Genetic Programming and Evolvable Machines (2021) 22:73–100
https://doi.org/10.1007/s10710-020-09391-4

Choosing function sets with better generalisation performance for symbolic regression models

Miguel Nicolau[1] · **Alexandros Agapitos**[2]

Received: 27 April 2019 / Revised: 24 February 2020 / Published online: 12 May 2020

Abstract

Supervised learning by means of Genetic Programming (GP) aims at the evolutionary synthesis of a model that achieves a balance between *approximating* the target function on the training data and *generalising* on new data. The model space searched by the Evolutionary Algorithm is populated by compositions of primitive functions defined in a *function set*. Since the target function is unknown, the choice of function set's constituent elements is primarily guided by the makeup of function sets traditionally used in the GP literature. Our work builds upon previous research of the effects of protected arithmetic operators (i.e. division, logarithm, power) on the output value of an evolved model for input data points not encountered during training. The scope is to benchmark the approximation/generalisation of models evolved using different function set choices across a range of 43 symbolic regression problems. The salient outcomes are as follows. Firstly, Koza's protected operators of division and exponentiation have a detrimental effect on generalisation, and should therefore be avoided. This result is invariant of the use of moderately sized validation sets for model selection. Secondly, the performance of the recently introduced analytic quotient operator is comparable to that of the sinusoidal operator on average, with their combination being advantageous to both approximation and generalisation. These findings are consistent across two different system implementations, those of standard expression-tree GP and linear Grammatical Evolution. We highlight that this study employed very large test sets, which create confidence when benchmarking the effect of different combinations of primitive functions on model generalisation. Our aim is to encourage GP researchers and practitioners to use similar stringent means of assessing generalisation of evolved models where possible, and also to avoid certain primitive functions that are known to be inappropriate.

Keywords Symbolic regression · Genetic Programming · Machine learning · Generalisation · Overfitting · Data-driven modelling

✉ Miguel Nicolau
Miguel.Nicolau@ucd.ie

Extended author information available on the last page of the article

1 Introduction

Supervised learning based on a finite training sample has two conflicting objectives, those of *approximating* a target function on the training data, and *generalising* on new examples not seen during training. Statistical learning theory [71] dictates that in order to attain a good trade-off, one needs to carefully match the number of training examples to the complexity of the output model. Genetic Programming (GP) [41] searches a model space, which is populated by compositions of primitive functions that are collectively defined in a primitive *function set*. There is certainly a relationship between the complexity of an evolved response surface, model-size constraints and makeup of the function set, albeit a not well-understood one. Previous work has mainly focused on studying the relationship between model size and generalisation [2, 4, 46, 64, 70], or the relationship between response surface complexity and generalisation [51, 55, 72, 75], with no regards to the constituent elements of function set.

Since the target function is unknown and GP searches for models in symbolic form, GP practitioners often resort to function sets containing diverse general-purpose functions, hoping that the resulting model space will contain a good model, and that the training data will enable GP to pin down this model. The choice of primitive functions is largely dependent on function sets that are traditionally used in GP literature. The vast majority of GP applications to symbolic regression employ either the subset of basic arithmetic operators of addition, subtraction, multiplication, and some variant of protected division, or that subset combined with other popular operators, such as the trigonometric operators sine and cosine, and protected versions of logarithm, square root, and exponential functions, (e^x, x^y, and occasionally x^2 and x^3).

Keijzer [30] empirically demonstrated the shortcomings of Koza-style protected division [35] in generating asymptotes to fit the training data. These asymptotes produce very large values in the output range of an evolved function when testing the function on data points outside the training set, and therefore hinder generalisation. Ni et al. [50] approached the shortcomings of Koza-style protected division both from the point of view of asymptotes and their effect on generalisation, but also from the point of view of discontinuity that results from these asymptotes. The authors proposed the replacement of Koza-style protected division by the differentiable Analytic Quotient (AQ) operator. In conjunction with the addition, subtraction, and multiplication operators, AQ enables the evolution of function compositions that are twice differentiable and are therefore amenable to regularisation techniques such as penalisation of response surface curvature. The authors found that the AQ operator resulted in consistently lower generalisation error over a range of symbolic regression problems as opposed to runs that used Koza-style protected division.

Our extensive literature review suggests that the GP community has ignored both studies, and to a large extent continues to use the Koza-style protected division operator, along with the protected versions of logarithm, square root, and power. The current study aims at reinforcing the awareness about the detrimental

effects that some of these operators have on the generalisation of GP-evolved models, and propose alternative function set setups that are more robust on average across a range of symbolic regression problems. Robustness is quantified in terms of both approximation (training set error) and generalisation (test set error) performance. Finally, to ensure that our results are not tied to a particular GP system implementation, we investigate the approximation/generalisation effects of function sets using two different systems: a standard expression-tree GP system implemented in Java, and a Grammatical Evolution system implemented in C++.

The rest of the paper is structured as follows: Sect. 2 discusses the interaction between GP model generalisation and primitive operators, and reviews protected operators, and alternatives to the use of these; Sect. 3 reviews the use of function sets in published research papers and in software packages; Sect. 4 describes the experiment setup, the benchmark problems tackled, the systems employed, and the performance measures used, while Sect. 5 analyses the performance results. Finally, conclusions are drawn in Sect. 6.

2 Background

2.1 Function sets, regularisation, and generalisation

The goal of GP in a supervised learning setting is to find a function $h : x \mapsto y$ given the information provided in the form of a finite set of training examples $\{(x_i, y_i)\}_{i=1}^{N}$, where y is the response variable and $x \in \mathbb{R}^d$ is a vector of explanatory variables. The focus of the current work is on problems where $y \in \mathbb{R}$, giving rise to symbolic regression with GP [35].

Given a loss function $L(y, h(x))$ and a joint distribution $P(x, y)$ for sampling examples, the goal of supervised learning is formulated through the following minimisation problem:

$$h^*(x) = \arg\min_{h \in H} \mathbb{E}_{x,y}\left[L(y, h(x))\right] \tag{1}$$

where H is the hypothesis set (i.e. the set of candidate function compositions h). The typical choice of loss function for the case of symbolic regression is the *squared loss* defined as $L(y, h(x)) = (y - h(x))^2$.

The expected value $\mathbb{E}_{x,y}[L(y, h(x))]$ is referred to as *generalisation error*, where both x and y are drawn from their joint distribution. It is the quantity that a supervised learning method needs to minimise in order for $h^*(x)$ to exhibit the property of *generalisation*. In practice, the expectation in Eq. 1 is approximated using an average, which is calculated on a set of examples that is disjoint from the training set.

Generalisation in GP has received considerable attention from the GP community. A recently published survey article on the topic can be found in [1]. Therein, the approximation-generalisation tradeoff is introduced through the bias/variance decomposition of the generalisation error based on the squared error measure. Model complexity is the determinant factor of this tradeoff, with too simple models resulting in low variance/high bias (underfitting), and with more complex models

resulting in high variance/low bias (overfitting). We wish to motivate our study of the effects of function sets to the generalisation of evolved programs partly through the method of *regularisation* as a means of model complexity control discussed in this section, and partly through the occurrence of certain types of *pathologies* (i.e. infinite or very large values) in the output of evolved functions discussed in a following section.

Regularisation works by augmenting the loss function with a model complexity penalty function. In a special case of m-th order Tikhonov regularisation for GP [51], the model complexity penalty function takes the form of the L_2 norm of the first m ($m = 0, 1, 2, \ldots$) derivatives of the evolved function. As examples, the zeroth-order regulariser ($m = 0$) penalises evolved functions with many extreme output values, the first order regulariser ($m = 1$) penalises functions that change rapidly, and the second order regulariser ($m = 2$) penalises functions with large curvature. The choice of primitive functions biases the search towards certain function compositions with associated derivatives, which result is different levels of model complexity as seen through the lenses of Tikhonov regularisation. If the selection of the appropriate level of model complexity is crucial to achieving the right balance between approximation and generalisation, then this selection can be controlled to some extent by the choice of respective primitive function elements.

2.2 Protected operators in GP

In order to evaluate a GP tree, the closure property must be satisfied, i.e. the output type and value of any function in the function set must be accepted as input for all such functions. Yet some arithmetic functions may generate undefined results (such as division by zero).

Koza [35] addressed this problem by defining *protected* versions of arithmetic functions, capable of avoiding the generation of undefined values. These are:

– protected division (also known as % or simply `pdiv`):

$$pdiv(x_1, x_2) = \begin{cases} x_1/x_2 & \text{if } x_2 \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

– protected logarithm (also known as `plog`):

$$plog(x) = \begin{cases} log(|x|) & \text{if } x \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

– protected square root (also known as `psqrt`):

$$psqrt(x) = \sqrt{|x|}$$

– protected exponentiation (also known as `srexpt` or `ppow`)[1]:

[1] In reality, `ppow` was defined [35] as $srexpt(x_1, x_2) = |x_1|^{x_2}$, but this does not take into account the fact that raising 0 to a negative number results in a division by zero.

