On the Nature of the Phenotype in Tree Genetic Programming

Wolfgang Banzhaf*
banzhafw@msu.edu
Department of Computer Science and Engineering,
Michigan State University
East Lansing, Michigan, USA

Illya Bakurov
bakurov1@msu.edu
Department of Computer Science and Engineering,
Michigan State University
East Lansing, Michigan, USA

ABSTRACT

In this contribution, we discuss the basic concepts of genotypes and phenotypes in tree-based GP (TGP), and then analyze their behavior using five real-world datasets. We show that TGP exhibits the same behavior that we can observe in other GP representations: At the genotypic level trees show frequently unchecked growth with seemingly ineffective code, but on the phenotypic level, much smaller trees can be observed. To generate phenotypes, we provide a unique technique for removing semantically ineffective code from GP trees. The approach extracts considerably simpler phenotypes while not being limited to local operations in the genotype. We generalize this transformation based on a problem-independent parameter that enables a further simplification of the exact phenotype by coarse-graining to produce approximate phenotypes. The concept of these phenotypes (exact and approximate) allows us to clarify what evolved solutions truly predict, making GP models considered at the phenotypic level much better interpretable.

CCS CONCEPTS

• **Computing methodologies → Matching**.

KEYWORDS

Genetic Programming, Genotype-Phenotype Map, Simplication, Neutrality, Explainability, Symbolic Regression

ACM Reference Format:
Wolfgang Banzhaf and Illya Bakurov. 2024. On the Nature of the Phenotype in Tree Genetic Programming. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3638529.3654129

1 INTRODUCTION

Since early in its development, GP has been associated with Machine Learning (ML) tasks [10, 38]. What is special about GP compared to other EC approaches is that its expected result is an active solution (the program, formula, model, etc.) that itself behaves in some way to process inputs into outputs. Thus it is not enough to simply consider a static solution to an optimization problem, but what is required is the ability to modify a solution in both its content and its complexity. The genetic operators for GP must therefore be able to grow and shrink the complexity of solutions in the evolutionary process, an ability that comes with its own particular challenges.

All this is well known and has led to some surprises early on in the history of GP [4, 10, 38]: To the bafflement of many researchers, a substantial portion of the code produced by GP was ineffective (neutral, non-effective, junk, etc.), which did not seem to play a role in the actual behavior of the evolved algorithms, but was produced by the evolutionary process. Researchers found that if they waited long enough in evolutionary runs, the overwhelming majority of code was of this type, a phenomenon coined 'bloat'.

Several theories have been put forward as to what the reasons are for this apparent inefficiency of code with its attendant strain on computational resources with respect to both memory and compute cycles. Obviously, some of the genetic material (genotypes) does not have an impact on the behavior which we can conceptualize by introducing a map between genotypes and behavioral determinants (phenotypes). Most GP representations today have such a mapping, even if it is somewhat hidden from view.

Three of the most typical representations employed in GP are expression trees in tree GP (TGP), linear sequences of instructions in linear GP (LGP) and circuit-type graphs in Cartesian GP (CGP). Here we shall be mostly concerned with TGP, and want to only briefly touch the other two representations. All three require genotypes (lists of program nodes in TGP, numbers encoding instructions in LGP, and numbers encoding graphs in CGP) and phenotypes (effective subtrees in TGP, effective subset of instructions in LGP and effective sub-graphs in CGP). To draw a clear distinction: genotypes are manipulated by genetic operations while phenotypes are the effective encoded structures that provide the behavioral function of the algorithms. Because genotypes are larger than phenotypes, taking into account that some of the code is non-effective or neutral (either structurally or semantically), it is the phenotypes that provide an understanding of what the algorithms do.

In an ML context, GP provides a user with a completely different experience in terms of transparency and interpretability of the final models when compared to "black-box" models of, say, deep neural networks. By manipulating discrete structures, the inner workings of a model are clearer, and the relationships between features and predictions are explicitly defined in GP. As a result, evolved solutions can offer insights into how the model arrives at its predictions, allowing humans to understand the decision-making process. Although GP is not the only "white-box" method in the spectrum of ML methods, it allows for unprecedented flexibility of representations and modularity, making it suitable for numerous tasks: classification [32, 39, 53], regression [15, 40, 51], feature engineering [54, 69], manifold learning [45], active learning [29], image classification [33, 68, 71, 77], image segmentation [9], image enhancement [16, 17], automatic generation of ML pipelines [58]

*Both authors contributed equally to this research.

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654129

and even neural architecture search [6, 27]. The area of explainable AI receives consistently more attention from both practitioners and researchers [47, 72] and GP has gained popularity where human-interpretable solutions are paramount. Real-world examples include medical image segmentation [18], prediction of human oral bioavailability of drugs [65], skin cancer classification from lesion images [1], and even conception of models of visual perception [8], etc. Besides being able to provide interpretable models, there is evidence that GP can also help to unlock the behaviour of black-box models [9, 26].

Given that GP phenotypes are even smaller than genotypes, interpretability is enhanced by focussing on phenotypes when analysing GP models. In this contribution, we study the relationship between genotypes and phenotypes in TGP, and develop a unique simplification technique that allows us to remove semantically ineffective code from GP genotype trees based on numerical simplification. Particularly, in this work, we show that:

- Trees of TGP represent both the genotype and phenotype.
- Smaller phenotypes are hidden within the larger genotypic trees but can be extracted via simplification algorithms without affecting their behavior, which facilitates models' understanding.
- Phenotypes can be further simplified to shrink model size but this might impact generalization ability.
- We observe the population dynamics of genotypes and phenotypes throughout the evolutionary process to facilitate our understanding of the genotype-phenotype relationship.

The paper is organized as follows: Section 2 introduces the necessary theoretical background by providing a brief discussion of the bloat phenomenon in TGP, an analogy to phenomena in living systems, some approaches to remedy this situation and a generic look at the difference between genotypes and phenotypes. Section 3 describes our algorithm to extract phenotypes from genotypes in TGP. We also introduce the notion of an approximate phenotype. Section 4 characterizes the datasets used in our study, discusses the hyper-parameters used, and shows the results obtained. Finally, Section 5 draws the main conclusions and proposes future research ideas.

2 BACKGROUND

As a natural result of genetic operators applied to tree genotypes, GP trees tend to grow in size during evolution in search of a better match to the desired behaviour. However, growth is not always justified by an improvement in performance (such as the generalization on unseen data). It has long been observed that standard TGP tends to produce excessively large and redundant trees without a corresponding improvement in terms of fitness [4, 38]. Moreover, it was shown that beyond a certain program length, the fitness distribution of individuals converges to a limit [43]. The apparent unnecessary growth of GP trees is often called the bloat phenomenon and several justifications were proposed for its existence. For decades, researchers tried to address this problem. We briefly review part of this voluminous literature here, but cannot address it in its full breadth.

The consequences of bloat in TGP are hard to underestimate: the amount of computer memory to store a population grows and concomitantly the evaluation time of trees increases, making the system less attractive for real-time applications. Moreover, unnecessarily large genotypes make visual inspection and interpretation difficult, if not impossible. In LGP and CGP the mechanisms for bloat result mainly in structural ineffective code. The difficulty for TGP is that its bloat is mostly semantic [11, 48].

2.1 Bloat

In binary-string Genetic Algorithms (GAs), it was found that highly fit building blocks become attached, by coincidence, to adjacent unfit building blocks. As a result, these joint entities were propagated throughout the population using recombination, subsequently preventing highly fit building blocks at adjacent locations from joining [50]. W. Tackett demonstrated that this phenomenon, called 'hitchhiking', also exists in TGP and that it was a consequence of recombination acting in concert with fitness-based selection [68].

