Geometric Semantic Genetic Programming with Normalized and Standardized Random Programs

Illya Bakurov[1,4], José Manuel Muñoz Contreras[2], Mauro Castelli[1], Nuno Rodrigues[3], Sara Silva[3], Leonardo Trujillo[2,3*], Leonardo Vanneschi[1]

[1] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal

[2] Departamento de Ingeniería Eléctrica y Electrónica, Tecnológico Nacional de México/IT de Tijuana, 22500, Tijuana BC, México

[3] LASIGE, Department of Informatics, Faculty of Sciences, University of Lisbon, Portugal

[4] Department of Computer Science & Engineering and Beacon Center for the Study of Evolution in Action, Michigan State University, 48824 Michigan, USA

[*]Corresponding author

This is the Author Peer Reviewed version of the following article published by Springer:

Bakurov, I., Muñoz Contreras, J. M., Castelli, M., Rodrigues, N., Silva, S., Trujillo, L., & Vanneschi, L. (2024). Geometric Semantic Genetic Programming with Normalized and Standardized Random Programs. Genetic Programming And Evolvable Machines, 25, 1-29. Article 6. Advance online publication. https://doi.org/10.1007/s10710-024-09479-1

Geometric Semantic Genetic Programming with Normalized and Standardized Random Programs

Illya Bakurov[1,4†], José Manuel Muñoz Contreras[2†], Mauro Castelli[1], Nuno Rodrigues[3], Sara Silva[3], Leonardo Trujillo[2,3*], Leonardo Vanneschi[1]

[1]NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal.
[2]Departamento de Ingeniería Eléctrica y Electrónica, Tecnológico Nacional de México/IT de Tijuana, 22500, Tijuana BC, México.
[3]LASIGE, Department of Informatics, Faculty of Sciences, University of Lisbon, Portugal.
[4]Department of Computer Science & Engineering and Beacon Center for the Study of Evolution in Action, Michigan State University, 48824 Michigan, USA.

*Corresponding author(s). E-mail(s): leonardo.trujillo@tectijuana.edu.mx;
Contributing authors: bakurov1@msu.edu, ibakurov@novaims.unl.pt; mane9986@gmail.com; mcastelli@novaims.unl.pt; nmrodrigues@ciencias.ulisboa.pt; sgsilva@ciencias.ulisboa.pt; lvanneschi@novaims.unl.pt;
†These authors contributed equally to this work.

Abstract

Geometric Semantic Genetic Programming (GSGP) represents one of the most promising developments in the area of Evolutionary Computation (EC) in the last decade. The results achieved by incorporating semantic awareness in the evolutionary process demonstrate the impact that geometric semantic operators have brought to the field of EC. An improvement to the geometric semantic mutation (GSM) operator is proposed, inspired by the results achieved by batch normalization in deep learning. While, in one of its most used versions, GSM relies on the use of the sigmoid function to constrain the semantics of two random programs responsible for perturbing the parent's semantics, here a different approach

is followed, which allows reducing the size of the resulting programs and overcoming the issues associated with the use of the sigmoid function, as commonly done in deep learning. The idea is to consider a single random program and use it to perturb the parent's semantics only after standardization or normalization. The experimental results demonstrate the suitability of the proposed approach: despite its simplicity, the presented GSM variants outperform standard GSGP on the studied benchmarks, with a difference in terms of performance that is statistically significant. Furthermore, the individuals generated by the new GSM variants are easier to simplify, allowing us to create accurate but significantly smaller solutions.

Keywords: Geometric Semantic Mutation, Internal Covariate Shift, Sigmoid Distribution Bias, Model Simplification.

1 Introduction

Genetic Programming (GP) is an evolutionary method introduced by Koza [23], based on the principles of Darwinian evolution. Since its inception, GP attracted the interest of researchers due to its capability to produce human-readable models and outperform human-designed solutions [24]. Despite the achievements of GP in several domains, researchers identified a limitation concerning the definition of the genetic operators (crossover and mutation) used to produce new individuals starting from a population of parent individuals. The standard genetic operators introduced by Koza rely on syntactic transformations of the parents to obtain new solutions, without considering how these syntactic transformations affect the output of the offspring. In other words, standard genetic operators execute transformations of the structure of the parent solutions without any known effect on the behaviour of the offspring. In particular, standard crossover creates two children by swapping two subprograms identified in the parents, while mutation, in its common implementation, replaces a subprogram with a randomly generated program. In fact, a minor modification in the parent structure might result in a large perturbation of its behaviour, or vice-versa. To overcome this limitation, GP researchers started to investigate the possibility of incorporating semantic awareness in the search process [38]. In the context of GP, the term semantics refers to the output vector of a given GP program when evaluated over a set of fitness cases [38]. Initial contributions attempting to exploit semantic awareness were based on the definition of indirect semantics-based methods [7, 28, 37]. In other words, standard syntax-based genetic operators were used, and, subsequently, semantic criteria were employed to determine whether to accept or reject the newly created solutions. Typical examples include the rejection of a solution if its fitness is worse than the fitness of the parent (for mutation) or worse than the fitness of both parents (for crossover) [6, 8, 28]. While these methods produced good results, they presented a limitation due to the high number of rejected individuals and the computational effort necessary to evaluate the semantic criteria. A milestone in this line of research was presented by Moraglio et al. [29] in 2012, when they defined genetic operators that allow direct semantic awareness in the search process, thus avoiding

the unnecessary creation and evaluation of individuals that may be rejected upon the assessment of the semantic criteria. Moraglio et al. proposed a novel variant of GP called Geometric Semantic Genetic Programming (GSGP), where the standard syntax-based genetic operators are replaced with the so-called Geometric Semantic Operators (GSOs). In particular, GSOs perform syntactic transformations on the parent solutions that have a known effect on the semantics of the offspring. Moreover, GSOs induce an unimodal fitness landscape for any supervised machine learning problem. These properties rapidly caught the interest of many researchers in the field and motivated several notable works [2–5, 9, 10, 14, 16, 26, 35, 39, 40]. Some of these works consisted on improving the novel genetic operators [1, 14, 27, 32]. For example, McDermot et al. proposed a variation of the Geometric Semantic Mutation (GSM) operator that reduces the amount of genetic material that is added at each mutation event, helping to keep trees small while retaining the geometric properties. Several works proposed to improve GSM by including an adaptive mechanism to optimally adjust the degree of the semantic perturbation, such that the error on the training data is minimized [14, 27]. Vanneschi et al. demonstrated that using a sigmoid function for bounding the semantics of the random programs used by the GSOs, applying it as an additional operator on the output of the programs, allowed a significant improvement in the generalization ability of the algorithm, evaluated on several real-world problems [9, 14, 39]. Using sigmoid as a bounding mechanism is motivated by the fact that random programs tend to produce arbitrary large semantics, thus making it difficult to incorporate these semantics into the parent programs through GSOs; bounding the random programs' semantics in $[0, 1]$ translated into a more controlled learning.

Nevertheless, the sigmoid function has a short linear regime and gets rapidly saturated for both large and small values of the corresponding input. Niolau & McDermott [31] empirically studied the distribution of the offspring in the semantic space resulting from unbounded GSM and empirically showed that: (i) the distributions are long-tailed in each dimension, thus can easily fall on the saturated regime of the sigmoid function; (ii) there is no radius within which all the outputs are contained for each dimension of the semantic space; and (iii) the variance in each dimension can differ due to disparate distributions of the training cases. Therefore, blindly using the sigmoid function to bound the output of random trees might result in poor semantics (i.e., without the necessary amount of entropy), which might decrease the capabilities of GSOs, and GSM in particular. At the same time, the evidence resulting from this study highlights the need to apply some form of transformation to mitigate the effects of unbounded GSM.

A similar problem associated with the use of the sigmoid function emerged in work done on deep artificial neural networks (DANNs) training [18]. Several techniques were presented to overcome this issue, like the use of rectified linear units (ReLUs) [30] and the so-called batch normalisation (BN) transformation [22], the latter being directly related to the work presented in this paper. In simple terms, BN consists of a normalization step that fixes the mean and variance of the distribution of the outputs of intermediate layers (activations or features) throughout the network.

To some extent, both an individual (program) generated through the generational application of GSM and a DANN can be seen as systems made of sequentially stacked

subsystems, such that each successive subsystem receives as input the output of the precedent one; this is illustrated in Figure 1 (Section 3.1 provides a detailed explanation). The present work is not the first to identify the parallels between GSM and DANNs. Gonçalves et al. proposed a novel neuroevolution algorithm that capitalizes upon the mechanics of GSM [15, 17] where artificial neural networks (ANNs) are used instead of syntax trees (a more common representation of GP programs). Motivated by the aforementioned similarity and inspired by the functioning of the BN transformation and its encouraging results, in this work the goal was to adapt this technique and incorporate it into GSGP.

The remainder of the paper is organized as follows. Section 2 recalls the definition of GSOs for regression problems and concepts from the field of DANNs that will be used throughout the paper. Section 3 presents the modification used in this study concerning the use of the sigmoid function in GSOs. Section 4 discusses the benchmarks considered in the experimental phase and the experimental settings. Section 5 discusses the achieved experimental results. Finally, Section 6 concludes the paper by summarizing the main findings and suggesting future research directions.

2 Background

2.1 Geometric Semantic Genetic Programming

Consider a symbolic regression problem with a set of input data $\mathbf{X}$ represented by $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ (training instances, observations or fitness cases). Let the vector $\mathbf{t} = [t_1, t_2, ..., t_n]$ represent their respective expected outputs (target values). In simpler terms, for each $i = 1, 2, ..., n$ the expected output for the input $\mathbf{x}_i$ is t_i. It is possible to view a GP individual (or program) P as a function that takes as input vector $\mathbf{x}_i$ and returns a scalar value $P(\mathbf{x}_i)$. According to [29], the vector $s_P = [P(\mathbf{x}_1), P(\mathbf{x}_2), ..., P(\mathbf{x}_n)]$ can be referred to as the *semantics* of P. This vector can be represented by a point in an n-dimensional space, which is called the *semantic space*. It is worth noting that the target vector $\mathbf{t}$ also represents a point in the semantic space.

As previously stated, GSGP is a modified version of GP that uses GSOs in place of the traditional crossover and mutation. The main goal of GSOs is to introduce modifications to the syntax of GP individuals that have a controlled and predictable impact on their semantics. As defined in [29], the GSOs[1] are:

Geometric Semantic Crossover (GSC)

Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, GSC returns the function $GSC(T) = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random function whose codomain values range in the interval $[0, 1]$.

Given two programs, representing the selected parent individuals, GSC's application produces another program (the offspring) whose semantics lie on the segment joining the two parents in the semantic space. The proof can be found in [29], but suffice

[1] Only the definitions of GSOs for symbolic regression problems are presented in this paper as they are the only ones used. For the definitions of GSOs in other domains, the reader is referred to [29].

is to say that the reasoning stems from the fact that the semantics of the offspring program is constructed as a linear combination of the semantics of the parents. The codomain of T_R is guaranteed to be in $[0, 1]$ using the aforementioned sigmoid function. In this sense, GSC is limited to finding a globally optimal solution only if this solution is inside the hypercube delimited in the semantic space by the individuals in the population [4, 11, 29].