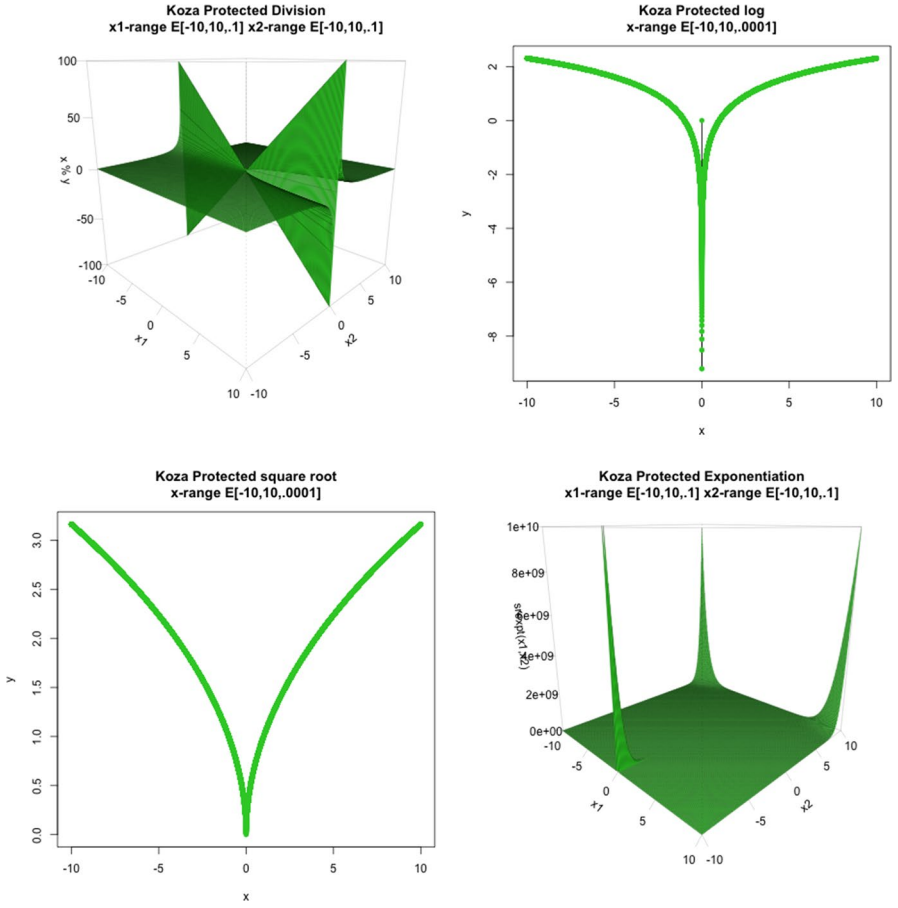


Fig. 1 From top-left to bottom-right: protected versions of x_1/x_2, $\log(x)$, $\sqrt{x}$, and $x_1^{x_2}$, grid sampled between $[-10, 10]$

$$ppow(x_1, x_2) = \begin{cases} |x_1|^{x_2} & \text{if } (x_1 \neq 0) \text{ or } (x_1 = x_2 = 0) \\ 0 & \text{otherwise} \end{cases}$$

These were originally defined with strict comparisons to zero (the approach taken in this work). Some researchers, in subsequent studies, have sometimes defined these operators based on thresholds (e.g. in *pdiv*, if $x_2 \geq 10^{-5}$ then x_1/x_2 else 1).

The problem with `pdiv`, `ppow`, and `plog` as far as generalisation is concerned is the presence of vertical asymptotes, as well as the discontinuities that result from the protection mechanism; see Fig. 1. During evolution, vertical asymptotes present in the range of certain primitives may be manifested in the form of "spikes" and discontinuities in the output of an evolved function, especially for parts of the input space not covered during training [30, 50]. On a more general note, the occurrence of vertical asymptotes and discontinuity are not inline with the most commonly used

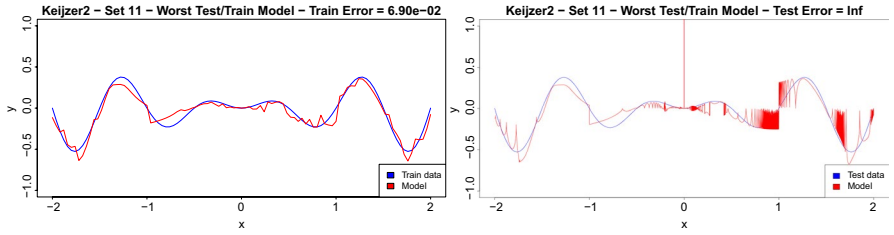


Fig. 2 Train and test data of the Keijzer2 problem, along with the predictions of a sample evolved model

form of prior information about the solution, which involves the assumption that the input-output mapping function is smooth (i.e. similar inputs produce similar outputs).

Division has a naturally occurring asymptote ($\lim_{x_2 \to 0} x_1/x_2 = \infty$), as does the logarithm function ($\lim_{x \to 0} \log x = \infty$); their protected versions do not remove these asymptotes, but avoid the undefined values of $x_1/0$ and $\log(x)$ (when $x <= 0$), respectively. The square root (and its protected version) do not have any asymptotes; the protected version avoids the creation of imaginary numbers when $x < 0$, although at the expense of creating a discontinuity in the output of the function. Finally, the power function has a naturally occurring asymptote ($\lim_{x_1 \to 0, x_2 < 0} {x_1}^{x_2} = \infty$), and two cases easily leading to infinities, when $|x_1|^{x_2} > \texttt{DBL\_MAX}$, where `DBL_MAX` is the largest value representable with the architecture used ($\approx 1.8e^{308}$ with IEEE 754 64-bit double type). Although not as extreme, these issues are also observed for the unary functions `inv` (implemented as $pdiv(1, x)$) and `exp` (implemented as $ppow(e, x)$).

These asymptotes may sometimes not occur during model building (i.e. using training data), but severely affect test performance of the evolved models. Figure 2 illustrates this. A model was evolved to solve the problem Keijzer2 (see Table 4), using a typical function (set 11, see Table 3), with a standard GP setup (see Table 5). Although some minor spikes are present in training data, they have a small detrimental effect on training fitness (0.069). When using a large test set with the same x-range, however, these asymptotes become more relevant and indeed lead to an infinity, when $x = 0$.

2.3 Alternatives to protected division

Keijzer [30] was the first to signal the dangers of `pdiv`. He proposed the use of *interval arithmetic*, a technique consisting in static analysis of generated models, maintaining at every application of a function the minimum and maximum values expected of its input(s). This allows the calculation of bounds for the function output values, and thus the detection of infinities and undefined values: for example, $f(x) = 1 \div x$, where x has a value range of $[-1, 1]$, would result in infinite bounds. Such models are deemed invalid, and can be assigned a very large error, or simply discarded. Interval arithmetic is however hard to implement, potentially costly to calculate, and not possible to use when the bounds of input variables are impossible

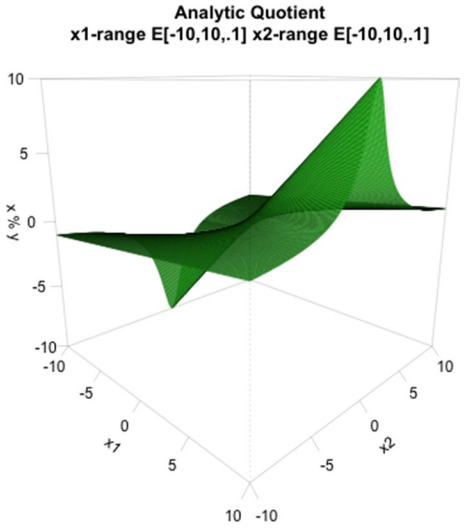


Fig. 3 Analytic Quotient operator, grid sampled between $[-10, 10]$

to specify.[2] In fact, Dick [16] has shown that interval arithmetic does not prevent invalid solutions from entering the population, and instead proposes to use GP with interval-aware search operators (crossover and mutation), improving generalisation performance.

Ni et al. [50] proposed instead to replace `pdiv` altogether. In their work, they proposed a replacement operator, Analytic Quotient (`aq`), defined as:

$$\mathtt{aq}(x_1, x_2) = \frac{x_1}{\sqrt{1 + {x_2}^2}} \tag{2}$$

This operator has the general properties of division, especially when $|x_2| \gg 1$, but reduces the range of division's (and `pdiv`'s) asymptote(s) to the value of x_1. This is shown in Fig. 3. It can be easily implemented in GE with a grammar transformation such as $<\mathtt{e}> ::= <\mathtt{e}> /(1 + \mathtt{pow}(<\mathtt{e}>, 2))$, where $<\mathtt{e}>$ is the recursive definition of an expression. In GP, several software packages either provide the `aq` operator, or provide ways to define your own functions (see Sect. 3.2).

3 Function sets in literature and in software

A typically unanswered question when using GP for symbolic regression is: *For problem x, which function set should I use?*. Although some benchmarks defined in the literature specify the function sets to be used [30, 32, 72], researchers do not necessarily abide to these specifications, and this does not answer the question as to

[2] In such cases, Keijzer [30] proposes using the ranges observed in the training set, but those ranges can change with unseen data, particularly when extrapolating, or when modelling time-series data.

Table 1 Functions under consideration and their use in GP journal publications in the period 2016–2019

Function	References
plus	[3, 5–8, 10–15, 17, 19, 23–27] [28, 29, 31, 33, 34, 36–39, 43, 44, 48, 49, 58–62] [9, 65–68, 76–78]
minus	[3, 5, 7, 8, 10–15, 17, 19, 23–28] [29, 31, 33, 34, 36–39, 43, 44, 48, 49, 58–62, 65] [9, 66–68, 76–78]
times	[3, 5–8, 10–15, 17, 19, 23–27] [28, 29, 31, 33, 34, 36–39, 43, 44, 48, 49, 58–62] [9, 65–68, 76–78]
div	[3, 5, 7, 8, 10–12, 14, 15, 19, 25–29, 31, 33, 34] [9, 36, 38, 39, 43, 44, 48, 49, 59, 61, 62, 65–68, 76, 78]
inv	[6, 15]
aq	
pow	[9, 12, 29, 66, 68]
exp	[3, 7, 9, 11, 15, 17, 19, 28, 31, 36, 44, 48, 59–61, 68, 76, 78]
log	[3, 7, 8, 10, 13, 15, 19, 23, 25, 28, 31, 36, 39, 44] [59–62] [9, 67, 68, 76–78]
sqrt	[3, 7, 8, 10–13, 15, 19, 25, 28, 29, 31, 39, 44, 48, 61, 65] [9, 66–68, 76]
sin	[3, 7, 13, 15, 17, 19, 23, 28, 31, 37, 44] [9, 48, 59–62, 66] [67, 68, 76–78]
cos	[3, 7, 9, 13, 15, 17, 19, 23, 28, 31, 44, 48, 59–62, 66, 67] [68, 76–78]
tanh	[3, 9, 13, 15, 19, 23, 28, 31, 37, 44, 60, 67, 68, 76, 78]

which function set should be used with real-world problems, particularly when no prior domain knowledge is available.