In [41, 42], W. Langdon and R. Poli show that fitness-based selection makes the evolutionary search converge to mainly finding candidate solutions with the same fitness as previously found solutions. In the absence of improvement, the search may become a random search for new representations of the best-so-far solution. Given the fact that the variable-length representation of TGP allows many more representations of a given behaviour to be longer, these solutions are expected to occur more often and representation length naturally tends to increase. In other words, a straightforward fitness-based approach contributes to bloat. These findings are in agreement with the earlier findings of W. Tackett [68] who concluded that (i) the average growth in size is proportional to the underlying selection pressure, and that (ii) bloat does not occur when fitness is completely ignored.

In [23] the authors provide a theoretical model of program length distribution in the population assuming a repeated application of subtree crossover (i.e., crossover with a uniform selection of points) on a flat fitness landscape. The model is validated empirically and it is proved that the reproduction process has a strong bias towards sampling shorter programs when crossover with a uniform selection of points is applied. It was demonstrated that, if a reasonable minimum program size exists for relatively fit programs, the repeated application of fitness-based selection and subtree crossover will cause bloat to occur and it will be more acute if allowing crossover to distribute a population before applying fitness-based selection.

Several researchers demonstrated that bloat is an evolutionary response to protect solutions from the destructive effects of operators, with crossover the main focus of their studies [3, 13, 49, 56, 66]. In particular, when a program contains large sections of code that do not have a significant effect on its behaviour (introns), the application of a given genetic operator in that section is expected to have no overall effect on the program's behaviour (i.e., it is neutral). Several studies demonstrate that genetic operators are more likely to deteriorate offspring fitness than increase it and that a large proportion of recombinations are fitness-neutral [44, 57, 66]. Thus, by increasing the proportion of introns to effective sections of code (exons), evolution increases the likelihood that the product of genetic operators will be neutral in terms of fitness [12]. However, if bloat makes it more difficult to deteriorate the fitness of a solution, it also makes it harder to improve it. In this sense, bloat constitutes a serious problem for a sustainable evolutionary process.

2.2 Simplification

Here we consider the concept of the phenotype in TGP as that part of an individuum that produces overall behavior. We use a simplification algorithm to extract the phenotype tree from a genotype tree, and keep them side by side to monitor their development throughout evolution. Simplification algorithms have been traditionally used in TGP, but not with a concept of phenotype in mind. We divide such contributions into two categories: (i) those that have no effect on the evolutionary trajectory of a run (observing approaches); and (ii) those that do affect the evolutionary trajectory (intervening approaches). As pointed out in [35], historically, the former approaches were first, but quickly overtaken by the latter. Javed et al. call the former 'offline' approaches vs. the latter called 'online'.

Under category (i), we count ”offline simplification”, properly defined as "the analysis and reduction in complexity" of an individual outside an evolutionary run, e.g. at the end of a run [30, 34, 38, 61, 67], or during a run, but not returned to evolution - the latter we call ”monitoring” [52, 56]. What researchers have done there, without saying it, is to observe the phenotype of an individual similar to what we do here, without explicitly calling it that.

Under category (ii), we count ”online simplification” or ”pruning” [2, 34, 36, 55, 74], as well as limiting the size of individuals in various ways [23, 24, 38, 44, 63], operator equalization and its variants [25, 62, 64], time-based approaches to controlling bloat [20, 21], fitness and selection approaches (like double tournaments etc.) [46, 59], and elitism [60, 73]. All of these methods work directly on the genotype, with the result of modifying that genotype and intervening in the evolutionary process. A recent review of simplification and bloat control techniques can be found in [35].

Here, we focus on the former category of approaches, that are observing the runs, rather than intervening, because we believe that understanding the underlying phenomena is important to engineer better and faster evolutionary processes.

2.3 Genotype and Phenotype - Definitions and Relations

It is useful to recall a crisp definition of the term ”genotype” and ”phenotype” in GP. In this line of work, we discern them sharply by asking the following questions:

- Which structure is manipulated by genetic operators (like mutation and crossover)? This structure we define as the genotype.
- Which structure determines behavior, i.e. has an impact on the output of the program? This we define as the phenotype.

The situation is depicted in Figure 1. The phenotype determines the behavior of an individual which in turn is judged by the fitness function.

In Biology, the notion of phenotype is somewhat vague. Anything that can be observed can be considered the phenotype [22]. So the definition of phenotype is relative to the intent of the observer. It could be one specific trait that is under observation, and thus the phenotype is restricted to that trait. Or it could encompass the entire organism. But in GP we can tie the phenotype to that part of the algorithm that makes a semantic contribution to the output. Thus we can tie it to the behavior of the algorithm that is then judged by a fitness function.

Between genotype and phenotype is the so-called genotype-phenotype map (GPM). In GP, this is normally a relatively simple many-to-one mapping between genotypes and phenotypes. In Biology, on the other hand, the genotype-phenotype map can be much more complicated and typically is dynamic, a feature that few GP GPMs have (see [19] for examples). It has been argued that the GPM is a key ingredient in evolvability in living organisms [37]. For GPMs in GP, the mapping is normally many-to-one, with many genotypes leading to the same structure with its subsequent behavior.

It is easy to see the difference in a linear GP approach [14]. As is well known, LGP individuals consist of sequences of instructions, some of which might not impact the overall behavior of an individual. In LGP this comes about because such non-effective instructions manipulate registers that have no impact on the output of a program. We can easily identify those instructions and exclude them from the phenotype of that individual. A similar, if less frequent, phenomenon has been termed ”dormant nodes” [34] in tree GP. Only what is effective in terms of causing effective behavior is relevant and thus belongs to the phenotype.

The problem for tree GP is that the parse trees we see under evolution are genotypes, not phenotypes, but somewhat hidden within such parse trees are the phenotypes of the TGP individuals [48]. Thus, it is only a subset of nodes in a tree that effectively impacts its semantics or behavior, and it is that subset we are interested in when we want to understand what an individual evolving in TGP does. The problem of bloat in TGP is a problem of the growth of genotypes under evolution, not of the phenotypes. Provided we have a means to extract phenotypes from genotypes, we can hope to better understand the solution semantics on the phenotype level.

The next section will describe an algorithm for extracting the phenotype of a TGP individual from its genotype. We shall use this algorithm and its variants (which extract approximations to the phenotype) to argue that the relevant part of a GP individual, i.e., the part that requires explanation and understanding when solving

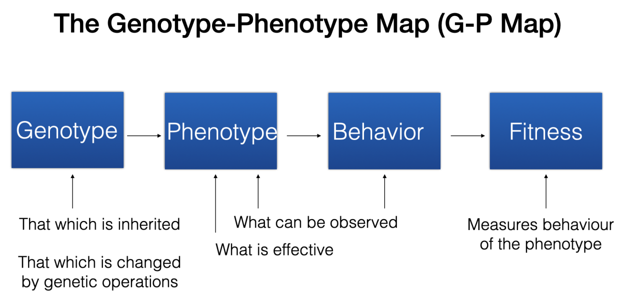
The Genotype-Phenotype Map (G-P Map)
Genotype
Phenotype
Behavior
Fitness
That which is inherited
That which is changed by genetic operations
What can be observed
What is effective
Measures behaviour of the phenotype

Figure 1: The relation between genotype, phenotype and behavior in Genetic Programming

a problem is only the phenotype and its behavior. Everything else is a distraction, perhaps necessary for the evolution of a solution, but not relevant to its behavior.

3 THE PHENOTYPE IN TREE GENETIC PROGRAMMING

Most researchers consider the size of the tree as a *proxy* for a solutions' complexity in TGP. Thus, extracting the phenotype from a genotype can reduce the complexity of a tree and enhance the interpretability of a solution while not changing the behavior, at least not significantly. In this study, the quality of a solution is measured as generalisation ability on regression problems. For each tree under evolution (genotype) we shall derive its phenotype and judge its behavior.

