standard form, which is limited to the comparison operators $>$, $=$ and $\neq$.

$$\neg(A > B) \equiv (A < B) \vee (A = B) \equiv (B > A) \vee (B = A) \equiv \neg(B < A) \tag{1}$$

$$\neg(A = B) \equiv A \neq B \equiv B \neq A \equiv \neg(B = A) \tag{2}$$

Even with this transformation, ambiguous representations are still possible for Boolean expressions involving equality and inequality due to reflexivity. In practice, however, this problem does not exist. All expressions involving equality and inequality are composed of a literal and a boolean constant $\top$ or $\bot$. An unambiguous representation is achieved by always assigning the boolean constant to the right operand. This preprocessing step additionally excludes ambiguities from our proposed mutation operator.

4.2 Results of Similarity Measures

We ran extensive evaluations on all defined metrics. Due to page restrictions, we only show a minor portion of the results here. Figure 7a shows the results of comparing the NTED structural metric with the performance difference in the given policies. Although some structure may be discernible, we disregard this metric as insufficient to describe policy behavior. Results were unfortunately similar for all studied structural metrics. However, this is somewhat understandable, since a simple change to one literal might have a dramatic effect on the policy's performance while having only a small effect on the structural metric.

Turning to the evaluation of semantic metrics: Figure 7b shows an exemplary result from these measures mapping the Weighted First Direction Agreement to the overall performance of the policy. Here, we can actually see a correlation in the data. This correlation is also present for comparing move-entropy to policy performance. Using statistical analyses, it could additionally be shown that both features together account for more than 80% of the variation in the data.

For behavioral similarity measures, only the slightest correlation was found between move agreement and performance difference. However, we see potential for further analysis, especially with different distributions of the analyzed board states.

All in all, we can cautiously answer our first research questions as follows: Structural similarity measures as such are not sufficient to give a good explanation on the performance of control policies. Behavioral similarity as such also seems to be insufficient in this regard. However, semantic measures are capable of explaining at least some of a policy's performance.

4.3 Results of first agreement mutation operator

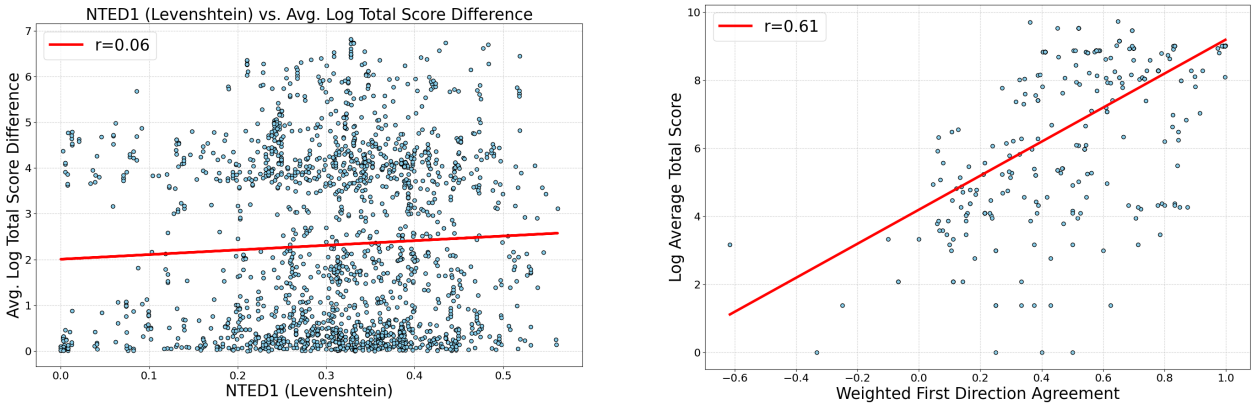
In this section, we compared the results of optimization runs with the newly developed first direction agreement mutator as well as without it. Figure 8 shows the results of running Berger et al.'s [2] genetic programming algorithm with the proposed mutator, motivated by the strong results of the semantic analysis.

To account for stochastic behavior, we ran each setup 50 times and report the mean performance of the maximum average highest tile achieved per run across generations as a proxy for policy performance. To determine whether the observed improvement in performance is statistically significant, we compared the final-generation performance per run, defined as the maximum average highest tile achieved at the last generation of each run. The configuration with the first direction mutator achieves a higher final performance (1021.3±123.3) compared to the baseline (927.1±112.0), with the difference being statistically significant (Mann–Whitney U test, $p = 2.89 \times 10^{-4}$).

The results show that the new mutator improves both speed and solution quality of the genetic programming approach, confirming the usefulness of incorporating explanation-based knowledge into the optimization algorithm and answering RQ2.

5 Discussion and Conclusion

Applicability to different domains Naturally, the importance of our results, while interesting for the original approach, suffers from the evaluation on a single domain. Nevertheless, most steps of our approach are transferrable to applications with comparable properties. Three main properties have to be fulfilled for our approach two be transferrable: First, the control problem needs to fulfill the Markov property, i.e., the next action should only be based on the current state of the system. Second, we assume a discrete action space. Number of actions and form, however, influence neither the



(a) Results from comparing the NTED similarity measure to performance values

(b) Results from comparing the weighted first direction match to performance values.