Geometric Semantic Mutation (GSM)

Given a parent function $T : \mathbb{R}^n \to \mathbb{R}$, GSM with mutation step ms returns the function $GSM(T) = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random functions.

The motivation for considering the difference between two random functions, T_{R1} and T_{R2}, is that in GSM it is essential to introduce a perturbation centered at zero. In fact, if T_{R1} and T_{R2} are random functions, $T_{R1} - T_{R2}$ is a random expression with an equal probability of being positive or negative. While this was not originally defined in GSM, later studies [14, 39] have demonstrated that limiting the codomain of T_{R1} and T_{R2} to a predefined interval such as $[0, 1]$, by imposing a sigmoid function as their root node, can enhance algorithm's generalization ability. This implies that $T_{R1} - T_{R2}$ ranges in the interval [-1, 1], and that multiplying it by a user-specified ms allows bounding this range in $[-ms, ms]$. It is worth highlighting that these studies used a standard modifying constant $\alpha = 1$ for the sigmoid function, i.e. $f(x) = \frac{1}{1+e^{-\alpha x}}$, where α is a modifying constant and x is the respective input; as such, in the present study, $\alpha = 1$ is being used.

A consequence of applying GSOs is the growth of the evolved programs. In the case of GSM, as defined in [29], a single application of the operator increments the size by $4 + |T_{R1}| + |T_{R2}|$ operators, where $|.|$ indicates the number of operators in the programs. In the long term, as the evolution proceeds, the resulting programs might become excessively large for mathematical analysis and practical applications. In this regard, McDermott et al. [27] proposed to simplify the operator while retaining the geometric properties. In particular, the authors suggested using only one random program instead of subtracting two random programs: $GSM(T) = T + ms \cdot T_{R1}$. Moreover, to enforce the expected value of the semantic distribution of $ms \cdot T_{R1}$ being centered around zero, ms is being drawn from a normal distribution with $\mu = 0$. In such a manner, GSM increments the program size by $3 + |T_{R1}|$ nodes per step; note that the authors did not apply any bounding on the random trees. However, the experimental evidence from three uni-variate time series datasets showed that the proposed approach numerically outperforms standard GP in only one problem while standard GSGP was outperformed in two problems. However, no statistical test was performed to validate the significance of the results.

On the other hand, several authors have proposed improvements over GSM by deterministically and optimally adapting the mutation step [14, 27]. The first work in this direction was done by McDermott et al. [27], entitled hereafter GSGP-D, which is based on the aforementioned one-program GSM and determines an optimal ms by differentiating the fitness function (MSE or RMSE) with respect to ms. The authors point out that this forces the T_R1 to always step in the direction of

the semantics vector and at the right length to optimize the fitness function. Later, Gonçalves et al. [14] proposed to rewrite the GSM as a general linear system of the form $P + RI \times ms = t \Leftrightarrow RI \times ms = (t - P)$, where RI results from subtracting the two random programs. Subsequently, the Moore-Penrose pseudo-inverse is used to compute the mutation step (ms) that minimizes the error on the training data. The results of this adaptive method, entitled hereafter GSGP-O, demonstrated that an adaptive mutation step applied over on the difference between two random programs can achieve competitive generalization in only a single application of the mutation operator, while standard GSGP would require hundreds of generations to reach the same result. These modifications will also be included in the experimental work. Although the two aforementioned methods share the same idea, there are fundamental differences that determined the choice of using GSGP-O in the present study. Given that GSGP-D relies upon a single random program and does not use any transformation for bounding its output, there are two fundamental implications: (i) the optimally computed ms no longer ensures that the distribution of $ms \cdot T_{R1}$ is zero-centered, making the operator deviate from the geometric properties defined in [29]; and (ii) the absence of bounds for the output of $ms \cdot T_{R1}$ makes the mutation less stable, which was found to negatively affect its generalization ability [39]. In fact, the experimental evidence shows that GSGP-D outperforms standard GP and GSGP in only one out of three problems [27]. Contrarily, GSGP-O retains geometric properties and consistently exhibits better performance when compared to both standard GP and GSGP [14]; note that in this study, the empirical evidence was statistically validated. It is worth noting that Gonçalves et al. experimented using both bounded and unbounded random programs, and the experimental evidence suggests that the former outperforms the latter.

Existing research is focused on GSM because empirical evidence demonstrated that, unlike in standard GP, a high mutation rate is fundamental for GSGP achieving good-quality solutions [11]. The reason is that GSC reduces, at every generation, the convex hull formed by the current population. Thus, if a good-quality solution is not enclosed in such a convex hull, GSC is not useful for substantially improving the quality of the best individual found. Given that GSM does not have this limitation of GSC, it is common to run GSGP using only the GSM operator. For this reason, the present work only uses the mutation operator for the GSGP algorithm.

2.2 Deep artificial neural networks

A DANN is a hierarchical sequence of layers, each comprising simple computational units called "neurons" [18]. The term "hierarchical" reflects the capability of DANNs to learn complicated concepts by constructing them from simpler ones. The term "sequence" implies that the neural architecture is a stack of layers connected sequentially such that each successive layer takes as input the output of the precedent layer (exceptions to this generalization can be found, for example, in residual learning framework [19] and its variants like in [20]). Under this perspective, one can think of a DANN as a network comprising several subnetworks, each aimed to minimize the loss function by manipulating the respective parameters [22] while producing outputs that constitute inputs for the proceeding subnetworks.

2.2.1 Training a DANN

Gradient descent is an optimization algorithm commonly used to train DANNs and relies upon the vector of partial derivatives of the loss function with respect to the network's parameters (i.e., the weights). The gradient is used for updating the value of each parameter [18]. In practice, however, this update is performed simultaneously for all parameters and across all the layers, despite the fact that the magnitude and direction of the updates are computed under the assumption that the other parameters remain constant. This might result in unexpected outcomes that make the learning process less stable, which translates into the impracticability of using a larger learning rate and converging faster.

Consider a neural network comprised of 2 layers of parameters L_{Θ_1} and L_{Θ_2}, respectively. In simple terms, the prediction of the network ($\hat{y}$) is given by two consecutive steps: (i) feeding the input X to the first parameterized layer L_{Θ_1} and (ii) feeding the respective output to the second parameterized layer L_{Θ_2}. Formally, $\hat{y} = L_{\Theta_2}(L_{\Theta_1}(X))$. During the update of parameters, which occurs simultaneously for all the parameters, the inputs to L_{Θ_2} are affected by the parameters' change in L_{Θ_1} while the change in L_{Θ_2} was performed under the assumption that those in L_{Θ_1} remain unchanged. Such an *unexpected* change in the distribution of layers' inputs may constitute a problem, as the layers might need to continuously adapt to a new distribution. As the network becomes deeper, this problem amplifies because small changes at the beginning of the network might translate into large changes for the input distributions of deeper layers. Such a phenomenon was called internal covariate shift (ICS) [22].

2.2.2 DANNs and the vanishing gradient

ICS can be particularly harmful to DANNs if a sigmoid activation function is used (or some other function with a saturated regime of non-linearity). Consider a layer l with a sigmoid activation function whose output is given by $\mathrm{a_l} = \frac{1}{1+\mathrm{e}^{-\mathrm{W_l x_l + b}}}$, where W_l, b_l and x_l represent the weight matrix, the bias term and the input at layer l. It is worth noting that the output of $\mathrm{a_l}$ depends not only on the weights and the bias term but also on the value of x_l which, by itself, is a function of the parameters of the previous layer. As $|x_l|$ increases, $\mathrm{a_l}$ tends towards the saturated regime (where the derivative tends to zero). Given the fact that DANNs learn using gradient descent with back-propagation, the presence of very small values in the long chain of multiplications to compute partial derivatives results in the so-called vanishing gradient problem - when the partial derivative becomes so small that it barely has any impact on the parameter update - which results in significantly longer training times or even stagnation of the learning process. In this context, "fixing", to some extent, the distribution of the input throughout the learning process can, at least, help the DANNs to converge faster. To better understand the further complications that ICS brings if a sigmoid activation function is used, consider a simple multi-layer perceptron made of one hidden layer with one neuron per layer. The activation function (a) is applied over the output of both hidden and output neurons. The gradient-based learning algorithm will perform parameter updates that are proportional to the partial derivative of the loss function (L) with respect to the current set of parameters (Θ). This update is controlled by

the learning rate (α), which tends to assume small values (a default learning rate of 0.001 is used in both Keras and PyTorch [12, 33]). To calculate the partial derivative for the first parameter in the network (Θ_1), one needs to make use of the chain rule: $\frac{\partial L(\Theta)}{\partial \Theta_1} = \left(\frac{\partial L(\Theta)}{\partial \mathrm{a}^{(2)}}\right)\left(\frac{\partial \mathrm{a}^{(2)}}{\partial z^{(2)}}\right)\left(\frac{\partial z^{(2)}}{\partial \mathrm{a}^{(1)}}\right)\left(\frac{\partial \mathrm{a}^{(1)}}{\partial z^{(1)}}\right)\left(\frac{\partial z^{(1)}}{\partial \Theta_1}\right)$, where ($z^1$, a^1) and (z^2, a^2) represent the sum of the weighted inputs and the application of the subsequent activation functions in the hidden and output neurons, respectively. Although the underlying network is small, one can find quite a few multiplicative terms. In practice, however, DANNs tend to be composed of many stacked layers, thus leading to many more multiplications in the calculation of the partial derivatives. Under this perspective, when updating DANNs, the parameters that are located at deeper layers might fail to update if the terms in this chain are small: with each successive layer, the partial derivative involves the multiplication of small numbers over and over, resulting in such a small value that it has no impact.

2.2.3 Batch Normalization Transformation

To overcome the aforementioned problems, Ioffe and Szegedy [22] proposed a normalization step that fixes the means and variances of the layers' input activations throughout the network, therefore preventing small changes in the parameters of a layer from amplifying as the data propagates through the DANN; the approach was called batch normalization (BN). Typically, DANNs tend to learn in batches (i.e., on random disjoint sets of data), hence the term "batch". BN consists of normalizing the activation vectors from hidden layers using the mean and variance of the current batch, aiming at speeding up the convergence of the backpropagation learning process in a DANN [25]. Specifically, any D-dimensional input x_l to layer l is normalized in each dimension d as follows: $n_l^d = \frac{x_l^d - E[x_l^d]}{\sqrt{Var[x_l^d] + \epsilon}}$, such that the expected value and the variance are computed on the batch of training data and ϵ serves as a stabilization constant. To restore the representation power of the network, the authors introduced two learnable parameters to ensure that the transformation introduced by BN at a given layer can represent the identity transformation (if it is necessary for a better convergence). In particular, the parameters γ_l^d and β_l^d allow scaling and shifting the normalized inputs for each dimension d of layer l as $\hat{n}_l^d = \gamma_l^d \hat{n}_l^d + \beta_l^d$. The authors proposed to apply the BN transformation before the activation function to reduce the saturation problem. At inference time, the BN transformation normalizes the unseen batch using a moving average of the mean and variance calculated from the batches during training.

