

Taras Kowaliw* and René Doursat

Bias-variance decomposition in Genetic Programming

DOI 10.1515/math-2016-0005

Received November 25, 2014; accepted January 7, 2015.

Abstract: We study properties of Linear Genetic Programming (LGP) through several regression and classification benchmarks. In each problem, we decompose the results into *bias* and *variance* components, and explore the effect of varying certain key parameters on the overall error and its decomposed contributions. These parameters are the maximum program size, the initial population, and the function set used. We confirm and quantify several insights into the practical usage of GP, most notably that (a) the variance between runs is primarily due to initialization rather than the selection of training samples, (b) parameters can be reasonably optimized to obtain gains in efficacy, and (c) functions detrimental to evolvability are easily eliminated, while functions well-suited to the problem can greatly improve performance—therefore, larger and more diverse function sets are always preferable.

Keywords: Analysis of algorithms, Bias-variance decomposition, Classification, Computational learning theory, Evolutionary computation, Genetic programming, Learning and adaptive systems, Non-parametric inference, Regression

MSC: 62G08, 62J10, 68Q32, 68T05, 68W40

1 Introduction

Bias-variance decomposition is a fundamental method in machine learning. It allows for the decomposition of the error rate of a predictor into two components: the *bias*, representing the systematic error made by a model, and the *variance*, representing the error associated with the particularities of a training set. There are many “non-parametric” estimators characterized by a learning error that always tends to zero as the number of samples becomes large. Unfortunately, these learners become computationally expensive to deal with as the number of training samples increases, especially when problem dimensionality is high. Given a fixed number of training samples, non-parametric estimators typically encounter the “bias-variance trade-off”, where greater complexity is required to exclude model bias but too much complexity will cause over-specialization to the training data. In light of this trade-off, several authors suggest that the “hard” part of solving a complex problem is precisely finding a proper *model* with a bias suited for the domain at hand [10, 11], for instance, via the inclusion of appropriate heuristics.

Genetic Programming (GP) refers to the use of evolutionary computation to generate computer programs or mathematical expressions. The most typical form of GP is the tree-based version pioneered by Koza [17]. There are many other types, however, including a family easily expressed as graph-based networks, which include Cartesian Genetic Programming [24], Parallel Distributed Genetic Programming [27], Linear Genetic Programming (LGP) [3], and others [26]. These forms of GP are often static-length, while the complexity of the program is derived from the ratio of neutral to active code in the representation. In this work, we concentrate on LGP, an attractive choice for

*Corresponding Author: Taras Kowaliw: Institut des Systèmes Complexes Paris Île-de-France (ISC-PIF), Centre National de la Recherche Scientifique (CNRS UPS3611), 113 rue Nationale, 75013 Paris, France, E-mail: taras@kowaliw.ca

René Doursat: Informatics Research Centre, School of Computing, Mathematics & Digital Technology, Manchester Metropolitan University, John Dalton Building, Chester Street, Manchester M1 5GD, UK, E-mail: r.doursat@mmu.ac.uk

applications due to its tendency to produce parsimonious solutions [3] and its capacity to be converted directly to machine code [28]. We use a variable-length version, in which the effective length of a program will be less than some varying maximum value.

Our goal in this study is to explore the application of LGP to several benchmark problems under the lens of bias-variance decomposition. Some analysis of this form has already been conducted for tree-based GP, where it was shown that GP is generally a low-bias approach for regression problems. Some authors have used variance adjustment [1, 13] or other strategies [8] to improve generalization of discovered solutions. In particular, Fitzgerald and Ryan hypothesize that low operator complexity (a smaller or simpler function set) corresponds to lower variance [7]. In this study, we find some evidence to the contrary.

Here we investigate more deeply this breakdown for several problems, both regression and classification, by considering the effect of various key parameters on this decomposition. First, we look at *program length*, a key parameter for the complexity of GP programs and the variable leading to the classic bias-variance trade-off. Next, we examine more closely choices involving *initialization* and *function sets* as potential means of reducing either the bias or the variance portions of the decomposition. In these latter cases, we do not attempt to produce yet other versions of the usual bias-variance trade-off, but rather explore means toward the amelioration of either component of error given realistic constraints. We believe this analysis will help guide practitioners in their choice of models and parameter setting.

2 Bias-variance decomposition

Let $x \in \mathbb{R}^p$ be a real-valued input vector, and $y \in \mathbb{R}$ be an output value, with joint probability distribution $P(x, y)$. Let our loss function in output space be $L(y, y')$, some measure of similarity between y and y' in \mathbb{R} . We seek a function f which, when provided with an input x , will predict an “appropriate” output value $f(x)$, i.e., one that tends to minimize the average loss at that point: $\int L(y, f(x))P(y|x)dy$.

2.1 Regression problems

For regression problems, the most common loss function is $L(y, f(x)) = (y - f(x))^2$. The best choice of predictor, in this case, is the simple conditional expectation, $f^*(x) = E[y|x] = \int yP(y|x)dy$, also known as the regression function.

Assume that we have some predictor f , generated through the use of a data sample T , and tuned by a set of parameter values λ . In the case of a stochastic system, it also takes an initial seed I . For a given λ , we will write the output of f at x as $f(x; T, I)$ to recall the other two dependencies. Then, the *mean square error* (MSE) of f at point x_0 can be expressed as the expected loss between $f(x_0)$ and $f^*(x_0)$ over various instances of T and I :

$$\text{mse}(x_0) = E_{T,I} \left[(f(x_0; T, I) - E[y|x_0])^2 \right] \quad (1)$$

in which we impose fixed-size training sets T . Note here that $\text{mse}(x_0)$ refers to the error of the *expected* predictor in x_0 , and as such, is a measure of the efficacy of the technique (and parameters) which spawned the predictor.

Via algebraic manipulation, and making the assumption that our problem is deterministic, we can break the MSE into two components:

$$\begin{aligned} \text{mse}(x_0) &= \text{bias}(x_0) + \text{var}(x_0) \\ \text{bias}(x_0) &= \left(\hat{f}(x_0) - E[y|x_0] \right)^2 \end{aligned} \quad (2)$$

$$\text{var}(x_0) = E_{T,I} \left[\left(f(x_0; T, I) - \hat{f}(x_0) \right)^2 \right] \quad (3)$$

where $\hat{f}(x_0) = E_{T,I}[f(x_0; T, I)]$ denotes the average response of the predictor over T and I . The bias-variance dilemma refers to the trade-off between the two components of error: whereas the bias should be reduced, to prevent

systematic error in predictions due to the model, doing so typically results in an increased variance, such as the propensity of a learner to rote memorization of training data¹.

Following previous work by Geman *et al.* [10], we will approximate these values as follows: 50 pairs of training set-seeds are generated, denoted $T^{(k)}$ and $I^{(k)}$, $k = 1, \dots, 50$. We calculate the average response for the evolutionary algorithm as

$$\hat{f}(x_0) \approx \frac{1}{50} \sum_{k=1}^{50} f(x_0; T^{(k)}, I^{(k)}). \quad (4)$$

Then, we estimate the bias(x_0) $\approx (\hat{f}(x_0) - E[y|x_0])^2$, the variance as

$$\text{var}(x_0) \approx \frac{1}{50} \sum_{k=1}^{50} \left(f(x_0; T^{(k)}, I^{(k)}) - \hat{f}(x_0) \right)^2 \quad (5)$$

and mse(x_0) as the sum of both terms.

2.2 Classification problems

For classification, it is necessary to use a binary loss function: $L(y, f(x)) = 1 - \delta(y, f(x))$, where δ is the Kronecker delta: $\delta(a, b) = 1$ if $a = b$, and 0 otherwise. Kohavi and Wolpert [14] have developed a bias-variance decomposition for the *mean misclassification rate* (MMR). Let us refer to the random variable which generates an actual answer to the input x as Y , and the random variable that generates an answer to input x via the learned function $f(x; T, I)$ as Y_f . Assuming a deterministic two-class problem, we can break the MMR into the usual two components:

$$\begin{aligned} \text{mmr}(x_0) &= \text{bias}(x_0) + \text{var}(x_0) \\ \text{bias}(x_0) &= \frac{1}{2} \sum_{y \in \{-1, +1\}} (P[Y=y|x] - P_{T,I}[Y_f=y|x])^2 \end{aligned} \quad (6)$$

$$\text{var}(x_0) = \frac{1}{2} \left(1 - \sum_{y \in \{-1, +1\}} (P_{T,I}[Y_f=y|x])^2 \right). \quad (7)$$

Note that $P[Y = y|x]$ will be 1 or 0, due to the determinism of the problem. The term $P_{T,I}[Y_f = y|x]$ is the probability of guessing value y via the learning algorithm over all possible training samples T and initial seeds I . As with previous expectations, we estimate this probability via 50 runs of the system.

