Genetic Programming with Epigenetic Local Search

William La Cava*
Department of Mechanical and Industrial Engineering
University of Massachusetts
Amherst, MA 01003
wlacava@umass.edu

Thomas Helmuth
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
thelmuth@umass.edu

Lee Spector
School of Cognitive Science
Hampshire College
Amherst, MA 01002
lspector@hampshire.edu

Kourosh Danai
Department of Mechanical and Industrial Engineering
University of Massachusetts
Amherst, MA 01003
danai@ecs.umass.edu

ABSTRACT

We focus on improving genetic programming through local search of the space of program structures using an inheritable epigenetic layer that specifies active and inactive genes. We explore several genetic programming implementations that represent the different properties that epigenetics can provide, such as passive structure, phenotypic plasticity, and inheritable gene regulation. We apply these implementations to several symbolic regression and program synthesis problems. For the symbolic regression problems, the results indicate that epigenetic local search consistently improves genetic programming by producing smaller solution programs with better fitness. Furthermore, we find that incorporating epigenetic modification as a mutation step in program synthesis problems can improve the ability of genetic programming to find exact solutions. By analyzing population homology we show that the epigenetic implementations maintain diversity in silenced portions of programs which may provide protection from premature convergence.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis*; J.2 [**Computer Applications**]: Physical Science and Engineering—*Engineering*

Keywords

genetic programming, epigenetics, regression, program synthesis

*corresponding author

GECCO '15, July 11 - 15, 2015, Madrid, Spain

ISBN 978-1-4503-3472-3/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739480.2754763

1. INTRODUCTION

In genetic programming (GP), the role of epigenetics, that is, regulation of gene expression, is traditionally ignored (with some exceptions, e.g. [7]), despite the fact that the expression of biological genes is highly regulated. It has recently become clear that epigenetic processes can provide several evolutionary benefits. For example, they allow the underlying genotype to encode various expressions and allow neutral variation through crossover and mutation of non-coding segments. This neutrality may allow populations to avoid evolutionary bottlenecks or allow them to respond to changing evolutionary pressures [14]. In addition, they provide for phenotypic plasticity, i.e. the ability of gene expression to change in response to environmental pressure [5]. Furthermore, epigenetics allow adaptations to gene expression to be inherited in offspring without explicit changes to the genotype. Indeed, it is now accepted that the previously discredited ideas of Lamarck, i.e. the inheritability of lifetime adaptations, are possible through epigenetic processes [14, 12]. Hence epigenetic processes may provide evolutionary benefits by performing "local search" for survival. We investigate that hypothesis by designing an epigenetics-enabled GP system to perform experiments for analysis.

Local search methods have been developed and integrated into evolutionary algorithms [10, 35, 23, 9], especially in genetic algorithms (GAs), through prescribed changes to the genotype. In GP, especially within the field of symbolic regression, the role of structure optimization is typically left to the GP process while local search is constrained to constant optimization via stochastic hill-climbing [3], linear [13] or non-linear regression [30]. While this often improves symbolic regression performance, the methods are inherently limited to problem domains that require constant optimization and cannot be readily applied to other problems like software synthesis, nor can they improve program topology. Other more generic local search methods like tree snipping [3] focus on improving secondary metrics like size or legibility. In these cases the local search is conducted at the genome level.

Our study of the application of epigenetics to GP is motivated by two main hypotheses: first, that generalized local search methods can improve the performance of GP across

problem domains; and second, that the benefits of epigenetic regulation observed in biology may confer analogous improvements on GP systems. Thus, we propose a new method for conducting topological optimization of genetic programs at level of gene expression via epigenetic local search. The contributions of this method are twofold: first, it introduces a generic method of topological search of the space of individual genotypes via modifications to gene expression. Second, unlike previous methods that focused on whether lifetime adaptations should be inheritable (the Lamarckian view) or only affect fitness (the Baldwinian view), epigenetic local search improves genetic programs without affecting the genotype *and* without discarding the acquired knowledge gained through the process. It does this by conducting local search on the epigenome rather than the genome and making these adaptations inheritable via evolutionary processes.

We begin by describing the method and its use in two stack-based GP environments. We then perform an experimental analysis of different epigenetic implementations on regression and program synthesis benchmarks. Finally we analyze population diversity to study how gene expression evolves for each implementation.

2. EPIGENETIC LINEAR GENETIC PROGRAMMING (ELGP)

We introduce epigenetic information into the GP representation by including an on/off condition on each element in an individual's genotype. Programs are encoded as postfix notation, linear genotypes with a corresponding set of on/off values, referred to as an epigenome. When evaluated together, the expressed program, i.e. phenotype, is produced by executing instructions that are on (active) and ignoring the instructions that are off (inactive). We refer to this general approach as epigenetic linear genetic programming (ELGP).

2.1 GP Representation

In order to accommodate the introduction of epigenetics, two stack-based GP systems are used. For the symbolic regression problems, a GP system known as ellenGP[1] is used. For the program synthesis problems, we implement an epigenetic extension to PushGP [28]. The stack-based GP systems used in our research are advantageous because the representation guarantees syntactic validity for arbitrary sequences of instructions. This allows instructions to be silenced or activated in a genotype without invalidating the program's ability to execute. This "syntax-free" property stands in contrast to tree-based representations that can become syntactically invalid due to changes to instructions and literals.