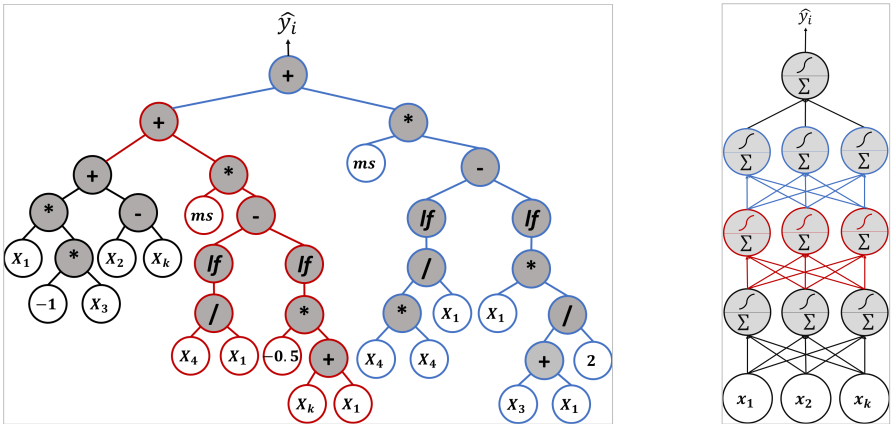
3 Standardization and Normalization of Random Programs

3.1 Motivation

Consider that Figure 1(a) represents a parent program[2] (in black) that was subject to GSM in two consecutive iterations (red and blue, respectively). The overall program can be seen as a stack of three subprograms connected sequentially via summation, such that each successive subprogram takes as input the output of the precedent subprogram (in this sense, each subprogram is represented with a different color). At this point, it becomes clear that programs built using GSM share similarities with DANNs (where layers can be seen as subprograms). To better perceive this, consider that Figure 1(b) represents a DANN made of three hidden layers: black, red, and blue, respectively. The overall network can be seen as a stack of three subnetworks connected sequentially, and one output neuron, such that each successive subnetwork takes as input the output of the precedent subnetworks, similarly to the program in Figure 1(a). If one pays attention to Figure 1, it becomes clear that the random composition of terminals before applying the sigmoid function is likely to produce outputs that fall on the saturated regime, where the co-domain approaches either zero or one (recall that the standard definition of the sigmoid function is being used, with the modifying constant set to 1). This makes part of the semantics of the random subprograms converge to single numbers, hindering GSM from exploring a broader semantic space.

The work of Nicolau & McDermott [31] is an important motivation for the present proposals, who empirically studied the bias of the Ramped Half & Half initialization method for random trees, particularly the effect on the output distribution, and concluded that: (i) a significant amount of outliers is being produced during the random composition of trees; (ii) the amount and their range depends on the function set; and (iii) the use of protected GP operators biases the semantics of random trees towards positive numbers. Thus, blindly applying sigmoid to the output of random trees in GSM can have negative effects. Moreover, it is important to note that, to support their conclusions, Nicolau & McDermott [31] used three synthesized datasets that used input features that were always generated under a zero-centered normal distribution: (i) one input feature and 500 observations $N(0, 1)$; (ii) ten input features and 500 observations $N(0, 1)$; and (iii) ten input features and 500 observations $N(0, 50)$. Although this allowed the authors to obtain convincing empirical evidence, data coming from real-world applications usually do not exhibit such a simple structure. For example, in the present study, six real-world problems are used, each with a varying number of observations and features (the latter exhibiting different scales and distributions). Such a degree of added complexity enhances the necessity to provide a method for limiting the output of the random programs used with GSM, allowing it to retain its geometric properties and competence when solving real-world problems.

[2] Given the nature of GSM and GSC, the ideas discussed here generalize to any type of program representation (trees, linear representations, graphs, etc.). The experimental work will use a linear genome representation for programs.



(a) Individual after two successive GSMs

(b) DANN with three hidden layers

Fig. 1 Visual assessment of similarities between an individual obtained using two successive applications of GSM (in red and blue, respectively) on a given parent program (in black), and a DANN with three hidden layers. Notice that nodes in gray represent functions, whereas those in white represent constant terms and terminals (the latter are also known as input features).

To address the aforementioned problem, this work proposes to incorporate the knowledge and best practices from deep learning into GSGP. Specifically, adapting the concept of BN, this work proposes standardization and normalization of the randomly generated programs used in conjunction with the GSM operator. The proposed standardization and normalization also avoids the use of two random programs (one random program will be enough) and does not require the sigmoid functions. Both of these contribute to the generation of much simpler individuals compared to the existing versions of GSGP.

3.2 Proposed semantic operators

Following the terminology of Section 2.1, let $H = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), ..., (\mathbf{x}_n, t_n)\}$ be a regression dataset where, for each $i = 1, 2, ..., n$, $\mathbf{x}_i$ represents an input observation and t_i the corresponding expected output (target value). Let T_R be a random program that is needed for applying a geometric semantic operator. Let $\mathcal{S}(T_R) = [T_R(\mathbf{x}_1), T_R(\mathbf{x}_2), ..., T_R(\mathbf{x}_n)]$ be the semantics of T_R and let m be the average of the values of vector $\mathcal{S}(T_R)$ and δ their standard deviation. Then, the standardization S_R of T_R is:

$$S_R = \frac{T_R - m}{\delta} \tag{1}$$

It is clear that S_R is a random expression centered in zero (in fact, the median of the normalized program is equal to zero). So, it is straightforward that S_R has all the needed characteristics to be used in GSM, replacing the difference between the two

random programs T_{R1} and T_{R2}. This idea will be at the basis of the novel GSGP-N1 method proposed in the continuation.

Let us now consider again the semantics of a random program T_R: $\mathcal{S}(T_R) = [T_R(\mathbf{x}_1), T_R(\mathbf{x}_2), ..., T_R(\mathbf{x}_n)]$ and let *min* be the minimum of the values in $\mathcal{S}(T_R)$ and *max* their maximum. Then, the normalization in $[0, 1]$ of T_R is:

$$N_R = \frac{T_R - min}{max - min} \tag{2}$$

If we now want a normalization in $[-1, 1]$ we can simply multiply N_R by 2 and subtract 1, obtaining:

$$M_R = 2 \cdot \frac{T_R - min}{max - min} - 1 \tag{3}$$

It is also clear that a program like M_R is appropriate for being used by GSM, and can potentially replace the difference between the two random programs T_{R1} and T_{R2}. This idea is at the basis of the method called GSGP-N2 studied in the continuation.

Compared to the traditionally used difference between two random programs, S_R (Equation (1)) and M_R (Equation (3)) are smaller, since they use only one random program. In this work, given a random program T_R, the GSM operator used by GSGP-N1 generates an individual T_{mut} defined as:

$$T_{mut}^{N1} = T + ms \cdot S_R \tag{4}$$

where S_R is defined in Equation (1). Analogously, the GSM operator used by GSGP-N2 generates an individual T_{mut} defined as:

$$T_{mut}^{N2} = T + ms \cdot M_R \tag{5}$$

where M_R is defined in Equation (3).

Both these methods will be employed by using a random value for ms at each mutation event (methods GSGP-N1 and GSGP-N2) and the optimal mutation step (methods GSGP-N1-O and GSGP-N2-O).

It is worth noting that once a random program T_R is generated, and its semantics is calculated, m, δ, *max* and *min* in Equations (1) and (3) are numeric constants. Finally, by not using a sigmoid function to wrap the output of the random programs, the proposed mutation operators generate individuals without exponential functions, thus facilitating their simplification, and potentially producing more interpretable models.

4 Experimental Setup

4.1 Implementation details

The experimental work is carried out using GSGP-CUDA, a CUDA-based implementation that runs on graphical processing units (GPUs) [35]. The implementation extends

the most efficient CPU-based implementation [9], where the evaluation of program syntax is only carried out at the beginning of a run, exploiting the fact that GSOs allow calculating fitness of the offspring by only considering the semantic vectors of the parent programs. Experimental evaluations of the CUDA-based implementation showed orders of magnitude improvements in terms of run time compared to the CPU version [35]. The CUDA implementation runs entirely on the GPU and uses a linear genome representation and stack-based processing. The original version was extended by integrating the proposed mutation operators with the computation of the optimal mutation step. Moreover, a scikitlearn interface was implemented to simplify the use and evaluation of the method. All experiments reported here were carried out on a desktop machine with an Intel(R) Xeon(R) CPU E3−1270 v3 @ 3.50GHz and 32 GB of RAM, an NVIDIA GeForce 2060 RTX GPU with 1920 CUDA cores and a base frequency of 1365 MHz, which also has a boost frequency of 1680 MHz, 12 GB of GDDR6 memory and a 192-bit memory interface with a bandwidth of 336 GB/s. The source code for GSGP-CUDA and the normalized GSM operators can be found in https://gitlab.com/Jmmc9122/gsgpcuda.

4.2 Test problems and experimental evaluation

Experiments were carried out employing the six real-world symbolic regression problems used in [35] and summarized in Table 1. All problems include one target variable and the number of features ranges from 6 to 25, while the number of samples goes from 2208 to 5000. Data partitioning was done using 30 difference random splits (one for each run), with 70% of the observations (selected randomly with uniform probability) used for training and the rest for testing. Performance was measured using several criteria, namely:

- Convergence during training. Convergence plots of the median training fitness over the 30 independent runs are presented, considering a fixed number of generations.
- Testing performance. Performance on the test set is evaluated using three measures: the root mean squared error (RMSE), the coefficient of determination R^2, and the mean absolute percentage error (MAPE).
- Model size. The model size is computed as the number of nodes of the best program in the population at the end of the evolution.

The model size is measured at two moments: before and after mathematical simplification. The non-simplified model is the program generated by GSGP (the raw model). The simplified one is obtained by applying the *simpify* function (from the Sympy library[3]) to the raw model. For better performance, *simpify* is applied to each linear term of the final expression, since applying it to the complete raw model sometimes produces errors or sub-optimal results. Once the model size is reduced in this way, *simpify* is applied once again to the complete model.

It is important to mention that *simpify* only performs a symbolic evaluation of the model. Basically, it removes introns from the expression but does not apply a more comprehensive simplification procedure. While this does not produce the best possible reduction in size, it is more stable since simplification can sometimes produce

[3]Version 1.10.1: https://github.com/sympy/sympy

Table 1 Real-world symbolic regression problems, as used in [35].

Problems	Instances	Features	Description
Concrete [42]	1030	8	Concrete compressive strength dataset based on building parameters.
Energy Cooling [36]	768	8	Cooling requirements of buildings as a function of building parameters.
Energy Heating [36]	768	8	Heating requirements of buildings as a function of building parameters.
Housing [34]	506	13	Housing values in suburbs of Boston.
Tower [41]	5000	25	Industrial dataset of chromatography measurements of a distillation tower.
Yacht [21]	308	6	Dataset regarding the hydrodynamic performance of sailing.

Table 2 Experimental parameters used for GSGP.

