Genetic Programming and Evolvable Machines (2024) 25:17
https://doi.org/10.1007/s10710-024-09488-0

Geometric semantic GP with linear scaling: Darwinian versus Lamarckian evolution

Giorgia Nadizar[1] · Berfin Sakallioglu[2] · Fraser Garrow[3] · Sara Silva[4] · Leonardo Vanneschi[2]

Received: 6 November 2023 / Revised: 29 April 2024 / Accepted: 12 May 2024

Abstract
Geometric Semantic Genetic Programming (GSGP) has shown notable success in symbolic regression with the introduction of Linear Scaling (LS). This achievement stems from the synergy of the geometric semantic genetic operators of GSGP with the scaling of the individuals for computing their fitness, which favours programs with a promising behaviour. However, the initial combination of GSGP and LS (GSGP-LS) underutilised the potential of LS, scaling individuals only for fitness evaluation, neglecting to incorporate improvements into their genetic material. In this paper we propose an advancement, GSGP with Lamarckian LS (GSGP-LLS), wherein we update the individuals in the population with their scaling coefficients in a Lamarckian fashion, i.e., by inheritance of acquired traits. We assess GSGP-LS and GSGP-LLS against standard GSGP for the task of symbolic regression on five hand-tailored benchmarks and six real-life problems. On the former ones, GSGP-LS and GSGP-LLS both consistently improve GSGP, though with no clear global superiority between them. On the real-world problems, instead, GSGP-LLS steadily outperforms GSGP-LS, achieving faster convergence and superior final performance. Notably, even in cases where LS induces overfitting on challenging problems, GSGP-LLS surpasses GSGP-LS, due to its slower and more localised optimisation steps.

Keywords Symbolic regression · Geometric semantic genetic programming · Linear scaling · Lamarckian evolution · Genetic programming

1 Introduction

Genetic Programming (GP) [1] has garnered increased attention in addressing symbolic regression problems [2–4], due to its ability to handle problems characterised by limited or uncertain data, its capacity to evolve models without

Extended author information available on the last page of the article predetermined mathematical forms, and its capability to perform automatic feature selection during learning. Traditionally, GP is employed to address symbolic regression problems using established loss measures, like, for instance, the root mean square error (RMSE), to evaluate fitness. While this approach remains widely adopted, it does have a limitation: certain solutions may receive unfavourable fitness values, despite their potential promise. This occurs, for example, when solutions closely resemble the target function in shape but differ in terms of slope and/or location within the Cartesian space. Linear scaling (LS) was introduced by Keijzer [5] to tackle this issue and improve the performance of GP on symbolic regression. LS modifies the fitness function, rescaling each individual by using their slope and intercept, two constants that can be easily calculated with a cost that is linear in the size of the training set. Since its introduction, the benefit of LS was demonstrated on many theoretical benchmark functions [5] and real-life applications [6–9]. These studies indicate that LS does not only improve standard GP on training data but can also bestow on GP a better generalisation ability, often outperforming standard GP also on unseen data. However, Costelloe and Ryan [10] pointed out that methods that improve training optimisation, including LS, may not always improve GP's generalisation ability as well.

A decade after the inception of LS, Moraglio et al. [11] introduced a novel variant of GP, known as Geometric Semantic GP (GSGP). GSGP differs from traditional GP by implementing specialised genetic operators referred to as Geometric Semantic Operators (GSOs). GSOs, despite acting directly on GP individuals' syntax, have an indirect and known effect on their semantics, notably yielding a unimodal error surface for supervised learning problems [11]. Several references in the literature have shown the success of GSGP, particularly in limiting overfitting, often surpassing standard GP on unseen data, when applied to real-world symbolic regression problems [11–13].

Given that LS entails a redefinition of the fitness and GSGP introduces novel genetic operators, which are typically regarded as separate elements within the GP framework, it seems reasonable to explore an integrated approach that joins these methodologies, with the objective of capitalising on the merits of both GSGP and LS. In accordance with this idea, a unified system merging GSGP and LS, denoted as GSGP-LS, was recently introduced in Nadizar et al. [14]. The outcomes presented in Nadizar et al. [14] exhibit commendable quality but are not without controversy. While GSGP-LS unquestionably outperforms GSGP across the majority of the assessed test problems, affirming the expected advantages of this fusion, it is intriguing that, on some particularly challenging datasets, GSGP-LS tends to overfit training data, yielding inferior performance when compared to GSGP on unseen data. This, upon initial analysis, seems to challenge the belief that LS invariably benefits GP, thereby alerting practitioners to the potential risk of overfitting in specific scenarios.