2.3 Integrated statistics

Finally, we will report the *integrated* forms of the MSE, MMR, bias and variance, respectively denoted $\widehat{\text{mse}}$, $\widehat{\text{mmr}}$, $\widehat{\text{bias}}$ and $\widehat{\text{var}}$, to compare predictors on the basis of a single global measure in each category. For regression problems, integrals are computed numerically over a large set of uniformly distributed samples $Q = \{x_0^j\}$ as follows:

$$\widehat{\text{mse}} = \frac{1}{|Q|} \sum_j \text{mse}(x_0^j) \quad (8)$$

where $|Q| = 360,000$. Note that since our numerical integration uses independently chosen samples, it can also be considered as an independent test set, and hence would detect any overfitting. For classification problems, we can similarly approximate the integrated mean misclassification rate, denoted $\widehat{\text{mmr}}$, over the test problem instances.

¹ This dilemma cannot be solved in general, but is often ameliorated via ensemble learning, as has been explored with GP [13]. We will not pursue this direction here, however.

3 Model and experimental design

3.1 Linear Genetic Programming

We follow Brameier and Banzhaf's definitions of LGP closely [3], with some minor modifications. An LGP individual consists of a collection of registers, n_{reg} , equal to the number of inputs, n_{in} , plus a number of additional registers initially filled with genetically defined constant values, n_{const} . Thus $n_{\text{reg}} = n_{\text{in}} + n_{\text{const}}$. Following this is a list of n program statements, with n ranging from 1 to a maximum program length, n_{prog} . A program is executed by loading the input values into the initial registers, executing the program statements, then reading the output from the 0-th register. Figure 1 shows an example program.

Fig. 1. An example LGP program, instantiating the 2D Euclidean distance function, written in pseudo-Java notation. Note the existence of ineffective ("neutral") code, commented out in light gray.

```
double LGP(double i0, double i1, double i2, double i3) {
    double R0 = i0;
    double R1 = i1;
    double R2 = i2;
    double R3 = i3;

    double R4 = 0.836;
    double R5 = 0.118;
    double R6 = 0.723;
    double R7 = 0.925;

    R4 = R1 - R0;
    R4 = square(R4);
    // R5 = R1 + R0;
    R2 = R3 - R2;
    R6 = square(R2);
    R1 = R6 + R4;
    R0 = sqrt(R1);
    // R7 = safepow(R6, R4);

    return R0;
}
```

Diagram annotations for Figure 1:

- input registers**: Points to the initialization of R_0, R_1, R_2, R_3 .
- additional registers (initialized with genetically specified constants)**: Points to the initialization of R_4, R_5, R_6, R_7 .
- length n_{reg}** : A bracket indicating the total number of registers from R_0 to R_7 .
- program statements length less than n_{prog}** : Points to the block of statements from $R_4 = R_1 - R_0$ to $// R_7 = \text{safepow}(R_6, R_4);$.
- output is always contents of first register**: Points to the `return R0;` statement.

An LGP individual is initialized by generating a sufficient number of constants to fill the additional registers, then generating a series of program statements. The constants are chosen uniformly and randomly from $[0,1]$. The number of program statements is selected randomly and uniformly between 1 and a maximum initialization length $n_{\text{init}} \leq n_{\text{prog}}$. The statements are generated by selecting three registers r_a , r_b and r_c uniformly and randomly from all the value registers n_{reg} , and then selecting a function g from the function set to generate the statement $r_c = g(r_a, r_b)$, or $r_c = g(r_a)$ if g takes only one variable. Finally, any output from the LGP individual is constrained within a certain range, where outlying values are rounded to the closest extreme bound. These problem-specific *output bounds* were added to prevent undue influence of singularities on statistical analysis. The global output function produced by the LGP individual is denoted f as before: it is equal to some (more or less complicated) composition of a certain number of functions g .

We use two function sets to explore our problems: one short list, G_{short} , and one long list, G_{long} . In some experiments, we utilize arbitrary subsets of G_{long} . All possible functions are listed in Table 1. Generally, in this article, we will write our LGP individuals as mathematical expressions. The reader should be aware that: (1) they are an expanded view of the program, since modules are written out explicitly, and (2) while we remove unused code, we do not remove any redundancy (i.e., a statement such as $a - a$ is not replaced by 0), in order to give a realistic view of the raw evolutionary outputs.

3.2 Evolutionary algorithm

For all problems, we use a steady-state evolutionary algorithm. In the beginning, a population of N_{pop} randomly initialized individuals f is created and each of them is evaluated by calculating its fitness $F(f)$ (see below). The population is maintained at a constant size N_{pop} throughout the evolutionary search by performing one-to-one replacements of individuals. During the search, an additional N_{new} evaluations are performed as follows: for each evaluation, a deterministic tournament is held between two randomly chosen individuals of the current population. The worse of the two is replaced by either a *cross* between the winner and a second tournament-selected individual (with probability p_{cross}), or a *mutation* of the winner (individual elements are mutated with probability p_{mut} and equal chances of macro- or micro-mutation). We also sometimes include a “parsimony pressure”: if the difference between the fitness of the two individuals selected for tournament is less than ΔF_{pars} , then we select the individual with the smaller number of program statements. In sum, while the population’s size remains N_{pop} , the total number of evaluations is $N_{\text{eval}} = N_{\text{pop}} + N_{\text{new}}$ and the total number of individuals that are effectively replaced is comprised between 0 and N_{new} .

Table 1. Pool of GP functions, G_{long} , with action on inputs a and b (where b is sometimes disregarded).

func. g	action	func. g	action	func. g	action
plus	$a + b$	sin	$\sin a$	max	$\max(a, b)$
minus	$a - b$	cos	$\cos a$	min	$\min(a, b)$
times	ab	abs	$ a $	dist	$ a - b $
div	a/b , or 1 if $ b < 0.00001$	inv	$1/a$, or 1 if $ a < 0.00001$	thresh	1 if $a > b$, 0 otherwise
pow	a^b , or 1 if undefined	log	$\log a$, or 1 if $a < 0.00001$	mag1	$ a + b $
sqrt	$\sqrt{ a }$	square	a^2	mag2	$\sqrt{a^2 + b^2}$

3.3 Four benchmarks

We perform our investigations on several benchmark problems. For each benchmark, we execute approximately 500 runs with randomly chosen parameter values similar to the original source. From these runs, we estimate the combination leading to the *lowest* test fitness (since the “fitness” represents an error or a mismatch to be minimized). The problems and their associated search parameters are summarized in Table 2.

3.3.1 MexHat

The first problem is the “Mexican hat” (MexHat), borrowed from [3] and so named after the shape of its 2D manifold in 3D space. The MexHat function is reduced here to its 2D expression, denoting $x = (a, b)$:

$$f_{\text{Mex}}^*(x) = \left(1 - \frac{a^2 + b^2}{4}\right) e^{\left(-\frac{a^2 + b^2}{8}\right)}. \quad (9)$$

Note that Euler’s number e is not included in the function set, and hence must be approximated genetically. For this regression problem, the fitness value F of an LGP individual f is defined as the *sum of squared errors* (SSE), with respect to the target f_{Mex}^* , approximated over the training samples $T = \{x_i\}$:

$$F_{\text{SSE}}(f) \approx \frac{1}{|T|} \sum_i (f(x_i) - f_{\text{Mex}}^*(x_i))^2. \quad (10)$$

Table 2. Summary of the three benchmark problems and their associated parameters.