The syntactic robustness of the stack-based approach is achieved mainly by ignoring the execution of instructions that have an arity larger than the current size of the stack. For example, if a $+$ operator attempts to execute and there is only one element on the stack, it instead does nothing. Furthermore, we base a program's behavior only on the top element(s) of the stack after execution which allows programs to contain unused arguments. The flexibility of this representation means that the genotypes of the following three programs $\mathbf{i_1}$, $\mathbf{i_2}$ and $\mathbf{i_3}$ all produce the identical phenotype $(x + y)$:

$$\begin{aligned}
\mathbf{i_1} &= [\ x\ \ y\ \ +\ ] &\Rightarrow\ \ (x+y)\\
\mathbf{i_2} &= [\ x\ \ y\ \ +\ \ -\ \ *\ \ /\ ] &\Rightarrow\ \ (x+y)\\
\mathbf{i_3} &= [\ z\ \ +\ \ x\ \ /\ \ x\ \ y\ \ +\ ] &\Rightarrow\ \ (x+y)
\end{aligned} \tag{1}$$

The executions of $-$, $*$ and $/$ in $\mathbf{i_2}$ are ignored due to insufficient stack size; in $\mathbf{i_3}$, the last element of the executed stack, $(x + y)$, is taken as the phenotype.

2.2 Epigenetic Learning and Evolution

The addition of epigenetic information makes it possible to alter the topology of and values expressed in the phenotype. For example, program $\mathbf{i_3}$ admits several phenotypes via epigenetic transformations, including,

$$\mathbf{i_3} \rightarrow \mathbf{i_3'} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ z & + & x & / & x & y & + \end{bmatrix} \Rightarrow (z+y)$$

$$\mathbf{i_3} \rightarrow \mathbf{i_3''} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ z & + & x & / & x & y & + \end{bmatrix} \Rightarrow (z/x)$$

$$\mathbf{i_3} \rightarrow \mathbf{i_3'''} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ z & + & x & / & x & y & + \end{bmatrix} \Rightarrow (z/x+y)$$

Similarly, program $\mathbf{i_2}$ in Eq. 1 admits the phenotypes $(x + y)$, $(x - y)$, $(x * y)$, and (x/y) via epigenetic transformations. Thus the epigenetic layer introduces flexibility to the expression of a genotype by treating introns (i.e. unexpressed genes) as partial solutions to explore locally in the search space.

The components of ELGP are shown in Figure 1. The epigenetic markers are initialized randomly with a probability of being active (50% for the experimental studies in Section 4). The extent to which epigenetic information is learned and inherited is a research question we study by exploring different implementations. The simplest topological search method, **Ep1M**, mutates the epigenetic layer of each individual each generation; the hill climber in Figure 1 is skipped. Thus for **Ep1M**, epigenetic mutations face only evolutionary pressures. In contrast, the epigenetic hill climbing (EHC) cases **EHC1** and **EHC5** use the epigenetic information explicitly to improve individuals each generation (the EHC is described in Section 2.2.2). The two methods execute one and five iterations of EHC each generation, respectively. Two control methods, **Baseline** and **Ep0**, are used as comparison. In the **Baseline** case, individuals are represented as basic genotypes as in Eq. 1. The **Ep0** case acts like **Baseline** but with half of the genes in initial code permanently silenced. As such **Ep0** controls for the effect that passive introns might have. Neither **Baseline** or **Ep0** use the right half of the system in Figure 1 (i.e. the program never enters epigenetic mutation).

2.2.1 Epigenetic Mutation

Whitley et al. [35] introduced Lamarckian updating to GAs by conducting local search of the bit strings within 1 Hamming distance of the current bit string. In theory it would be possible to treat the epigenome as a bit string and proceed similarly. However the cost of GP fitness evaluations render this approach intractable. Instead, each generation, the epigenome is uniformly mutated with a probability of 10% at each gene. The mutation flips the binary value of the epigenome at the gene, thus activating or silencing that gene. The operation is uniform with respect to the number

[1] http://www.github.com/lacava/ellen

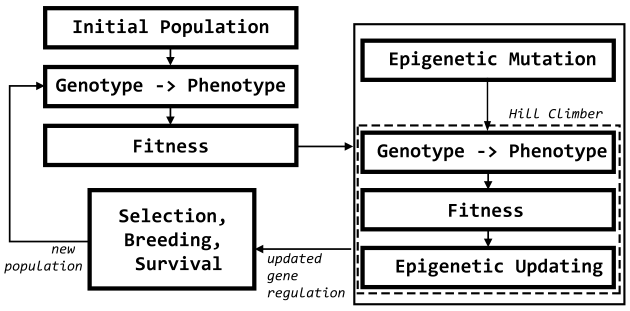

Figure 1: Block diagram of ELGP. After fitness evaluation and before selection, the population undergoes an iteration of epigenetic mutation, represented by the block on the right. For EHC1 and EHC5, the hill climbing step is then conducted.

of instructions. For the **EHC1** and **EHC5** implementations, the epigenetic mutation is followed by hill climbing, described next.