This paper aims at refining GSGP-LS by introducing a novel method referred to as GSGP with Lamarckian LS (GSGP-LLS). The genesis of GSGP-LLS stems from the observation that the method used to integrate LS in GSGP in Nadizar et al. [14] may not be the most effective. In fact, following the original recommendation from Keijzer [5], in Nadizar et al. [14] individuals were solely rescaled during the fitness evaluation phase, leaving their structures unaltered within the population. Consequently, only the root of individual trees was impacted by linear scaling during the evaluation, while the genetic material deeper within the trees remained unscaled. The approach we adopt here is converse to this: we apply linear scaling by incorporating the scaling directly in the genotype of the individual, essentially following the concept of Lamarckian evolution. This adjustment ensures that when the genetic material of that individual is employed by subsequent descendants in the evolution, it remains scaled. As a result, individuals within the population after several evolutionary iterations undergo more than just the rescaling of their root value; numerous internal subtrees are similarly affected by linear scaling. It is our expectation that this approach will facilitate the evolutionary process, enabling more gradual and localised optimisation steps, possibly contributing to mitigating overfitting, in particular on those challenging datasets where GSGP-LS was outperformed by GSGP. This paper effectively extends [14] by introducing GSGP-LLS into the experimental comparison, where it is assessed alongside GSGP and GSGP-LS across the same test problems.

The remainder of this paper is organised as follows. In Sect. 2 we review previous works relevant to this study, while in Sect. 3 we describe GSGP and LS. Section 4 presents GSGP-LLS, deepening on the difference between Darwinian and Lamarckian evolution. In Sect. 5. we describe our experimental setup, first presenting the used test problems and then discussing the parameter settings. In Sect. 6, we present and comment on the obtained results. Finally, in Sect. 7 we conclude the work and propose ideas for future research.

2 Previous and related work

Although similar ideas to LS had already been proposed for GP before Keijzer's contribution [5], the previous works involving multiple linear regression were considered costly and increased the likeliness of overfitting, since they introduced extra parameters and limitations to the system [15–18]. Conversely, Keijzer's work showed a dramatic improvement in the performance of GP for symbolic regression by applying LS to the error measure [5], at a limited computational cost. In his first contribution, Keijzer demonstrated the benefits of LS on several synthetic test functions. Shortly after, he published another article, where he gave theoretical corroboration to the success of LS [19].

After Keijzer's contribution, LS has been used in several benchmark problems and real-life applications. For instance, Archetti et al. [6], reported using LS with GP to improve the performance on several regression tasks related to the area of drug discovery. A few years later, the same authors also successfully applied LS with GP on another problem from the medical field, consisting in predicting the effect of an anticancer therapy on a specific cohort of patients [20]. In the same year, Raja et al. [7] also combined LS with GP for applications in the telecommunication area and concluded that the system that used LS outperformed the system that did not use it. A general trend has also been to integrate LS in GP systems that also contain other novel methods. For instance, Pennachin et al. [21] used affine arithmetic to improve both the performance and the robustness of GP for symbolic regression, and they also performed LS of outputs before fitness evaluation. The presented results indicate that the proposed system reduces the number of fitness evaluations needed during training and improves generalisation of GP, reducing overfitting. Similarly, Azad and Ryan [22] integrated LS and a method to maintain diversity in a GP system aimed at exploring lifetime learning. A few years later, Virgolin et al. [8] applied LS to a GP-based algorithm, called GP-GOMEA, on a symbolic regression problem from the area of oncology. Later, in another work where several other real-world datasets were employed [23], the same authors confirmed the power of LS, successfully integrating LS in a semantic backpropagation-based GP system. Recently, [9] tackled dynamic target problems by integrating LS with a GP system, using the hinge-loss functions to evolve a set of discriminant functions for multi-class classification. The authors reported on the advantage of the version that uses LS. Later, these results were confirmed and extended, providing an upper bound to the error in dynamic symbolic regression [9, 24] and classification [25].

Even though LS has been applied to GP several times, if we exclude [14], so far in the literature it is possible to find only one contribution in which LS has been integrated with GSGP: in 2015, Vanneschi et al. [26] applied LS to GSGP for tackling an application in the maritime awareness domain. The objective of that work was to predict the position of vessels at sea, based on information related to the vessels' past positions in a specific time interval, using AIS data. The proposed system was compared to two different GP variants and three non-evolutionary machine learning methods, outperforming all of them.

Despite the several successes on real-life applications, Costelloe et al. [10] remarked that several methods that improve GP's training performance, including LS, may not improve GP's generalisation ability as well. This consideration is important since it partially reflects some of the findings that were presented in Nadizar et al. [14].