3.1 Function sets used in published papers

Table 1 shows the most commonly used functions in recent literature, for symbolic regression problems. Out of 2,225 GP publications from the GP Bibliography [40] for the years 2016-2019, we extracted 852 journal articles. Of those articles, there were 69 that mentioned "regression" in their title or keywords. Out of these, there were 44 articles that clearly defined the GP function set(s) used. This data is

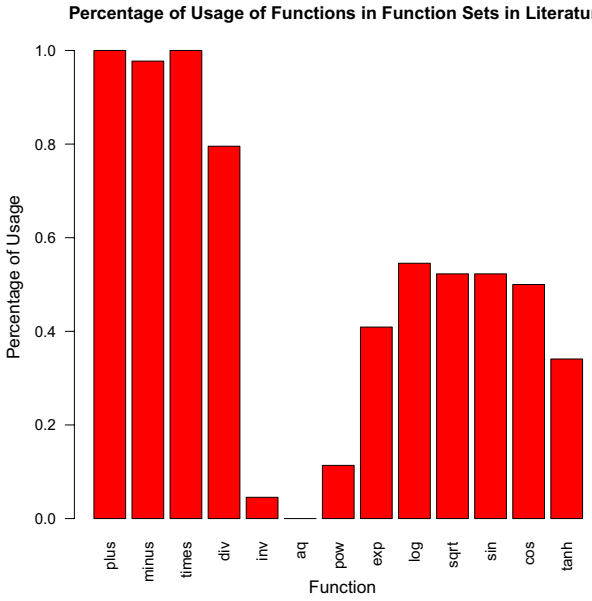


Fig. 4 Percentage of publications (from 44 journal articles extracted from literature) that use each of the functions under consideration in this study in their function sets

summarised in Fig. 4; when unspecified, it is assumed that protected versions of operators were used.

Although there were no journal papers matching our search criteria that used the Analytic Quotient function, there are a few examples of its use in recent literature [18, 47, 69].

3.2 Function sets provided in software packages

There are several Evolutionary Computation software packages available, that implement the Genetic Programming or Grammatical Evolution algorithms. Some (but not all) offer default function set choices, and protected implementations of arithmetic operators.

All of the following were chosen as actively maintained packages (latest version less than 2 years old), and all were accessed on the 9th September 2019. Their protected (or otherwise) operators are shown in Table 2.

3.2.1 ECJ

The ECJ framework [45] is one of the earliest and most famous EC implementations, first published in 1998, and written in Java. It seems to provide no default function set for GP. Although there is a protected version of division, the

Table 2 Software implementation of division, logarithm, square root and power operators in available evolutionary computation software packages

System	$div(x_1, x_2)$	$log(x)$	$sqrt(x)$	$pow(x_1, x_2)$
ECJ [45]	x_1/x_2 if $x_2 \neq 0$ 1 otherw.	$\log(\lvert x\rvert)$ if $x \neq 0$ 0 otherw.	$\sqrt{x}$	N/A
DEAP [21]	x_1/x_2 if $x_2 \neq 0$ 1 otherw.	N/A	N/A	N/A
GPLab [63]	x_1/x_2 if $x_2 \neq 0$ x_1 otherw.	$\log(\lvert x\rvert)$ if $x \neq 0$ 0 otherw.	$sqrt(x)$ if $x \geq 0$ 0 otherw.	${x_1}^{x_2}$ if no error 0 otherw.
PonyGE [20]	x_1/x_2 if $x_2 \neq 0$ 1 otherw.	$\log(1 + \lvert x\rvert)$	$\sqrt{\lvert x\rvert}$	$\lvert x_1\rvert^{x_2}$
Libgges [73]	x_1/x_2 if $x_2 \neq 0$ 1 otherw.	$log(\lvert x\rvert)$ if $x \neq 0$ 0 otherw.	$\sqrt{\lvert x\rvert}$	${x_1}^{x_2}$

inversion operator is unprotected; it also does not provide protected versions of square root and power.

3.2.2 DEAP

DEAP [21] is a heavily parallelised evolutionary computation framework, written in Python. Although it provides a protected version of division, it leaves the implementation of other protected operators to the user. Its default function set for symbolic regression with GP is $(+, -, *, \texttt{pdiv}, neg, \cos, \sin)$.

3.2.3 GPLab

GPLab [63] is a Matlab implementation of GP. Other than the operators shown in Table 2, it also provides protected versions of $\log_2(x)$ and $\log_{10}(x)$. No default function set seems to be provided.

3.2.4 PonyGE2

PonyGE2 [20] is a software package implementing GE in Python. It provides two GE symbolic regression default function sets: $(+, -, *, \texttt{pdiv}, \sin, \tanh, \exp, \texttt{plog})$, and $(+, -, *, \texttt{aq})$. It is the only software package from those analysed that provides an implementation of the Analytic Quotient operator.

3.2.5 libgges

libgges [73] is a C implementation of several grammar-guided GP approaches, including GE. Its default function set is $(+, -, *, \texttt{pdiv}, \sin, \cos, \exp, \texttt{inv}, \texttt{plog})$. The inversion operator is implemented as $pdiv(1, x)$.

4 Experiment design

4.1 Definition of pathologies in evolved response surfaces

In what impacts generalisation, we define pathologies as: *output values that are infinite (Inf), undefined (NaN), or excessively large*. Examples include: 1/0 (represented in some programming languages as Inf, others as NaN), 10^{1000} (Inf), *log*(−1) (NaN), *sqrt*(−1) (NaN), etc. By excessively large, we define values which are not representable with 32bit architecture, i.e. values above 3.40282e+38, such as 100^{20}, exp(50), etc.

4.2 Function set setups

Table 3 specifies the composition of the function sets considered in this study. All function sets contain as subset the *basic* function set, composed of the operators of addition, subtraction and multiplication (function set 01). We define four classes of function sets as follows:

1. *Minimal Sets* are designed to investigate the effect of each added function on its own, combined only with the three basic arithmetic operators.
2. *Full Sets* are designed to investigate the use of all functions together, and the potential for evolution to select those that are adequate for specific problems.
3. *Pathological Sets* investigate the effect of combining specific functions (`pdiv`, `pinv`, `ppow` and `exp`) known to create discontinuities or "spikes" in the output of evolved functions [30, 50], as well as infinite values and undefined values [72].
4. *Balanced Sets* are different combinations of functions other than `pdiv`, `pinv`, `ppow`, `exp`, and `aq` functions. Here our aim is to investigate approximation and generalisation without the use of division primitives, or other primitives that may result in a pathology.
5. *Balanced Sets with* `aq` are defined in terms of combining functions other than `pdiv`, `pinv`, `ppow`, exp, and aim to test the synergy of `aq` with the rest of primitive functions.

4.3 Benchmark problems

To test the function sets defined, we used a large set of 43 benchmark problems from the literature; these are shown in Table 4. Synthetic problems (the first 35) are defined and sampled according to their source publications, apart from the actual number of samples generated (see below). Some of the real-world problems (Housing, Heating and Cooling) have corresponding versions with normalised predictors and response. Also note that some problems from the main sources of benchmarks were not used, when their underlying generating functions

Table 3 Function sets under comparison

ID	add	sub	mul	pdiv	pinv	aq	ppow	exp	plog	psqrt	sin	tanh
Minimal Sets												
01	✓	✓	✓	–	–	–	–	–	–	–	–	–
02	✓	✓	✓	✓	–	–	–	–	–	–	–	–
03	✓	✓	✓	–	✓	–	–	–	–	–	–	–
04	✓	✓	✓	–	–	✓	–	–	–	–	–	–
05	✓	✓	✓	–	–	–	✓	–	–	–	–	–
06	✓	✓	✓	–	–	–	–	✓	–	–	–	–
07	✓	✓	✓	–	–	–	–	–	✓	–	–	–
08	✓	✓	✓	–	–	–	–	–	–	✓	–	–
09	✓	✓	✓	–	–	–	–	–	–	–	✓	–
10	✓	✓	✓	–	–	–	–	–	–	–	–	✓
Full Sets												
11	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓
12	✓	✓	✓	✓	–	–	✓	–	✓	✓	✓	✓
13	✓	✓	✓	–	✓	–	–	✓	✓	✓	✓	✓
14	✓	✓	✓	–	–	✓	✓	✓	✓	✓	✓	✓
15	✓	✓	✓	–	–	✓	✓	–	✓	✓	✓	✓
16	✓	✓	✓	–	–	✓	–	✓	✓	✓	✓	✓
Pathological Sets												
17	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–
18	✓	✓	✓	✓	–	–	✓	–	–	–	–	–
19	✓	✓	✓	–	✓	–	–	✓	–	–	–	–
Balanced Sets												
20	✓	✓	✓	–	–	–	–	–	✓	✓	–	–
21	✓	✓	✓	–	–	–	–	–	✓	–	✓	–
22	✓	✓	✓	–	–	–	–	–	–	✓	✓	–

Table 3 (continued)

ID	add	sub	mul	pdiv	pinv	aq	ppow	exp	plog	psqrt	sin	tanh
23	✓	✓	✓	–	–	–	–	–	✓	✓	✓	–
24	✓	✓	✓	–	–	–	–	–	✓	–	–	✓
25	✓	✓	✓	–	–	–	–	–	–	✓	–	✓
26	✓	✓	✓	–	–	–	–	–	✓	✓	–	✓
27	✓	✓	✓	–	–	–	–	–	–	–	✓	✓
28	✓	✓	✓	–	–	–	–	–	✓	–	✓	✓
29	✓	✓	✓	–	–	–	–	–	–	✓	✓	✓
30	✓	✓	✓	–	–	–	–	–	✓	✓	✓	✓
Balanced Sets with AQ												
31	✓	✓	✓	–	–	✓	–	–	✓	–	–	–
32	✓	✓	✓	–	–	✓	–	–	–	✓	–	–
33	✓	✓	✓	–	–	✓	–	–	✓	✓	–	–
34	✓	✓	✓	–	–	✓	–	–	–	–	✓	–
35	✓	✓	✓	–	–	✓	–	–	✓	–	✓	–
36	✓	✓	✓	–	–	✓	–	–	–	✓	✓	–
37	✓	✓	✓	–	–	✓	–	–	✓	✓	✓	–
38	✓	✓	✓	–	–	✓	–	–	–	–	–	✓
39	✓	✓	✓	–	–	✓	–	–	✓	–	–	✓
40	✓	✓	✓	–	–	✓	–	–	–	✓	–	✓
41	✓	✓	✓	–	–	✓	–	–	✓	✓	–	✓
42	✓	✓	✓	–	–	✓	–	–	–	–	✓	✓
43	✓	✓	✓	–	–	✓	–	–	✓	–	✓	✓
44	✓	✓	✓	–	–	✓	–	–	–	✓	✓	✓
45	✓	✓	✓	–	–	✓	–	–	✓	✓	✓	✓

consist of a single function present in some of the compared function sets: Keijzer7 ($f(x) = \log x$), Keijzer8 ($f(x) = \sqrt{x}$), and Keijzer10 ($f(x_1, x_2) = x_1{}^{x_2}$).