The impact of the simplification algorithm on the solutions' behavior will depend on the type of simplifications we apply to the genotype, and can be two-fold: (i) the behaviour is preserved, since the removed code has absolutely no impact, or (ii) it is changed, as the removed code contributed measurably to the behavior. We refer to the former as exact simplification, whereas the latter is referred to as approximate simplification. For approximate simplification, a problem-independent parameter t is introduced to control for different levels of approximation accuracy and represents the percentile of the distance distribution within a tree (see Section 3.1 for more details).

Consider a tree T implemented as a list of program elements (or nodes). Given a set of n training instances, the corresponding n-dimensional output vector is represented by $\hat{y}$ and is called its semantics. Let T_s represent a given subtree in T, rooted at node s, and $\hat{y}_s$ the respective semantics (T_s can also be a terminal node). During fitness evaluation, calculating $\hat{y}$ requires calculating $\hat{y}_s$ for each T_s in T. Specifically, the evaluation algorithm recursively traverses the tree in a bottom-up manner, starting from the deepest levels of T and computes the outputs for each successive T_s in T up the tree.

Algorithm 1 Pseudo-code for the proposed simplification method.

```
(1) Compute $\hat{y}_s \ \forall \ T_s \in T$.
(2) Given some similarity measure $f(x, y)$, create $M$.
(3) Given some value of $t \geq 0$, reduce $M$ to semantically equivalent
    pairs (i.e., $\forall \ (\hat{y}_{s_i}, \hat{y}_{s_j}), \ f(\hat{y}_{s_i}, \hat{y}_{s_j}) \leq p(t)$).
(4) While $M$ has entries:
    (a) Select the largest tree $T_{s_i} \in M$.
    (b) Select the smallest semantically equivalent tree $T_{s_j}$ for $T_{s_i}$.
    (c) Replace $T_{s_i}$ with $T_{s_j}$.
    (d) Remove all trees contained by $T_{s_i}$ (i.e., $\forall \ T_{s_{i_j}} \in T_{s_i}$).
    (e) Remove $T_{s_i}$ from $M$.
```

The proposed simplification algorithm capitalizes upon the mechanics of this fitness evaluation. In this way, we can avoid redundant function calls and calculations. In particular, when $\hat{y}_s$ is calculated for each T_s (which includes terminal nodes), the semantics are temporarily stored in a dictionary D of the form $D = \{idx_s : y_s\}$, where idx_s represents the index of node s (the root of T_s). As a worst-case, assume a fully grown binary tree of depth d with $2^{d+1} - 1$ subtrees; thus, D will store $2^{d+1} - 1$ vectors of length n. We use D to create a similarity matrix M between every pair of semantics y_{s_i} and y_{s_j}. In practice, we only calculate and store the lower triangular matrix since we assume symmetry, $f(\hat{y}_{s_i}, \hat{y}_{s_j}) = f(\hat{y}_{s_j}, \hat{y}_{s_i})$ and $f(\hat{y}_{s_i}, \hat{y}_{s_i}) = 0$, where f is some similarity measure. Being this the most computationally demanding step, the worst-case computational effort requires $N \times (N-1)/2$ operations for the lower triangle of an $N \times N$ matrix. Technically, D and M may be constructed from incomplete semantics to save memory and computing time. M is then filtered to store only semantically equivalent pairs. When the exact simplification is desired, T_{s_j} is said to be equivalent to T_{s_i} if $f(\hat{y}_{s_i}, \hat{y}_{s_j}) \cong 0$, where $\cong$ denotes equality within floating-point rounding errors. In the case of an approximation, the equivalence means $f(\hat{y}_{s_i}, \hat{y}_{s_j}) \leq p(t)$, $t > 0$. We denote the exact equivalence (the exact phenotype) with $t = 0$. Additional meta-data is provided with M: the length of each subtree and whether one subtree contains another.

The proposed algorithm will then use this information as follows: First, the largest tree T_{s_i} is selected from M and, for it, the smallest semantically equivalent tree T_{s_j} is chosen. Second, T_{s_i} is replaced by T_{s_j}. Finally, T_{s_i} is removed from M along with the subtrees it contained. The procedure is iterated while M has entries, see Algorithm 1.

Additionally, a dedicated *global library* can be provided and maintained for creating M to accelerate redundancy removal. It can be user-specified or created dynamically by caching all, or the most commonly found, semantically equivalent trees; a combination of both methods is also possible. However, a user-specified library requires additional effort and domain knowledge, and the maintenance of a global library would significantly increase both the memory and computational burden of the system. In this work, we rely on local tree information alone.

3.1 Approximate simplification accuracy

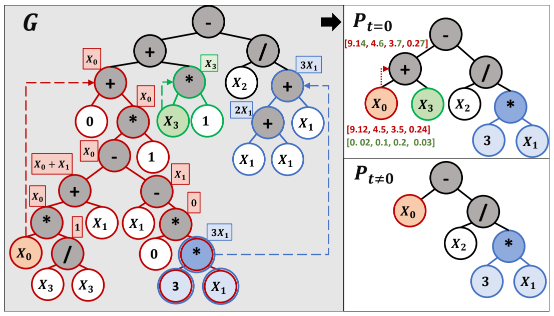

Figure 2: Illustrative mapping from genotype to phenotype. The genotype is represented with G (left box), whereas the exact and approximate phenotypes are represented with $P_{t=0}$ and $P_{t=10}$ (right boxes). Left: Semantics indicated by small colored boxes; Right: Semantics listed at selected nodes for all inputs.

We introduce a relaxation on the exact phenotype extraction algorithm which translates into an additional degree of simplification by coarse-graining the phenotype. This is achieved by a problem-independent parameter t representing the t^{th} percentile of the similarity distribution generated when simplifying a given tree T. Specifically, when M is constructed for a given tree, the values it contains represent a similarity distribution. In this sense, $p(t)$ is the value of the t^{th} percentile of that distribution and inversely represents the approximation accuracy for the phenotype extraction. In our experiments, we explore four different values of t: 2.5%, 5%, 10% and 20%. Thus T_{s_j} is said to be compatible to T_{s_i} if $f(\hat{y}_{s_i}, \hat{y}_{s_j}) \leq p(t)$. Given the fact that the input data for different regression problems have different distributions, including the range of values of the target, by defining $p(t)$ as the t^{th} percentile of the similarity distribution, a robust and problem-independent approach for approximate simplification is achieved. From this perspective, the exact simplification corresponds to $t = 0$. To accommodate for floating-point rounding errors, in our experiments, two semantic vectors are considered *equal* if the similarity between them, after being rounded to five decimals, equals zero.

Figure 2 provides a demonstration of the method. It is divided into three areas: G, $P_{t=0}$ and $P_{t\neq0}$. The grey area G depicts the genotype, while the other two represent different phenotypes, simplifications depending on the value of t. For $t = 0$, we perform exact simplification, thus extracting the true phenotype. For $t \neq 0$, we perform approximate simplification. From the figure, $t = 0$ requires only three iterations of the algorithm (represented in red, blue and green subtrees, respectively). Note that, however, more redundancies are present within the subtrees. For example, at the bottom of the red subtree, on the right, we find the expression $0 \times 3 \times X_1 = 0$. However, with X_0 being the smallest semantically compatible subtree of the large red subtree rooted on +, the algorithm will replace the red subtree completely with X_0, avoiding an unnecessary replacement of $0 \times 3 \times X_1$ with 0. Additionally, the algorithm can perform simplification by leveraging information from different branches. For the subtrees in blue, the semantics of $3\times X_1$ is the same as that of $X_1+X_1+X_1$. Although both subtrees are on different branches, this algorithm still identifies this redundancy and replaces the smallest with the largest.