Lamarckian evolution, a notion discredited in traditional biology, has found renewed interest in the realm of digital evolution. It was explored within the context of evolutionary algorithms by several researchers. For instance, Gruau and Whitley [27] integrated a learning component into the development process of their grammar trees. The grammar trees underwent enhancement through learning during recombination. These enhancements were then incorporated back into the chromosome, reflecting a Lamarckian evolutionary approach. Their objective was to facilitate early learning by enabling the impact of learning on development. Whitley et al. [28] emphasised the potential for utilising Lamarckian strategies to expedite results, recognising their efficiency in accelerating the search process. Ross [29] stated that Lamarckian localised optimisation tends to enhance the fitness of the individuals in a population of Genetic Algorithms (GAs), consequently boosting the search performance. However, he cautioned that if fitness evaluations during localised searches are computationally expensive, the use of Lamarckian evolution could incur high costs. Therefore, the practical application of Lamarckian evolution within GAs should be judiciously weighed against the computational expenses relative to the problem being investigated. Starting from the understanding that individual learning can enhance evolution, Mingo and Aler [30] integrated the Lamarck mechanism in their grammatical evolution system guided by reinforcement. This integration involved substituting the original genotype with information learned by the phenotype. Incorporating local learning into GP, Topchy and Punch [31] enabled the tuned performance of individuals to directly impact the genome, aligning with the Lamarckian principle of evolution. Their modifications aimed at improving final fitness and speed. In their research, La Cava and Spector [32] introduced a GP method that effectively harnessed the benefits of Lamarckian updating, particularly its ability to drive fast convergence. This was achieved by conserving inheritable phenotypic improvements in offspring. Merta and Brandejský [33] conducted a comparative analysis of Lamarckian and Baldwinian approaches to lifetime adaptation in GP for symbolic regression. Their experiments demonstrated that Lamarckian evolution exhibited faster performance than standard GP for symbolic regression of third-degree polynomials, even considering the additional costs and computational time involved.

3 Background

3.1 Geometric semantic genetic programming

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ be the set of input data (also referred to as training instances, observations or fitness cases) of a symbolic regression problem, and $\mathbf{t} = [t_1, t_2, ..., t_n]$ the vector of the respective expected (scalar) output or target values (in other words, for each $i = 1, 2, ..., n$, t_i is the expected output corresponding to input $\mathbf{x}_i$). A GP individual (or program) P can be seen as a function that, for each input vector $\mathbf{x}_i$ returns the scalar value $P(\mathbf{x}_i)$. Following [11], we call *semantics* of P the vector $s_P = [P(\mathbf{x}_1), P(\mathbf{x}_2), ..., P(\mathbf{x}_n)]$. This vector can be represented as a point in an n-dimensional space, that we call *semantic space*. Note that the target vector $\mathbf{t}$ itself is a point in the semantic space. As explained above, GSGP is a variant of GP where the standard crossover and mutation are replaced by Geometric Semantic Operators (GSOs). The objective of GSOs is to define modifications on the syntax of GP individuals that have a precise effect on their semantics. In particular, Geometric Semantic Crossover (GSC) generates one offspring whose semantics stands in the line joining the semantics of the two parents in the semantic space, while Geometric Semantic Mutation (GSM), by mutating an individual i, allows us to obtain another individual j such that the semantics of j stands inside a ball of a given predetermined radius centered in the semantics of i. One of the reasons why GSOs became popular is because GSOs induce a unimodal error surface (on training data) for any supervised learning problem where fitness is calculated using an error measure between outputs and targets. In other words, when using GSOs the error surface on training data is guaranteed to not have any locally optimal solution. This property holds, for instance, for any regression or classification problem, independently of how big and how complex data are. A detailed explanation of the reason why the error surface is unimodal and why this is important can be found in Vanneschi [34]. The definitions of the GSOs are, as given in Moraglio et al. [11], respectively[1]:

Geometric Semantic Crossover (GSC) Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, GSC returns the function $T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random function, i.e., a randomly generated tree, whose output values range in the interval [0, 1].

Geometric Semantic Mutation (GSM) Given a parent function $T : \mathbb{R}^n \rightarrow \mathbb{R}$, GSM with mutation step ms returns the function $T_M = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random functions, i.e., randomly generated trees.

The reason why GSM uses two random trees T_{R1} and T_{R2} is that the amount of modification caused by GSM must be centred in zero. In other words, a random expression is needed that has the same probability of being positive or negative. As pointed out in Moraglio and Mambrini [35], any isotropic Gaussian random function centred in zero can, in principle, be replaced with the term $(T_{R1} - T_{R2})$ in the definition of GSM. Even though this is not in the original definition of GSM, later contributions [13, 34, 36] have clearly shown that limiting the codomain of T_{R1} and T_{R2} in a predefined interval (for instance [0, 1], as it is done for T_R in GSC) helps improve the generalisation ability of GSGP. For this reason, as in several previous works [12, 34], also in this paper we constrain the outputs of T_R, T_{R1}, and T_{R2} by wrapping them in a logistic function. As reported in Refs. [11, 34], GSOs have the drawback of generating larger offspring than the parents, and this entails a rapid growth of the size of the individuals in the population—to as many as 10^8 nodes, in our case. To counteract this problem, in Refs. [12, 37, 38] implementations of GSOs were proposed, that make GSGP not only usable in practice but also significantly faster than standard GP. This is possible through a smart representation of GP individuals that allows us to not store their genotypes during the evolution. The implementation presented in [39] also employs the same idea, and it is the one used here.