The problems were purposefully sampled with small training (100) and large test (100,000) set sizes, in a manner similar to previous studies [50] on the effect of function sets in generalisation. The data was either grid-sampled or randomly sampled, according to the original publications, and using the same predictor ranges. For the Tower, Heating, Cooling and Housing problems, the first 100 samples of the available datasets [42] were used for training, and the remaining for testing.

4.4 Genetic programming systems

The Evolutionary Algorithm parameters for both systems are given in Table 5. For GP, an Ephemeral Random Constant in the range of $[-1, 1]$ is added to the terminal set. The probability of selecting an inner-node as a crossover point is set to 90%, while the probability of selecting a leaf-node is set to 10%. If mutation is chosen, then there is an equal probability that either subtree mutation or point mutation is applied. In subtree mutation, the tree-generation procedure is *grow* or *full* [35], each applied with equal probability. In point mutation, the probability of mutating a node is set to $2/\text{treesize}$. No reproduction operator is used.

For GE, a depthless variant of Probabilistic Tree-Creation 2 (PTC2) [52] is used for initialisation. The standard linear representation of GE is used [56], including linear versions of crossover and mutation. The GECodonValue technique [53] is used to generate 100 constants in the range of [0.0, 9.9]. The grammars were designed in a balanced way [54].

Finally, the fitness function used for both systems is defined as the Root Mean Squared Error (RMSE) computed on the training sample.

4.5 Performance metrics

Following the methodology in [22], we tested the robustness of the performance of a function set in terms of how well it performs on average across the 43 benchmark problems relative to the rest of the function sets. Firstly, we compute Mean RMSE for each function set on each benchmark problem by averaging the resulting best-of-generation RMSEs across 50 independent runs for generation numbers 50 and 100. Subsequently, for each function set and benchmark problem we compute *error ratios* by dividing the Mean RMSE of the function set by the smallest Mean RMSE obtained across all 45 function sets compared on that benchmark problem. That is, the error ratio of function set j is defined as $E_j = e_j/(\min_{1 \le k \le 45} e_k)$, where e_j is the mean-of-50-runs RMSE of a function set on a benchmark problem at generation 50 or 100. As a result, for each of the 43 problems, the best function set receives the error ratio of 1.0, while the other function sets receive a larger value. If a particular function set was best (smallest Mean RMSE) for all 43 benchmark problems, then its distribution of error ratios concentrates all the mass in the value of 1.0. We visualise the distribution of error ratios for different function sets by means of boxplots.

Table 4 Benchmark problems used

Benchmark problem	Input size	Source	Train set	Test set
Keijzer1	1	[30]	$E[-1, 1, 0.02]$	$E[-1, 1, 0.00002]$
Keijzer2	1	[30]	$E[-2, 2, 0.04]$	$E[-2, 2, 0.00004]$
Keijzer3	1	[30]	$E[-3, 3, 0.06]$	$E[-3, 3, 0.00006]$
Keijzer4	1	[30]	$E[0, 10, 0.1]$	$E[0.05, 10.05, 0.0001]$
Keijzer5	3	[30]	$U[-1, 1, 100]$	$U[-1, 1, 100000]$
			$U[1, 2, 100]$	$U[-1, 1, 100000]$
			$U[-1, 1, 100]$	$U[-1, 1, 100000]$
Keijzer6	1	[30]	$E[1, 100, 1]$	$E[1, 100000, 1]$
Keijzer9	1	[30]	$E[1, 100, 1]$	$E[1, 100, 0.00099]$
Keijzer11	2	[30]	$U[-3, 3, 100]$	$E[-3, 3, 0.01895]$
Keijzer12	2	[30]	$U[-3, 3, 100]$	$E[-3, 3, 0.01895]$
Keijzer13	2	[30]	$U[-3, 3, 100]$	$E[-3, 3, 0.01895]$
Keijzer14	2	[30]	$U[-3, 3, 100]$	$E[-3, 3, 0.01895]$
Keijzer15	2	[30]	$U[-3, 3, 100]$	$E[-3, 3, 0.01895]$
Vladislavleva1	2	[72]	$U[0.3, 4, 100]$	$E[-0.2, 4.2, 0.0139]$
Vladislavleva2	1	[72]	$E[0.05, 10, 0.1]$	$E[-0.5, 10.5, 0.00011]$
Vladislavleva3	2	[72]	$E[0.05, 10, 0.3]$	$E[-0.5, 10.5, 0.01101]$
			$E[0.05, 10.05, 5]$	$E[-0.5, 10.5, 0.1101]$
Vladislavleva4	5	[72]	$U[0.05, 6.05, 100]$	$U[-0.25, 6.35, 100000]$
Vladislavleva5	3	[72]	$U[0.05, 2, 100]$	$E[-0.05, 2.1, 0.04275]$
			$U[1, 2, 100]$	$E[0.95, 2.05, 0.0285]$
			$U[0.05, 2, 100]$	$E[-0.05, 2.1, 0.04275]$
Vladislavleva6	2	[72]	$U[0.1, 5.9, 100]$	$E[-0.05, 6.05, 0.0193]$
Vladislavleva7	2	[72]	$U[0.05, 6.05, 100]$	$U[-0.05, 6.35, 100000]$
Vladislavleva8	2	[72]	$U[0.05, 6.05, 100]$	$E[-0.05, 6.35, 0.02025]$
Korns1	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns2	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns3	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns4	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns5	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns6	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns7	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns8	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns9	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns10	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns11	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns12	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns13	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns14	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Korns15	5	[32]	$U[-50, 50, 100]$	$U[-50, 50, 100000]$
Pagie1	5	[57]	$E[-5, 5, 1.1]$	$E[-5, 5, 0.0316]$

Table 4 (continued)

Benchmark problem	Input size	Source	Train set	Test set
Tower	5	[72]	*S*[1:100]	*S*[101:4989]
Heating	8	[42]	*S*[1:100]	*S*[101:768]
HeatingNorm	8	[42]	*SN*[1:100]	*SN*[101:768]
Cooling	8	[42]	*S*[1:100]	*S*[101:768]
CoolingNorm	8	[42]	*SN*[1:100]	*SN*[101:768]
Housing	13	[42]	*S*[1:100]	*S*[101:506]
HousingNorm	13	[42]	*S*[1:100]	*SN*[101:506]

For sample selection, $E[a, b, c]$ is a grid of points evenly spaced with an interval of c, from a to b inclusive; $U[a, b, c]$ means c uniform random samples drawn from a to b inclusive; $S[a{:}b]$ means select from the ath to the bth observation (inclusive) from the original dataset; and $SN[a{:}b]$ means select from the ath to the bth observation (inclusive) from a normalised dataset. Some problems define each input variable differently

Table 5 Evolutionary Algorithm parameters

Parameter	GP	GE
Replacement	Elitist	
Elitism	1% of population size	
Population size	1000	
Tournament size	4	5
No. of generations	100	
Initialisation	Ramped half'n'half	PTC2
Init. depth	2–6	–
Init. size	–	3–15
Max. tree depth	20	–
Crossover probability	90%	50%
Mutation probability	10%	10%

Experiments showed that certain function sets produce very large best-of-generation testing RMSE, which impacted Mean RMSE computed over a number of independent runs. This degraded the readability of boxplot graphs of error ratios. For the sake of plotting clarity, we decided to cap all values of error ratios to the value of 5.

Comparing the distribution of error ratios for training and testing Mean RMSE between generations 50 and 100 intends to identify which of the function sets are more susceptible to overfitting that may result from a two-fold increase in search effort. In addition, it allows us to investigate the approximation/generalisation trade-off, with good-performing function sets exhibiting good relative performance in both training and testing cases on average.

We resorted to the use of Mean RMSE instead of Median RMSE in order to factor in pathological behaviour that may be exhibited from any of the 2150 models (50 runs × 43 problems) that were evolved using a particular function set.

5 Performance results

5.1 Approximation performance

We first benchmark approximation performance (RMSE on training data) derived from different function sets. The box-plots in Fig. 5 illustrate the distribution of error ratios, as detailed in Sect. 4.5, at both 50 and 100 generations.

Figure 5 suggests that Full Sets consistently achieve some of the better training performance, with regard to the distribution of error ratios, for both GP and GE systems, along with Balanced Sets with `aq`, particularly when the `sin` operator is included. This is also visible in the Minimal Set results: sets 04 (with `aq`) and 09 (with `sin`) are the best performing of that group.

Based on existing results on the universal approximation capability of Multilayer Perceptrons with arbitrary squashing functions, one could put forward a similar argument for the case of GP provided that the function set contains a Tauber–Wiener function such as `sin` or `cos` (or some other squashing functions) and a linear function, as discussed in [74]. The empirical results attained with the functions of sine and hyperbolic tangent are in support of this hypothesis.

The worst performing set (set 01) is the one composed solely of the three arithmetic operators. Although it should be noted that all sets in the Minimal Sets exhibit larger interquartile ranges, which indicates inconsistency in the results obtained for different problems. Finally, comparing sets 02 and 04 confirms that replacing Koza's protected division with `aq` improves the approximation capability of the evolved models.

The results are remarkably similar between GP and GE, independently of their different representation, search operators, and solution size, which indicates that the findings are strongly associated to the function sets used.

Looking at the differences in the distribution of error ratios between generations 50 and 100, note that results for the Minimal and Pathological sets are shifting away from the ideal distribution (centred at the value of 1). This suggests that function sets in those groups result in faster convergence as measured by training RMSE, while other sets allow for further fine-tuning of the generated models. This observation is less pronounced in the case of GE.