To better follow the replacement logic for $t = 0$, at the root of the we provide the respective exact phenotype as a symbolic expression.

To better understand what happens when $t \neq 1$, let's assume that the result of adding X_3 to X_0 has a small effect on the overall behavior of T because variable X_3 has *small* values. The semantics of each node and the resulting operation are provided as lists, colored according to the node they relate to. Assuming some value of $t \neq 0$ (like 10), $f(\hat{y}_{X_0}, \hat{y}_+) \leq p(10)$; whereas $f(\hat{y}_{X_3}, \hat{y}_+) > p(10)$. Thus, X_0 will replace the subtree rooted at +, resulting in an approximate phenotype. It is semantically different from that obtained through exact simplification of the genotype.

4 EXPERIMENTS

4.1 Datasets

We assess our method on five real-world regression problems, which are described in Table 1.

4.2 Experimental settings

Table 2 lists the hyper-parameters (HPs) used in this study, along with cross-validation settings. The HPs were selected following common practice found across the literature to avoid a computationally demanding tuning phase. No limit to tree depth was applied during evolution in order not to interfere with the evolutionary process and to allow bloat to reveal itself in all its *splendour*. Although this resulted in a heavy computational demand, it allowed us to perform an unbiased assessment of monitoring the phenotypes. We used two sets of experiments: (i) one with a low mutation and high crossover rate, (ii) the other with a high mutation and low crossover rate. Low mutation / high crossover encourages a more exploitative search, whereas high mutation / low crossover tends towards exploration. These cases have different implications on the genotype of candidate solutions: high recombination of already existing genetic material in the former, and a higher disruption and novelty in the latter. Following previous studies about bloat [56, 57], we expected to observe more bloat to occur with higher crossover rates.

We rely on GPOL [7] to conduct our experiments. GPOL is a flexible and efficient multi-purpose optimization library in Python that covers a wide range of stochastic iterative search algorithms, including GP. Its modular implementation allows for solving optimization problems, like the one in this study, and easily incorporates new methods. The library is open-source and can be found by following this link. The implementation of the proposed approach can be found there.

Table 1: Five datasets used in the empirical study.

Dataset	#Instances	#Features	Target range
Boston [28]	506	13	[5, 50]
Bioav [5]	358	241	[0.4, 100.0]
Heating [70]	768	8	[6.01, 43.1]
Slump [76]	103	7	[0, 29]
Strength [75]	1005	8	[0, 1145]

Table 2: Summary of the hyper-parameters. Note that $P(C)$ and $P(M)$ indicate the crossover and the mutation probabilities, respectively.

Parameters	Values
№Train/test split	70/30%
Cross-validation	Monte-Carlo (repeated random subsampling)
№runs	30
№ generations	100
Population's size	100
Functions (F)	{+, -, x, /}
Initialization	Ramped Half&Half (RHH) with max depth of 5
Selection	tournament with size 4
Genetic operators	{swap crossover, subtree mutation}
$P(C)$	{0.8, 0.2}
$P(M)$	{0.2, 0.8}
Maximum depth limit	Not applied
Stopping criteria	Maximum № generations

4.3 Experimental findings

This section demonstrates the effects of the GPM on the population dynamics across five perspectives: size, diversity, fitness, deviation

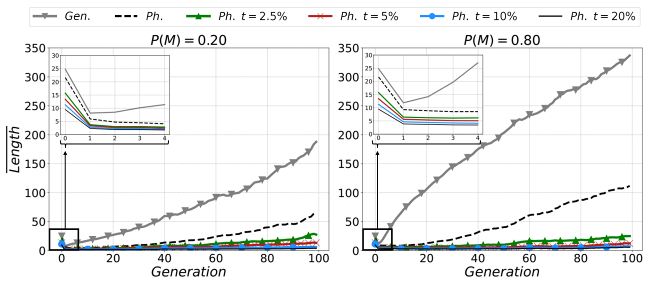

Figure 3: Growing average population length of genotypes, exact and approximate phenotypes, except for first generations (inset). Left: low mutation/high crossover; Right: high mutation/low crossover.

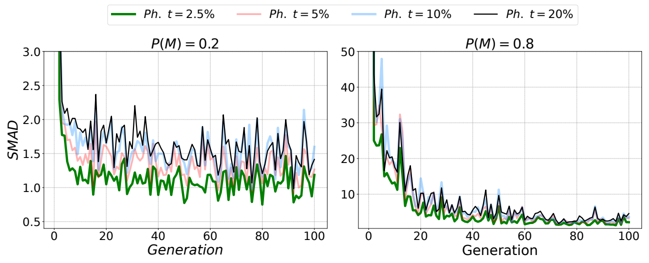

Figure 4: Mean absolute deviation between the semantics of the exact and approximated phenotypes. Note difference in scale.

between the genotype and phenotype semantics, and variation of the proportion of terminals. Each perspective is studied with two different mutation/crossover rates (low/high and high/low), provided as sub-figures. Standard TGP normally only reports what we call the genotype and is used as a baseline. In the plots, it is represented as a gray line. The exact phenotype is represented with a black dashed line, whereas the approximate phenotype is shown in green, red, blue and black solid lines, referring to different degrees of approximation (2.5%, 5%, 10%, 20%, respectively).

4.3.1 Size. Figure 3 shows the average population length over 100 generations. The subplot on the left shows the low mutation/high crossover. On the right, a high mutation/low crossover can be found. The inception plots provide a closer look at the first 5 generations. From the figure:

- Trees in S-TGP (here, the genotype) grow quickly, except for the first 1-2 generations.
- Phenotypes also grow after some generations, but much slower, with approximate phenotypes ordered according to approximation accuracy t.
- The average length of the individuals is larger when a higher mutation rate is used.
- The latter observation suggests that mutation, unlike what was expected, is more destructive than crossover for the applications considered. This makes the evolutionary process foster bloat as a protective measure. Similar results were found by [66] where higher mutation provoked larger growth.
- Monitoring the exact phenotype shows a reduction in the average length of individuals by a factor of 2-3, compared to the genotype.
- Approximate phenotypes show a 10 to 20-fold reduction in the average length of individuals.
- The sudden drop in the size of individuals in the first generation is an indication that selection prefers small and fitter individuals out of the randomly composed trees by RHH. Phenotypes stay close or drop even further in subsequent generations.

4.3.2 Semantics of Approximate Phenotypes. Figure 4 depicts the semantics mean absolute deviation (SMAD) between the genotype (T) and the respective phenotype approximations ($T_t,\ t \in \{2.5,\ 5,\ 10,\ 20\}$). Specifically, given the training dataset X with n training instances, and the respective outputs of T and T_t, represented as $\hat{y}_T$ and $\hat{y}_{T_t}$, $MAD(\hat{y_T},\ \hat{y_{T_t}}) = \frac{\sum |\hat{y}_T - \hat{y}_{T_t}|}{n}$. To enhance the patterns, we applied a higher of transparency to those lines representing the intermediate approximations ($t = 5\%$ and $t = 10\%$). On the left, the low mutation/high crossover HP setting is shown, whereas the high mutation/low crossover setting can be found on the right. From the figure:

- There is a notable decrease in SMAD in the first generations for all the cases, likely associated with the fact that trees are still more random combinations of program elements rather than refined input-output mappings. Thus, even mild modification of their structure results in substantial semantic perturbations when compared to those in later stages of the evolution.
- In general terms, SMAD exhibits an exponential decay. For $P(M) = 0.2$, it is more pronounced whereas for $P(M) = 0.8$ the decay is slower.
- For both mutation rates, one can conclude that the larger the degree of phenotypic approximation, the larger the SMAD.
- The difference between $t = 2.5\%$ and $t = 20\%$ is particularly notable: a higher degree of approximation results in bolder simplification and, thus larger SMAD.
- When $P(M) = 0.8$, the SMAD tends to exhibit significantly larger values, see the scale of the subfigure is different by an order of magnitude.
- This fact might unveil the nature of redundancies between different HPs. Recall that when $P(M) = 0.2$, one expects less exploration diversity, while with $P(M) = 0.8$, a larger diversity is expected.
- More disruptive changes in behavior happen when the search is more explorative.