	MexHat	DistND	Spiral	Cenparmi
problem parameters				
type	regression	regression	classification	classification
dimension	2	$2N$	2	144
axis range	$[-4, 4]$	$[-10, 10]$	$[-2\pi, 2\pi]$	$\{0, \dots, 255\}$
fitness	SSE	SSE	MR	MR
$ T $	400	2000	194	300
evolutionary parameters				
$n_{\text{in}} + n_{\text{const}}$	$2 + 4$	$2N + 2N$	$2 + 2$	$144 + 19$
n_{prog}	50	40	30	36
n_{init}	10	20	20	28
p_{mut}	0.07	0.15	0.03	0.11
p_{cross}	0	0.0	0.15	0.69
ΔF_{pars}	0.0005	0.001	0.001	—
N_{pop}	1000	1000	9000	43000
N_{eval}	$5 \cdot 10^5$	10^6 if $N < 3$, 10^7 otherwise	$2 \cdot 10^6$	$2 \cdot 10^6$
output bounds	$[-10, 10]$	$[-10^4, 10^4]$	—	—
G_{short}	plus, minus, times, div, pow	plus, minus, times, div, abs, square, sqrt	plus, minus, times, div, sin, cos, thresh	plus, minus, times, div
G_{long}	See Table 1			

3.3.2 DistND

Next, we evaluate several “distance” problems (Dist1D, Dist2D, ..., DistND). These are a series of regression problems based on Euclidean distance in any even number of dimensions:

$$f_{\text{Dist}}^*(x) = \sqrt{\sum_{j=1}^N (a_j - b_j)^2}. \quad (11)$$

Note that, since $x = (a_1, \dots, a_N, b_1, \dots, b_N)$, the dimensionality of the problem is actually $2N$. The distance functions are useful for investigating a related series of problems in increasing dimensionality. The fitness function F_{SSE} is calculated as in Eq. (10), with target f_{Dist}^* .

3.3.3 Spiral

We also include two classification problems. The first one is the artificial “spiral” problem, as described by Lang and Witbrock [19]. Here, the target f_{Spir}^* is a binary function whose domains are intertwined in a double spiral pattern (Figure 2). There has been significant research on this problem, including in the GP community, due to both its difficulty and its capacity to be easily visualized [5]. For this classification problem, the fitness function is an approximation of the *misclassification rate* (MR), i.e., the average of all binary mismatches:

$$F_{\text{MR}}(f) \approx \frac{1}{|T|} \sum_i \left(1 - \delta(f(x_i), f_{\text{Spir}}^*(x_i)) \right). \quad (12)$$

Fig. 2. Output examples from a GP classification run. Positive and negative samples are drawn as white and black dots, while the pattern produced by the GP individual is drawn as light orange or dark blue.

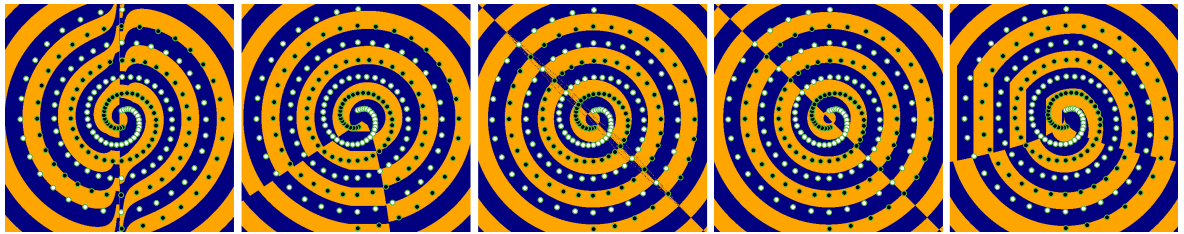
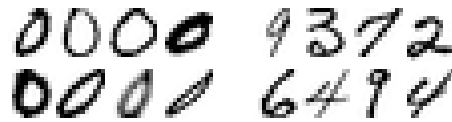


Fig. 3. 16 samples from the Cenparmi database, divided into classes (left) “zero” and (right) “not-zero”. Note that the machine learner does not have access to geometric adjacency information about the pixels.



3.3.4 Cenparmi

The second classification problem is the Cenparmi database of hand-written characters, collected by the CENPARMI lab at Concordia University (Figure 3). This real-world supervised learning challenge consists of 6000 image samples of hand-written digits, and constitutes a high-dimensional problem with tightly correlated features. We scaled the images to a size of $12 \times 12 = 144$ integer inputs between 0 and 255 representing gray-level pixels. Our treatment made the classification problem binary by distinguishing between a selected class and the remainder of the set (e.g., between “4” and “not-4” instances). At the start of each run, we randomly selected the particular class, involving $|T| = 300$ training samples from the training pool, and $|V| = 600$ test samples from the test pool. Note that our approach to the database remained “naive” on purpose, i.e., we did not include geometric information about the relative location of the pixels. Our goal here was to test the bias-variance limits of genetic programming, not achieve competitive performance. The fitness function F_{MR} was calculated as in Eq. (12), with target f_{Cenp}^* representing the correct binary answer for the chosen class. Note that the state-of-the-art MR, across all learning techniques and available information, is approximately 0.02 [20].

4 Overall results

4.1 Initial exploration

As expected, LGP was generally successful at evolving good regression functions and classifiers. Some examples of outputs for the Spiral problem are shown in Figure 2, and for the MexHat problem in Table 3. In further sections we will look closely at the variance associated with the individual runs of the evolutionary algorithm. Recall that we are discussing the variation of the solutions in terms of their behaviour in input, i.e., “phenotypic”, space. While this is typically the object of interest during the use of genetic programming—as practitioners care how well they achieve some objective or fit some data—it should be noted that this is not the same as variance in genotypic space. In other words, two largely different mathematical expressions (genotypes) might have nearly identical performance (phenotypes), while, conversely, two genetic programs differing by a single instruction might produce dramatically different output functions.

The question here is about *genetic convergence*, that is, the propensity of evolutionary methods to find the same or equivalent mathematical expressions for the same problem on different runs. Indeed, this is a difficult concept to evaluate, since while there are ways to detect when *some* related programs are similar, there is no way, in general, to determine if two arbitrary programs are equivalent. There exist several measures of genotypic dissimilarity (termed

“diversity”, after their typical usage²) for GP. For instance, *edit distance* (a measure of the number of steps required to transform one program into another, adapted for LGP in [3]), *entropy*, and *behavioural diversity* (comparing distributions of program outputs) are known to correlate well with expected fitness for some problem domains [4, 12]. Unfortunately, in the case of edit distance and entropy, typical applications of these measures to GP tend to make the assumption that individuals are genetically related, hence are not useful for programs generated via independent means. Furthermore, some identities, such as the capacity to construct one primitive function from combinations of other functions, are not detectable.

Informally speaking, in most cases that we examined, some form of genetic convergence was the norm. For instance, consider the solutions evolved from the Dist2D problem using the G_{short} function set (see Table 2). We show below the two functions of $x = (a_1, a_2, b_1, b_2)$ that had the best fitness values among 33 inexact solutions:

$$f_{\text{Dist}}^1(x) = \sqrt{\left\| \left((b_1 - a_1 \cdot 1) \right)^2 + \left(\left(\sqrt{|0.001|} - a_2 \right) + b_2 \right)^2 \right\|}$$

$$f_{\text{Dist}}^2(x) = \sqrt{\left(\sqrt{\left\| (a_2 - b_2 + 0.053) \right\|^2 + \left\| (b_1 - a_1) \right\|^2} \right)^2 + \left| \frac{0.053}{0.499} \right|}$$

with $F_{\text{SSE}}(f_{\text{Dist}}^1) = 0.0006$ and $F_{\text{SSE}}(f_{\text{Dist}}^2) = 0.0126$. In these cases, the evolved solutions strikingly “resemble” the target function $f_{\text{Dist}}^*(x) = ((a_1 - b_1)^2 + (a_2 - b_2)^2)^{1/2}$, genetically speaking. To obtain exactly f_{Dist}^* , all that would be required is minor tweaks and some elimination of redundancy. One could reasonably expect that additional computational effort would achieve at least the former, and under parsimony pressure, possibly the latter.

In the case of the Spiral classification problem, we also observed convergence to a similar *form* of solution, even if there were differences between the individual runs. For instance, using the G_{long} function set, 5 out of 50 runs found zero-fitness solutions to the problem. Each of the 5 runs admits a similar structure of concentric rings (Figure 2), in which some additional singular boundary, like a flaw in a crystal, separates regions of rings to compensate for the ascending radius of the spirals. All five solutions make prodigious use of the *div*, *mag2*, and *thresh* functions, and one of either *sin* or *cos*. While this strategy is relatively consistent for the best of the LGP runs, it is by no means the only solution to the problem generally. For instance, techniques using constructive neural networks generate quite different output patterns, including true spirals [6].

On the other hand, consider the best solutions to the MexHat problem using its G_{long} function set. The three best solutions are shown in Table 3. Notice how all three individuals depend on $a^2 + b^2$, i.e., they discovered the radial symmetry of the MexHat target. Otherwise, these solutions display much more genetic variance than in the previously discussed problems. The edit distance between these statements is evidently quite high, as both the statement structure and the functions used differ wildly. Regardless, the outputs of these functions in the 2D plane are quite similar, and do not significantly overfit. Hence, despite great genotypic variation, they are all highly successful examples of regression solutions.