2.2.2 Epigenetic Hill Climbing

In order to mimic the acquisition of lifetime learning by epigenetic adaptation, the EHC implementations evaluate epigenetic changes $\mathbf{i} \rightarrow \mathbf{i}'$ to determine whether individuals should be updated. **EHC1** and **EHC5** undergo one and five iterations of epigenetic mutation, and at each iteration the epigenetic changes are tested for acceptance.

For the program synthesis problems, epigenetic changes to an individual are kept only if the fitness is improved or does not change, i.e. $f_{\mathbf{i}'} \leq f_{\mathbf{i}}$ (fitness f is being minimized). For the symbolic regression problems, we exploit the epigenetic layer's potential to reduce equation nonlinearity in order to break fitness ties. The complexity $C_{\mathbf{i}}$ of program $\mathbf{i}$ with active genotype $\mathbf{g_a} = \left[\begin{array}{ccc} g_{a_1} & \cdots & g_{a_\ell} \end{array} \right]$ is defined as $C_{\mathbf{i}} = \sum_{q=1}^{\ell} c(g_{a_q})$, where component function nonlinearities [26] are defined as

$$c(g_a) = \begin{cases} 4 & : \quad (g_a = \log) \vee (g_a = \exp) \\ 3 & : \quad (g_a = \sin) \vee (g_a = \cos) \\ 2 & : \quad (g_a = /) \\ 1 & : \quad \text{otherwise} \end{cases}$$

Lower-complexity programs with equivalent fitness are accepted, giving the condition

$$pass = (f_{\mathbf{i}'} < f_{\mathbf{i}}) \vee ((f_{\mathbf{i}'} = f_{\mathbf{i}}) \wedge (C_{\mathbf{i}'} < C_{\mathbf{i}})) \qquad (2)$$

If the epigenetically mutated individual $\mathbf{i}'$ does not pass Eq. 2, the changes are discarded and $\mathbf{i}$ is kept in the population. Otherwise $\mathbf{i}$ is replaced with $\mathbf{i}'$.

2.2.3 Epigenetic Inheritance

A key feature of ELGP is the inheritance of epigenetic values throughout the evolutionary process. During crossover the epigenetic values of the parent genes are kept intact such that the child receives the epigenetic states of the genes it has inherited. If a new gene is introduced via genetic mutation, that gene has the same probability of being active as the initial genes of the population (50% here).

3. RELATED WORK

There has been some work to incorporate epigenetic learning into GP, notably by Tanev [29]. In this case the focus was to model histone modification through a double cell representation and use it to solve a predator-prey problem. Unlike our approach, Tanev did not treat lifetime epigenetic modifications as inheritable, despite recent studies that support this idea in biology [31, 15, 6].

There have been a number of studies on the effects of non-coding segments in GP, some of which have found that the structural presence of introns can protect genotypes from destructive crossover operations (i.e. operations that produce children less fit than their parents) [21, 4]. Non-coding segments were found to be useful in evo-devo for evolution of arbitrary shapes as well [8]. In each of these studies, introns were either declared explicitly or measured during evolution, rather than actively manipulated by the system itself. Our preliminary study of epigenetic initialization found rates of beneficial crossover to be highest with the probability set to 50% [18].

Several GP systems use similar stack-based or linear genome representations such as Push-forth [17] and Gene Expression Programming [7] that could trivially implement the epigenetic layer suggested in this paper. In addition, there are methods that leverage neutrality (i.e. different genotypes with the same fitness) by creating a genotype - phenotype mapping, for example Cartesian GP [20] and Binary GP [2]. Rather than redefining the GP representation completely, our goal with ELGP is to explore whether local search of gene expression can be a viable, generic GP extension; hence its application to two different GP systems in this paper. As mentioned earlier, there are a plethora of studies on local search methods for improving GP by Lamarckian or Baldwinian means, yet very few that have considered these changes to occur at the epigenetic level rather than the genotype level. A notable exception is Multiple Regression GP [1], in which parameter values are implied at each node location and updated by linear regression. Still, the tangible improvements brought about by this and most other local search methods for symbolic regression are achieved by parametric, rather than topological, search.

4. EXPERIMENTAL ANALYSIS

4.1 Symbolic Regression with ellenGP

4.1.1 Algorithm Settings

In order to make our results relevant to state-of-the-art symbolic regression tools that often leverage Pareto optimization [26, 24], we use age-fitness Pareto survival [25], in which each individual is assigned an age equal to the number of generations since its oldest ancestor was created. Each generation, a new individual is introduced to the population as a means of random restart. Selection for breeding is random, and during breeding we create a number of children equal to the overall population size. At the end of each generation, we conduct culling tournaments of size 2 to reduce the set consisting of the current population and the newly created children down to the population size. In each tournament, we remove an individual from the population if their competitor Pareto-dominates them by the metrics of age (younger is better) and fitness. Culling continues until the population reaches its original size or until it conducts a

Table 1: Symbolic regression problem settings.