Parameter	Values
Genes	1024
Population Size	1024
Number of generations	200
Mutation rate	1
Terminal set	Features and random constants
Function set	$+, -, \times, pd$
Survival	Keep-best elitism

very long run times, especially when the raw model is extremely large. It is assumed that models using standardized and normalized programs will be simpler and easier to reduce in size than those that employ a sigmoid function because they do not include exponential terms.

4.3 Variants and parameters

For the set of problems summarized in Table 1, six variants of GSGP are evaluated:

- GSGP: Baseline GSGP with standard GSM; random mutation step.
- GSGP-O: Baseline GSGP with standard GSM; optimal mutation step.
- GSGP-N1: standardization of random programs employed by GSM, as in Equation (1); random mutation step.
- GSGP-N1-O: standardization of random programs employed by GSM, as in Equation (1); optimal mutation step.
- GSGP-N2: min-max normalization of random programs employed by GSM, as in Equation (3); random mutation step.
- GSGP-N2-O: min-max normalization of random programs employed by GSM, as in Equation (3); optimal mutation step.

Standard GP is not included in the comparisons since previous works have conclusively shown that GSGP outperforms it in terms of efficiency and accuracy [39]. All these GSGP variants adopt the parameters summarized in Table 2. As previously discussed, none of them uses crossover. The GSGP implementation uses a linear

representation, so the genes parameter specifies the total number of elements in an individual. However, the size of the program and the specific expression that produces the semantics of a program might be substantially smaller since individuals are randomly generated without imposing restrictions. The interpreter uses a stack to compute the output of a program and non-valid operations are skipped. Random constants are generated in the range $[-10, 10]$ and the protected division pd returns the numerator when the denominator is zero. For the standard GSM, a sigmoid function limits the output of the random programs. Finally, the computation of the optimal mutation step can sometimes become numerically unstable, producing very large ms values and making the models unstable. A simple heuristic was used by limiting ms values to the range $[-5000, 5000]$ as a form of regularization. While better approaches might be devised, this heuristic was sufficient to generate robust models in these experiments.

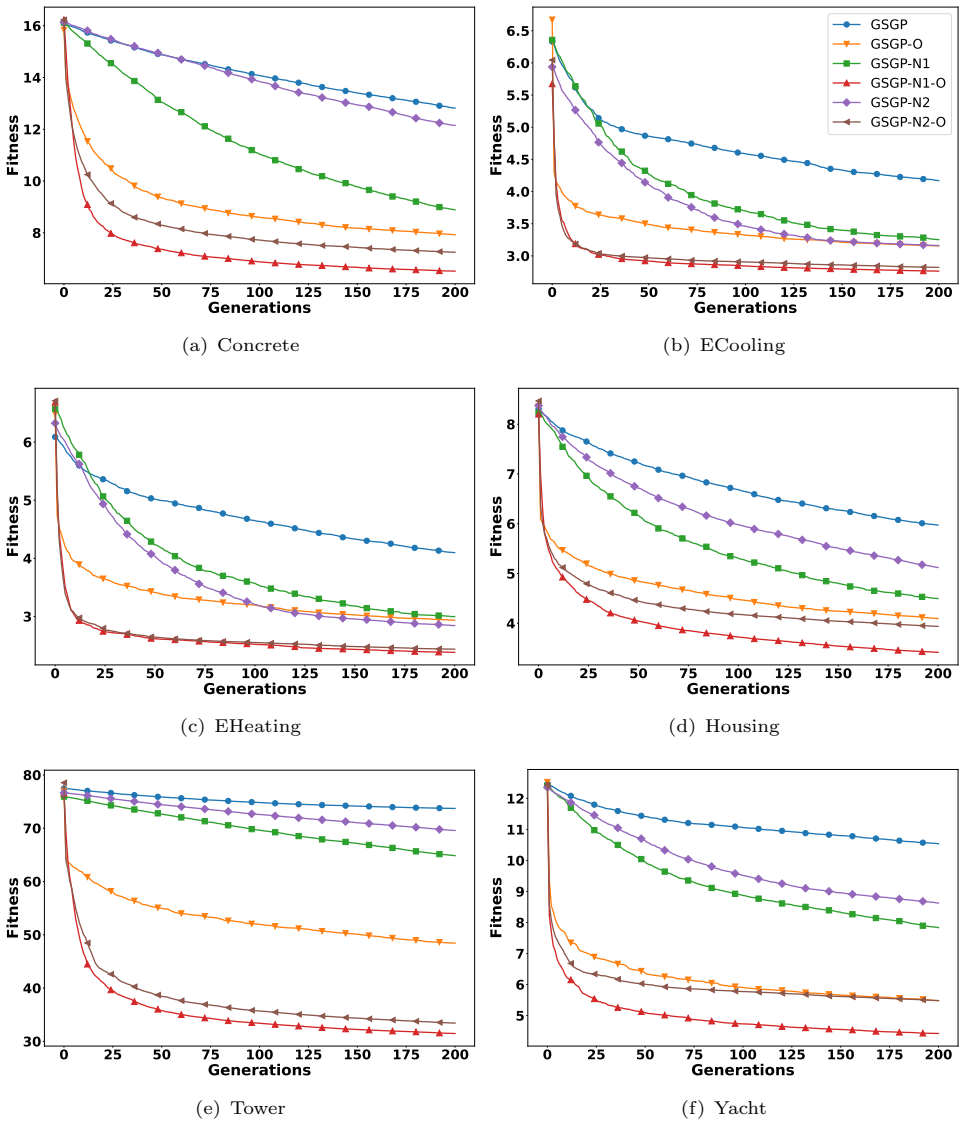
5 Results and Discussion

Figure 2 shows the convergence of all the studied variants on the training set. From these results, it is clear that all variants that use an optimal mutation step converge faster on all the studied problems, as expected. Moreover, the normalized versions (N1 and N2) achieve the fastest convergence on all the problems, with GSGP-N2-O performing slightly better. Focusing on the methods that use a random mutation, GSGP-N1 exhibits the fastest convergence while GSGP exhibits the slowest.

Table 3 summarizes the results of the best model found when evaluated on the test set, showing the median and IQR of the five performance measures: RMSE, R^2, MAPE, size of the raw model, and size of the simplified model. These same results are presented graphically using violin plots with: RMSE reported in Figure 3, R^2 in Figure 4, MAPE in Figure 5, raw model size in Figure 6, and simplified model size in Figure 7. GSGP-N1-O overfits during training and produces testing scores that fall outside the range of values produced by the other methods, thus skewing the plots. For this reason, this variant is excluded from the figures for RMSE, R^2, and MAPE.

Overall, the best performance is obtained with a normalization approach and a method that uses an optimal mutation step. In particular, GSGP-N2-O achieves the best performance in terms of accuracy, both in median performance and in robustness (based on the IQR), with a few exceptions. When it did not achieve the best performance, it produced the second-best result. GSGP-O also achieves good accuracy, particularly on the Yacht problem. GSGP-N1 was by far the most competitive method of those that used a random mutation step. On the other hand, GSGP-N1-O exhibited the worst accuracy on the testing data, clearly overfitting the training set, as shown in Figure 2. The reason may be the numerical instabilities induced by the standardization, which produces many values close to zero. On the other hand, the min-max normalization produces more values closer to 1 and -1, which probably limits overfitting.

Regarding model size, it is clear that the normalized methods produce smaller models, which was expected since the GSM only requires one random program instead of two. Moreover, methods that employ an optimal mutation step also tend to produce smaller models, although there is a large overlap between the distributions produced



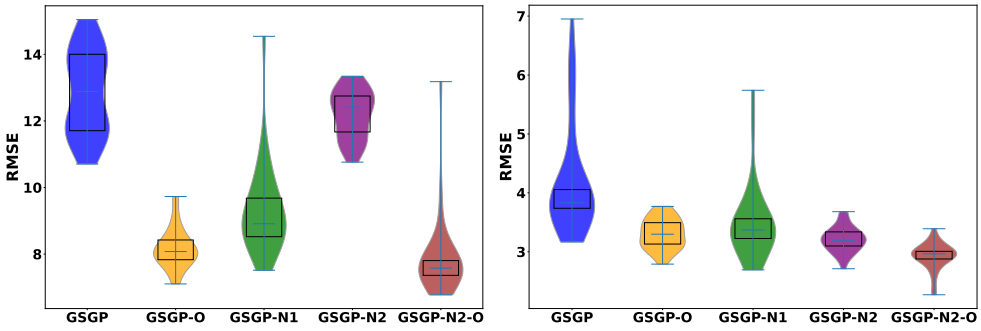
(a) Concrete (b) ECooling

(c) EHeating (d) Housing

(e) Tower (f) Yacht

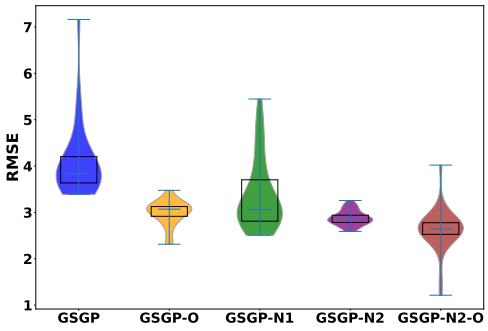
Fig. 2 Convergence plots showing the mean best fitness value per generation during training for all the GSGP variations evaluated.

with and without this computation. However, the maximum peeks in program size are produced by the variants that use an optimal mutation step in most problems. Finally, the size of the models after simplification is where an order of magnitude improvement is achieved. Both GSGP-N1-O and GSGP-N2-O produce the smallest models. As hypothesized, since these methods do not use a sigmoid function in the GSM, they are more amenable to automatic simplification.