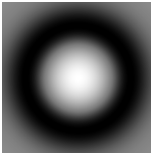
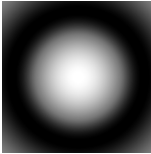
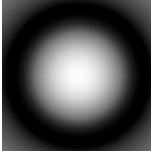

While our description is rather informal (the development of a more rigorous measure of genetic convergence lying beyond the scope of this study), we believe that it highlights the possibility of phenotypic convergence in the absence of genotypic convergence.

4.2 Typical bias-variance numbers

The bias-variance decomposition of each of our benchmark problems, including five different instances of DistND, is shown in Table 4. Parameters were set as indicated in Table 2. As proceeding sections will show, the listed values are typical. Nearly everywhere, the variance portion of the error dominates the bias component, often by several multiples. This is generally consistent with a view of GP as a low-bias/high-variance approach, which suggests that overfitting should be of concern for GP practitioners. In all cases, the use of G_{long} also outperforms G_{short} , especially

² Often, authors are concerned not with the consistency of solutions between runs, but instead with the *encouragement* of diversity in a particular population to prevent premature convergence.

Table 3. The three best solutions on a run of the MexHat problem using the G_{long} function set.

expression $f(x)$, where $x = (a, b)$	fitness $F(f)$	action on $[-4, 4]^2$
$f_{\text{Mex}}^*(x) = \left(1 - \frac{a^2 + b^2}{4}\right) e^{\left(-\frac{a^2 + b^2}{8}\right)}$	0	
$f_{\text{Mex}}^1(x) = \frac{\cos(\sqrt{\alpha b^2 + \alpha a^2})}{\sqrt{(\beta \sqrt{\alpha b^2 + \alpha a^2})^2 + \left(\cos\left(\beta \sqrt{\alpha b^2 + \alpha a^2} \beta^{\sin(\sqrt{\alpha b^2 + \alpha a^2})}\right)\right)^2}}$ <p>with $\alpha = 0.785^2$ and $\beta = 0.742$.</p>	0.00004	
$f_{\text{Mex}}^2(x) = \log\left(\max\left\{\log\left(\sqrt{\left \sqrt{a^2 + b^2} - 0.174\right }\right), \left \beta \sqrt{a^2 + b^2}^{\alpha \sqrt{a^2 + b^2}}\right + \left \alpha + \beta \sqrt{a^2 + b^2}^{\alpha \sqrt{a^2 + b^2}}\right \right\}\right)$ <p>with $\alpha = \sin(0.673)$ and $\beta = \sin(0.535)$.</p>	0.00009	
$f_{\text{Mex}}^3(x) = \min\left\{\sqrt{\left \sqrt{(\alpha \sqrt{a^2 + b^2})^2 + \alpha^2}\right }, \beta + \sqrt{(\alpha \sqrt{a^2 + b^2})^2 + \alpha^2}\right\} \cos(\beta \sqrt{a^2 + b^2})$ <p>with $\alpha = 0.626$, $\beta = \sin(0.905^{0.905})$.</p>	0.00010	

in the bias values. This is true despite the fact that some of the particular functions are known to be detrimental to evolvability (see Section 5.4).

5 Detailed analysis

In this section, we examine the effects of varying one of four control parameters separately from the others. First, we look at a key parameter of GP complexity, the maximum program length n_{prog} . It is here that we expect to see the classic bias-variance trade-off, and the existence of a range corresponding to the optimal point in that trade-off.

Next, we examine parameters related to the genetic initialization I , population size N_{pop} , and choice of function set G . Our goal here is to explore the potential for reducing the error due to variance and bias, respectively, in a manner achievable by a GP practitioner. As such, in these latter experiments we aim not to generate new forms of the bias-variance trade-off, but instead, to study the error components under computationally constrained experimentation.

5.1 Varying maximum LGP length

First, a series of experiments were undertaken in which the maximum length of the LGP expressions was varied. The n_{init} value (the maximum initial number of program statements) was chosen randomly and uniformly from the range $[1, 150]$, and the maximum LGP program length was set to $n_{\text{prog}} = 2n_{\text{init}}$. Over 200 values of n_{prog} (including repeats) were explored, and $\widehat{\text{mse}}$ (or $\widehat{\text{mmr}}$), $\widehat{\text{bias}}$ and $\widehat{\text{var}}$ were computed for each value. The G_{short} function set was used throughout.

Table 4. Integrated error and bias-variance decomposition on the target benchmarks. “Exact” solutions, for the regression problems DistND, refer to individuals whose symbolic expression reduces exactly to the target function f^* (impossible for MexHat, in the absence of Euler’s number e) and, for the classification problems Spiral and Cenparmi, to those achieving a zero fitness.

problem	func. set	error	bias	var.	#exact
regression problems		\widehat{mse}	\widehat{bias}	\widehat{var}	
MexHat	G_{short}	0.0486	0.0161	0.0324	—
	G_{long}	0.0021	0.0002	0.0019	—
Dist1D	G_{short}	0.0000	0.0000	0.0000	50
	G_{long}	0.0000	0.0000	0.0000	50
Dist2D	G_{short}	2.4742	0.2668	2.2073	17
	G_{long}	0.0000	0.0000	0.0000	50
Dist3D	G_{short}	13.895	6.509	7.386	2
	G_{long}	0.6081	0.0178	0.5903	43
Dist4D	G_{short}	14.068	8.103	5.966	0
	G_{long}	1.227	0.093	1.134	44
Dist5D	G_{short}	16.395	11.267	5.129	0
	G_{long}	5.972	1.528	4.444	4
classification problems		\widehat{mmr}	\widehat{bias}	\widehat{var}	
Spiral	G_{short}	0.1614	0.0428	0.1186	0
	G_{long}	0.1173	0.0235	0.0938	5
Cenparmi	G_{short}	0.0993	0.0681	0.0312	0
	G_{long}	0.0931	0.0613	0.0318	0

Our first question dealt with the best choice of model for the integrated error quantities over an independent parameter λ (such as n_{prog} here). Based on experimentation with several curve types, we elected to fit \widehat{mse} (or \widehat{mmr}) and \widehat{bias} to a four-parameter model $err_4(\lambda) = \alpha e^{\beta\lambda} + \gamma\lambda^2 + \epsilon$, and \widehat{var} to either the same curve, or to a straight line. Our choices are motivated in Annex B. Figure 4 shows the results. The data fits closely to the expected view of the bias-variance decomposition of a non-parametric learner over a complexity measure³. Indeed, as the maximum complexity of the evolved solutions increases, the bias term quickly drops to a level close to zero. Simultaneously, however, the variance term rises, showing an increased propensity to overfit.

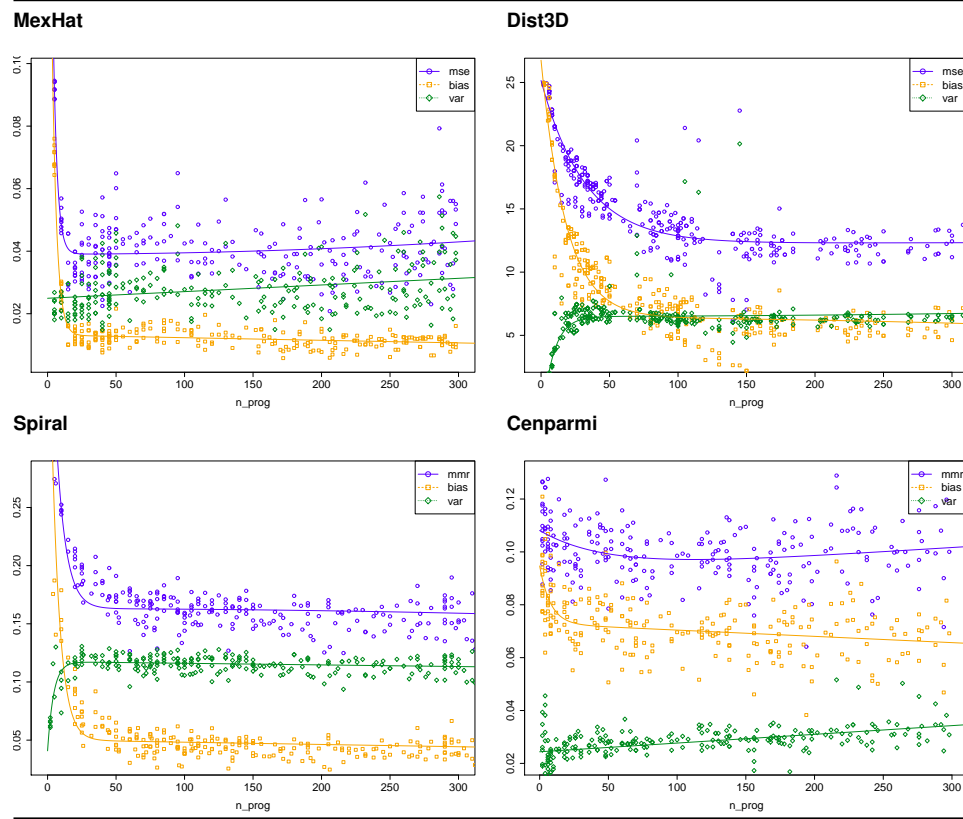
Clearly, selecting a maximum length too low will significantly sabotage results. An important question is, on the contrary, whether a practitioner could plausibly be expected to choose higher or intermediate values so as to favor good results. To verify this, we broke our independent variable n_{prog} into a series of equally sized bands of length 50. We report the mean \widehat{mse} (or \widehat{mmr}) over the best band, as opposed to over all runs, including the improvement and the certainty (according to a two-sample t -test):