Setting		Value
Population size		1000
Crossover / Mutation		80/20%
Program length limits		[3, 50]
ERC range		[-10,10]
Termination criterion		max point evals or $f < 1.0\text{E-}6$
Trials		50 Tower & Keijzer-6, 100 rest
Problem	Terminal Set	Point Evals
Pagie-1	$\{x, y, +, -, *, /, 1.0\}$	2.5E+10
Nguyen-7	$\{x, +, -, *, /,$ exp, log, ERC$\}$	2.5E+10
Uball5D	$\{x_1 \ \ldots \ x_5, +, -, *, /,$ ERC$\}$	2.5E+10
Keijzer-6	$\{x, +, -, *, /,$ exp, log, ERC$\}$	2.5E+10
Tower	$\{x_1 \ \ldots \ x_{25}, +, -, *, /,$ sin, cos, exp, log, ERC$\}$	1.0E+11

maximum number of tournaments (10 times the population size in our implementation), at which point individuals with the worst fitness are removed.

The fitness metric is designed to minimize error and maximize correlation so that both mean absolute error (MAE) and the closeness of the output and target shapes, i.e. the coefficient of determination (R^2), can be compared in the results. For target $\mathbf{y}^*$ and output $\mathbf{y_i}$ $f_\mathbf{i}$ is defined as:

$$f_\mathbf{i} = \frac{1}{N}\sum_{q=1}^{N} |\mathbf{y}^*(k_q) - \mathbf{y_i}(k_q)|/R^2_\mathbf{i} \quad (3)$$

$$R^2_\mathbf{i} = \frac{(\mathrm{cov}(\mathbf{y}^*, \mathbf{y_i}))^2}{\mathrm{var}(\mathbf{y}^*)\mathrm{var}(\mathbf{y_i})} \quad (4)$$

Run-time settings for the algorithm are shown in Table 1. A one-point crossover operator is used to produce two children from two parents. The mutation operator is applied uniformly to the chosen parent with a probability of 2.5% at each gene. If a constant gene is picked for mutation and ephemeral random constants (ERCs) are being used, the constant is perturbed by Gaussian noise with standard deviation equal to half the magnitude of the constant. Otherwise the instruction is mutated to a randomly chosen gene.

For problems utilizing ERCs, one iteration of parameter hill climbing is conducted each generation. The hill climber perturbs all constant values in the active genotype by Gaussian noise with a standard deviation equal to 10% of the value of the constant. These changes are kept if they result in a lower fitness for the individual.

Each trial was allocated a maximum number of point evaluations, i.e. gene executions, to normalize for the different program sizes among methods. A GP run will exit early if the fitness condition $f < 10^{-6}$ is achieved before the designated number of point evaluations has been reached. In practice, this fitness termination condition was sufficient to guarantee exact solutions for the problems studied.

4.1.2 Optimizations

The following optimizations are applied to the ellenGP system in order to reduce the number of point evaluations required to evaluate the fitness of an individual that has undergone epigenetic mutation. The majority of run-time in most GP systems (including ours) is spent in fitness evaluation, thus motivating the need for methods that can reduce the number of point evaluations required.

Fitness Escape.

EHC requires additional fitness evaluations in order to determine whether the prescribed epigenetic changes will be kept. Given that the fitness $f_\mathbf{i}$ of program $\mathbf{i}$ monotonically increases with the evaluation of more fitness cases $k_{1\ldots n}$, evaluation of the epigenetically mutated individual $\mathbf{i}'$ can be halted if at any point the fitness $f_{\mathbf{i}'}(1\ldots k_j) > f_\mathbf{i}(1\ldots k_n)$ for $1 \leq j < n$. Since fitness is always equal to or larger than MAE (see Eq. 3), the halt condition can be defined conservatively using the mean absolute error (MAE) of $\mathbf{i}'$ and the fitness of $\mathbf{i}$ as

$$\frac{1}{N}\sum_{q=1}^{j} |\mathbf{y}^*(k_q) - \mathbf{y}_{\mathbf{i}'}(k_q)| > f_\mathbf{i} \quad (5)$$

Stack Tracing.

In GP tree representations, typically the output of a node in the program is dependent only on the outputs of its child nodes (and those children's children and so forth). We can say conservatively with ellenGP representations that no instruction in the stack is dependent on an instruction to its right. Therefore, when a gene is silenced or activated, only the outputs of the genes to its right in the genotype are affected, hence only part of the program needs to be reevaluated. To avoid repeated instruction evaluations during epigenetic hill climbing, we save the intermediate program outputs of each gene and after epigenetic mutation reevaluate only those genes to the right of the left-most location of mutation. Saving the stack outputs is a trade-off between memory and time resources since it requires more memory to save the intermediate outputs but requires fewer point evaluations to evaluate epigenetically mutated individuals. The trade-off is favorable in our implementation because processor resources are much more limited than memory resources. Similar partial evaluation strategies have been proposed, e.g. in [19].