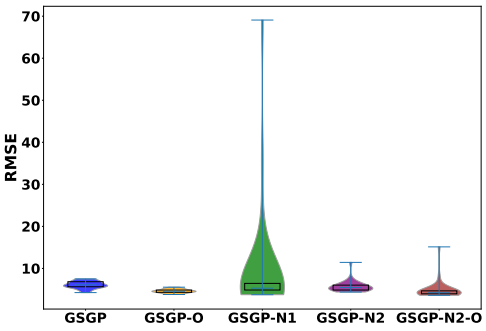


(a) Concrete

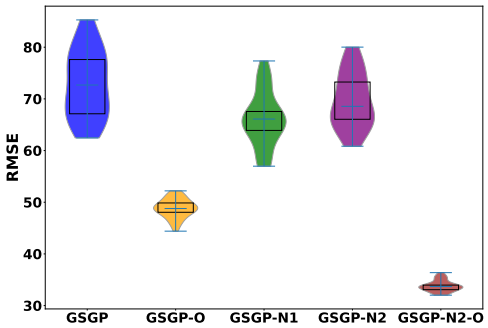
(b) ECooling



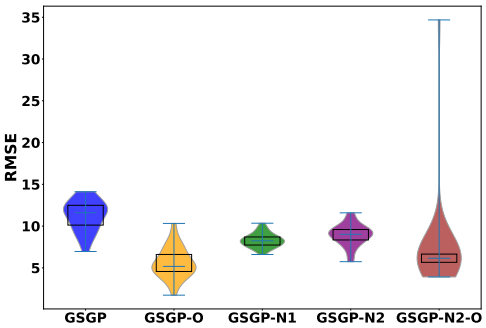
(c) EHeating



(d) Housing

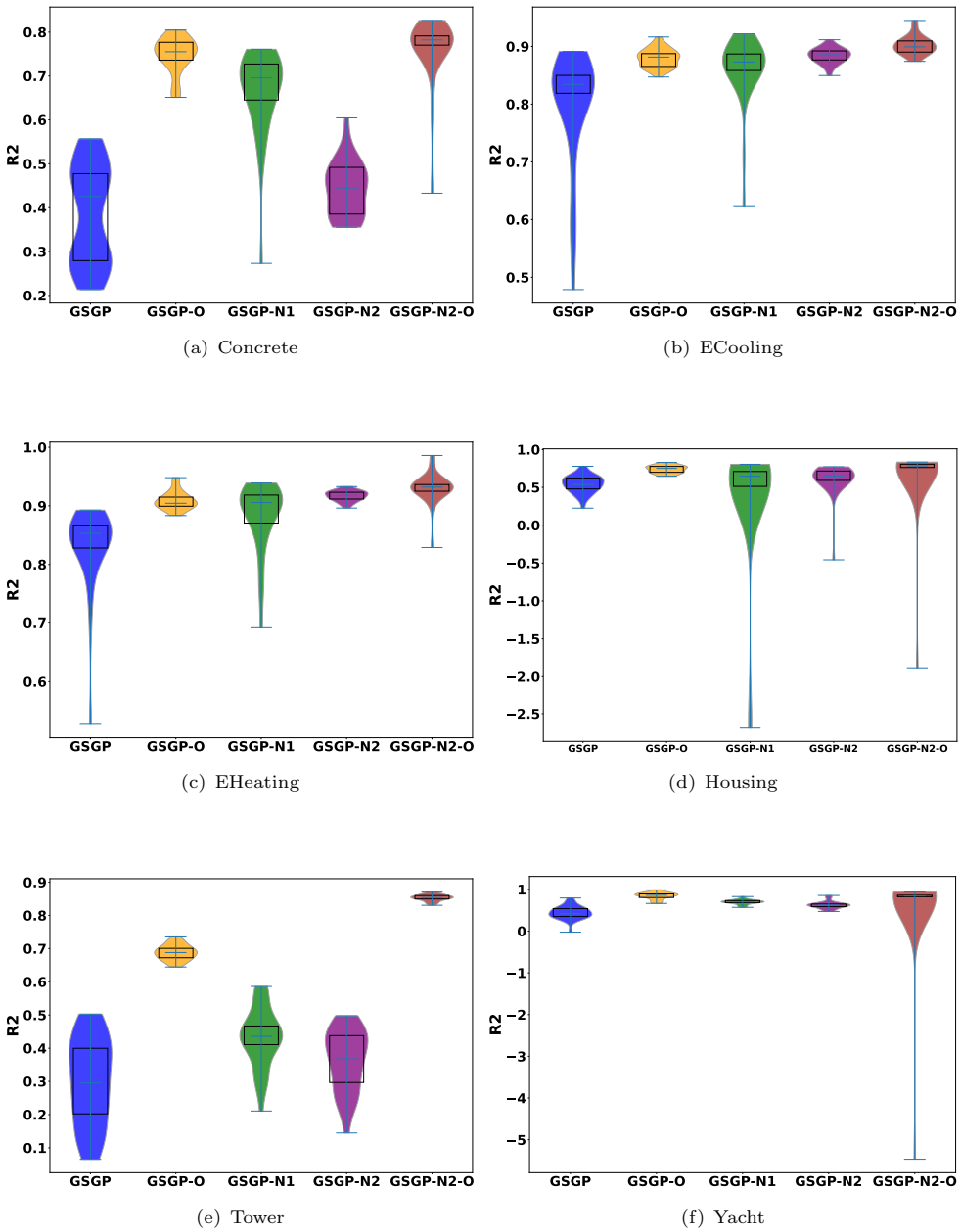


(e) Tower



(f) Yacht

Fig. 3 Violin plots of RMSE scores obtained on the test set for all GSGP variants evaluated.

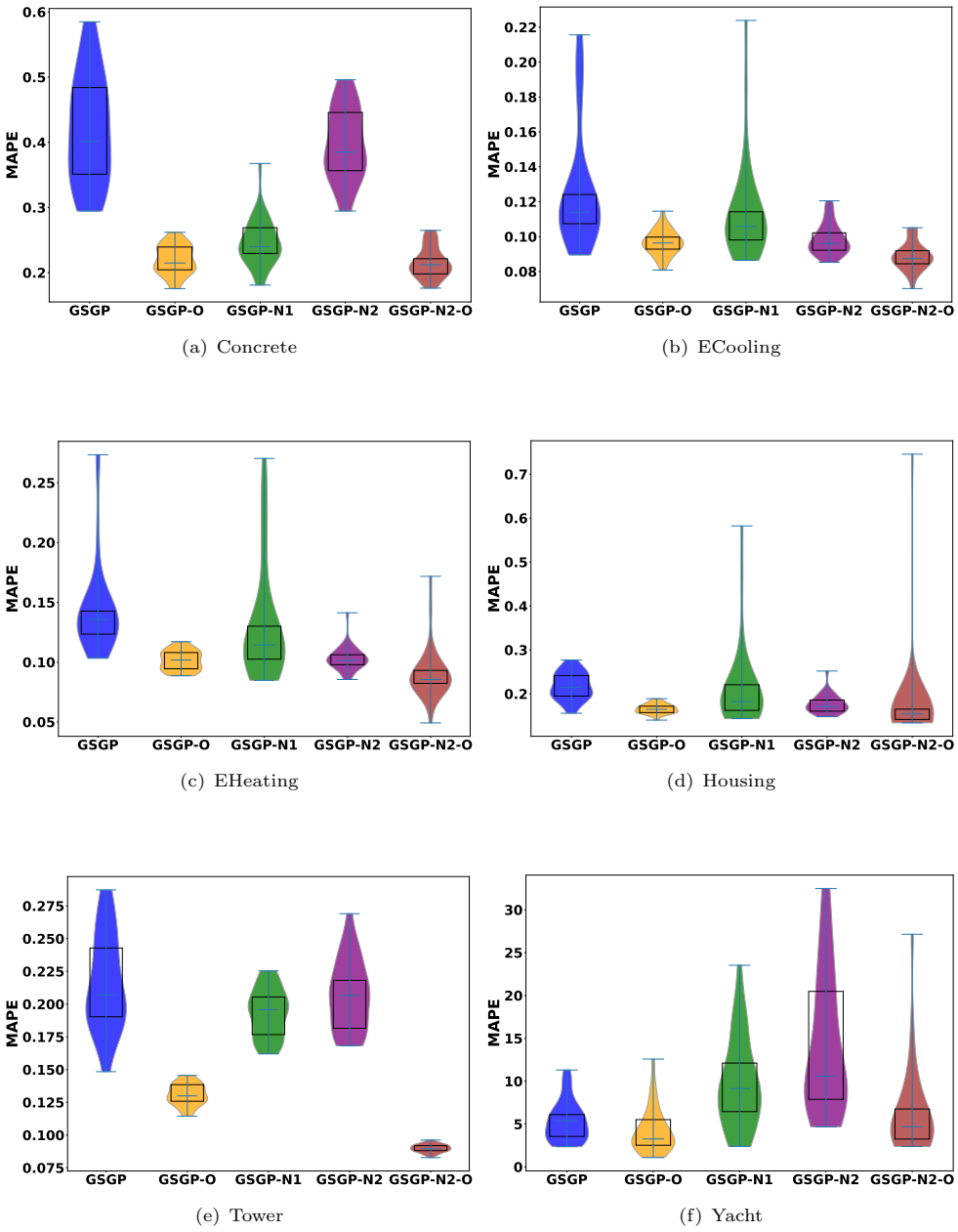


(a) Concrete (b) ECooling

(c) EHeating (d) Housing

(e) Tower (f) Yacht

Fig. 4 Violin plots of R2 scores obtained on the test set for all GSGP variants evaluated.

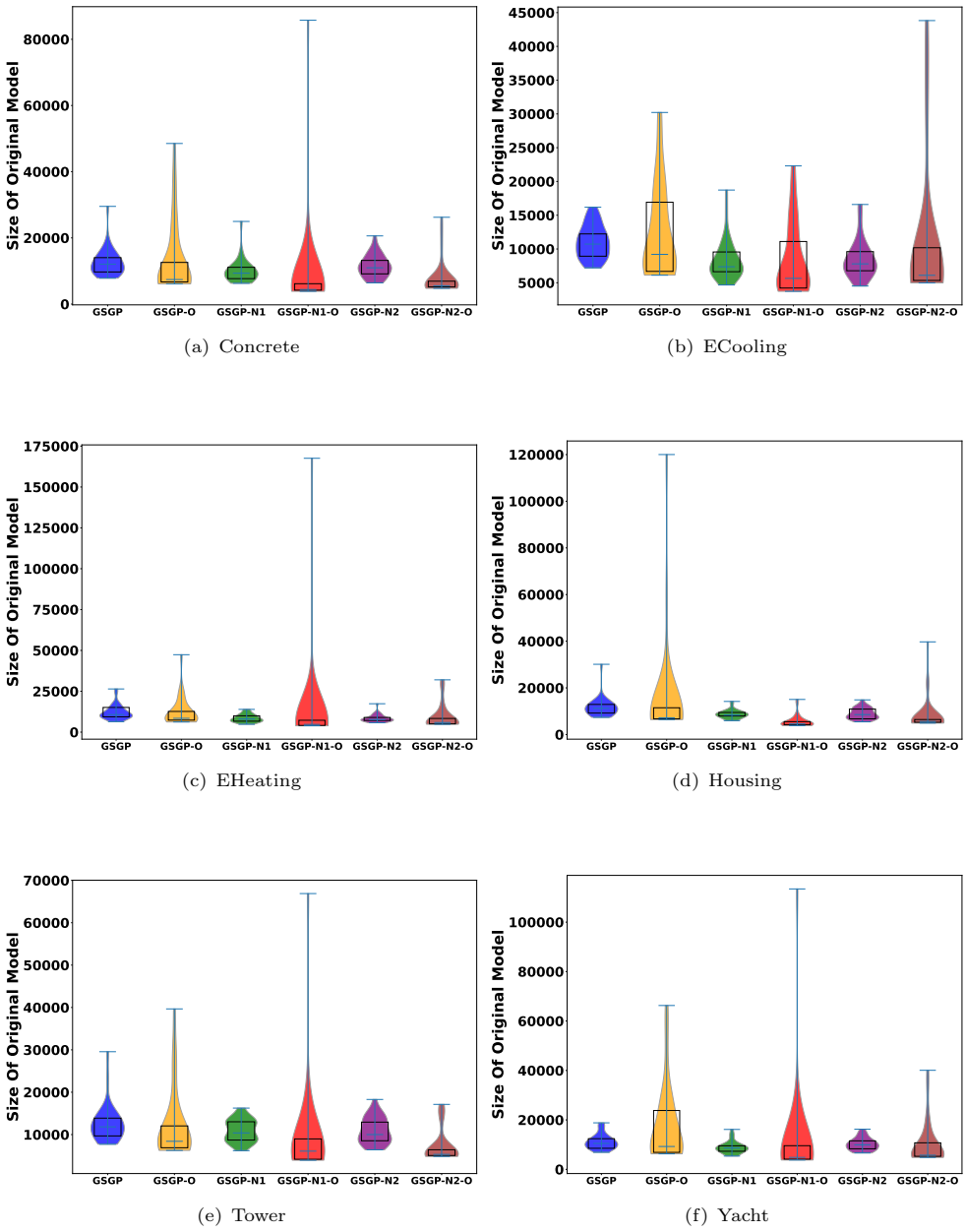


(a) Concrete (b) ECooling

(c) EHeating (d) Housing

(e) Tower (f) Yacht

Fig. 5 Violin plots of MAPE scores obtained on the test set for all GSGP variants evaluated.