3.2 Linear scaling

Linear Scaling (LS) [5] is a method that was introduced to facilitate the task of GP of searching for the best function matching a set of known data. It consists in calculating the slope and intercept of the formula coded by a GP individual. Let $P(\mathbf{x}_i)$ be the output of a GP individual P on the i-th observation of the training set. Using the same notation as in Sect. 3.1, a linear regression on the target values $\mathbf{t}$ can be performed using the equations:

$$b = \frac{\sum_{i=1}^{n} \left[\left(t_i - \bar{t}\right) \left(P(\mathbf{x}_i) - \overline{P}\right) \right]}{\sum_{i=1}^{n} \left(P(\mathbf{x}_i) - \overline{P}\right)^2}, \qquad a = \bar{t} - b\,\overline{P} \tag{1}$$

[1] Only the definitions of the GSOs for symbolic regression problems are given here since they are the only ones used in this work. For the definition of GSOs for other domains, we refer the reader to Moraglio et al. [11].

where n is the number of training observations (fitness cases) and $\overline{P}$ and $\bar{t}$ denote the average output and the average target value, respectively. Values b and a respectively calculate the slope and intercept of the set of outputs $P(\mathbf{x}_i)$, such that the sum of the squared errors between $\mathbf{t}$ and $a + bP$ is minimised. After this, any error measure can be calculated on the scaled formula $a + bP$, for instance the RMSE. If a is different from 0 and b is different from 1, the procedure outlined above is guaranteed to reduce the RMSE for any formula P [5]. The cost of calculating the slope and intercept is linear in the size of the training set. By efficiently calculating the slope and intercept for each individual, the burden of searching for these two constants is removed from the evolution. GP is then free to search for the expression whose shape is most similar to that of the target function.

3.3 Lamarckian evolution

Lamarckian evolution, also known as Lamarckism, is a theory of evolution that was proposed by the French biologist Jean-Baptiste Lamarck in the early 19th century. Lamarck's theory was an attempt to explain how species change over time through a mechanism involving the inheritance of acquired characteristics. Namely, Lamarck proposed that an organism could acquire new traits or characteristics during its lifetime, for instance as a result of its interactions with the environment. These acquired traits were believed to be passed on to the organism's offspring, leading to evolutionary change. It contradicts with Darwinian evolution by emphasising the direct impact of individual experiences on genetic inheritance.

Despite its substantial rejection in the field of Biology, Lamarckian evolution has demonstrated efficacy in artificial evolution applications within the realm of computing. Unlike in natural scenarios, computer programs use simple mapping of phenotypic traits to genotypes, and the reversal of phenotypes back to their corresponding genotypes is frequently manageable [29, 32]. In the context of GP, Lamarckian evolution involves encoding acquired traits directly into the genotype, impacting fitness distribution and genotypic values [31, 40]. The concept of inheriting certain acquired characteristics from one generation to the next, in the context of the integration of GSGP and LS, will be elaborated in Sect. 4.

4 LS and GSGP: Darwinian versus Lamarckian evolution

Building on top of the notions presented in Sect. 3, we hereby describe how we incorporate LS in GSGP. Namely, we consider two settings: one where evolution is performed in a standard Darwinian fashion, as already proposed in Nadizar et al. [14], and one where evolution follows the principles of Lamarckism.

In Darwinian evolution, we perform the standard GSGP evolutionary loop, applying LS for the fitness evaluation only, giving rise to GSGP with LS (GSGP-LS). In this case, the only difference with respect to standard GSGP lies in the fitness evaluation, where we first compute the scaling coefficients on the training set, as described in Sect. 3.2, and then rescale the individual accordingly to compute its error. Clearly, when evaluating on the test set, we do not re-compute the scaling coefficients, but we use the ones computed on the training set. This corresponds to the recommendation of the original formulation of LS by Keijzer, applied on GSGP instead of GP. It is important to note that in this case the coefficients serve only to compute the fitness of an individual, and are discarded thereafter. Hence, these coefficients, which likely yielded an improvement of an individual, do not become available as genetic material.

Conversely, when applying Lamarckian evolution, in what we call GSGP with Lamarckian LS (GSGP-LLS), the scaling coefficients do become part of the individual's genotype, and can therefore be used to generate fitter offspring. In more detail, given a function T, we first compute its scaling coefficients a and b, and then replace the initial individual T with $T_s = a + bT$, which has the coefficients embedded in its representation. Thus, the next generation will not inherit from T, but from T_s, which has acquired some traits, i.e., the scaling coefficients, after its interaction with the environment, i.e., after the evaluation on some data, as Lamarckism suggests. The rationale behind the idea of including the coefficients in the genotype is based on the fact that GSGP is able to induce a unimodal error surface. When we move from T to T_s, we make a step in the semantic space towards a more favourable area of the fitness landscape. Thus, we expect it to be more convenient to start from T_s to generate the offspring, rather than from T, as we could, in principle, get even closer to the target via mutation or crossover of a fitter individual. Nonetheless, there are no guarantees of GSGP-LLS being better than GSGP or GSGP-LS: although the starting points for computing the new population might be better, as it is usual in traditional GSGP, we have no warranty of moving in the right direction, e.g., when performing a mutation. Another point worth mentioning is that GSGP-LLS, differently from GSGP or GSGP-LS, further increases the size of the tree at each generation by appending four nodes to it ($+$, $\times$, a, b). However, this does not hinder the applicability of the method, because we do not constrain the size of the tree and we leverage a GSGP implementation which is efficient regardless of the tree size, thanks to a mechanism which exploits the output of the evolving programs instead of storing their genotype [39].