4.3.3 Diversity. Population diversity has been long considered an important characteristic of the evolutionary process. Losing diversity, especially at the beginning, frequently translates into premature convergence. We here examine genotypic diversity defined as the number of unique trees per generation. Figure 5 depicts the mean of genotypic diversity in the two HP settings: low mutation/high crossover on the left, and high mutation/low crossover on the right. Analysis of the figure suggests that:

- During the first few generations we find a strong decrease in genotypic diversity. The decrease is more dramatic, however, for $P(M) = 0.2$ as less exploration takes place.

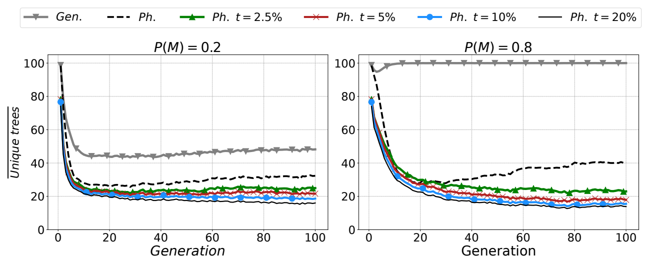

Figure 5: Diversity across evolutionary runs. Genotyic diversity stays high for high mutation scenario.

- Different lines seem to stack one upon the other and the order is a function of the approximation degree. In other words, the rougher the phenotype approximation, the smaller the diversity of the resulting population.
- In general terms, the diversity exhibits an exponential decay. For the high mutation setting, the decay is slower as more exploration takes place.
- There is an exception to the trend lines, however, when looking at the genotypic diversity for $P(M) = 0.8$. Individual genotypes tend to remain different in terms of their genomes across the evolutionary run.
- After 40-50 generations of the evolutionary process, the number of unique trees for the approximated phenotypes varies in a small range, around 15-25. This is observed for both $P(M) = 0.2$ and $P(M) = 0.8$. In other words, the genotypic diversity of the populations consisting of the approximated phenotype converges to the same level, regardless of the crossover/mutation rates.
- We can observe a growing trend for the exact phenotype (black dashed line) after 20 generations. This can be an indication that even the phenotype is not protected against bloat. Note that this phenomenon is more pronounced in the high-mutation setup, which was found to be associated with higher bloat in Figure 3 (possibly caused by destructive effects of the mutation operator).

4.3.4 Fitness. For each generation, we record the average population fitness, for both training and test data. Fitness here is error, so smaller is better. Figure 6 depicts median values for the two different mutation/crossover rates, represented here in distinct rows. Training data are depicted on the left, test data on the right. A higher line transparency was used for intermediate levels of phenotype approximations to enhance the main patterns. In the figure we can observe:

- There is a dramatic improvement in population fitness in the first few generations, which can be related to diversity loss.
- Globally, fitness/error exhibits an exponential reduction. For a higher mutation setting, the decay is slower as more exploration takes place.
- The rougher the phenotype approximation (and the smaller the phenotype), the better the fitness of the resulting individuals.
- This can be attributed to the presence of noise in the data that notably affects the population. Approximation seems to reduce these effects, leading to predictions that are closer to the overall trend.

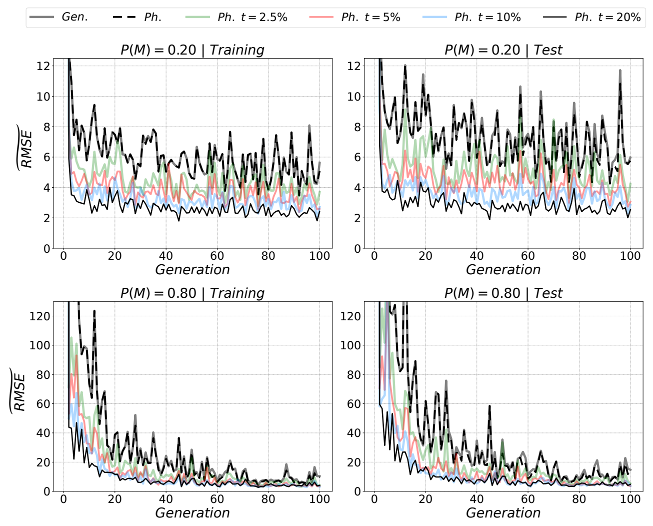

Figure 6: Median fitness in the population. Approximate phenotypes exhibit better fitness.

- Fitness progress achieved with $P(M) = 0.8$ are notably larger than those observed for $P(M) = 0.2$, particularly in the first half of runs.
- The conclusions hold for both training and test data.

Figure 7 shows the median fitness of the best/elite individuals (selected on the training data). The structure of the figure conforms to that of Figure 6. Unlike what was observed with population fitness, larger phenotypic approximation results in virtually no progress in elite fitness and we see two reasons for this. The first is the objective of the search - minimizing RMSE - a global measure that benefits individuals who are generally good, potentially neglecting training cases that are not widely well fit. The second is the fact that elites can be seen as a reflection of the exploitative nature of the search. Approximate simplification might remove components of elite individuals that contribute to their success, pushing towards the exploration side of the search. In these conditions, there will be cases where the final best individual in the run cannot be simplified at all. To remedy this, we plan to employ Lexicase selection [31, 40] and a more careful simplification for the elite solutions in the future works.

The overlap between the lines representing the exact phenotype ($p(t) = 0$) and the genotype tree is because semantics are the same. Some infrequent mismatches, however, can be observed due to minor floating-point rounding errors that sometimes occur in large trees, percolating through the tree. From all the generated results, these correspond to 1% of cases and mostly in one of the problems.

4.3.5 Structure. Figure 8 shows the proportion of terminals (input features and constants) in trees during evolutionary runs. The rest of the nodes are operators. We can observe that:

- Genotypes exhibit a notable reduction of terminal proportion in the first few generations. This is associated with a growth in tree depth (consequently more function nodes).

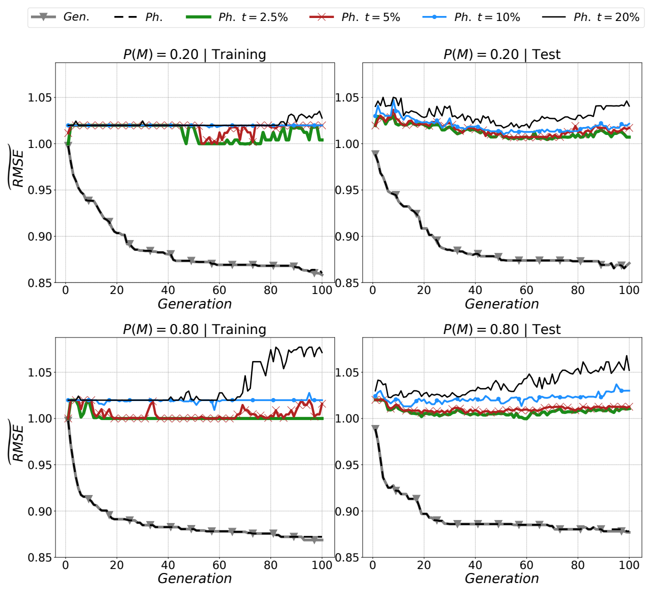

Figure 7: Median fitness of elite individuals. The fitness of phenotypes deteriorates with approximation.