4.1.3 Problems

Five problems were chosen from the benchmark suite suggested by White et. al. [34]. The first four problems have the following forms[2]:

$$\text{Pagie-1 [22]} : \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}} \quad (6)$$

$$\text{Nguyen-7 [32]} : \ln(x+1) + \ln(x^2+1) \quad (7)$$

$$\text{UBall5D [33]} : \frac{10}{5 + \sum_{i=1}^{5}(x_i - 3)^2} \quad (8)$$

$$\text{Keijzer-6 [16]} : \sum_{i}^{x} \frac{1}{i} \quad (9)$$

The settings for each problem are summarized in Table 1. Each of these problems use the training and validation sets provided in the original papers except for the Nguyen-7 validation set (not originally provided). We test validation on Nguyen-7 using 20 evenly spaced points from $x \in [2, 4]$. Keijzer-6 is unique here in that it does not admit an analytic solution, and can only be approximated. The final problem is an industrial data set known as the Tower problem[3] that uses data recorded from a chemical distillation tower. The

[2]UBall5D is also known as Vladislavleva-4.

[3]http://symbolicregression.com/?q=towerProblem

data is split into training and validation sets according the scaled extrapolation method used in [33]. The training set is comprised of data in the scaled range [0.02, 0.98] of input variables x_6, x_8, x_{11}, x_{17}, and x_{23}, and the validation set is comprised of data in $[0, 0.02) \cup (0.98, 1]$.

4.2 General Program Synthesis with PushGP

4.2.1 Algorithm settings

We conduct the program synthesis experiments in PushGP using standard tournament selection of size 7, a population size of 1000, and 100 trials for all problems. Table 2 shows the settings for each problem.

The new linear-genome PushGP allows us to use entirely uniform genetic operators. The crossover operator, which we use 20% of the time, alternates between parents, with a 1% chance of alternation at each instruction and an alignment deviation of 10, similar to ULTRA [27]. The mutation operator, which we use 20% of the time, gives a 1% chance of changing each instruction to a randomly selected one. The uniform close mutation operator, which we use 10% of the time, gives a 10% chance of adding or removing a closing parenthesis after each instruction. Finally, 50% of the time we perform a crossover followed by a mutation.

4.2.2 Problems

We perform tests on five program synthesis benchmark problems [11]. The problems used are:

- *Replace Space with Newline (RSWN)*: Given a string input, print the string, replacing spaces with newlines. The input string will not have tabs or newlines, but may have multiple spaces in a row. It will have maximum length of 20 characters. The program should also return the integer count of the non-whitespace characters.
- *String Lengths Backwards (SLB)*: Given a vector of strings with length ≤ 50, where each string has length ≤ 50, print the length of each string in the vector starting with the last and ending with the first.
- *Vector Average (VA)*: Given a vector of floats with length in [1,50], with each float in [-1000,1000], return the average of those floats.
- *Negative To Zero (N2Z)*: Given a vector of integers in [-1000,1000] with length ≤ 50, return the vector where all negative integers have been replaced by 0.
- *Syllables (Syl)*: Given a string (max length 20, containing symbols, spaces, digits, and lowercase letters), count the number of occurrences of vowels (a, e, i, o, u, y) in the string and print that number as X in "The number of syllables is X".

Instruction sets are determined by selecting potentially relevant data types for each problem, and then using all instructions that utilize those data types (shown in Table 2).

4.3 Results

We discuss the results of the symbolic regression and program synthesis experiments in the following two sections. In the third section, we look at population homologies for specific symbolic regression and program synthesis problems in order to shed light on the performance differences observed between methods.

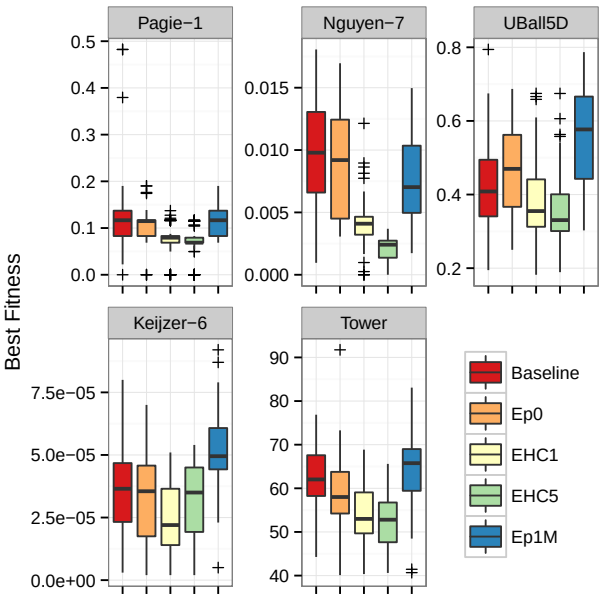

Figure 2: Boxplot comparison of best-of-run training fitnesses for the symbolic regression problems.

4.3.1 Symbolic Regression

The symbolic regression results are summarized in Table 3, and the best-of-run fitnesses are shown in Figure 2. We compare training performance in terms of exact solutions, mean best MAE, mean best R^2, and mean active program size (averaged from population averages recorded each generation for each trial). For the validation comparisons we report the percent of solutions that "generalize", defined as having a real-valued, finite output on validation data. We assign a poor fitness to non-generalizing solutions when calculating (non-parametric) statistics and report the median values of MAE and R^2 on the validation sets to account for outliers (i.e. non-generalizing models).