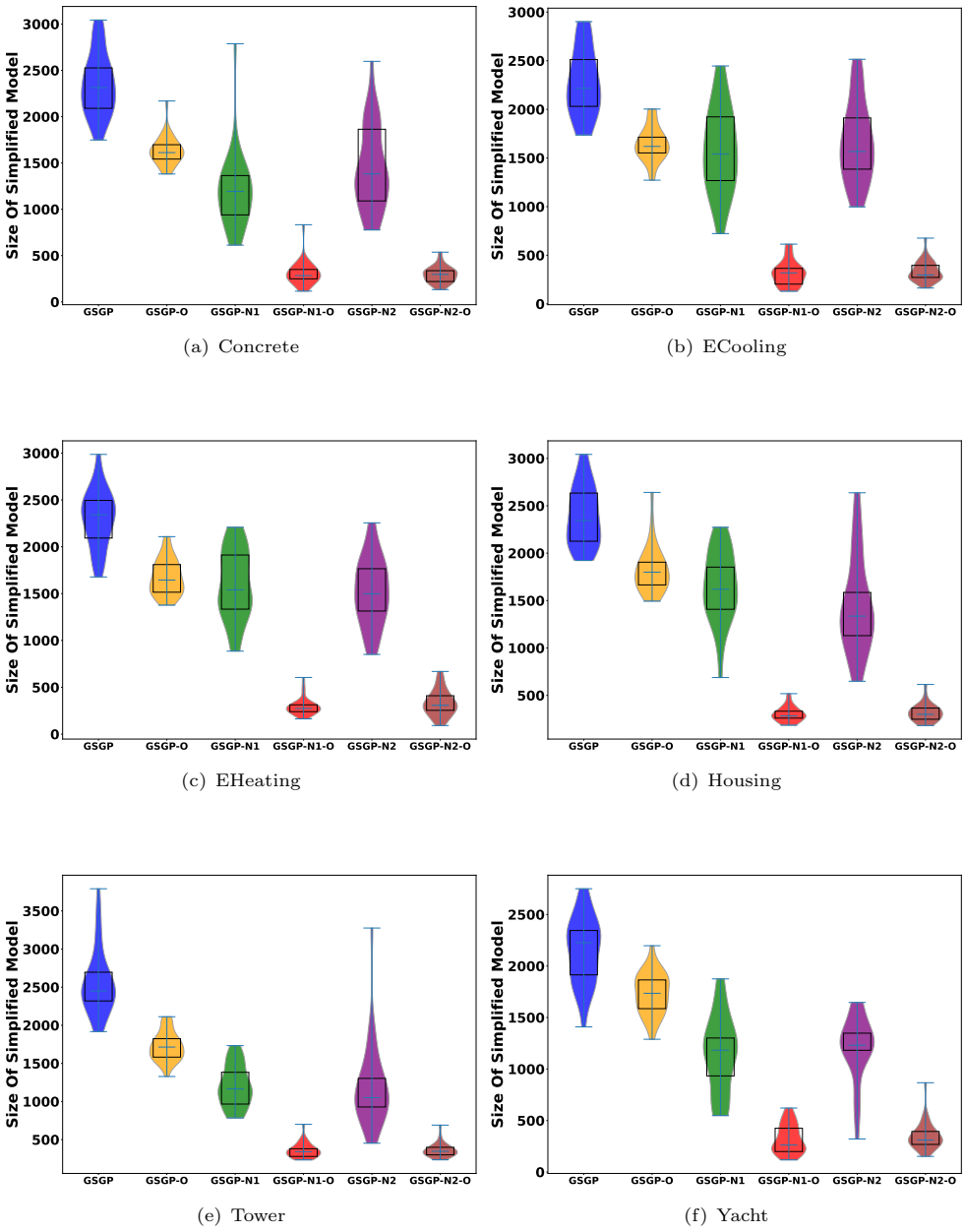


(a) Concrete (b) ECooling

(c) EHeating (d) Housing

(e) Tower (f) Yacht

Fig. 6 Violin plots of model sizes before simplification, for all GSGP variants evaluated.



(a) Concrete (b) ECooling

(c) EHeating (d) Housing

(e) Tower (f) Yacht

Fig. 7 Violin plots of model sizes after simplification, for all GSGP variants evaluated.

Table 3 Comparison of GSGP variants showing median and IQR (median/IQR) on the test set, with bold indicating the best results.

Problem	GSGP	GSGP-O	GSGP-N1	GSGP-N1-O	GSGP-N2	GSGP-N2-O
RMSE						
Concrete	13.1/2.27	8.12/0.54	8.92/1.12	17.03/14.69	12.30/1.06	**7.57/0.43**
ECooling	3.83/0.32	3.32/0.36	3.40/0.57	7.21/7.90	3.20/0.23	**2.96/0.14**
EHeating	3.85/0.62	3.07/**0.20**	3.07/0.92	8.91/11.63	2.90/**0.20**	**2.66/0.24**
Housing	5.96/1.17	4.57/**0.51**	5.52/1.54	12.01/17.35	5.44/1.14	**4.24/0.73**
Tower	73.02/10.30	48.70/2.06	65.81/4.19	36.80/10.35	68.31/7.12	**33.65/0.88**
Yacht	11.63/2.33	**5.24**/1.94	8.22/**0.95**	9.14/12.06	9.14/1.09	6.11/0.98
R^2						
Concrete	0.36/0.19	0.75/0.04	0.69/0.09	-0.05/1.60	0.44/0.09	**0.78/0.02**
ECooling	0.83/0.03	0.88/0.02	**0.90**/0.06	0.40/1.56	0.88/**0.01**	0.89/0.02
EHeating	0.85/0.02	0.90/**0.01**	0.87/0.04	0.21/2.55	0.91/**0.01**	**0.93/0.01**
Housing	0.53/0.14	0.75/**0.07**	0.63/0.19	-0.6/6.52	0.65/0.11	**0.78/0.07**
Tower	0.28/0.21	0.68/0.02	0.43/0.06	0.82/0.11	0.37/0.13	**0.85/0.01**
Yacht	0.42/0.18	**0.86**/0.07	0.70/0.04	0.66/1.4	0.62/0.08	0.84/**0.03**
MAPE						
Concrete	0.41/0.12	**0.21**/0.03	0.23/0.04	0.53/0.56	0.39/0.08	**0.21/0.02**
ECooling	0.11/0.01	0.09/0.00	0.11/0.04	0.33/0.42	0.09/0.01	**0.08/0.00**
EHeating	0.13/**0.01**	0.10/**0.01**	0.10/0.02	0.48/0.67	0.10/0.01	**0.08/0.01**
Housing	0.21/0.04	0.16/**0.01**	0.18/0.06	0.53/0.93	0.17/0.02	**0.15/0.2**
Tower	0.20/0.05	0.12/0.01	0.19/0.02	0.09/0.02	0.20/0.03	**0.08/0.00**
Yacht	4.85/2.90	**3.27/2.79**	9.13/6.69	10.6/16.3	13.3/7.58	4.80/5.3
Raw Model Size						
Concrete	12297/4559	7410/5598	9403/3506	**4404/1836**	10854/3949	5552/2617
ECooling	10857/3220	9839/10276	7998/3207	**5453**/6745	7643/**2851**	5917/4434
EHeating	10921/5810	8508/5074	7810/3525	**4887**/3180	8454/3449	5858/**3029**
Housing	11228/3577	7062/4315	8542/1490	**4544/1191**	8293/4091	5605/1376
Tower	11861/4141	8582/5994	10273/4545	5854/4739	9916/4386	**5327/1566**
Yacht	10533/3741	9068/16945	8565/**2104**	**4746**/5322	9600/3947	5789/6320
Simplified Model Size						
Concrete	2316/433	1608/150	1193/423	**283/105**	1384/714	300/111
ECooling	2236/507	1632/172	1506/571	316/168	1566/512	**296/127**
EHeating	2350/384	1645/281	1437/671	**274/71**	1618/690	305/153
Housing	2377/496	1803/236	1620/446	**281/89**	1335/467,	310/115
Tower	2466/385	1729/248	1161/404	**341/94**	1040/374	347/101
Yacht	2222/421	1731/288	1175/509	**258**/223	1258/698	312/**121**

Table 4 presents the average rank of the algorithms based on median performance across all problems and for each of the performance metrics. Statistical significance was tested using the Friedman multigroup test for the ranking of each method based on median performance on each problem. This is a non-parametric test where the null hypothesis is that all the columns (GSGP variants) effects are the same. Performing the test for all five measures, the p-values were all below $4e^{-04}$, allowing us to reject the null hypothesis with high confidence. For the pairwise test, post-hoc analysis of the Friedman test is carried out using the Tukey-Kramer method to account for multiple comparisons. The resulting p-values are shown in Table 5 for each pairwise comparison. In general, these results confirm that GSGP-N2-O achieves the best performance since it is the only one that achieves statistically significant results when compared with other methods. It is important to note, moreover, that when the number of compared methods is small relative to the number of problems, the Friedman test can underestimate the difference between methods [13].

Another aspect that is analyzed is the composition of the best model found in each run. All of the GSGP variants use "keep best" elitism, where the best parent replaces the worst of the offspring when it is also better than the best of the offspring.

Table 4 Average rank based on median performance considering all problems and each performance measure; bold indicates top rank.

Measure	GSGP	GSGP-O	GSGP-N1	GSGP-N1-O	GSGP-N2	GSGP-N2-O
RMSE	5.3	2.3	3.6	5.1	3.3	**1.1**
R^2	5,3	2.5	3.1	5	3.6	**1.3**
MAPE	4.5	2	4.1	5.1	4	**1.1**
Raw Size	6	3.8	4	**1.1**	4.1	1.8
Simplified Size	6	5	3.3	**1.1**	3.6	1.8

Table 5 Pairwise Friedman post-hoc analysis of algorithm ranks using the Tukey-Kramer to account for multiple comparisons. Bold values are below the $\alpha = 0.05$ confidence level.