5.2 Generalisation performance

Figure 6 illustrates the distribution of error ratios for the testing data. The evolved models tested for generalisation were selected as best-of-generation models based on training RMSE. The figure clearly shows that there is a decidedly different performance between different sets. Sets 02, 03, 05 and 06, testing the `pdiv`, `pinv`,

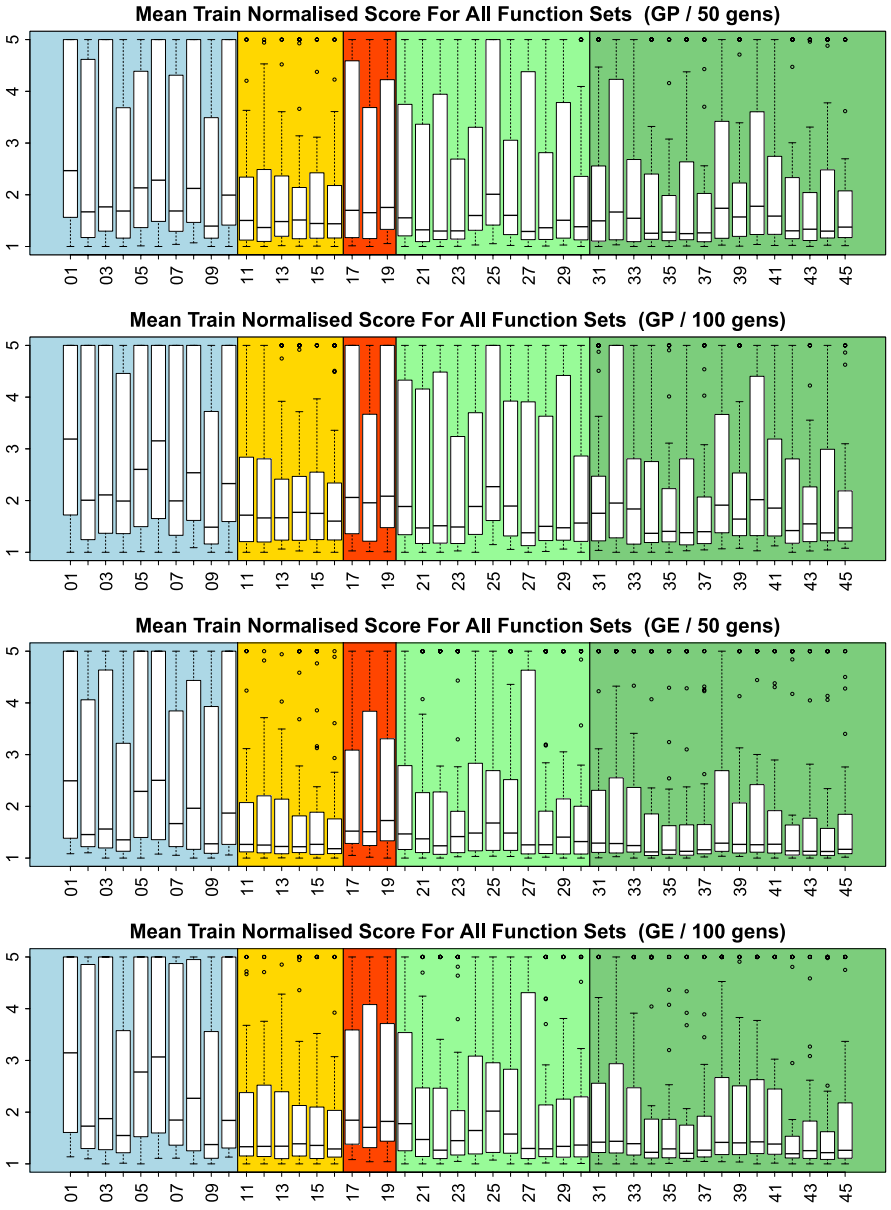


Fig. 5 GP and GE train RMSE error ratio distributions for all function sets, over 43 problems, at generation 50 and 100. For each function set/problem combination, the ratio is the mean train performance across all 50 runs at the reference generation, over that of the best function set for that problem. Ratios larger than 5 are capped at 5, for readability. Results obtained across 50 independent runs for each function set/problem combination

ppow and exp operators respectively, consistently generate mean RMSE performance which is substantially worse than all other sets. This is particularly the case with ppow. Pathological sets (sets 17, 18, and 19) display this behaviour even more markedly.

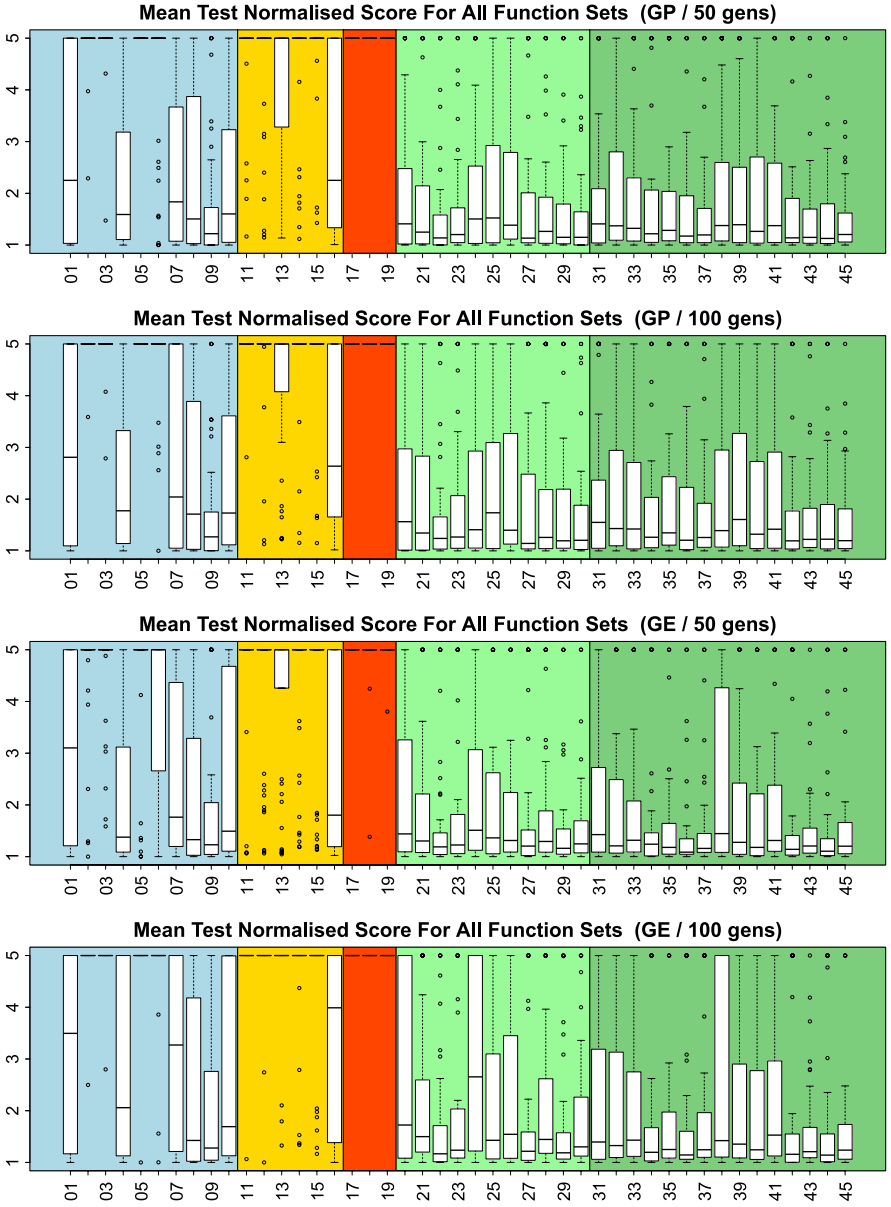


Fig. 6 GP and GE test RMSE error ratio distributions for all function sets, at generation 50 and 100. Test performance is measured with the best-in-training models, at the reference generations. Plots generated in the same manner as those of Fig. 5

Fig. 7 Mean train, validation, and test performance for five problems, using function sets 11, 17 and 34 ▶ (results averaged across 50 runs). Labels show the average generation at which best validation model was found. Fitness scale is capped at 1.0 for readability

Once again, the comparison between sets 02 and 04 shows that replacing `pdiv` with `aq` results in models with better test performance. Unlike `pdiv`, the `aq` operator did not produce any models generating extreme values when used with a large set of test data.

The generalisation performance of set 09 ($+, -, *$, `sin`) is quite remarkable, and provides further evidence of the advantage of combining a Tauber-Wiener function [74] and linear functions.

The results for the Full Sets (11-16) demonstrate that the good approximation performance yielded from the use of large function sets containing protected operators is traded-off for bad generalisation. The slight worsening of relative performance between generations 50 and 100 is evidence of overfitting.

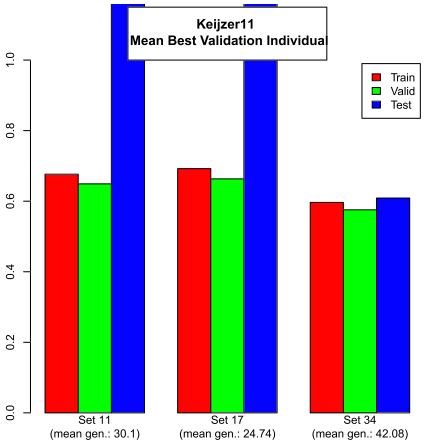
The Balanced sets clearly provide the best generalisation capability for the evolved models, with sets including the `aq` operator being marginally better. Within these, there is once again a clear improvement in generalisation performance, when the `sin` operator is included. Inclusion of the `tanh` operator also seems to slightly improve performance.