In conclusion, Table 1 illustrates a brief example highlighting the difference between GSGP-LS and GSGP-LLS. We consider a single individual T in generation 0, and its progress to generation 1, using only the mutation operator. For GSGP-LS, the progression is straightforward: the scaling coefficients are absent because they are used only for calculating the fitness, and they are never stored as part of the individual. For GSGP-LLS, the first column represents the intermediate individual before adding the coefficients, while the second column represents the final individual after adding the coefficients, that is the individual that is actually inserted into the new population.

Table 1 Progression of a single individual from generation 0 to generation 1 in GSGP-LS and GSGP-LLS, using only mutation

Gen	GSGP-LS	GSGP-LLS (intermediate)	GSGP-LLS (final)
0	T	T	$a_0 + b_0 T$
1	$T + ms(T_{R1} - T_{R2})$	$a_0 + b_0 T + ms(T_{R1} - T_{R2})$	$a_1 + b_1(a_0 + b_0 T + ms(T_{R1} - T_{R2}))$
…	…	…	…

For GSGP-LLS, the second column displays the final individual after the integration of the scaling coefficients in the genotype. It is this final individual that is inserted into the new population

5 Experimental setup

We investigate the effectiveness of LS and LLS when combined with GSGP. To do so, we compare the performance of traditional GSGP (simply GSGP from now on), GSGP with LS (GSGP-LS) and GSGP with LLS (GSGP-LLS). As in [14], we conduct the experimental comparison on five hand-tailored symbolic regression benchmarks and six real-life regression datasets. We implemented GSGP-LS and GSGP-LLS using the General Purpose Optimisation Library (GPOL) [39], a publicly available software platform that integrates numerous computational intelligence algorithms, including GSGP. LS and LLS have been integrated in the library on top of the existent GSGP implementation. This section describes the experimental study carried out: in Sect. 5.1 we overview the considered test problems and in Sect. 5.2 we describe the parameter settings.

5.1 Test problems

The five theoretical benchmarks that we have studied were taken directly from the paper that introduced LS [5]. They are:

- $f_5(x) = x^3 \exp^{-x} \cos(x) \sin(x)(\sin^2(x) * \cos(x) - 1)$
- $f_6(x, y, z) = \frac{30xz}{(x-10)y^2}$
- $f_7(x) = \sum_i^x 1/i$
- $f_8(x) = \log x$
- $f_9(x) = \sqrt{x}$

Besides being a good scientific practice to test a method (in this study, LS) on the same case studies that were used when it was introduced, motivations for choosing these benchmarks are the same as in Keijzer [5]. Namely, “many of the problems above mix trigonometry with polynomials, or make the problems in other ways highly non-linear”. Also, it is relevant to point out that, as stated in Keijzer [5], “being of low dimensionality does not make the problems easy”. Exactly as

Table 2 Intervals used as training and test set for the hand-tailored benchmarks used in this work (taken from [5])

Benchmark	Training set	Test set	Note
f_5	$x, z = \text{rnd}(-1, 1)$, $y = \text{rnd}(1, 2)$	$x, z = \text{rnd}(-1, 1)$, $y = \text{rnd}(1, 2)$	Train: 1000 cases, Test: 10000 cases
f_6	$x = [1 : 1 : 50]$	$x = [1 : 1 : 120]$	Extrapolation
f_7	$x = [1 : 1 : 100]$	$x = [1 : 0.1 : 100]$	Interpolation
f_8	$x = [0 : 1 : 100]$	$x = [1 : 0.1 : 100]$	Interpolation
f_9	$x = [0 : 1 : 100]$	$x = [1 : 0.1 : 100]$	Interpolation

Intervals are expressed using the notation *[start:step:stop]*

in Keijzer [5], we have used these benchmarks with the training and test intervals reported in Table 2.

The six real-world regression problems that we have employed, and that have often been used as case studies for GP experiments, are:

- *Boston Housing* [41]: a dataset provided by the Statistical Library and maintained by Carnegie Mellon University. The purpose is to forecast housing prices using data such as air pollution, criminality, pupil-teacher ratio, etc.
- *Concrete Compressive Strength* [42]: a dataset aimed at predicting the strength of concrete depending on the age, mixture and other features of the ingredients.
- *Parkinson Total UPDRS* [43]: composed of a range of biomedical voice measurements and other features of Parkinson's disease patients. The aim is to predict the clinician's Parkinson's disease symptom score on the UPDRS scale.
- *Bioavailability* [6]: consists in predicting the human oral bioavailability of a set of drug compounds, based on a set of molecular descriptors.
- *LD50* [6]: is also a problem in the field of pharmacokinetics. Its purpose is to predict the median lethal dose of a molecular compound, which is one of the most used measures to assess the toxicity of drugs.
- *PPB* [6]: is another dataset from the field of pharmacokinetics. Its aim is to predict the percentage of the initial drug dose which binds plasma proteins.