	GSGP	GSGP-O	GSGP-N1	GSGP-N1-O	GSGP-N2	GSGP-N2-O
RMSE						
GSGP	-	0.061	0.636	1.00	0.432	**0.001**
GSGP-O	-	-	0.820	0.091	0.940	0.889
GSGP-N1	-	-	-	0.733	0.999	0.188
GSGP-N1-O	-	-	-	-	0.533	**0.002**
GSGP-N2	-	-	-	-	-	0.338
GSGP-N2-O	-	-	-	-	-	-
4R^2						
GSGP	-	0.091	0.338	0.999	0.636	**0.002**
GSGP-O	-	-	0.989	0.188	0.889	0.889
GSGP-N1	-	-	-	0.533	0.997	0.533
GSGP-N1-O	-	-	-	-	0.820	**0.009**
GSGP-N2	-	-	-	-	-	0.256
GSGP-N2-O	-	-	-	-	-	-
MAPE						
GSGP	-	0.188	0.999	0.989	0.997	**0.024**
GSGP-O	-	-	0.338	**0.039**	0.432	0.972
GSGP-N1	-	-	-	0.940	1.00	0.061
GSGP-N1-O	-	-	-	-	0.889	**0.002**
GSGP-N2	-	-	-	-	-	0.091
GSGP-N2-O	-	-	-	-	-	-
Raw Model Size						
GSGP	-	0.338	0.432	**0.000**	0.533	**0.001**
GSGP-O	-	-	1.00	0.133	0.999	0.432
GSGP-N1	-	-	-	0.091	1.00	**0.0338**
GSGP-N1-O	-	-	-	-	0.061	0.989
GSGP-N2	-	-	-	-	-	0.256
GSGP-N2-O	-	-	-	-	-	-
Simplified Model Size						
GSGP	-	0.94	0.133	**0.000**	0.256	**0.001**
GSGP-O	-	-	0.636	**0.005**	0.820	**0.039**
GSGP-N1	-	-	-	0.338	0.999	0.733
GSGP-N1-O	-	-	-	-	0.188	0.989
GSGP-N2	-	-	-	-	-	0.533
GSGP-N2-O	-	-	-	-	-	-

In particular, Table 6 presents the median and IQR of the number of times a survival event due to elitism is present in the lineage of the best solution found. High median values indicate that elitism played a big role in the construction of a model, while the opposite is true for low median values. In general, results show that elitism is more important when using a random mutation step, while it has little or no influence when an optimal mutation step is used. Moreover, elitism is slightly more important when a

Table 6 Comparison of GSGP variants showing median and IQR (median/IQR) of the number of elitism survival events present in the evolutionary lineage of the best solution found.

Problem	GSGP	GSGP-O	GSGP-N1	GSGP-N1-O	GSGP-N2	GSGP-N2-O
Concrete	38.5/9.50	1.0/2.0	45.5/5.75	0.5/1.0	55.5/11.0	1.0/1.0
ECooling	44.0/15.75	2.0/3.0	103/36.75	1.0/1.0	99.5/31.5	1.0/2.0
EHeating	42.0/14.25	2.0/1.0	90/22.0	0.0/1.0	91.5/20.75	1.0/1.0
Housing	45.0/8.50	1.0/1.75	74.5/13.0	0.0/1.0	69.5/15.0	0.0/1.0
Tower	32.0/6.00	3.5/3.0	35.0/10.75	0.0/1.0	39.5/19.25	0.0/1.0
Yacht	40.00/10.5	1.0/1.75	54.5/19.0	1.0/1.0	64.5/12.0	1.0/1.0

random mutation step is used along with a normalized GSM compared to the standard GSM. It seems that when an optimal mutation step is used, the best individual can be improved upon at almost every generation, while random mutations are far less likely to do so. Normalized versions of GSM reduce the likelihood of improving the best model found so far during a run when combined with a random mutation step.

5.1 Comparison with rescaled input data

To determine the impact of the normalized semantics on GSGP, one additional analysis is performed. The experiments described above are repeated, but min-max normalization is first applied to rescale the data to the range $[-1, 1]$. In these tests, GSGP and GSGP-O are used as baselines, GSGP-N1 since it was the best-performing method with a random mutation step, and GSGP-N2-O since it achieved the best performance overall. Moreover, performance is measured using R^2 on the test set to account for the differences in scale. The methods trained and tested on the normalized datasets have a D appended to their acronym: GSGP-D, GSGP-O-D, GSGP-N1-D, and GSGP-N2-O-D. Table 7 shows all the results, including the ones achieved with these methods with the raw and the min-max normalized data. A more detailed comparison is presented in Figure 8 using boxplots to compare the methods[4]. In almost all cases, performance improved when using the normalized datasets, with GSGP-N1 being the only exception. However, the best results were once again obtained when GSM uses normalized random programs using min-max normalization; on three of the six problems, GSGP-N2-O performed better than GSGP-N2-O-D.

A similar analysis is presented in Table 8, but in this case, problem data are standardized to mean zero and standard deviation 1. For this analysis, the methods have an S appended to their acronym: GSGP-S, GSGP-O-S, GSGP-N1-S, and GSGP-N2-O-S. A similar trend can be seen, to the one observed before for the baseline methods, with GSGP-S and GSGP-O-S outperforming GSGP and GSGP-O. However, the GSGP variants using normalized semantics did not perform as well, with performance dropping in most cases. Nonetheless, comparing Table 7 and Table 8, the best performance is still achieved when using a normalized approach.

6 Conclusions and Future Work

Inspired by a common practice in deep learning to stabilize the distribution of inputs to a subnetwork called batch normalization (BN), this work proposes two novel variants

[4]Boxplots are used instead of violin plots for easier depiction.

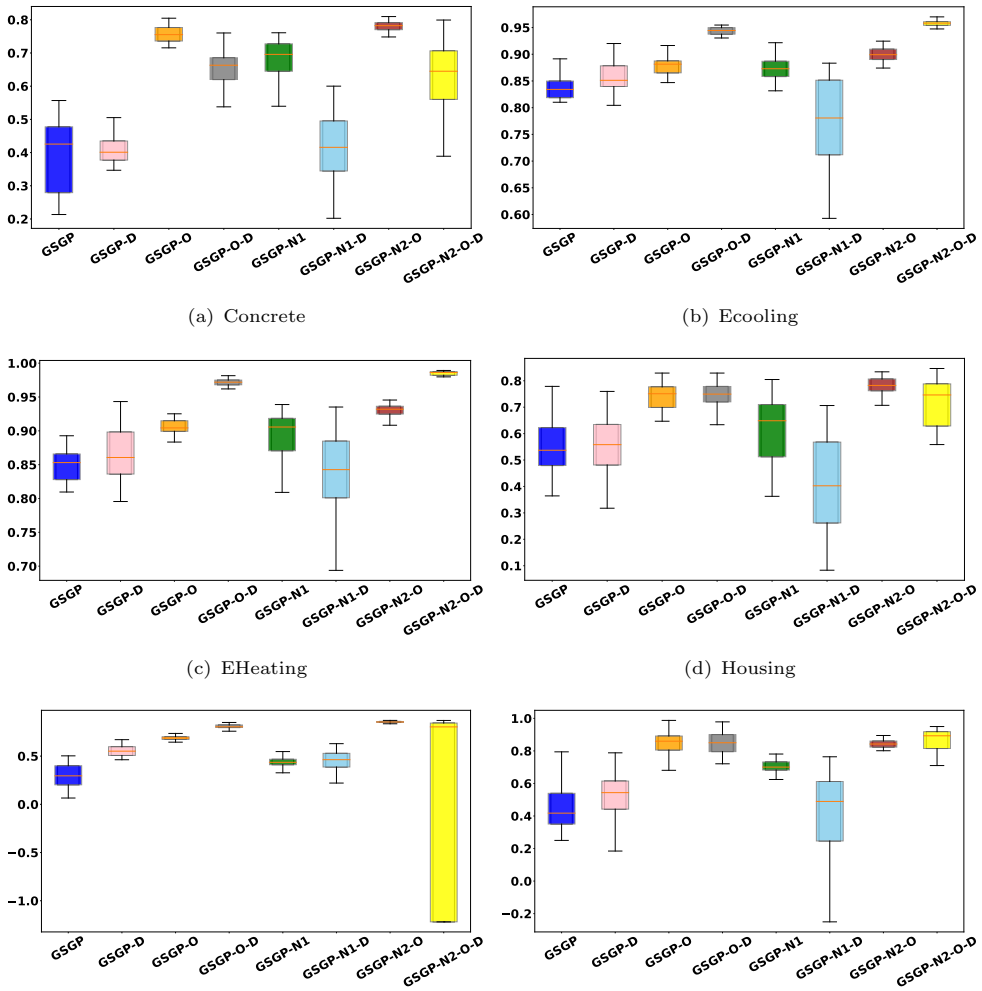
Table 7 Comparison of GSGP variants using min-max normalized datasets, showing median and IQR (median/IQR) R^2 scores on the test set. Bold indicates the best results for each problem.

Problem	GSGP	GSGP-D	GSGP-O	GSGP-O-D
Concrete	0.36/0.19	0.39/0.05	0.75/0.04	0.65/0.06
ECooling	0.83/0.03	0.85/0.04	0.88/0.02	0.94/0.01
EHeating	0.85/0.02	0.85/0.06	0.90/0.01	0.97/0.00
Housing	0.53/0.14	0.54/0.14	0.75/**0.07**	0.75/0.05
Tower	0.28/0.21	0.55/0.08	0.68/0.02	0.80/0.02
Yacht	0.42/0.18	0.54/0.18	0.86/0.07	0.85/0.10
Problem	GSGP-N1	GSGP-N1-D	GSGP-N2-O	GSGP-N2-O-D
Concrete	0.69/0.09	0.42/0.15	**0.78/0.02**	0.64/0.14
ECooling	0.90/0.06	0.77/0.13	0.89/0.02	**0.95/0.00**
EHeating	0.87/0.04	0.84/0.08	0.93/0.01	**0.98/0.00**
Housing	0.63/0.19	0.40/0.28	**0.78/0.07**	0.74/0.15
Tower	0.43/0.06	0.46/0.13	**0.85/0.01**	0.80/1.9
Yacht	0.70/0.04	0.47/0.36	0.84/**0.03**	**0.89/0.14**

Table 8 Comparison of GSGP variants using standardized data datasets, showing median and IQR (median/IQR) R^2 scores on the test set. Bold indicates the best results for each problem.