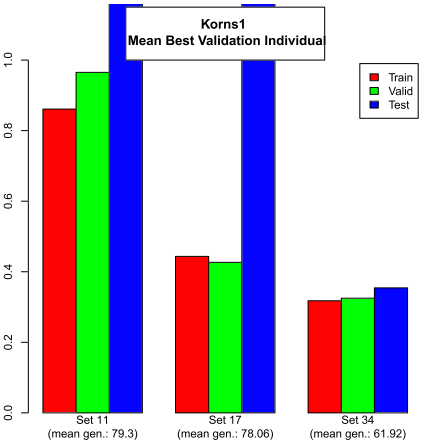
5.3 Effectiveness of validation-based model selection

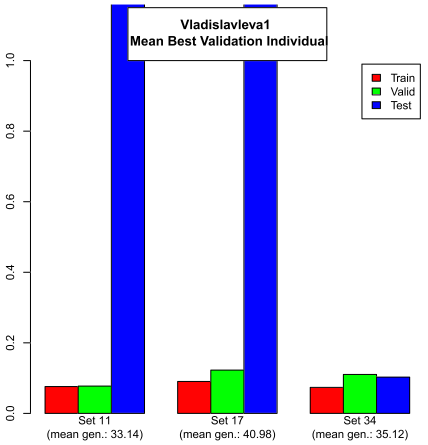
The experiments conducted deliberately used typical experimental setups found in literature, including the simplistic approach to overfitting by stopping evolution after 50 or 100 generations. Given that we are particularly concerned with generalisation performance, and that some of the protected operators can have a negative effect over it, it would make sense to use a better approach to model selection.

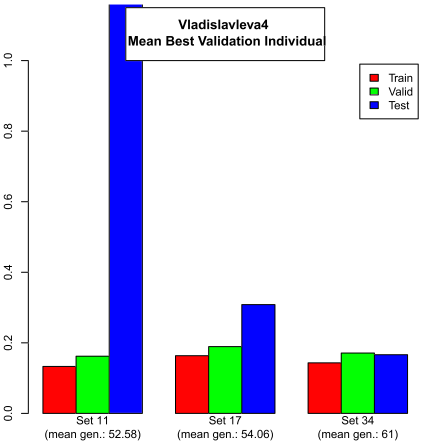
To that end, we re-ran our experiment, using the same setup as in Table 5, but transferring 100 data points from the test set to a validation set (all sets non-overlapping). At the end of each generation (100 generations), the best (training) model is recorded, along with its validation performance. At the end of each run (50 runs), the model with the best recorded validation performance is selected for testing.

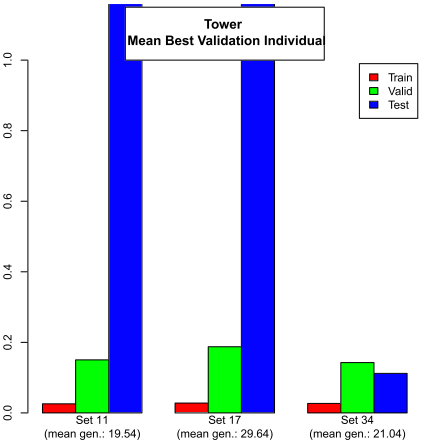
Figure 7 shows some of the results obtained, for three relevant function sets on five problems, using GE. The plots show, for each function set, the mean train, validation, and test performance, for the model with the best validation performance from each run, along with the mean generation at which it was found. The results show that model selection is not an effective solution to avoid the generalisation issues that protected operators create. Even when using best-of-run validation models, the occurrence of pathologies is evident (plots are capped at fitness value 1.0 for readability). This is not an issue with a set composed of aq and sine.


Keijzer11
Mean Best Validation Individual
Train
Valid
Test
0.0
0.2
0.4
0.6
0.8
1.0
Set 11
(mean gen.: 30.1)
Set 17
(mean gen.: 24.74)
Set 34
(mean gen.: 42.08)


Korns1
Mean Best Validation Individual
Train
Valid
Test
0.0
0.2
0.4
0.6
0.8
1.0
Set 11
(mean gen.: 79.3)
Set 17
(mean gen.: 78.06)
Set 34
(mean gen.: 61.92)


Vladislavleva1
Mean Best Validation Individual
Train
Valid
Test
0.0
0.2
0.4
0.6
0.8
1.0
Set 11
(mean gen.: 33.14)
Set 17
(mean gen.: 40.98)
Set 34
(mean gen.: 35.12)


Vladislavleva4
Mean Best Validation Individual
Train
Valid
Test
0.0
0.2
0.4
0.6
0.8
1.0
Set 11
(mean gen.: 52.58)
Set 17
(mean gen.: 54.06)
Set 34
(mean gen.: 61)


Tower
Mean Best Validation Individual
Train
Valid
Test
0.0
0.2
0.4
0.6
0.8
1.0
Set 11
(mean gen.: 19.54)
Set 17
(mean gen.: 29.64)
Set 34
(mean gen.: 21.04)

5.4 A closer look at the search landscape

Given that protected operators negatively affect generalisation performance, it seems logical to expect that they would also have a negative impact in training performance, by shaping a more rugged search landscape.

To test this, the same problems from Sect. 5.3 were used. Best-of-run individuals (at generation 100) were subjected to a random walk of 50 steps, where each random step consists of applying a single node mutation to their expressions. We did not modify the rest of the phenotype, so same-arity nodes were chosen as suitable replacements (e.g. `pdiv` could be replaced with `ppow`, `exp` with `plog`, `x1` with `x2`, etc). Figure 8 shows the results obtained.

These show that, in the case of protected operators, the neighbours of best-of-run individuals that are reachable through point mutation contain programs that exhibit pathologies in their outputs, both with training and test data. This is in contrast to the neighbourhoods of function set 34 (aq and sine), that are pathology free. The search landscape hence appears more rugged in the former case, and it is therefore more difficult to search.

6 Conclusion

The aim of this study was to investigate the effect of GP function set to the approximation versus generalisation trade-off inherent in a supervised learning setting using a finite set of training examples. Previous research results demonstrated that certain function primitives may result in evolved function output values that negatively impact generalisation, i.e., infinite, undefined, or excessively large output values on test data. Nonetheless, these results are largely ignored: a recent literature review on the most commonly used functions for symbolic regression revealed that, with notable exceptions, the majority of studies continue to use these function primitives.

In an attempt to reinforce awareness on the effect of certain function primitives to the generalisation capabilities of evolved models, as well as shed new light on their approximation counterpart, we conducted a large empirical study that compared training and test error performance that accrues from a diverse range of function set setups, and we further analysed the occurrence of pathologies in the output of evolved programs. Previous studies mainly focused on protected division, therefore the present work extends the results to protected versions of inversion, power, exponential, logarithm, and square root operators.

Results showed that the protected versions of division, inversion, power and exponential functions negatively impact the generalisation performance of the resulting models, and their use should therefore be discontinued. The negative effect of these protected operators is invariant of the use of moderately sized validation sets for selecting the best-of-run program.

Division is beneficial, nonetheless, using the analytic quotient implementation, which serves as an easy and effective plug-in replacement for protected division. Across the range of benchmark problems, the combination of function primitives for

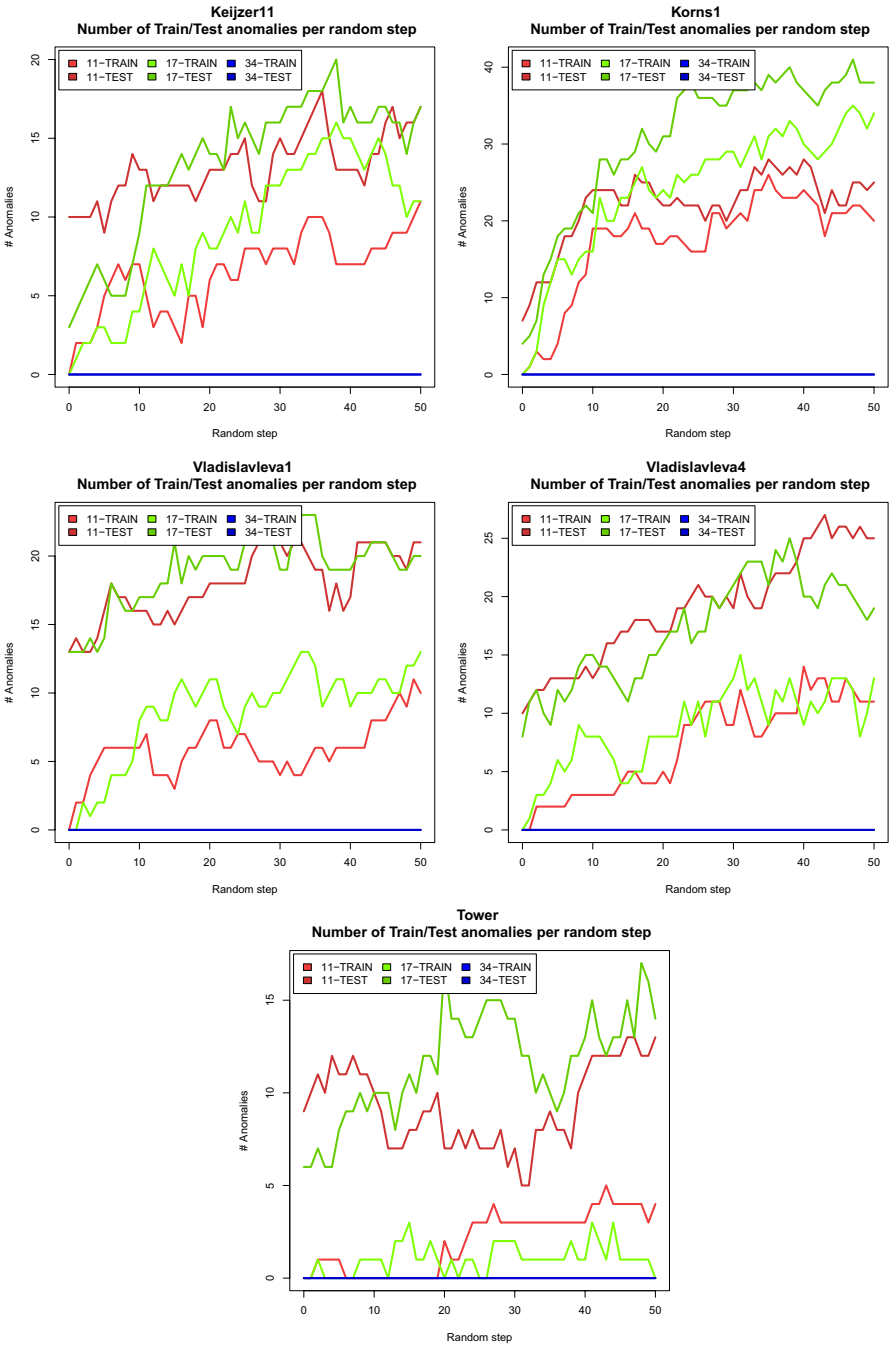


Fig. 8 Percentage of runs exhibiting pathologies, in train or test performance, when the best-of-run individual is subjected to a random walk of node mutation steps (results from 50 independent runs). Note that some runs (with function sets 11 and 17) already begin with test performance exhibiting pathologies

addition, subtraction, multiplication, analytic quotient and sine yielded one of the best generalisation/approximation trade-offs.