Table 3 reports the number of instances and attributes for each one of these datasets. Among these datasets, the Bioavailability one was criticised in Dick et al. [44], partially because of a lack of preprocessing, since it includes features that contain no information as well as contradictory relationships between the dependent and independent variables. However, according to many authors who have used this dataset, these characteristics are interesting and should be integrated in a reasonable benchmark suite, because they allow us to test the ability of our algorithms to deal with the difficulties and ambiguities that are typical of real-world data. It is not our objective to discuss what characteristics a good benchmark suite should possess (the interested reader is referred to [45–48] for such a discussion). We simply observe that the Bioavailability dataset, as well as the PPB and LD50 datasets, have

Table 3 Number of instances and attributes of the datasets

Problems	Instances	Attributes
Boston	506	14
Concrete	1030	9
Parkinson	5875	20
Bioavailability	260	247
LD50	234	627
PPB	131	627

been used in several previous GP studies, clearly indicating a trend for overfitting to emerge [12, 13, 36]. We thus use these three datasets as a sort of stress-test suite to assess the generalisation ability of GSGP with LS and LLS, compared to GSGP.

5.2 Parameter settings

The objective of this work is to compare the studied LS variants on the GSGP algorithm. Our focus is not on obtaining the best possible results on the considered test problems. For this reason, instead of optimising the hyperparameters, which would probably lead us to the use of different parameters for each problem, we have preferred to use a relatively standard parameter setting, taken as much as possible from the literature. This enabled us to use the same setting across all the experimental cases. Table 4 reports the employed parameters for each configuration.

We use a population of 100 individuals and run for 500 generations. The populations are initialised with the Ramped Half-and-Half method [1], with maximum initial tree depth of 6, and with no depth limit imposed during the evolution [11]. We employ the same function and terminal sets for each configuration, with the four basic arithmetic operators and no random constants, as in [13, 34, 36]. In the function set, $\div_p$ refers to the protected division function that returns 1.0 if the denominator is less than 0.001. Tournament selection is used with elitism of size 1 (best individual copied unchanged into the next generation). Regarding the genetic operators, we use the GSOs described in Sect. 3.1. The genetic operator probabilities follow the general guidelines for GSGP, without any particular tuning. GSGP uses a logistic wrapper on all random trees, as described in Sect. 3.1. As suggested in [13], the mutation step (*ms* parameter in the definition of GSM in Sect. 3.1) is a random number between 0 and 1, that is generated independently of the previous ones at each mutation event (note that the value of *ms* could also be optimised via gradient descent, as in [49]).

For each of the studied problems, we performed 30 independent runs for each configuration. The execution of each run took approximately one minute, with the exception of the theoretical benchmark f_5 and the Parkinson dataset, which took around 70-80 seconds, given their larger amount of instances. We conducted the experimental evaluation on a Virtual Machine running on VMware ESXi, 7.0.3 with

Table 4 Parameter settings used in the experiments

Parameter	GSGP/GSGP-LS/GSGP-LLS
Generations	500
Population size	100
Initialisation	Ramped
Max. init. depth	6
Max. depth	∞
Function set	$\{+, -, \times, \div_p\}$
Terminal set	{ *features* }
Selection	Tournament
Tournament size	2
Elites	1
Genetic operators	GSOs
Crossover probability	0.3
Mutation probability	0.7
Mutation step	$\mathcal{U}(0, 1]$
Random tree wrapper	Logistic

Ubuntu 22.04.05, 16 VCPU, 64GB RAM, and Nvidia A100 (VGPU with 20GB of VRAM, MIG mode).

Concerning the training-test partitioning, for the theoretical benchmarks we have used the intervals reported in Table 2, which are the same as in [5]. For the real-life problems, at each run, we have selected at random, with uniform probability, 70% of the observations to form the training set, while the remaining 30% were used as test set. This well-known and widely used subsampling method is also referred to as Montecarlo crossvalidation [50, 51]. It is important to point out that, in the same run, all the configurations used the same partitions, and the partitions change (because they are randomly generated each time) from one run to the other. The results reported in the next section are the medians and interquartile range, computed over the performed 30 runs, of the fitness on the training and on the test set of the individual with the best fitness on the training set at each generation.

6 Experimental results

Figure 1 reports the evolution of training and test fitness for GSGP, GSGP-LS, and GSGP-LLS on the theoretical benchmarks.

From these plots, we can observe that both GSGP-LS and GSGP-LLS outperform GSGP on the training set for all the case studies. On the test set, except for function f_6, both GSGP-LS and GSGP-LLS also outperform GSGP. As for function f_6, GPSP-LS slowly reaches the same fitness as GSGP (and would probably outperform it if given more generations) whereas GSGP-LLS outperforms both.