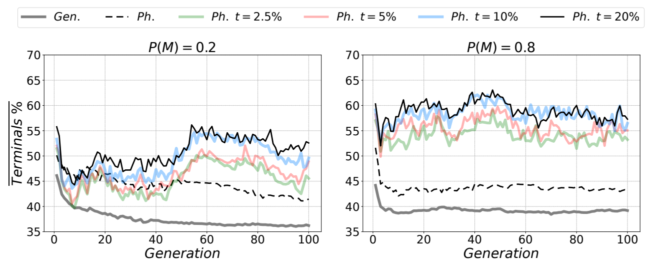

Figure 8: Terminal nodes percentage in the elite trees. Being smaller, phenotypes have a higher percentage of terminals.

- Later in the search, the proportion of terminals stabilizes around 35-40%.
- As was previously observed in other figures, lines tend to follow an order as a function of the approximation degree.
- The rougher the approximation, the larger the proportion of terminals to functions. This is expected since rougher approximation results in smaller phenotypic trees, with a smaller number of levels and fewer function nodes.
- A larger proportion of terminals can be observed with a higher mutation rate (subplot on the right). We consider this to be related to a larger simplification observed for this setting (see Figure 3).

Figure 9 provides an example of the exact ($t = 0$) and approximated ($t = \{2.5, 5, 10, 20\}$) phenotype(s) automatically extracted using our approach for the elite observed at generation 95 of run 2 for the Slump problem. The genotype (G) is represented in the gray box. The exact phenotype is represented in the upper right box ($P_{\{0,\,2.5\}}$), obtained by removing the red subtree, due to zero constant multiplication (nodes circled red). This happens to be the same phenotype as for the approximate phenotype with $t = 2.5\%$ which

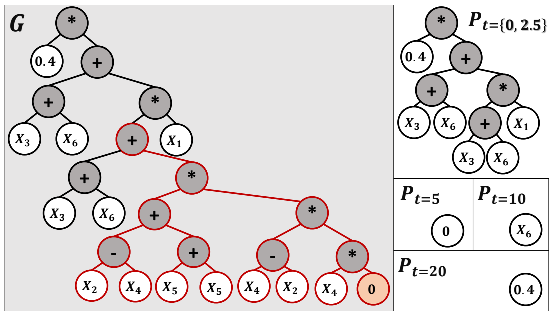

Figure 9: Genotype and exact/approximated phenotypes with our approach on the Slump problem. See text for details.

did not produce a smaller tree. The rougher approximations consist of single-node trees, which are of small utility for this application.

5 CONCLUSIONS

Traditionally, only the genotype has been considered in Tree Genetic Programming (TGP). Here we study the notion of phenotype and its relationship to the genotype in TGP to clarify what evolved solutions truly predict. We developed a unique simplification method that maps a genotype to its phenotype by removing semantically ineffective code from the genotype. This mapping shows that smaller phenotypes are hidden within larger genotypic trees but can be extracted via simplification, achieving a size reduction of up to 20 folds, which facilitates understanding. We also study how simplification affects population dynamics on a set of five real-world problems, averaging over effects of individual problems. Particularly, analysis of the diversity suggests that the genotype population is normally based on a small number of unique phenotypes, even when evolution is more explorative.

When we extract approximate phenotypes, defined by behavior that is in the statistical neighborhood of the target behavior, we observe larger semantic perturbations in the early stages of evolution. In the end, however, approximated phenotypes allow for better population fitness and smaller phenotypes, although elite solutions show deterioration when approximated. Phenotypes have not been explore in Tree GP to date but can provide valuable insights into evolutionary dynamics and hopefully help in the development of new bloat control methods. It might also lead to more efficient simplification algorithm and just generally lead the way to improved evolutionary search algorithms.

ACKNOWLEDGEMENTS

We gratefully ackowledge funding from the John R. Koza endowment to Michigan State University. MSU's High Performance Computing Center provided the compute power for this research.

REFERENCES

[1] Qurrat Ul Ain, Harith Al-Sahaf, Bing Xue, and Mengjie Zhang. 2022. Genetic programming for automatic skin cancer image classification. *Expert Systems with Applications* 197 (2022), 116680. https://doi.org/10.1016/j.eswa.2022.116680

[2] Eva Alfaro-Cid, JJ Merelo, F Fernández de Vega, Anna Isabel Esparcia-Alcázar, and Ken Sharman. 2010. Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation* 18, 2 (2010), 305–332.

[3] Lee Altenberg. 1994. The Evolution of Evolvability in Genetic Programming. In *Advances in Genetic Programming, Volume 1*. The MIT Press, 47–74. https://doi.org/10.7551/mitpress/1108.003.0008 arXiv:https://direct.mit.edu/book/chapter-pdf/1962544/9780262277181_c000300.pdf

[4] Peter J. Angeline. 1994. *Genetic Programming and Emergent Intelligence.* MIT Press, Cambridge, MA, USA, 75–97.

[5] Francesco Archetti, Stefano Lanzeni, Enza Messina, and Leonardo Vanneschi. 2006. Genetic Programming for Human Oral Bioavailability of Drugs *(GECCO '06)*. Association for Computing Machinery, New York, NY, USA, 255–262. https://doi.org/10.1145/1143997.1144042

[6] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. 2019. DENSER: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines* 20, 1 (March 1 2019), 5–35. https://doi.org/10.1007/s10710-018-9339-y

[7] Illya Bakurov, Marco Buzzelli, Mauro Castelli, Leonardo Vanneschi, and Raimondo Schettini. 2021. General Purpose Optimization Library (GPOL): A Flexible and Efficient Multi-Purpose Optimization Library in Python. *Applied Sciences* 11, 11 (2021). https://doi.org/10.3390/app11114774

[8] Illya Bakurov, Marco Buzzelli, Raimondo Schettini, Mauro Castelli, and Leonardo Vanneschi. 2023. Full-Reference Image Quality Expression via Genetic Programming. *IEEE Transactions on Image Processing* 32 (2023), 1458–1473. https://doi.org/10.1109/TIP.2023.3244662

[9] Illya Bakurov, Marco Buzzelli, Raimondo Schettini, Mauro Castelli, and Leonardo Vanneschi. 2023. Semantic segmentation network stacking with genetic programming. *Genetic Programming and Evolvable Machines* 24, 2 (Dec. 2023), Article number: 15. https://doi.org/doi:10.1007/s10710-023-09464-0 Special Issue on Highlights of Genetic Programming 2022 Events.

[10] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. 1998. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[11] Wolfgang Banzhaf, Ting Hu, and Gabriela Ochoa. 2024. How the Combinatorics of Neutral Spaces Leads Genetic Programming to Discover Simple Solutions. *Genetic Programming Theory and Practice XX* (2024), 1–22.

[12] Wolfgang Banzhaf and William B. Langdon. 2002. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines* 3 (2002), 81–91.

[13] Tobias Blickle and Lothar Thiele. 1994. Genetic Programming and Redundancy. In *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, J. Hopf (Ed.). Max-Planck-Institut für Informatik (MPI-I-94-241), Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, 33–38. http://www.tik.ee.ethz.ch/~tec/publications/bt94/GPandRedundancy.ps.gz

[14] M.F. Brameier and W. Banzhaf. 2007. *Linear Genetic Programming.* Springer US. https://books.google.com/books?id=AhZJ9SIChnQC

[15] Mauro Castelli and Luca Manzoni. 2019. GSGP-C++ 2.0: A geometric semantic genetic programming framework. *SoftwareX* 10 (2019), 100313. https://doi.org/10.1016/j.softx.2019.100313

[16] João Correia, Leonardo Vieira, Nereida Rodriguez-Fernandez, Juan Romero, and Penousal Machado. 2021. Evolving Image Enhancement Pipelines. In *Artificial Intelligence in Music, Sound, Art and Design*, Juan Romero, Tiago Martins, and Nereida Rodríguez-Fernández (Eds.). Springer International Publishing, Cham, 82–97.