In addition, a summary about the support for different primitive functions and their protection mechanism in open source GP software packages was provided in order to facilitate interested readers in adopting the current results.

The study of the relationship between function set and model generalisation is part of the broader ongoing research of studying the interplay between model complexity, model-size constraints and function set makeup. In the introductory sections we motivated the generalisation impact of certain primitive functions through the lenses of regularisation as a means of model complexity control. We plan to investigate this issue as an immediate next step. Furthermore, we plan to study the effect of conditional `if-then-else` constructs for evolving non-continuous functions.

References

1. A. Agapitos, R. Loughran, M. Nicolau, S. Lucas, M. O'Neill, A. Brabazon, A survey of statistical machine learning elements in genetic programming. IEEE Trans. Evol. Comput. **23**(6), 1029–1048 (2019)
2. R.M.A. Azad, C. Ryan, Variance based selection to improve test set performance in genetic programming, in *GECCO'11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (ACM, Dublin, 2011), pp. 1315–1322
3. P. Barmpalexis, A. Karagianni, G. Karasavvaides, K. Kachrimanis, Comparison of multi-linear regression, particle swarm optimization artificial neural networks and genetic programming in the development of mini-tablets. Int. J. Pharm. **551**(1), 166–176 (2018). https://doi.org/10.1016/j.ijpharm.2018.09.026
4. M. Castelli, L. Manzoni, S. Silva, L. Vanneschi, A comparison of the generalization ability of different genetic programming frameworks, in *IEEE Congress on Evolutionary Computation (CEC 2010)* (IEEE Press, Barcelona, 2010)
5. Q. Chen, B. Xue, M. Zhang, Improving generalisation of genetic programming for symbolic regression with angle-driven geometric semantic operators. IEEE Trans. Evol. Comput. **23**(3), 488–502 (2019). https://doi.org/10.1109/TEVC.2018.2869621
6. Q. Chen, M. Zhang, B. Xue, Feature selection to improve generalisation of genetic programming for high-dimensional symbolic regression. IEEE Trans. Evol. Comput. **21**(5), 792–806 (2017). https://doi.org/10.1109/TEVC.2017.2683489
7. Q. Chen, M. Zhang, B. Xue, Structural risk minimisation-driven genetic programming for enhancing generalisation in symbolic regression. IEEE Trans. Evol. Comput. **23**(4), 703–717 (2019). https://doi.org/10.1109/TEVC.2018.2881392
8. O. Claveria, E. Monte, S. Torra, Assessment of the effect of the financial crisis on agents expectations through symbolic regression. Appl. Econ. Lett. **24**(9), 648–652 (2017). https://doi.org/10.1080/13504851.2016.1218419
9. L.F. dal Piccol Sotto, V.V. de Melo, Studying bloat control and maintenance of effective code in linear genetic programming for symbolic regression. Neurocomputing **180**, 79–93 (2016). https://doi.org/10.1016/j.neucom.2015.10.109. Progress in Intelligent Systems Design Selected papers from the 4th Brazilian Conference on Intelligent Systems (BRACIS 2014)
10. O. Claveria, E. Monte, S. Torra, Using survey data to forecast real activity with evolutionary algorithms: a cross-country analysis. J. Appl. Econ. **20**(2), 329–349 (2017). https://doi.org/10.1016/S1514-0326(17)30015-6
11. G. D'Angelo, R. Pilla, C. Tascini, S. Rampone, A proposal for distinguishing between bacterial and viral meningitis using genetic programming and decision trees. Soft Comput. **23**(22), 11775–11791 (2019). https://doi.org/10.1007/s00500-018-03729-y

12. I. De Falco, A. Della Cioppa, T. Koutny, M. Krcma, U. Scafuri, E. Tarantino, Genetic programming-based induction of a glucose-dynamics model for telemedicine. J. Netw. Comput. Appl. **119**, 1–13 (2018). https://doi.org/10.1016/j.jnca.2018.06.007
13. F.O. de Franca, A greedy search tree heuristic for symbolic regression. Inf. Sci. **442**, 18–32 (2018). https://doi.org/10.1016/j.ins.2018.02.040
14. V.V. de Melo, W. Banzhaf, Improving the prediction of material properties of concrete using kaizen programming with simulated annealing. Neurocomputing **246**, 25–44 (2017). https://doi.org/10.1016/j.neucom.2016.12.077
15. V.V. de Melo, W. Banzhaf, Automatic feature engineering for regression models with machine learning: an evolutionary computation and statistics hybrid. Inf. Sci. **430–431**, 287–313 (2018). https://doi.org/10.1016/j.ins.2017.11.041
16. G. Dick, Revisiting interval arithmetic for regression problems in genetic programming, in *Genetic and Evolutionary Computation Conference—GECCO 2017, Berlin, Germany, July 15–19, 2017, Companion, Proceedings*, ed. by G. Ochoa (ACM 2017), pp. 129–130. https://doi.org/10.1145/3067695.3076107
17. A.I. Diveev, N.B. Konyrbaev, E.A. Sofronova, Method of binary analytic programming to look for optimal mathematical expression. Procedia Comput. Sci. **103**, 597–604 (2017). XII International Symposium Intelligent Systems, INTELS 2016, 5–7 October 2016. Moscow, Russia (2016). https://doi.org/10.1016/j.procs.2017.01.073
18. T. Dou, P. Rockett, Comparison of semantic-based local search methods for multiobjective genetic programming. Genet. Program Evol. Mach. **19**(4), 535–563 (2018). https://doi.org/10.1007/s10710-018-9325-4
19. I. Fajfar, T. Tuma, Creation of numerical constants in robust gene expression programming. Entropy **20**(10), 756 (2018). https://doi.org/10.3390/e20100756
20. M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, M. O'Neill, E. Hemberg, *PonyGE2: Grammatical Evolution in Python*. http://ncra.ucd.ie/Site/GEVA.html (2019)
21. F.A. Fortin, F.M. De Rainville, M.A. Gardner, M. Parizeau, C. Gagné, *DEAP*. https://github.com/DEAP/deap (2019)
22. J.H. Friedman, Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**, 1189–1232 (2000)
23. A. Garg, J.S.L. Lam, B.N. Panda, A hybrid computational intelligence framework in modelling of coal-oil agglomeration phenomenon. Appl. Soft Comput. **55**, 402–412 (2017). https://doi.org/10.1016/j.asoc.2017.01.054
24. B. Ghaddar, N. Sakr, Y. Asiedu, Spare parts stocking analysis using genetic programming. Eur. J. Oper. Res. **252**(1), 136–144 (2016). https://doi.org/10.1016/j.ejor.2015.12.041
25. E.M. Golafshani, A. Behnood, Automatic regression methods for formulation of elastic modulus of recycled aggregate concrete. Appl. Soft Comput. **64**, 377–400 (2018). https://doi.org/10.1016/j.asoc.2017.12.030
26. I. Gonzalez-Taboada, B. Gonzalez-Fonteboa, F. Martinez-Abella, J.L. Perez-Ordonez, Prediction of the mechanical properties of structural recycled concrete using multivariable regression and genetic programming. Constr. Build. Mater. **106**, 480–499 (2016). https://doi.org/10.1016/j.conbuildmat.2015.12.136
27. M.A. Haeri, M.M. Ebadzadeh, G. Folino, Statistical genetic programming for symbolic regression. Appl. Soft Comput. **60**, 447–469 (2017). https://doi.org/10.1016/j.asoc.2017.06.050
28. S.A. Hosseini, A. Tavana, S.M. Abdolahi, S. Darvishmaslak, Prediction of blast induced ground vibrations in quarry sites: a comparison of GP, RSM and MARS. Soil Dyn. Earthq. Eng. **119**, 118–129 (2019). https://doi.org/10.1016/j.soildyn.2019.01.011
29. A. Kattan, A. Agapitos, Y.S. Ong, A.A. Alghamedi, M. O'Neill, GP made faster with semantic surrogate modelling. Inf. Sci. **355–356**, 169–185 (2016). https://doi.org/10.1016/j.ins.2016.03.030
30. M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in *Genetic Programming. Proceedings of EuroGP'2003, LNCS*, vol. 2610, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Springer, Essex, 2003), pp. 70–82
31. M. Khandelwal, R.S. Faradonbeh, M. Monjezi, D.J. Armaghani, M.Z.B.A. Majid, S. Yagiz, Function development for appraising brittleness of intact rocks using genetic programming and non-linear multiple regression models. Eng. Comput. **33**(1), 13–21 (2017). https://doi.org/10.1007/s00366-016-0452-3