benchmark	best n_{prog} band	mean \widehat{mse} or \widehat{mmr} (all)	mean \widehat{mse} or \widehat{mmr} (best)	improve	certainty
MexHat	[200, 250]	0.0423	0.0381	11%	< 0.03
Dist3D	[150, 200]	15.41	12.16	27%	<< 0.001
Spiral	[150, 200]	0.1675	0.1539	9%	<< 0.001
Cenparmi	[150, 200]	0.1004	0.0944	6%	< 0.002

Hence, we conclude that some reasonable amount of one-dimensional experimentation with n_{prog} could be expected to lead to improvements.

³ The one set of curves which break this rule are the Spiral curves, where no increase in error rate is seen as the variance points remain flat: we note, however, that the training and test data are the same for this problem, which excludes the possibility of overfitting.

Fig. 4. Best-fit curves of $\widehat{\text{bias}}$ (orange, using err_4), $\widehat{\text{var}}$ (green, using err_4 or a straight line) and $\widehat{\text{mse}}$ or $\widehat{\text{mmr}}$ (blue, using err_4), varying over n_{prog} .



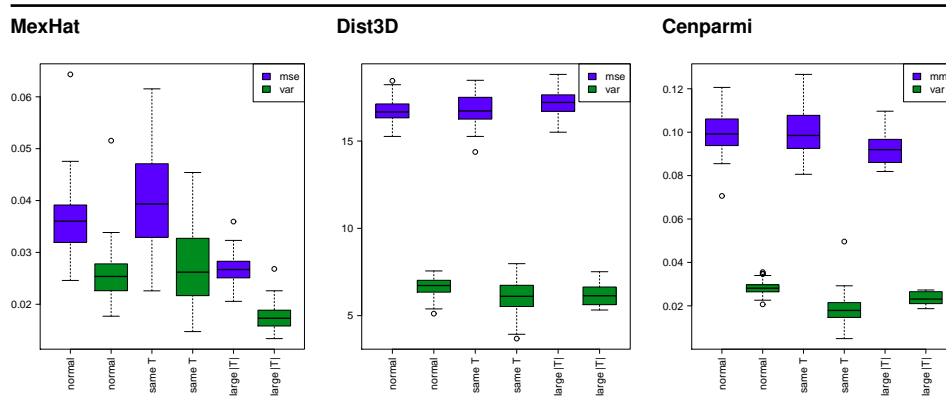
5.2 Variance due to genetic initialization

Here, we confirm our intuition regarding the role of choice of random seed for the efficacy of the evolutionary algorithm. Our goal is to estimate the proportion of variance resulting from the training samples T versus the random seed I . For our regression problems, MexHat and DistND, using the G_{short} function set, we computed $\widehat{\text{mse}}$, $\widehat{\text{bias}}$ and $\widehat{\text{var}}$ as described in Section 2. Note that the Spiral classification problem uses a static set of samples, hence could not be analyzed in this fashion. For the other classification problem, Cenparmi, we calculated $\widehat{\text{mmr}}$. First, we used different random seeds $I^{(k)}$ with the same training sample set T (“same T ”); then, we used different seeds $I^{(k)}$ paired with different sample sets $T^{(k)}$ (“normal”). In both cases, the size of the sample set $|T|$ was fixed, as indicated in Table 2. Finally, we computed a third experiment in which the training sets were larger (“large $|T|$ ”), with $|T^{(k)}| = 6400$ for the regression benchmarks and $|T^{(k)}| = 600$ for the Cenparmi benchmark. The results over approximately 50 runs for each trial are shown in Figure 5

Comparing the “normal” runs against the “same T ” runs, we see statistically significant gains in performance for the latter in the case of the Cenparmi benchmark only, although the absolute difference is small. This implies a smaller role for the particularities of training set selection in the generation of variance, relative to the role of the initialization seed. Similarly, comparing the “normal” runs against the “large $|T|$ ” runs shows statistically significant gains for the latter in the MexHat benchmark only. This time, the reduction in variance due to the increased training set size is approximately 40% of total variance, leaving 60% due to initialization seed. For the other two benchmarks, there is negligible difference in $\widehat{\text{var}}$. Again, we see that the selection of the initialization seed has more influence on the variance than the size of the training set, even when increased by a factor 16.

Therefore, it is clear that in these examples *the majority of the variance associated with the error rates stems from the initial sample of genetic space*. We would expect this to be reflected in the final genetic outputs.

Fig. 5. A comparison of the effect of selection of training set T on \widehat{mse} or \widehat{mmr} and \widehat{var} on three benchmarks. Plots are “Tukey-style” boxplots: dark lines are median values, boxes are based on quintiles, whiskers represent the 95% confidence interval, circles are outliers.



5.3 Varying the population size

In this third series of experiments, we elected to explore the effect of the population size N_{pop} (the steady number of evolved individuals f) on the performance of the algorithm, given a constant number of evaluations N_{eval} . This parameter plays here the role of a trade-off, which involves the amount of initial exploration taken by the EA (in a larger population), as opposed to the exploitation of the better individuals (in a smaller population). In order to avoid greater amounts of computation, we maintain the number of evaluations $N_{\text{eval}} = N_{\text{pop}} + N_{\text{new}}$ constant, i.e., diminish the number of individuals created via genetic operators, N_{new} , as N_{pop} grows.

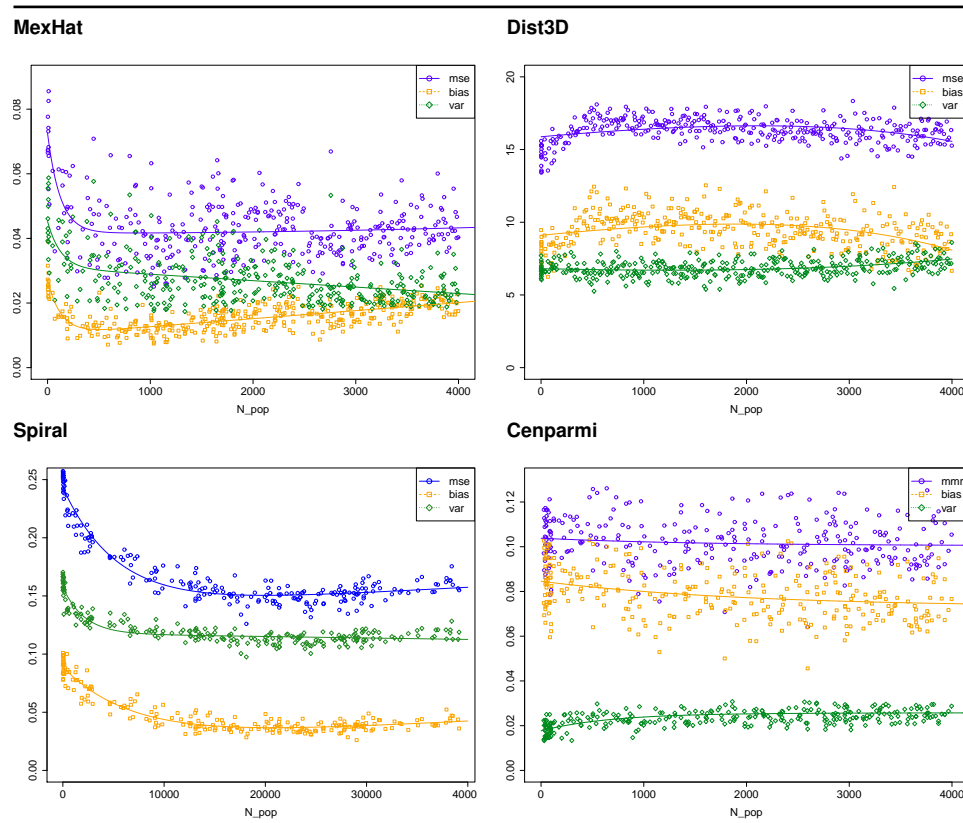
We computed over 200 samples for each problem, with ranges of [1, 4000]. Due to the different role of parameter N_{pop} , i.e., that of a trade-off rather than a measure of model complexity like n_{prog} , we re-evaluated our choice of fitting curves to model the data and decided to use err_4 for all three error measures. These results are shown in Figure 6.