Overall, exact solutions are found only for the Pagie-1 and Nguyen-7 problems; for both cases, **EHC1** and **EHC5** find approximately 3 to 5 times more exact solutions than **Baseline** or **Ep0**. In all training cases, a significant improvement to MAE and R^2 is achieved by **EHC1** and/or **EHC5** except when all methods converge to mean best $R^2 = 1$ (Nguyen-7 and Keijzer-6). For the validation cases, **EHC1** and **EHC5** are significantly better on Pagie-1, Nguyen-7 and UBall5D, but on Keijzer-6 and Tower the results are not significant aside from **EHC1** R^2 on Keijzer-6 and **Ep1M** R^2 on Tower. Surprisingly, **Ep1M** significantly improves validation R^2 on Tower even though it has the poorest training fitness, perhaps due to the small program sizes produced by that method for that problem.

For all problems, **Ep1M** has the smallest average program sizes, followed by **Ep0**, **EHC1** and **EHC5**. All the epigenetic programs have significantly smaller programs than the **Baseline** case. In general the symbolic regression problems seem to benefit from some level of epigenetic hill climbing, both in terms of fitness convergence and program size.

Table 2: PushGP settings for the program synthesis problems.

Problem	Replace Space with Newline	String Lengths Backwards	Vector Average	Negative To Zero	Syllables
Max Point Evals	3.0E+10	1.0E+10	1.0E+10	1.0E+10	1.0E+10
Max Program Size	800	300	400	500	800
Max Point Evals per Program Evaluation	1600	600	800	1500	1600
Instruction Set (all instructions in listed types)	:integer :boolean :string :char :exec :print	:string :vector_string :integer :boolean :exec :print	:vector_float :float :integer :exec	:integer :boolean :vector_integer :exec	:integer :boolean :string :char :exec :print
Terminal Set	input, '\n', ' ', visible char ERC, string ERC	input, integer ERC [-100, 100]	input	input, 0, []	input, "The number of syllables is ", "aeiouy", 'a', 'e', 'i', 'o', 'u', 'y', visible char ERC, string ERC
Error Function	printed string Levenshtein distance, integer absolute error	printed string Levenshtein distance	float absolute error	Levenshtein distance between vectors	printed string Levenshtein distance, printed integer absolute error

4.3.2 *Program Synthesis*

The results for the program synthesis problems are summarized in Table 4. Unlike symbolic regression problems, the goal is to generate a program that passes all tests, and so the results are compared only in terms of the number of exact solutions found. By this measure, the best performance case is always **Ep1M** or **EHC1**, although this performance boost is only significant for **Ep1M** on the RSWN and VA problems. Surprisingly, **Ep1M** outperforms **EHC1**, unlike in the symbolic regression case. We discuss reasons for why this might occur in the following section.

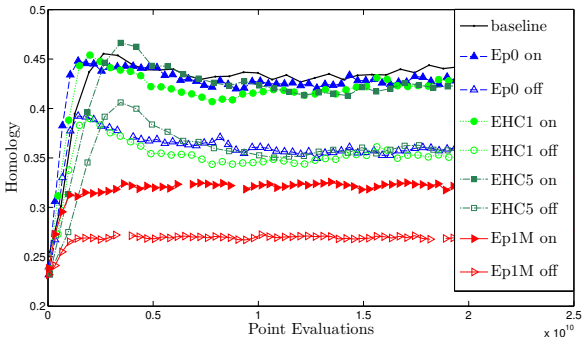

Figure 3: Active (on) and inactive (off) genotype homology for Uball5D problem, averaged over 100 trials.

4.3.3 *Homology*

We hypothesize that epigenetics provide protection from premature convergence in populations by preserving sections of the genome from fitness pressure and allowing them to drift genetically, thus providing an avenue for introduction of diversity and continued progress. We look at population homology in detail to determine whether this phenomenon of preserved diversity is evident. We define the homology H of a population using a Levenshtein distance comparison of P randomly sampled pairs of individuals $(|\mathbf{i}_j, \mathbf{i}_m|_L)$, normalized by the length, $|\cdot|$, of the longer individual:

$$H = 1 - \frac{1}{P}\sum_{n=1}^{P}\frac{|\mathbf{i}_j, \mathbf{i}_m|_L}{max(|\mathbf{i}_j|, |\mathbf{i}_m|)} \quad (10)$$

We sample homology separately for the active, inactive, and total (whole genome) portions of the genotypes with $P = 200$. In general we find that inactive genotypes have lower homology (i.e. higher diversity) than active genotypes for every epigenetic case. The homologies for the UBall5D problem are shown in Figure 3 as an example. The differences between active and inactive homologies are very significant in this example ($p < 10^{-14}$). The active gene homologies for **Ep0**, **EHC1**, and **EHC5** are slightly (yet significantly) lower than **Baseline** as well ($p < 10^{-13}$). This suggests the improvement in fitness convergence for the symbolic regression problems is linked to the extra diversity available during epigenetic local search. **Ep1M** displays much lower homology in both active and inactive genotypes, a trend also evident when comparing total genome homology for the VA problem in Figure 4. The program synthesis problems may therefore benefit mostly from increased genetic diversity over the total genome (afforded by **Ep1M**). We hypothesize that increased diversity in this context assures larger phenotypic coverage and hence a higher likelihood of finding exact solutions in non-convex fitness landscapes.