32. M.F. Korns, Accuracy in symbolic regression, in *Genetic Programming Theory and Practice IX, Genetic and Evolutionary Computation*, ed. by R. Riolo, E. Vladislavleva, J.H. Moore (Springer, New York, 2011), pp. 129–151
33. M. Kovacic, A. Mihevc, M. Tercelj, *Roll wear modeling using genetic programming—industry case study* (Mater, Technol, 2019). 10.17222/mit.2018.104
34. M. Kovacic, A. Turnsek, D. Ocvirk, G. Gantar, Increasing the tensile strength and elongation of 16mncrs5 steel using genetic programming. Mater. Technol. **51**(6), 883–888 (2017). 10.17222/mit.2016.293
35. J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
36. G. Kronberger, M. Kommenda, E. Lughofer, S. Saminger-Platz, A. Promberger, F. Nickel, S. Winkler, M. Affenzeller, Using robust generalized fuzzy modeling and enhanced symbolic regression to model tribological systems. Appl. Soft Comput. **69**, 610–624 (2018). https://doi.org/10.1016/j.asoc.2018.04.048
37. J. Kubalik, E. Alibekov, R. Babuska, Optimal control via reinforcement learning with symbolic policy approximation. IFAC-PapersOnLine **50**(1), 4162–4167 (2017). https://doi.org/10.1016/j.ifacol.2017.08.805. 20th IFAC World Congress
38. J. Kubalik, E. Alibekov, J. Zegklitz, R. Babuska, Hybrid single node genetic programming for symbolic regression. Trans. Comput. Collect. Intell. **9770**, 61–82 (2016). https://doi.org/10.1007/978-3-662-53525-7_4
39. H.C. Kwak, S. Kho, Predicting crash risk and identifying crash precursors on korean expressways using loop detector data. Accid. Anal. Prev. **88**, 9–19 (2016). https://doi.org/10.1016/j.aap.2015.12.004
40. W.B. Langdon, *The Genetic Programming Bibliography*. http://www.gpbib.cs.ucl.ac.uk/ (2020)
41. W.B. Langdon, R. Poli, *Foundations of Genetic Programming* (Springer, New York, 2002)
42. M. Lichman, *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml (2013)
43. H. Liu, H. Lin, X. Jiang, X. Mao, Q. Liu, B. Li, Estimation of mass matrix in machine tool's weak components research by using symbolic regression. Comput. Ind. Eng. **127**, 998–1011 (2019). https://doi.org/10.1016/j.cie.2018.11.033
44. Q. Lu, J. Ren, Z. Wang, Using genetic programming with prior formula knowledge to solve symbolic regression problem. Comput. Intell. Neurosci. (2016). https://doi.org/10.1155/2016/1021378
45. S. Luke, E.O. Scott, L. Panait, G. Balan, S. Paus, Z. Skolicki, R. Kicinger, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, A. Desai, A. Chircop, J. Compton, W. Haddon, S. Donnelly, B. Jamil, J. Zelibor, E. Kangas, F. Abidi, H. Mooers, J. O'Beirne, L. Manzoni, K.A. Talukder, S. McKay, J. McDermott, J. Zou, A. Rutherford, D. Freelan, E. Wei, S. Rajendran, A. Dhawan, B. Brumbac, J. Hilty, A. Kabir, ECJ 27: A Java-based Evolutionary Computation Research System. https://cs.gmu.edu/~eclab/projects/ecj (2019)
46. Mahler, S., Robilliard, D., Fonlupt, C.: Tarpeian bloat control and generalization accuracy, in *Proceedings of the 8th European Conference on Genetic Programming, Lecture Notes in Computer Science*, vol. 3447. ed. by M. Keijzer, A. Tettamanzi, P. Collet, J.I. van Hemert, M. Tomassini, (Springer, Lausanne, 2005), pp. 203–214
47. L.F. Miranda, L.O.V.B. Oliveira, J.F.B.S. Martins, G.L. Pappa, How noisy data affects geometric semantic genetic programming, in *Genetic and Evolutionary Computation Conference—GECCO 2017, Berlin, Germany, July 15–19, 2017, Companion, Proceedings*, ed. by G. Ochoa (ACM, 2017), pp. 985–992. https://doi.org/10.1145/3071178.3071300
48. J.L. Montana, C.L. Alonso, C.E. Borges, C. Tirnauca, Model-driven regularization approach to straight line program genetic programming. Expert Syst. Appl. **57**, 76–90 (2016). https://doi.org/10.1016/j.eswa.2016.03.003
49. S.S. Mousavi Astarabadi, M.M. Ebadzadeh, A decomposition method for symbolic regression problems. Appl. Soft Comput. **62**, 514–523 (2018). https://doi.org/10.1016/j.asoc.2017.10.041
50. J. Ni, R.H. Drieberg, P.I. Rockett, The use of an analytic quotient operator in genetic programming. IEEE Trans. Evol. Comput. **17**(1), 146–152 (2013)
51. J. Ni, P. Rockett, Tikhonov regularization as a complexity measure in multiobjective genetic programming. IEEE Trans. Evol. Comput. **19**(2), 157–166 (2015)
52. M. Nicolau, Understanding grammatical evolution: initialisation. Genet. Program Evolv Mach. **18**(4), 1–41 (2017). https://doi.org/10.1007/s10710-017-9309-9
53. M. Nicolau, I. Dempsey, Introducing grammar based extensions for grammatical evolution, in *IEEE Congress on Evolutionary Computation (CEC 2006)*, pp. 2663–2670

54. M. Nicolau, M. O'Neill, A. Brabazon, Termination in grammatical evolution: Grammar design, wrapping, and tails, in *IEEE Congress on Evolutionary Computation (CEC 2012)* (2012), pp. 1–8
55. N.Y. Nikolaev, H. Iba, Regularization approach to inductive genetic programming. IEEE Trans. Evol. Comput. **54**(4), 359–375 (2001)
56. M. O'Neill, C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Genetic programming*, vol. 4 (Kluwer, Alphen aan den Rijn, 2003)
57. L. Pagie, P. Hogeweg, Evolutionary consequences of coevolving targets. Evol. Comput. **5**(4), 401–418 (1997)
58. X. Pan, M.K. Uddin, B. Ai, X. Pan, Influential factors of carbon emissions intensity in oecd countries: evidence from symbolic regression. J. Clean. Prod. (2019). https://doi.org/10.1016/j.jclepro.2019.02.195
59. T.P. Pawlak, K. Krawiec, Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. Evol. Comput. **26**(2), 177–212 (2018). https://doi.org/10.1162/EVCO_a_00205
60. S. Polanco-Martagon, J. Ruiz-Ascencio, M.A. Duarte-Villasenor, Symbolic modeling of the Pareto-optimal sets of two unity gain cells. DYNA **83**(197), 128–137 (2016). 10.15446/dyna.v83n197.50919
61. M. Quade, M. Abel, K. Shafi, R.K. Niven, B.R. Noack, Prediction of dynamical systems by symbolic regression. Phys. Rev. E **94**, 012214 (2016). https://doi.org/10.1103/PhysRevE.94.012214
62. S.S. Rathore, S. Kumar, Towards an ensemble based system for predicting the number of software faults. Expert Syst. Appl. **82**, 357–382 (2017). https://doi.org/10.1016/j.eswa.2017.04.014
63. S. Silva, *GPLAB: A Genetic Programming Toolbox for MATLAB*. http://gplab.sourceforge.net/download.html(2019)
64. S. Silva, L. Vanneschi, Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction, in *GECCO'09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (ACM, Montreal, 2009), pp. 1115–1122
65. A. Sohani, M. Zabihigivi, M.H. Moradi, H. Sayyaadi, H.H. Balyani, A comprehensive performance investigation of cellulose evaporative cooling pad systems using predictive approaches. Appl. Therm. Eng. **110**, 1589–1608 (2017). https://doi.org/10.1016/j.applthermaleng.2016.08.216
66. R. Taghizadeh-Mehrjardi, K. Nabiollahi, R. Kerry, Digital mapping of soil organic carbon at multiple depths using different data mining techniques in Baneh Region, Iran. Geoderma **266**, 98–110 (2016). https://doi.org/10.1016/j.geoderma.2015.12.003
67. A. Tahmassebi, A.H. Gandomi, Building energy consumption forecast using multi-objective genetic programming. Measurement **118**, 164–171 (2018). https://doi.org/10.1016/j.measurement.2018.01.032
68. Y. Tao, Y.J. Chen, X. Fu, B. Jiang, Y. Zhang, Evolutionary ensemble learning algorithm to modeling of warfarin dose prediction for chinese. IEEE J. Biomed. Health Inform. (2018). https://doi.org/10.1109/JBHI.2018.2812165
69. P.T. Thuong, N.X. Hoai, X. Yao, Combining conformal prediction and genetic programming for symbolic interval regression, in *Genetic and Evolutionary Computation Conference—GECCO 2017, Berlin, Germany, July 15–19, 2017, Companion, Proceedings*, ed. by G. Ochoa, (ACM, 2017), pp. 1001–1008. https://doi.org/10.1145/3071178.3071280
70. L. Vanneschi, M. Castelli, S. Silva, Measuring bloat, overfitting and functional complexity in genetic programming, in *GECCO'10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (ACM, Portland, USA, 2010), pp. 877–884
71. V. Vladimir, *The Nature of Statistical Learning Theory* (Springer, New York, 1999)
72. E.J. Vladislavleva, G.F. Smits, D. den Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009)
73. P. Whigham, G. Dick, J. Maclaurin, C.A. Owen, *libgges: Grammar-Guided Evolutionary Search*. https://github.com/DEAP/deap (2019)
74. X. Yao, Universal approximation by genetic programming, in *Foundations of Genetic Programming*, ed. by T. Haynes, W.B. Langdon, U.M. O'Reilly, R. Poli, J. Rosca (Orlando, Florida, USA, 1999), pp. 66–67
75. Y.S. Yeun, W.S. Ruy, Y.S. Yang, N.J. Kim, Implementing linear models in genetic programming. IEEE Trans. Evol. Comput. **8**(6), 542–566 (2004)

76. E. Flores, M. Abatal, A. Bassam, L. Trujillo, P. Juarez-Smith, Y. El Hamzaoui, Modeling the adsorption of phenols and nitrophenols by activated carbon using genetic programming. J. Clean. Prod. **161**, 860–870 (2017). https://doi.org/10.1016/j.jclepro.2017.05.192
77. A. Zameer, J. Arshad, A. Khan, M.A.Z. Raja, Intelligent and robust prediction of short term wind power using genetic programming based ensemble of neural networks. Energy Convers. Manag. **134**, 361–372 (2017). https://doi.org/10.1016/j.enconman.2016.12.032
78. J. Zhong, W. Cai, M. Lees, L. Luo, Automatic model construction for the behavior of human crowds. Appl. Soft Comput. **56**, 368–378 (2017). https://doi.org/10.1016/j.asoc.2017.03.020

Affiliations

Miguel Nicolau[1] · **Alexandros Agapitos**[2]

Alexandros Agapitos
alexagapitos@gmail.com

[1] College of Business, University College Dublin, Dublin, Ireland

[2] Ireland Research Center, Huawei Technologies Ltd., Dublin, Ireland