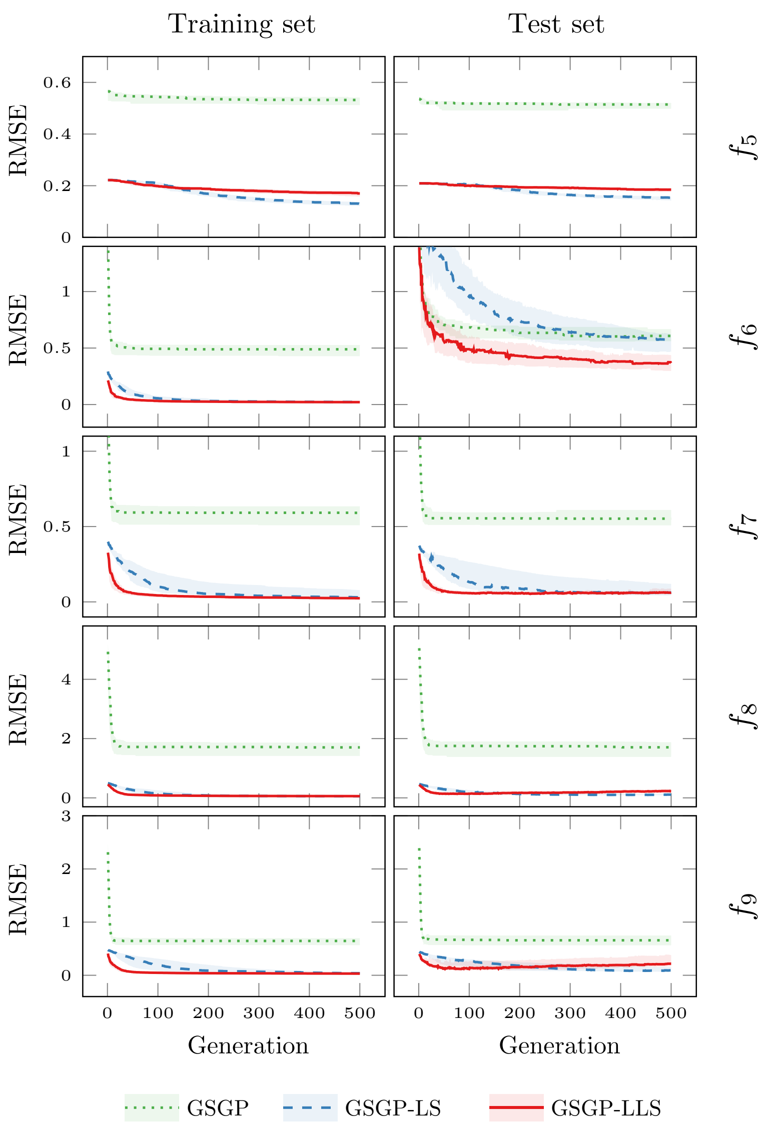

Fig. 1 Comparison between GSGP, GSGP-LS, and GSGP-LLS on Keijzer's theoretical benchmarks. Evolution of median fitness and interquartile range (in 30 independent runs) of the best individual on the training and test sets

On the remaining problems, GSGP-LLS performs the same as GSGP on function f_7, and slightly worse than GSGP on the remaining three functions.

To assess the statistical significance of these results, we performed a pairwise Mann–Whitney U test with significance level $\alpha = 0.05/3$, the /3 being derived from the Bonferroni correction, for both training and test sets, for each problem,

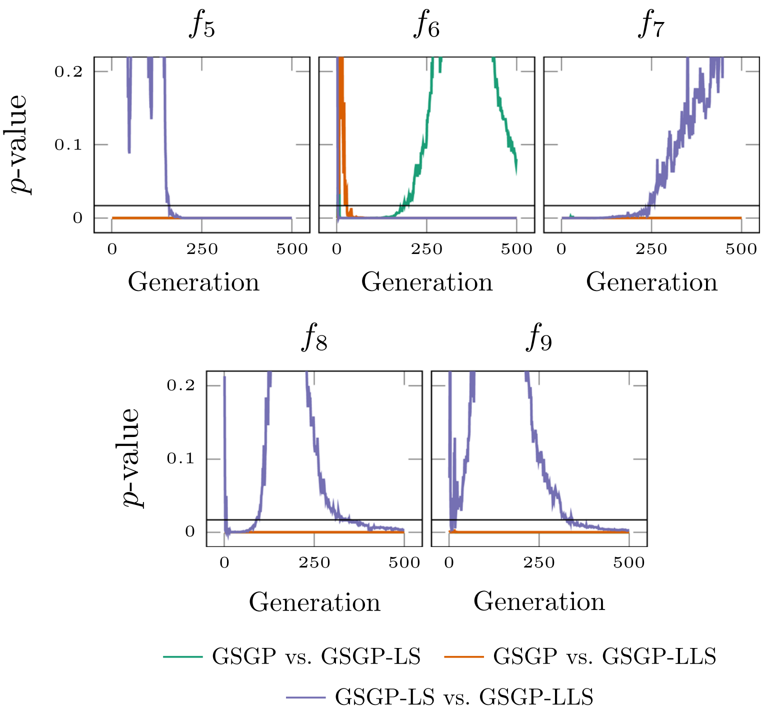

Fig. 2 Evolution of p-values of the Mann–Whitney U test from the pairwise comparisons between the three methods on the test data of Keijzer's benchmarks. Whenever the green line is not visible, it means it is behind the orange line (Color figure online)