Figure 7: Results from comparing similarity measures to performance

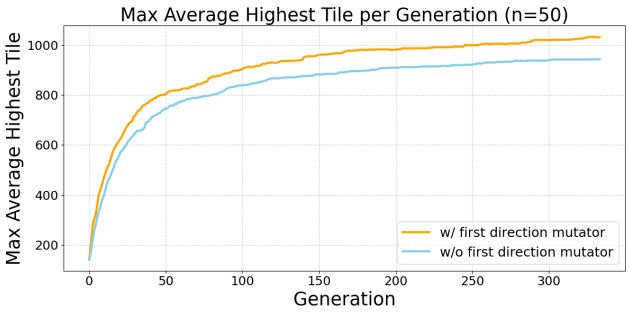


Figure 8: Results from comparing behavioral metrics against performance measures. Results have been averaged over 50 runs per setup.

concept introduced by Berger et al. nor the analyzed metrics in this work. Third, the reduction of the state space to the property space should (a) be possible and (b) lead to decreased computational complexity. Nevertheless, this third property is not strictly necessary, it simply improves the optimization approach. Rotational symmetry is not a direct requirement, it rather necessitates preprocessing of policies, i.e., our approach can deal with rotational symmetry but does not require it. So, in theory, our approach should be applicable to any discrete control problem that fulfills the Markov property.

Future Work The most obvious direction for future work would involve confirming the applicability to different discrete control problems in practice and not only by qualitative assessment. Additionally, the move entropy metric could serve as the basis for another improved mutator. Furthermore, online adaption of mutators could be investigated from collected runtime information during optimization. Finally, some metrics (especially the behavioral metrics) could be improved by investigating different designs for their computation.

Conclusion The importance of explainability has gained traction in optimization in recent years. While much previous research has focused mainly on the interpretability of machine learning systems, explaining the results of optimization algorithms is becoming increasingly important. In this work, we analyze the work of Berger et al., who developed a genetic programming algorithm to optimize interpretable policies for the game 2048. We analyze the structure of the policies and their behavior during the game. This provides insight into why the algorithm chose the resulting policy and also allows us to design an additional mutation operator for Berger et al.'s algorithm. When compared against their original optimization process, the addition of this new mutation operator actually improves solution quality. These results validate the assumption that a good understanding of an algorithm's inner workings can enable improvement of the algorithm. The results of this research are therefore relevant not only in terms of increasing users' trust in optimization results, but also for helping researchers gain a deeper understanding of the inner workings of metaheuristics and potentially adapt them to improve optimization results. We are interested in extending this work to different policy settings, thereby increasing its relevance to real-world use cases.

Acknowledgments

This work was funded by the German Research Foundation (DFG) within the CAUSE RTG 2972 under project number 513623283.

References

[1] Rangaswami Balakrishnan and Kanna Ranganathan. 2012. *A textbook of graph theory.* Springer Science & Business Media.
[2] Bernhard J Berger, Christina Plump, and Rolf Drechsler. 2024. EvoAl 2048. *arXiv preprint arXiv:2408.16780* (2024).
[3] Horst Bunke and Kim Shearer. 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters* 19, 3-4 (1998), 255–259.
[4] EvoAl Project. [n. d.]. *EvoAl.* https://doi.org/10.5281/zenodo.17494874
[5] Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).
[6] Maarten Houbraken, Sofie Demeyer, Tom Michoel, Pieter Audenaert, Didier Colle, and Mario Pickavet. 2014. The Index-based Subgraph Matching Algorithm with General Symmetries (ISMAGS): exploiting symmetry for faster subgraph enumeration. *PloS one* 9, 5 (2014), e97896.
[7] Uday Kamath and John Liu. 2021. Introduction to interpretability and explainability. In *Explainable artificial intelligence: An introduction to interpretable machine*

learning. Springer, 1–26.

[8] Joseph B Kruskal. 1983. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review* 25, 2 (1983), 201–237.

[9] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. 2020. Explainable ai: A review of machine learning interpretability methods. *Entropy* 23, 1 (2020), 18.

[10] Dang Minh, H Xiang Wang, Y Fen Li, and Tan N Nguyen. 2022. Explainable artificial intelligence: a comprehensive review. *Artificial Intelligence Review* 55, 5 (2022), 3503–3568.

[11] Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)* 40, 1 (2015), 1–40.

[12] Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. *Information Systems* 56 (2016), 157–173.

[13] Juan Ramón Rico-Juan and Luisa Micó. 2003. Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. *Pattern Recognition Letters* 24, 9 (2003), 1417–1426. https://doi.org/10.1016/S0167-8655(02)00382-3

[14] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence* 1, 5 (2019), 206–215.

[15] Claude E Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423.

[16] Dylan Walsh. 2025. *A New Look at the Economics of AI.* https://mitsloan.mit.edu/ideas-made-to-matter/a-new-look-economics-ai MIT Management Sloan School, Massachusetts Institute of Technology Sloan School of Management.