[17] João Correia, Daniel Lopes, Leonardo Vieira, Nereida Rodriguez-Fernandez, Adrian Carballal, Juan Romero, and Penousal Machado. 2022. Experiments in Evolutionary Image Enhancement with ELAINE. *Genetic Programming and Evolvable Machines* 23, 4 (dec 2022), 557–579. https://doi.org/10.1007/s10710-022-09445-9

[18] Kévin Cortacero, Brienne McKenzie, Sabina Müller, Roxana Khazen, Fanny Lafouresse, Gaëlle Corsaut, Nathalie Van Acker, François-Xavier Frenois, Laurence Lamant, Nicolas Meyer, Béatrice Vergier, Dennis G. Wilson, Hervé Luga, Oskar Staufer, Michael L. Dustin, Salvatore Valitutti, and Sylvain Cussat-Blanc. 2023. Evolutionary design of explainable algorithms for biomedical image segmentation. *Nature Communications* 14, 1 (2023), 7112. https://doi.org/10.1038/s41467-023-42664-x

[19] Sylvain Cussat-Blanc, Kyle Harrington, and Wolfgang Banzhaf. 2019. Artificial gene regulatory networks—a review. *Artificial life* 24, 4 (2019), 296–328.

[20] Francisco Fernández de Vega, Gustavo Olague, Francisco Chávez, Daniel Lanza, Wolfgang Banzhaf, and Erik Goodman. 2020. It is time for new perspectives on how to fight bloat in GP. *Genetic Programming Theory and Practice XVII* (2020), 25–38.

[21] Francisco Fernández de Vega, Gustavo Olague, Daniel Lanza, Wolfgang Banzhaf, Erik Goodman, Jose Menendez-Clavijo, Axel Martinez, et al. 2020. Time and individual duration in genetic programming. *IEEE Access* 8 (2020), 38692–38713.

[22] Dominique de Vienne. 2022. What is a phenotype? History and new developments of the concept. *Genetica* 150, 3 (2022), 153–158.

[23] Stephen Dignum and Riccardo Poli. 2007. Generalisation of the Limiting Distribution of Program Sizes in Tree-Based Genetic Programming and Analysis of Its Effects on Bloat. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, England) *(GECCO '07)*. Association for Computing Machinery, New York, NY, USA, 1588–1595. https://doi.org/10.1145/1276958.1277277

[24] Stephen Dignum and Riccardo Poli. 2008. Crossover, Sampling, Bloat and the Harmful Effects of Size Limits. In *Genetic Programming*, Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcázar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 158–169.

[25] Stephen Dignum and Riccardo Poli. 2008. Operator Equalisation and Bloat Free GP. In *European Conference on Genetic Programming*. https://api.semanticscholar.org/CorpusID:1863814

[26] Benjamin P. Evans, Bing Xue, and Mengjie Zhang. 2019. What's inside the Black-Box? A Genetic Programming Method for Interpreting Complex Machine Learning Models *(GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 1012–1020. https://doi.org/10.1145/3321707.3321726

[27] Ivo Goncalves, Sara Silva, and Carlos M. Fonseca. 2015. Semantic Learning Machine: A Feedforward Neural Network Construction Algorithm Inspired by Geometric Semantic Genetic Programming. In *Progress in Artificial Intelligence - 17th Portuguese Conference on Artificial Intelligence, EPIA 2015 (Lecture Notes in Computer Science, Vol. 9273)*, Francisco C. Pereira, Penousal Machado, Ernesto Costa, and Amilcar Cardoso (Eds.). Springer, Coimbra, Portugal, 280–285. https://doi.org/doi:10.1007/978-3-319-23485-4_28

[28] David Harrison and Daniel L Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management* 5, 1 (1978), 81–102. https://doi.org/10.1016/0095-0696(78)90006-2

[29] Nathan Haut, Bill Punch, and Wolfgang Banzhaf. 2023. Active Learning Informs Symbolic Regression Model Development in Genetic Programming. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (Lisbon, Portugal) *(GECCO '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 587–590. https://doi.org/10.1145/3583133.3590577

[30] Thomas Helmuth, Nicholas Freitag McPhee, Edward Pantridge, and Lee Spector. 2017. Improving generalization of evolved programs through automatic simplification. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Berlin, Germany) *(GECCO '17)*. Association for Computing Machinery, New York, NY, USA, 937–944. https://doi.org/10.1145/3071178.3071330

[31] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (2015), 630–643. https://doi.org/10.1109/TEVC.2014.2362729

[32] Vijay Ingalalli, Sara Silva, Mauro Castelli, and Leonardo Vanneschi. 2014. A Multi-dimensional Genetic Programming Approach for Multi-class Classification Problems. In *Genetic Programming*, Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo García-Sánchez, Juan J. Merelo, Victor M. Rivas Santos, and Kevin Sim (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 48–60.

[33] Muhammad Iqbal, Bing Xue, Harith Al-Sahaf, and Mengjie Zhang. 2017. Cross-Domain Reuse of Extracted Knowledge in Genetic Programming for Image Classification. *IEEE Transactions on Evolutionary Computation* 21, 4 (2017), 569–587. https://doi.org/10.1109/TEVC.2017.2657556

[34] David Jackson. 2010. The Identification and Exploitation of Dormancy in Genetic Programming. *Genetic Programming and Evolvable Machines* 11, 1 (mar 2010), 89–121. https://doi.org/10.1007/s10710-009-9086-1

[35] Noman Javed, Fernand Gobet, and Peter Lane. 2022. Simplification of genetic programs: a literature survey. *Data Mining and Knowledge Discovery* 36, 4 (07 2022), 1279–1300. https://doi.org/10.1007/s10618-022-00830-7

[36] David Kinzett, Mark Johnston, and Mengjie Zhang. 2009. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence* 2 (2009), 151–168.

[37] Marc W Kirschner and John C Gerhart. 2005. *The plausibility of life: Resolving Darwin's dilemma.* Yale University Press.

[38] J.R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

[39] William La Cava, Sara Silva, Kourosh Danai, Lee Spector, Leonardo Vanneschi, and Jason H. Moore. 2019. Multidimensional genetic programming for multiclass classification. *Swarm and Evolutionary Computation* 44 (2019), 260–272. https://doi.org/10.1016/j.swevo.2018.03.015

[40] William La Cava, Lee Spector, and Kourosh Danai. 2016. Epsilon-Lexicase Selection for Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (Denver, Colorado, USA) *(GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 741–748. https://doi.org/10.1145/2908812.2908898

[41] W. B. Langdon and R. Poli. 1998. Fitness Causes Bloat. In *Soft Computing in Engineering Design and Manufacturing*, P. K. Chawdhry, R. Roy, and R. K. Pant

(Eds.). Springer London, London, 13–22.

[42] W. B. Langdon and R. Poli. 1998. Fitness causes bloat: Mutation. In *Genetic Programming*, Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 37–48.

[43] William B. Langdon and Riccardo Poli. 2002. *Foundations of Genetic Programming.* https://api.semanticscholar.org/CorpusID:13348347

[44] W. B. Langdon, T. Soule, R. Poli, and J. A . Foster. 1999. The Evolution of Size and Shape. In *Advances in Genetic Programming, Volume 3.* The MIT Press. https://doi.org/10.7551/mitpress/1110.003.0012 arXiv:https://direct.mit.edu/book/chapter-pdf/2183015/9780262284127_cah.pdf

[45] Andrew Lensen, Bing Xue, and Mengjie Zhang. 2022. Genetic Programming for Manifold Learning: Preserving Local Topology. *IEEE Transactions on Evolutionary Computation* 26, 4 (2022), 661–675. https://doi.org/10.1109/TEVC.2021.3106672

[46] Sean Luke and Liviu Panait. 2002. Fighting Bloat with Nonparametric Parsimony Pressure. In *Parallel Problem Solving from Nature — PPSN VII*, Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, Hans-Paul Schwefel, and José-Luis Fernández-Villacañas (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 411–421.