In two cases, we observe that the variance component of error drops to some minimal level, and then plateaus. This suggests that following some critical population size, an adequate sample of genotypic space is found. The bias component of error also drops initially, and then gradually begins to climb. This is likely due to a decrease in evolutionary evaluations, where unnecessarily large initial population sizes encroach on the time devoted to exploitation in the algorithm. (It is unlikely that the bias is caused by the discovery of difficult-to-find local minima via larger samples, since these would not only increase bias but also lower variance.) In the other two cases, no significant effect on variance was observed, suggesting that small populations sample the genetic space sufficiently well.

A key point here is that the lowest values of variance for all these problems is still significantly higher than the variance we associate with the selection of the training set (see Section 4.1). That is, *we cannot reasonably expect larger initial samples of the genomic space to eliminate the variance due to initialization.*

An interesting effect can be seen with the Dist3D benchmark: namely, the best results were observed with a very small population, followed by an increase in error rates, and finally a decrease. Indeed, the error scores seen at the larger population sizes are significantly better than the middle range. This difference is driven largely by bias, not variance. We are at a loss to explain this behaviour.

Again, we asked whether or not a practitioner could hope to select optimal values of N_{pop} in order to increase success. We broke the possible values into bands of size 500. We summarize our results below (noting that no significant changes were observed with the Cenparmi runs):

Fig. 6. Best-fit curves, varying over N_{pop} .

benchmark	best n_{prog} band	mean $\widehat{\text{mse}}$ or $\widehat{\text{mmr}}$ (best)	mean $\widehat{\text{mse}}$ or $\widehat{\text{mmr}}$ (all)	improve	certainty
MexHat	[2500, 3000]	0.0393	0.0436	10%	< 0.0002
Dist3D	[0, 500]	15.55	16.32	5%	$\ll 0.001$
Spiral	[3500, 4000]	0.1484	0.1709	15%	$\ll 0.001$

The conclusion here is that, in some cases, modest but significant improvements can be made by adjusting N_{pop} .

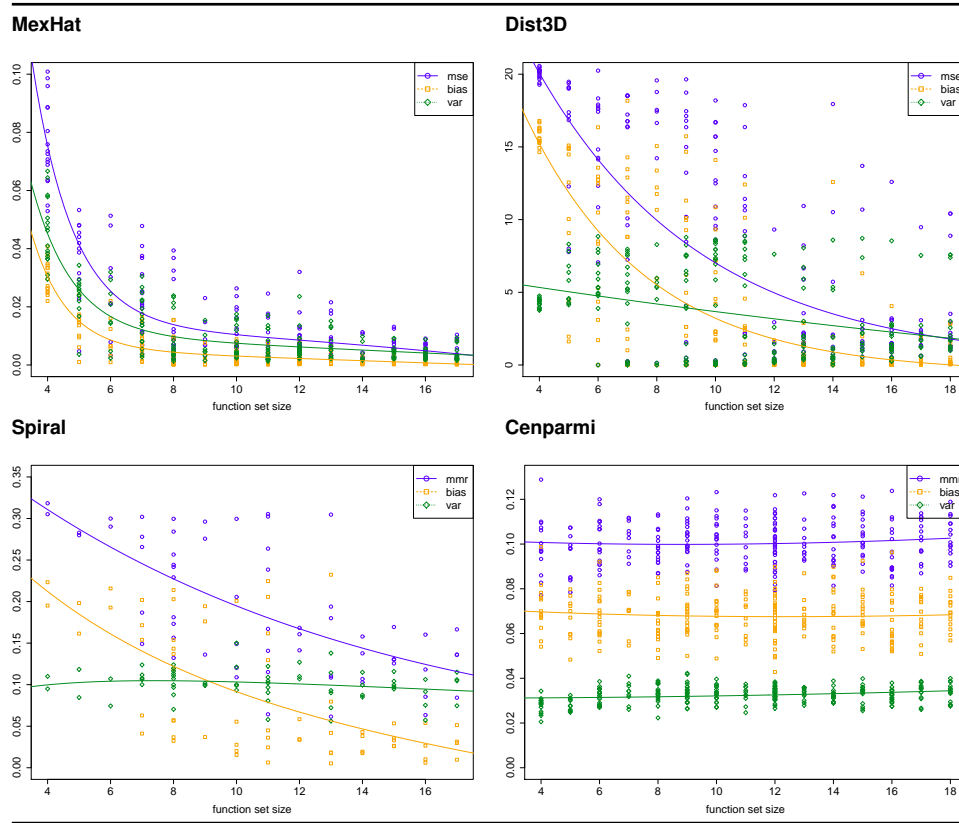
5.4 Varying the function set

In a final series of experiments, we elected to vary the size of the function set, $|G|$, and its membership. Here, each run uses a random subset of G_{long} as a selection of available choices for the evolutionary algorithm. In each subset G , the basic functions {plus, minus, times, div} were included by default. Next, an integer was chosen randomly and uniformly in $[0, 14]$ and that many additional functions were drawn from G_{long} (see Table 1) to form the pool available to the evolutionary algorithm. Each function was equally likely to be chosen by genetic initialization or mutation. Results are shown in Figure 7.

For all benchmarks save Cenparmi, there is a sharp increase in performance with the number of functions included (in the case of Cenparmi, the performance is unchanged at all sizes). It is immediately evident that the expected performance, in terms of $\widehat{\text{mse}}$ or $\widehat{\text{mmr}}$, improves rapidly with more functions. Although there is some drop in variance, too, especially with values near four functions, the primary gains are made via reduction in $\widehat{\text{bias}}$, until the value drops nearly to zero. The $\widehat{\text{var}}$ score, on the other hand, appears to plateau before this.

The fact that $\widehat{\text{var}}$ does not begin to increase with more functions is interesting. It suggests that the addition of new choices to the function set is not an increase in model complexity, i.e., that it does not generally enable the

Fig. 7. Best-fit curves, varying over function set size $|G|$. Note the ill-fit curves for Spiral and Dist3D near the mid-range values, due to the arbitrary presence or absence of particularly useful functions.



production of things previously impossible. Instead, we should view it as a means of skewing the distribution of solutions so as to make relevant solutions more probable. Thus, we propose that the function set can be used to control the bias of the system, that is, *introduce heuristics that may (or may not) be appropriate to a given problem*.

If the above hypothesis is correct, we should be able to see changes to output associated with the addition of particular functions while the set size is held constant. To test this, we generated over 50 runs of the system where the function set was selected as above, but the size was fixed to 11 (the necessarily included functions {plus, minus, times, div} along with 7 additional randomly chosen functions from G_{long}). For each function, we compared the mean of $\widehat{\text{mse}}$ in those runs which included the function versus those runs which did not.