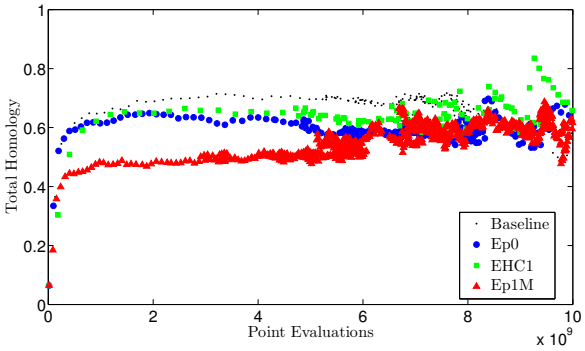

Figure 4: Total (whole genome) homology for the Vector Average problem.

5. CONCLUSIONS

The results suggest that epigenetic local search can be a promising addition to GP. We have shown that the epigenetic layer is able to preserve diversity in the inactive

Table 3: Comparison of best-of-run results for the symbolic regression benchmark problems. * and underline indicate significant ($p < 0.05$) improvement over Baseline and Ep0, respectively. Exact solution p-values are based on pairwise chi-squared tests with Holm correction. Fitness and size p-values are based on pairwise Wilcoxon rank-sum tests.

Problem	Exact Solutions (%)	Training		Validation			Mean Active
Method		Mean MAE	Mean R^2	Generalize (%)	Median MAE	Median R^2	Program Size
Pagie-1							
Baseline	1	0.114	0.930	100	0.116	0.960	31.7
Ep0	3	0.099	0.960	100	0.109	0.971	*19.21
EHC1	*12	*0.071	*0.968	100	*0.080	*0.972	*20.92
EHC5	*15	*0.065	*0.970	100	*0.078	*0.963	*21.35
Ep1M	0	0.104	0.962	100	0.111	*0.971	*18.3
Nguyen-7							
Baseline	10	5.04E-4	1	95	0.074	0.999	28.22
Ep0	10	7.08E-4	1	98	0.061	0.999	*17.77
EHC1	*27	4.34E-4	1	99	*0.024	0.999	*20.25
EHC5	*31	*3.46E-4	1	94	*0.019	*1.000	*21.20
Ep1M	11	1.10E-3	1	99	0.031	0.999	*15.41
UBall5D							
Baseline	0	0.130	0.325	100	0.165	0.124	33.20
Ep0	0	0.130	0.288	100	0.149	0.163	*18.70
EHC1	0	*0.122	0.334	100	*0.128	*0.287	*20.22
EHC5	0	*0.119	*0.352	100	*0.122	*0.293	*22.14
Ep1M	0	0.132	0.259	100	0.149	0.160	*17.77
Keijzer-6							
Baseline	n/a	3.65E-5	1	88	4.93E-5	1	25.84
Ep0	n/a	3.26E-5	1	90	4.31E-5	1	*16.62
EHC1	n/a	*2.42E-5	1	90	4.08E-5	*1	*19.44
EHC5	n/a	3.05E-5	1	94	4.53E-5	1	*20.23
Ep1M	n/a	5.10E-5	1	92	6.47E-5	1	*16.06
Tower							
Baseline	n/a	41.03	0.663	96	42.55	0.663	22.57
Ep0	n/a	40.08	0.682	98	42.99	0.667	*16.67
EHC1	n/a	*37.55	*0.706	100	38.56	0.705	*18.28
EHC5	n/a	*37.23	*0.715	98	40.64	0.697	*19.37
Ep1M	n/a	41.91	0.660	100	40.46	*0.752	*13.47

Table 4: Results of program synthesis runs. * and underline indicate significant ($p < 0.05$) improvement over Baseline and Ep0, respectively, according to chi-squared tests with Holm correction.

Problem	Solutions on training / validation set (%)			
	Baseline	Ep0	EHC1	Ep1M
RSWN	61/58	66/65	61/60	*85/83
SLB	14/14	11/10	23/23	15/14
VA	12/12	10/10	20/20	*43/43
N2Z	9/9	9/8	9/8	21/19
Syl	10/10	2/2	5/5	11/11

sequences of genes and that, for the benchmark problems studied, epigenetic methods outperform a baseline implementation of GP in terms of fitness minimization, exact solutions, and program sizes. We have only considered epigenetic learning by mutation and hill climbing here, but we hope the results encourage further research into the use of epigenetic methods for structure optimization in GP.