Problem	GSGP	GSGP-S	GSGP-O	GSGP-O-S
Concrete	0.36/0.19	0.52/0.12	0.75/0.04	0.72/0.06
ECooling	0.83/0.03	0.87/0.03	0.88/0.02	**0.93/0.00**
EHeating	0.85/0.02	0.88/0.02	0.90/0.01	**0.96/0.00**
Housing	0.53/0.14	0.60/0.12	0.75/0.07	0.75/0.04
Tower	0.28/0.21	0.62/0.06	0.68/0.02	0.82/0.02
Yacht	0.42/0.18	0.70/0.09	0.86/0.07	**0.85/0.08**
Problem	GSGP-N1	GSGP-N1-S	GSGP-N2-O	GSGP-N2-O-S
Concrete	0.69/0.09	0.49/0.12	**0.78/0.02**	0.53/0.14
ECooling	0.90/0.06	0.87/0.02	0.89/0.02	0.86/0.02
EHeating	0.87/0.04	0.88/0.02	0.93/0.01	0.88/0.02
Housing	0.63/0.19	0.62/0.12	**0.78/0.07**	0.52/0.34
Tower	0.43/0.06	0.61/0.14	**0.85/0.01**	0.65/0.49
Yacht	0.70/0.04	0.65/0.14	0.84/0.03	0.64/0.09

of Geometric Semantic Genetic Programming (GSGP) that decrease the complexity of the Geometric Semantic Mutation (GSM) operator and remove the need for a sigmoid bounding function. In traditional GSGP, GSM uses the difference between two random programs, each of which is often wrapped with a sigmoid function since experimental evidence has indicated that this practice limits overfitting. Both of the proposed GSGP variants use GSM with different genetic material compared to standard GSGP. In particular, both variants use only one random program instead of two, generating smaller offspring compared to traditional GSM. Moreover, the proposed variants aim at stabilizing the output distribution of random programs without requiring a sigmoid function. Using the sigmoid function produces complex and difficult to interpret solutions, while the proposed GSM variants only introduce numeric constants into the programs (the constants used to standardize and normalize the random programs), which leads to models that are more easily simplified. Both of the proposed variants



(a) Concrete (b) Ecooling

(c) EHeating (d) Housing

(e) Tower (f) Yacht

Fig. 8 Boxplot comparison of R2 scores on the test data using both raw and min-max normalized data.

have been studied using a random mutation step or an optimal mutation step at each mutation event. The experimental results, obtained on six real-life symbolic regression problems, showed the advantage of the proposed variants. In particular, besides the expected advantages in terms of model size, the proposed variants outperform standard GSGP on most of the studied problems in terms of predictive accuracy. While the variant that uses standardized random programs with an optimal mutation step is the one that obtained the best training performance, the variant that uses normalized random programs with an optimal mutation step is the one that outperformed all the other studied methods on unseen data.

In future work, one goal is to use more efficient techniques to simplify the programs produced by the new operators. Also, a hybrid approach that combines the standardization and normalization techniques deserves to be investigated. Last but not least, future work will focus on the study of these new operators on more test problems and real-life applications, and the possible extension of the approach to other forms of GP and other learning domains.

Compliance with Ethical Standards

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Author Contribution

Conceptualization: Illya Bakurov, Mauro Castelli, Nuno Rodrigues, Sara Silva, Leonardo Vanneschi; Methodology: Illya Bakurov, José Manuel Muñoz Contreras, Leonardo Trujillo, Leonardo Vanneschi; Formal analysis and investigation: Illya Bakurov, José Manuel Muñoz Contreras, Leonardo Trujillo; Writing - original draft preparation: Illya Bakurov, José Manuel Muñoz Contreras, Mauro Castelli, Nuno Rodrigues, Leonardo Vanneschi; Writing - review and editing: Mauro Castelli, Sara Silva, Leonardo Trujillo, Leonardo Vanneschi; Funding acquisition: Leonardo Trujillo, Leonardo Vanneschi, Mauro Castelli, Sara Silva; Supervision: Leonardo Trujillo, Leonardo Vanneschi.

Competing Interests

Several authors are board members of Genetic Programming and Evolvable Machines. Leonardo Trujillo, Sara Silva and Leonardo Vanneschi are Associate Editors, while Mauro Castelli is on the Editorial Board.

Acknowledgments

This work was partially supported by FCT, Portugal, through funding of research units MagIC/NOVA IMS (UIDB/04152/2020) and LASIGE (UIDB/00408/2020 and UIDP/00408/2020). This work also was supported by CONACYT (Mexico) project CF-2023-I-724, TecNM (Mexico) project 16788.23-P. The second author was supported by CONACYT scholarship 771416; the fourth author was supported by FCT PhD grant 2021/05322/BD.

References

[1] Albinati J, Pappa GL, Otero FE, et al (2015) The effect of distinct geometric semantic crossover operators in regression problems. In: Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings 18, Springer, pp 3–15

[2] Bakurov I, Vanneschi L, Castelli M, et al (2018) Edda-v2 – an improvement of the evolutionary demes despeciation algorithm. In: Auger A, Fonseca CM, Lourenço N, et al (eds) Parallel Problem Solving from Nature – PPSN XV. Springer International Publishing, Cham, pp 185–196

[3] Bakurov I, Buzzelli M, Castelli M, et al (2021) General purpose optimization library (gpol): A flexible and efficient multi-purpose optimization library in python. Applied Sciences 11(11). https://doi.org/10.3390/app11114774

[4] Bakurov I, Castelli M, Gau O, et al (2021) Genetic programming for stacked generalization. Swarm and Evolutionary Computation 65:100913. https://doi.org/https://doi.org/10.1016/j.swevo.2021.100913

[5] Bakurov I, Castelli M, Fontanella F, et al (2022) A novel binary classification approach based on geometric semantic genetic programming. Swarm and Evolutionary Computation 69:101028. https://doi.org/https://doi.org/10.1016/j.swevo.2021.101028

[6] Beadle L, Johnson CG (2008) Semantically driven crossover in genetic programming. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, pp 111–116

[7] Beadle L, Johnson CG (2009) Semantic analysis of program initialisation in genetic programming. Genetic Programming and Evolvable Machines 10:307–337

[8] Beadle L, Johnson CG (2009) Semantically driven mutation in genetic programming. In: 2009 IEEE Congress on Evolutionary Computation, IEEE, pp 1336–1342

[9] Castelli M, Silva S, Vanneschi L (2015) A c++ framework for geometric semantic genetic programming. Genetic Programming and Evolvable Machines 16(1):73–81

[10] Castelli M, Trujillo L, Vanneschi L, et al (2015) Geometric semantic genetic programming with local search. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. Association for Computing Machinery, New York, NY, USA, GECCO '15, p 999–1006, https://doi.org/10.1145/2739480.2754795

[11] Castelli M, Manzoni L, Gonçalves I, et al (2016) An analysis of geometric semantic crossover: A computational geometry approach. In: International Joint Conference on Computational Intelligence

[12] Chollet F, et al (2015) Keras. https://keras.io

[13] Derrac J, García S, Molina D, et al (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and

swarm intelligence algorithms. Swarm and Evolutionary Computation 1(1):3–18. https://doi.org/10.1016/j.swevo.2011.02.002, URL https://doi.org/10.1016/j.swevo.2011.02.002

[14] Gonçalves I, Silva S, Fonseca CM (2015) On the generalization ability of geometric semantic genetic programming. In: Machado P, Heywood MI, McDermott J, et al (eds) Genetic Programming. Springer International Publishing, Cham, pp 41–52

[15] Gonçalves I, Silva S, Fonseca CM (2015) Semantic learning machine: A feed-forward neural network construction algorithm inspired by geometric semantic genetic programming. In: Pereira F, Machado P, Costa E, et al (eds) Progress in Artificial Intelligence. Springer International Publishing, Cham, pp 280–285

[16] Gonçalves I, Silva S, Fonseca CM, et al (2017) Unsure when to stop? In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, https://doi.org/10.1145/3071178.3071328

[17] Gonçalves I (2017) An exploration of generalization and overfitting in genetic programming: Standard and geometric semantic approaches. Phd thesis, Department of Informatics Engineering, University of Coimbra, Portugal., Coimbra, Portugal, available at https://www.cisuc.uc.pt/download-file/13946/sfxgEyeIRXv2dxxWgZS5

[18] Goodfellow IJ, Bengio Y, Courville A (2016) Deep Learning. MIT Press, Cambridge, MA, USA, http://www.deeplearningbook.org

[19] He K, Zhang X, Ren S, et al (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 770–778, https://doi.org/10.1109/CVPR.2016.90

[20] Huang G, Liu Z, Maaten LVD, et al (2017) Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, Los Alamitos, CA, USA, pp 2261–2269, https://doi.org/10.1109/CVPR.2017.243, URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.243

[21] I. Ortigosa JGR. Lopez (2007) A neural networks approach to residuary resistance of sailing yachts prediction. In: Proceedings of the International Conference on Marine Engineering MARINE, p 250

[22] Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. JMLR.org, ICML'15, p 448–456

[23] Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT press

[24] Koza JR (2010) Human-competitive results produced by genetic programming. Genetic programming and evolvable machines 11:251–284

[25] LeCun YA, Bottou L, Orr GB, et al (2012) Efficient BackProp, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 9–48. https://doi.org/10.1007/978-3-642-35289-8_3, URL https://doi.org/10.1007/978-3-642-35289-8_3

[26] Martins JFBS, Oliveira LOVB, Miranda LF, et al (2018) Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, New York, NY, USA, GECCO '18, pp 1151–1158

[27] McDermott J, Agapitos A, Brabazon A, et al (2014) Geometric semantic genetic programming for financial data. In: Applications of Evolutionary Computation: 17th European Conference, EvoApplications 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17, Springer, pp 215–226

[28] McPhee NF, Ohs B, Hutchison T (2008) Semantic building blocks in genetic programming. In: Genetic Programming: 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008. Proceedings 11, Springer, pp 134–145

[29] Moraglio A, Krawiec K, Johnson C (2012) Geometric semantic genetic programming. In: Coello C, Cutello V, Deb K, et al (eds) Parallel Problem Solving from Nature - PPSN XII, Lecture Notes in Computer Science, vol 7491. Springer Berlin Heidelberg, p 21–31

[30] Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning. Omnipress, Madison, WI, USA, ICML'10, p 807–814

[31] Nicolau M, McDermott J (2020) Genetic programming symbolic regression: What is the prior on the prediction? Genetic Programming Theory and Practice XVII pp 201–225

[32] Oliveira LOV, Otero FE, Pappa GL (2016) A dispersion operator for geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp 773–780

[33] Paszke A, Gross S, Massa F, et al (2019) PyTorch: An Imperative Style, High-Performance Deep Learning Library, Curran Associates Inc., Red Hook, NY, USA

[34] Quinlan JR (1993) Combining instance-based and model-based learning. In: Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993, pp 236–243

[35] Trujillo L, Muñoz Contreras JM, Hernandez DE, et al (2022) Gsgp-cuda — a cuda framework for geometric semantic genetic programming. SoftwareX 18:101085. https://doi.org/https://doi.org/10.1016/j.softx.2022.101085

[36] Tsanas A, Xifara A (2012) Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy and Buildings 49:560–567

[37] Uy NQ, Hoai NX, O'Neill M, et al (2011) Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines 12:91–119

[38] Vanneschi L, Silva S, Castelli M, et al (2014) Geometric semantic genetic programming for real life applications. Genetic programming theory and practice xi pp 191–209

[39] Vanneschi L, Silva S, Castelli M, et al (2014) Geometric Semantic Genetic Programming for Real Life Applications, Springer New York, New York, NY, pp 191–209

[40] Vanneschi L, Bakurov I, Castelli M (2017) An initialization technique for geometric semantic gp based on demes evolution and despeciation. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp 113–120, https://doi.org/10.1109/CEC.2017.7969303

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

[42] Yeh IC (1998) Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete Research 28(12):1797 – 1808