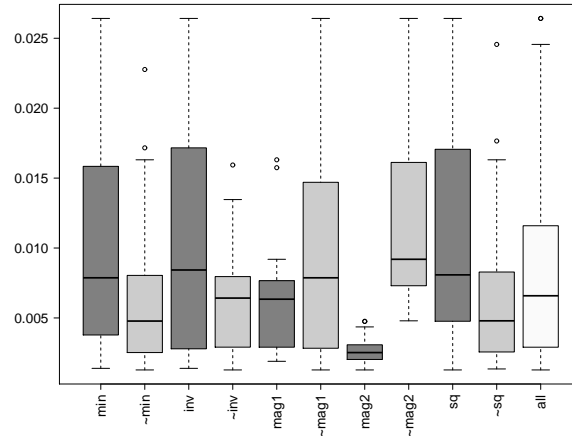
Indeed, we discovered several important results. Below we list all those functions with significant certainty ($p < 0.05$) for the MexHat problem, noting that the mean $\widehat{\text{mse}}$ score over all runs is 0.0082 (see Figure 8 for a graphic comparison):

function g for MexHat	$\widehat{\text{mse}}$ without g	$\widehat{\text{mse}}$ with g	$\Delta \widehat{\text{mse}}$	cert.
min	0.0064	0.0102	+0.0038	$p < 0.03$
inv	0.0062	0.0106	+0.0044	$p < 0.01$
mag1	0.0097	0.0061	-0.0036	$p < 0.02$
mag2	0.0119	0.0027	-0.0092	$p \ll 0.01$
square	0.0065	0.0107	+0.0042	$p < 0.02$

The majority of the functions had an adverse effect (8 out of 14 increased $\widehat{\text{mse}}$), implying that either they tended towards overfitting or that evolutionary effort was wasted on removing them from the potential solutions. The most significant single function, mag2, had a highly beneficial effect, decreasing the expected $\widehat{\text{mse}}$ score by about 67%.

Most of the improvement in efficacy when augmenting the function set can probably be ascribed to this single function.

Fig. 8. Comparison of the mean $\widehat{\text{mse}}$ score on runs with function set sizes of 11, grouped by the presence or absence of particular functions g . Boxplot conventions as in Fig. 9.



Similarly, for the Spiral problem, where the mean $\widehat{\text{mse}}$ score for all runs is 0.1549, we found:

function g for Spiral	$\widehat{\text{mse}}$ without g	$\widehat{\text{mse}}$ with g	$\Delta \widehat{\text{mse}}$	cert.
cos	0.1922	0.1134	-0.0788	$p \ll 0.01$
thresh	0.1251	0.1765	+0.0514	$p < 0.03$
mag2	0.1981	0.1324	-0.0657	$p < 0.01$

The full benefits of increasing the function set can be accounted for by the inclusion of two favourable functions, cos and mag2, undoing the damage caused by the other functions, which had a mostly adverse effect (7 of the remaining 12). The same pattern is seen with the Dist3D problem (where the mean $\widehat{\text{mse}}$ score for all runs is 5.175):

function g for Dist3D	$\widehat{\text{mse}}$ without g	$\widehat{\text{mse}}$ with g	$\Delta \widehat{\text{mse}}$	cert.
sin	4.099	7.137	-3.038	$p < 0.05$
mag2	9.556	0.414	+9.142	$p \ll 0.01$

The most useful function, mag2, accounts for all gains when increasing the function set. This is not surprising, as mag2 is a repeated element in the target solution f_{Dist3D} . Again, the majority of additional functions (10 of 13) have an adverse effect on the problem (increasing the fitness), but a very useful function can compensate this, and provide large gains to overall performance, primarily through elimination of bias.

For the Cenparmi problem, there are several moderately significant functions, but none whose effect increased or decreased error by more than 0.005.

6 Conclusions

Our study has generally confirmed the view of GP as a low-bias and high-variance approach to regression and classification problems. Furthermore, our analysis of variation on the maximum program length n_{prog} has shown results consistent with the bias-variance decomposition of a non-parametric learning technique.

We have reached several key conclusions from this study. While these results have been seen in particular contexts in the literature, here we contrast their effects between benchmarks, and quantify the expected effect. The conclusions are:

- **Initialization creates the most variance:** The variance associated with GP runs is largely due to the initialization seed, and secondarily to the selection of training samples. Further, increasing the sample of the genomic space taken in the population cannot be realistically expected to reduce this variance.
- **Parameters can be optimized:** For all three parameters that we examined (maximum program length n_{prog} , population size N_{pop} , and function set G), one-dimensional selection of a reasonably sized band of values usually led to significant improvements in overall results. Of the three parameters, the largest gains were obtained by making minor changes to the function sets: indeed, in three of four benchmarks, the inclusion of one appropriately chosen function affected performance more than the best expected gains from tuning the other two parameters. In none of the benchmarks was the inclusion of all functions detrimental. The consistency of these results between benchmarks suggests that this conclusion can be generalized.
- **Population size effects are unclear:** The choice of population size, N_{pop} , led to largely inconsistent results. For two benchmarks, variance could be decreased with larger initial populations. Along with this decrease was an increase in bias, due to the lessened efforts devoted to genetic optimization. For the other two benchmarks, significant changes in variance were not seen.
- **Larger function sets are better:** Regarding the choice of function set G for inclusion in the genetic search space, the widest possible space was consistently preferred by evolution, reflected in a steady decrease of regression or classifier bias to near-zero levels. This was true despite the fact that the majority of functions were demonstrably detrimental to the evolvability of the problem. Thus it appears easier for evolution to eliminate ill-suited heuristics than to construct well-suited heuristics from more primitive operators. In particular, when increasing the function set size, we found no increase in either the average error or the variance of the results, thus providing evidence against the hypothesis of Fitzgerald and Ryan [7].
- **Well-chosen functions are best:** In most cases we explored here, there existed some non-standard functions in the larger function set very well suited to the problem at hand. It is these functions which accounted for the majority of gain in efficacy.

7 Future directions

Today, GP is increasingly being applied to knowledge extraction, where a symbolic description of a given database is desired. For instance, GP serves to extract scientific hypothesis from selected databases [2, 30, 33]; extract symbolic features from image databases [15, 16, 25, 34]; explore the space of network generators to create predictive descriptions of agent behaviours or complex networks [22, 23]; and other engineering-related applications [18]. In all these tasks, the genetic component of the evolved solution has definite meaning, possibly independently of the evaluation of the solution.

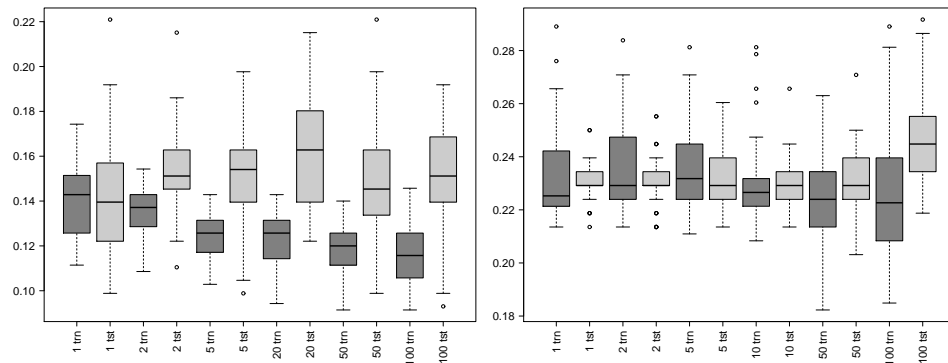
The most popular philosophy of science generally admits any model which makes useful and testable (falsifiable) predictions, and is parsimonious [21]. These conditions, however, are the product of an age in which the general assumption was made that only a few competing hypothesis would be available at any time, and hence, that determination of the most accurate or parsimonious solution would be simple. In the case of automated knowledge extraction, the possibility exists that indefinitely many models can be posited without any clear means of determining a best one: generalization becomes a multi-dimensional question, and parsimony, if at all definable, is potentially subject to the non-computability of minimal program length.

While in some cases human-understandable (or even elegant) solutions are discovered [2, 30], generally speaking, little attention has been paid to the matter. This study has shown that these issues are problem-dependent: in cases where a clear solution existed in the space (such as the DistND regression problems) genotypic convergence was possible, while in other cases (such as the MexHat regression problem) many competing genotypically distinct solutions existed. The consequences of consistent genetic diversity on the capacity to extract knowledge automatically remains to be investigated.

A A note on several other UCI databases

In the course of conducting this research, we also experimented with several popular data sets from the University of California, Irvine (UCI) [9]. They were explored in some detail, but ultimately rejected as inappropriate for this style of research. Specifically, we worked with the Breast Cancer Wisconsin (Diagnostic) data set [32], the Pima Indians Diabetes data set [31] (both original and corrected versions), and the Statlog (Australian Credit Approval) data set [29]. Performance was systematically tested by measuring MR for different program lengths: $n_{\text{prog}} \in \{1, 2, 20, 50, 100\}$. We discovered that in all three cases the naive application of GP was incapable of improvement when given additional complexity (i.e., increasing n_{prog}), relative to the natural stochasticity due to the selection of training and test sets. For all data sets, the difference in MR on randomly chosen test samples was not significantly different between $n_{\text{prog}} = 2$ and $n_{\text{prog}} = 100$ (over 30 runs, a textbook t -test did not discover any trends with $p < 0.1$). This implied that a simple threshold on one or two input variables was the best discoverable performance by a naive technique.