6. ACKNOWLEDGMENTS

We thank Nicholas McPhee and anonymous reviewers for helping improve this paper. This work is partially supported by the NSF-sponsored IGERT: Offshore Wind Energy Engineering, Environmental Science, and Policy (Grant Number 1068864), as well as Grant No. 1017817, 1129139, and 1331283. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] I. Arnaldo, K. Krawiec, and U.-M. O'Reilly. Multiple regression genetic programming. In *Proceedings of GECCO '14*, pages 879–886. ACM Press, 2014.

[2] W. Banzhaf. Genotype-phenotype-mapping and neutral variation - a case study in genetic programming. In *PPSN III*, pages 322–332. Springer, 1994.

[3] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *PNAS*, 104(24):9943–9948, 2007.

[4] M. Brameier and W. Banzhaf. *Linear Genetic Programming*, volume 1. Springer, 1 edition, 2007.

[5] B. G. Dias and K. J. Ressler. PACAP and the PAC1 receptor in post-traumatic stress disorder. *Neuropsychopharmacology*, 38(1):245–246, Jan. 2013.

[6] B. G. Dias and K. J. Ressler. Parental olfactory experience influences behavior and neural structure in

subsequent generations. *Nature Neuroscience*, 17(1):89–96, Jan. 2014.

[7] C. Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *arXiv:cs/0102027*, Feb. 2001. Complex Systems, 13(2): 87-129, 2001.

[8] A. Fontana. Epigenetic tracking: Biological implications. In *Advances in Artificial Life. Darwin Meets von Neumann*, number 5777 in Lecture Notes in Computer Science, pages 10–17. Springer Berlin Heidelberg, Jan. 2011.

[9] C. Giraud-Carrier. Unifying learning with evolution through baldwinian evolution and lamarckism. In *Advances in Computational Intelligence and Learning*, pages 159–168. Springer, 2002.

[10] F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation*, 1(3):213–233, Sept. 1993.

[11] T. Helmuth and L. Spector. General program synthesis benchmark suite. In *Proceedings of GECCO '15*, 2015.

[12] R. Holliday. Epigenetics: a historical overview. *Epigenetics*, 1(2):76–80, 2006.

[13] H. Iba and T. Sato. Genetic programming with local hill-climbing. Technical Report ETL-TR-94-4, Electrotechnical Laboratory, 1-1-4 Umezono, Tsukuba-city, Ibaraki, 305, Japan, 1994.

[14] E. Jablonka and M. J. Lamb. The changing concept of epigenetics. *Annals of the New York Academy of Sciences*, 981(1):82–96, 2002.

[15] G. Kaati, L. O. Bygren, and S. Edvinsson. Cardiovascular and diabetes mortality determined by nutrition during parents' and grandparents' slow growth period. *European Journal of Human Genetics*, 10(11):682, Nov. 2002.

[16] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *Genetic Programming*, number 2610 in Lecture Notes in Computer Science, pages 70–82. Springer Berlin Heidelberg, Jan. 2003.

[17] M. Keijzer. Push-forth: A light-weight, strongly-typed, stack-based genetic programming language. In *Proceedings of GECCO '13 Companion*, GECCO '13 Companion, pages 1635–1640, New York, NY, USA, 2013. ACM.

[18] W. La Cava, L. Spector, K. Danai, and M. Lackner. Evolving differential equations with developmental linear genetic programming and epigenetic hill climbing. In *Companion proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 141–142. ACM Press, 2014.

[19] W. B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston, 1998.

[20] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming*, pages 121–132. Springer, 2000.

[21] P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA, 1995.

[22] L. Pagie and P. Hogeweg. Evolutionary consequences of coevolving targets. *Evolutionary computation*, 5(4):401–418, 1997.

[23] B. J. Ross. A lamarckian evolution strategy for genetic algorithms. *Practical handbook of genetic algorithms: complex coding systems*, 3:1–16, 1999.

[24] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[25] M. Schmidt and H. Lipson. Age-fitness pareto optimization. In *Genetic Programming Theory and Practice VIII*, pages 129–146. Springer, 2011.

[26] G. F. Smits and M. Kotanchek. Pareto-front exploitation in symbolic regression. In *Genetic Programming Theory and Practice II*, pages 283–299. Springer, 2005.

[27] L. Spector and T. Helmuth. Uniform linear transformation with repair and alternation in genetic programming. *Genetic Programming Theory and Practice XI*, 2013.

[28] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the push programming language. *GPEM*, 3(1):7–40, 2002.

[29] I. Tanev and K. Yuta. Epigenetic programming: Genetic programming incorporating epigenetic learning through modification of histones. *Information Sciences*, 178(23):4469–4481, Dec. 2008.

[30] A. Topchy and W. F. Punch. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of GECCO '01*, pages 155–162, 2001.

[31] B. M. Turner. Histone acetylation and an epigenetic code. *Bioessays*, 22(9):836–845, 2000.

[32] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *GPEM*, 12(2):91–119, 2011.

[33] E. Vladislavleva, G. Smits, and D. den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.

[34] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly, and S. Luke. Better GP benchmarks: community survey results and proposals. *GPEM*, 14(1):3–29, Dec. 2012.

[35] D. Whitley, V. S. Gordon, and K. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In *PPSN III*, pages 5–15. Springer, 1994.