[47] Aniek F. Markus, Jan A. Kors, and Peter R. Rijnbeek. 2021. The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and evaluation strategies. *Journal of Biomedical Informatics* 113 (2021), 103655. https://doi.org/10.1016/j.jbi.2020.103655

[48] James McDermott, Edgar Galván-Lopéz, and Michael O'Neill. 2011. *A Fine-Grained View of Phenotypes and Locality in Genetic Programming.* Springer New York, New York, NY, 57–76. https://doi.org/10.1007/978-1-4614-1770-5_4

[49] Nicholas Freitag McPhee and Justin Darwin Miller. 1995. Accurate Replication in Genetic Programming. In *Proceedings of the 6th International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 303–309.

[50] Melanie Mitchell, John Holland, and Stephanie Forrest. 1993. When will a Genetic Algorithm Outperform Hill Climbing. In *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector (Eds.), Vol. 6. Morgan-Kaufmann. https://proceedings.neurips.cc/paper_files/paper/1993/file/ab88b15733f543179858600245108dd8-Paper.pdf

[51] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. 2012. Geometric Semantic Genetic Programming. In *Parallel Problem Solving from Nature - PPSN XII*, Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 21–31.

[52] Naoki Mori, Bob McKay, Nguyen Xuan Hoai, Daryl Essam, and Saori Takeuchi. 2008. A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. In *SCIS & ISIS SCIS & ISIS 2008.* Japan Society for Fuzzy Theory and Intelligent Informatics, 1671–1676.

[53] Luis Muñoz, Sara Silva, and Leonardo Trujillo. 2015. M3GP – Multiclass Classification with GP. In *Genetic Programming*, Penousal Machado, Malcolm I. Heywood, James McDermott, Mauro Castelli, Pablo García-Sánchez, Paolo Burelli, Sebastian Risi, and Kevin Sim (Eds.). Springer International Publishing, Cham, 78–91.

[54] Kourosh Neshatian, Mengjie Zhang, and Peter Andreae. 2012. A Filter Approach to Multiple Feature Construction for Symbolic Learning Classifiers Using Genetic Programming. *IEEE Transactions on Evolutionary Computation* 16, 5 (2012), 645–661. https://doi.org/10.1109/TEVC.2011.2166158

[55] Quang Uy Nguyen and Thi Huong Chu. 2020. Semantic approximation for reducing code bloat in genetic programming. *Swarm and Evolutionary Computation* 58 (2020), 100729.

[56] Peter Nordin and W. Banzhaf. 1995. Complexity Compression and Evolution. In *International Conference on Genetic Algorithms.* 310–317. https://api.semanticscholar.org/CorpusID:16415863

[57] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. 1996. Explicitly Defined Introns and Destructive Crossover in Genetic Programming. In *Advances in Genetic Programming, Volume 2.* The MIT Press. https://doi.org/10.7551/mitpress/1109.003.0010 arXiv:https://direct.mit.edu/book/chapter-pdf/2186519/9780262290791_caf.pdf

[58] Randal S. Olson and Jason H. Moore. 2019. *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning.* Springer International Publishing, Cham, 151–160. https://doi.org/10.1007/978-3-030-05318-5_8

[59] Riccardo Poli. 2003. A simple but theoretically-motivated method to control bloat in genetic programming. In *European Conference on Genetic Programming.* Springer, 204–217.

[60] Riccardo Poli, Nicholas Freitag McPhee, and Leonardo Vanneschi. 2008. Elitism Reduces Bloat in Genetic Programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (Atlanta, GA, USA) *(GECCO '08).* Association for Computing Machinery, New York, NY, USA, 1343–1344. https://doi.org/10.1145/1389095.1389355

[61] Peter Rockett. 2020. Pruning of genetic programming trees using permutation tests. *Evolutionary Intelligence* 13, 4 (2020), 649–661. https://doi.org/doi:10.1007/s12065-020-00379-8

[62] Sara Silva. 2011. Reassembling Operator Equalisation: A Secret Revealed. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11).* Association for Computing Machinery, New York, NY, USA, 1395–1402. https://doi.org/10.1145/2001576.2001764

[63] Sara Silva and Ernesto Costa. 2009. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines* 10 (2009), 141–179. https://api.semanticscholar.org/CorpusID:10925054

[64] Sara Silva and Leonardo Vanneschi. 2009. Operator Equalisation, Bloat and Overfitting: A Study on Human Oral Bioavailability Prediction. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (Montreal, Québec, Canada) *(GECCO '09).* Association for Computing Machinery, New York, NY, USA, 1115–1122. https://doi.org/10.1145/1569901.1570051

[65] Sara Silva and Leonardo Vanneschi. 2012. Bloat Free Genetic Programming: Application to Human Oral Bioavailability Prediction. *Int. J. Data Min. Bioinformatics* 6, 6 (nov 2012), 585–601. https://doi.org/10.1504/IJDMB.2012.050266

[66] Terence Soule and Robert B. Heckendorn. 2002. An Analysis of the Causes of Code Growth in Genetic Programming. *Genetic Programming and Evolvable Machines* 3, 3 (Sept. 2002), 283–309. https://doi.org/doi:10.1023/A:1020115409250

[67] Lee Spector and Thomas Helmuth. 2014. Effective simplification of evolved push programs using a simple, stochastic hill-climber. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation.* 147–148.

[68] Walter Alden Tackett. 1994. *Recombination, Selection, and the Genetic Construction of Computer Programs.* Ph. D. Dissertation. USA.

[69] Binh Tran, Bing Xue, and Mengjie Zhang. 2019. Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recognition* 93 (2019), 404–417. https://doi.org/10.1016/j.patcog.2019.05.006

[70] Athanasios Tsanas and Angeliki Xifara. 2012. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings* 49 (2012), 560–567. https://doi.org/10.1016/j.enbuild.2012.03.003

[71] Qurrat Ul Ain, Bing Xue, Harith Al-Sahaf, and Mengjie Zhang. 2017. Genetic programming for skin cancer detection in dermoscopic images. In *2017 IEEE Congress on Evolutionary Computation (CEC)* (Donostia, San Sebastián, Spain). IEEE Press, 2420–2427. https://doi.org/10.1109/CEC.2017.7969598

[72] Heidi Vainio-Pekka, Mamia Ori-Otse Agbese, Marianna Jantunen, Ville Vakkuri, Tommi Mikkonen, Rebekah Rousi, and Pekka Abrahamsson. 2023. The Role of Explainable AI in the Research Field of AI Ethics. *ACM Trans. Interact. Intell. Syst.* 13, 4, Article 26 (dec 2023), 39 pages. https://doi.org/10.1145/3599974

[73] Peter A Whigham and Grant Dick. 2009. Implicitly controlling bloat in genetic programming. *IEEE Transactions on Evolutionary Computation* 14, 2 (2009), 173–190.

[74] Phillip Wong and Mengjie Zhang. 2006. Algebraic simplification of GP programs during evolution. In *Proc. 8th Annual Conference on Genetic and Evolutionary Computation (GECCO-2006).* ACM Press, New York, 927–934.

[75] I-Cheng Yeh. 2007. Concrete Compressive Strength. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5PK67.

[76] I-Cheng Yeh. 2009. Concrete Slump Test. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5FG7D.

[77] Mengjie Zhang and Will Smart. 2004. Multiclass Object Classification Using Genetic Programming. In *Applications of Evolutionary Computing*, Günther R. Raidl, Stefano Cagnoni, Jürgen Branke, David Wolfe Corne, Rolf Drechsler, Yaochu Jin, Colin G. Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.