Fig. 9. Distribution of training MR (dark gray) and test MR (light gray) for at least 30 runs. (Left) Results of LGP on the credit card data set varying over n_{prog} . (Right) Results of neural networks on the corrected diabetes data set varying over the number of hidden nodes.



Lest our results be interpreted as a failure of our particular approach to GP, we re-ran the same experiments using another non-parametric learner, a neural network. Specifically, neural networks with a varying number of hidden nodes were trained and tested on the above databases, and trained via backpropagation (using a sigmoid activation function $\phi(v) = 1.7159 \cdot \tanh(2v/3)$, and 50 epochs of training). The number of hidden neurons used varied over the set $\{1, 2, 5, 10, 50, 100\}$ and 30 runs. In all cases, the test error did not change significantly, save for the Diabetes database, where in fact the test error worsened significantly.

An illustration of these results is shown in Figure 9. In conclusion of these findings, we deemed the above three databases too noisy for non-parametric learning, and recommend future researchers to proceed with caution.

B Curve selection

Selection of a model (curve) for data fitting was carried out using the MexHat domain, using the G_{short} function set, and over 100 runs. All curves were fit using the Gauss-Newton method of non-linear regression. Goodness-of-fit error was the residual standard error. All polynomials up to degree seven were fit. We also tested three curves designed to resemble expected curve shape (from previous experiments with MSE):

$$\begin{aligned} \text{err}_5(\lambda) &= \alpha e^{\beta\lambda} + \gamma\lambda^2 + \delta\lambda + \epsilon \\ \text{err}_4(\lambda) &= \alpha e^{\beta\lambda} + \gamma\lambda^2 + \epsilon \\ \text{err}_{4'}(\lambda) &= \alpha e^{\beta\lambda} + \delta\lambda + \epsilon \end{aligned} \quad (13)$$

The best error rate for fitting the mse data was achieved by the err_4 curve (0.00809), slightly outperforming the other two exp curves, and even outperforming the more complex 7-term polynomial (0.00865). Further simplifications to the exp curves rapidly increased the error rate. Hence, we elected to use err_4 as a default guess for all curves, with other err curves substituted in the case of an improvement of error greater than 0.001. The var curves were typically modelled via straight lines, unless a curve improved error by more than 0.001.

Acknowledgement: This work was supported by an ISC-PIF/DIM 2010 Région Paris Ile-de-France fellowship to T.K.

References

- [1] A. Agapitos, A. Brabazon, and M. O'Neill, *Controlling overfitting in symbolic regression based on a bias/variance error decomposition*, Parallel Problem Solving from Nature (PPSN XII), Springer, 2012, pp. 438–447.
- [2] M.J. Baptist, V. Babovic, J. Rodriguez-Uthurburu, M. Keijzer, R.E. Uittenbogaard, A. Mynett, and A. Verwey, *On inducing equations for vegetation resistance*, Journal of Hydraulic Research **45** (2007), no. 4, 435–450.
- [3] M.F. Brameier and W. Banzhaf, *Linear genetic programming*, Springer, 2006.
- [4] E.K. Burke, S. Gustafson, and G. Kendall, *Diversity in genetic programming: an analysis of measures and correlation with fitness*, Evolutionary Computation, IEEE Transactions on **8** (2004), no. 1, 47–62.
- [5] S.K. Chalup and L.S. Wiklendt, *Variations of the two-spiral task*, Connection Science **19** (2007), no. 2, 183–199.
- [6] S.E. Fahlman and C. Lebiere, *The cascade-correlation learning architecture*, Advances in neural information processing systems 2 (David S. Touretzky, ed.), Morgan Kaufmann Publishers Inc., 1990, pp. 524–532.
- [7] J. Fitzgerald and C. Ryan, *On size, complexity and generalisation error in GP*, Proceedings of the 2014 conference on Genetic and evolutionary computation, ACM, 2014, pp. 903–910.
- [8] ———, *Selection bias and generalisation error in genetic programming*, Sixth International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN2014, 2014.
- [9] A. Frank and A. Asuncion, *UCI machine learning repository* (<http://archive.ics.uci.edu/ml/>), 2011.
- [10] S. Geman, E. Bienenstock, and R. Doursat, *Neural networks and the bias/variance dilemma*, Neural Computation **4** (1992), no. 1, 1–58.
- [11] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed., Springer, 2008.
- [12] David Jackson, *Phenotypic diversity in initial genetic programming populations*, Genetic Programming, Springer, 2010, pp. 98–109.
- [13] M. Keijzer and V. Babovic, *Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations*, Proceedings of the European Conference on Genetic Programming (London, UK), Springer-Verlag, 2000, pp. 76–90.
- [14] R. Kohavi and D.H. Wolpert, *Bias plus variance decomposition for zero-one loss functions*, Machine Learning: Proceedings of the Thirteenth International Conference (L. Saitta, ed.), Morgan Kaufmann Publishers, Inc., 1996.
- [15] T. Kowaliw and W. Banzhaf, *The unconstrained automated generation of cell image features for medical diagnosis*, Conference on Genetic and evolutionary computation (GECCO), 2012, pp. 1103–1110.
- [16] T. Kowaliw, J. McCormack, and A. Dorin, *Evolutionary automated recognition and characterization of an individual's artistic style*, IEEE Congress on Evolutionary Computation (CEC), 2010.
- [17] J. Koza, *Genetic programming: On the programming of computers by means of natural selection*, MIT Press, 1992.
- [18] J.R. Koza, *Human-competitive results produced by genetic programming*, Genetic Programming and Evolvable Machines **11** (2010), no. 3-4, 251–284.
- [19] K.J. Lang and M.J. Wittbrock, *Learning to tell two spirals apart*, Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 1988.
- [20] C.L. Liu, K. Nakashima, H. Sako, and H. Fujisawa, *Handwritten digit recognition: benchmarking of state-of-the-art techniques*, Pattern Recognition **36** (2003), no. 10, 2271–2285.
- [21] J. Losee, *A historical introduction to the philosophy of science*, 4th ed., Oxford University Press, 2001.
- [22] T. Menezes and C. Roth, *Automatic discovery of agent based models: An application to social anthropology*, Advs. Complex Syst. **16** (2013), no. 1350027.
- [23] ———, *Symbolic regression of generative network models*, Scientific Reports **4** (2013), no. 6284.
- [24] J.F. Miller, *Cartesian genetic programming*, Cartesian Genetic Programming (Julian F. Miller, ed.), Natural Computing Series, Springer, 2011, pp. 17–34.
- [25] G. Olague and L. Trujillo, *Interest point detection through multiobjective genetic programming*, Applied Soft Computing **12** (2012), no. 8, 2566–2582.

- [26] M. Oltean, C. Groșan, L. Dioșan, and C. Mihăilă, *Genetic programming with linear representation: a survey*, International Journal on Artificial Intelligence Tools **18** (2009), no. 02, 197–238.
- [27] R. Poli, *Parallel distributed genetic programming*, New Ideas in Optimization (D. Corne, M. Dorigo, and F. Glover, eds.), McGraw-Hill, 1999.
- [28] R. Poli, W.B. Langdon, and N.F. McPhee, *A field guide to genetic programming*, Lulu Enterprises, 2008.
- [29] J.R. Quinlan, *C4.5: programs for machine learning*, Morgan Kaufmann, San Francisco, CA, USA, 1993.
- [30] M. Schmidt and H. Lipson, *Distilling Free-Form Natural Laws from Experimental Data*, Science **324** (2009), no. 5923, 81–85.
- [31] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, and R.S. Johannes, *Using the adap learning algorithm to forecast the onset of diabetes mellitus*, Johns Hopkins APL Technical Digest **10** (1988), 262–266.
- [32] W. Street, W. Wolberg, and O. Mangasarian, *Nuclear feature extraction for breast tumor diagnosis*, IS&T/SPIE 1993 International Symposium on Electronic Imaging, vol. 1905, 1993, pp. 861–870.
- [33] J.B. Voytek and B. Voytek, *Automated cognome construction and semi-automated hypothesis generation*, Journal of Neuroscience Methods **208** (2012), no. 1, 92–100.
- [34] M. Zhang, *Improving object detection performance with genetic programming*, International Journal on Artificial Intelligence Tools **16** (2007), no. 05, 849–873.