at each generation, with the null hypothesis that the distribution of the RMSE of the best individual originated from the 30 runs is the same for the two compared methods (GSGP vs. GSGP-LS, GSGP vs. GSGP-LLS, GSGP-LS vs. GSGP-LLS). The evolution of the p-values on the test set is shown in Fig. 2. We show the significance threshold in each plot (black horizontal line at 0.017) and we clip the y-axis at 0.2 to ease the visual examination of the results. As for the training set, although we do not show the plots, we report that the pairwise differences are always statistically significant, with the exception of those between GSGP-LS and GSGP-LLS for function f_5 when the two lines cross (around generation 150) and for function f_8 after approximately 300 generations.

These plots clearly confirm that GSGP is significantly worse than both GSGP-LS and GSGP-LLS on the test set for all the studied theoretical benchmarks, except for function f_6. All other reported differences are statistically significant.

Figure 3 reports the evolution of training and test fitness for GSGP, GSGP-LS, and GSGP-LLS on the real-life problems.

Again, both GSGP-LS and GSGP-LLS consistently outperform GSGP on the training set for all considered problems. Concerning the results on the test set, instead, we notice that although GSGP-LS outperforms GSGP on three of the considered problems, it suffers from overfitting issues on the Bioavailability, LD50, and PPB datasets, as already pointed out in [14] (in the figure, these three cases are separated from the previous three by means of a horizontal dashed line). As for GSGP-LLS, it outperforms GSGP-LS in all cases, although sometimes by a small margin. It also outperforms GSGP on the three top problems, achieving practically the same results as GSGP on the bottom problems.

Figure 4 reports the evolution of the *p*-values on the test set. Analogously to the case of the theoretical benchmarks, also for the real-life problems we do not show the p-values of the Mann–Whitney U test on the training set. However, we can report that, for all the problems, those *p*-values confirm that the differences (between GSGP and GSGP-LS, GSGP and GSGP-LLS, and GSGP-LS and GSGP-LLS) are statistically significant on the training set, with the only exception of those between GSGP-LS and GSGP-LLS in the early stages of evolution for Bioavailability, LD50, and PPB.

Considering the results obtained on the test set for the real-life problems, at first glance we can clearly divide them into two groups: (1) the first group, including the Boston, Concrete, and Parkinson datasets, for which the GSGP-LS and GSGP-LLS methods are always significantly better than GSGP (*p*-values of the order of 10^{-15}), and (2) the second group, involving the remaining datasets, for which the results are more controversial. For the second group, although the results are less clear, we notice some common trends. When comparing GSGP with GSGP-LS, the *p*-values are initially smaller than the threshold; they grow very fast and then decay, eventually crossing the significance level for Bioavailability and LD50. This, together with the plots of Fig. 3, tells us that there are three distinct phases during the evolution: (1) initially, GSGP-LS significantly outperforms GSGP, (2) then, for some generations, the two methods have comparable performance, and (3) in the end, GSGP-LS becomes (significantly) worse due to overfitting. As for the comparison of GSGP with GSGP-LLS, the behaviour also repeats itself on the three problems: like with GSGP-LS, the *p*-values begin small and then rise, except that with GSGP-LLS they rise slower and remain higher until the end. Looking again at Fig. 3, what is happening is that GSGP-LLS also starts better than GSGP and then the differences vanish. Finally, the *p*-values comparing GSGP-LS with GSGP-LLS, together with the plots of Fig. 3, reveal that GSGP-LLS is significantly better than GSGP-LS on the test set for two of these difficult problems, being the same for the remaining one. However, the fact that, for these three problems, both GSGP-LS and GSGP-LLS exhibit test error curves that increase from the very first generations should also be noticed. We extend our analysis and discussion of this matter later in this section.

Last, it is interesting to notice that, for both training and test sets, the initial RMSE values of both GSGP-LS and GSGP-LLS are already lower than the final RMSE values of GSGP. On the training set, this outcome was expected, given the known benefits of LS on the initial population [5]. In addition, although GSGP plateaus fairly soon during evolution, the performance of both GSGP-LS and GSGP-LLS normally improves steadily across generations on the training set (which sometimes leads to overfitting). This is probably caused by the fact that GSGP has to look for constants *a* and *b* of Eq. (1), while both GSGP-LS and GSGP-LLS already have those constants calculated. This gives GSGP-LS and

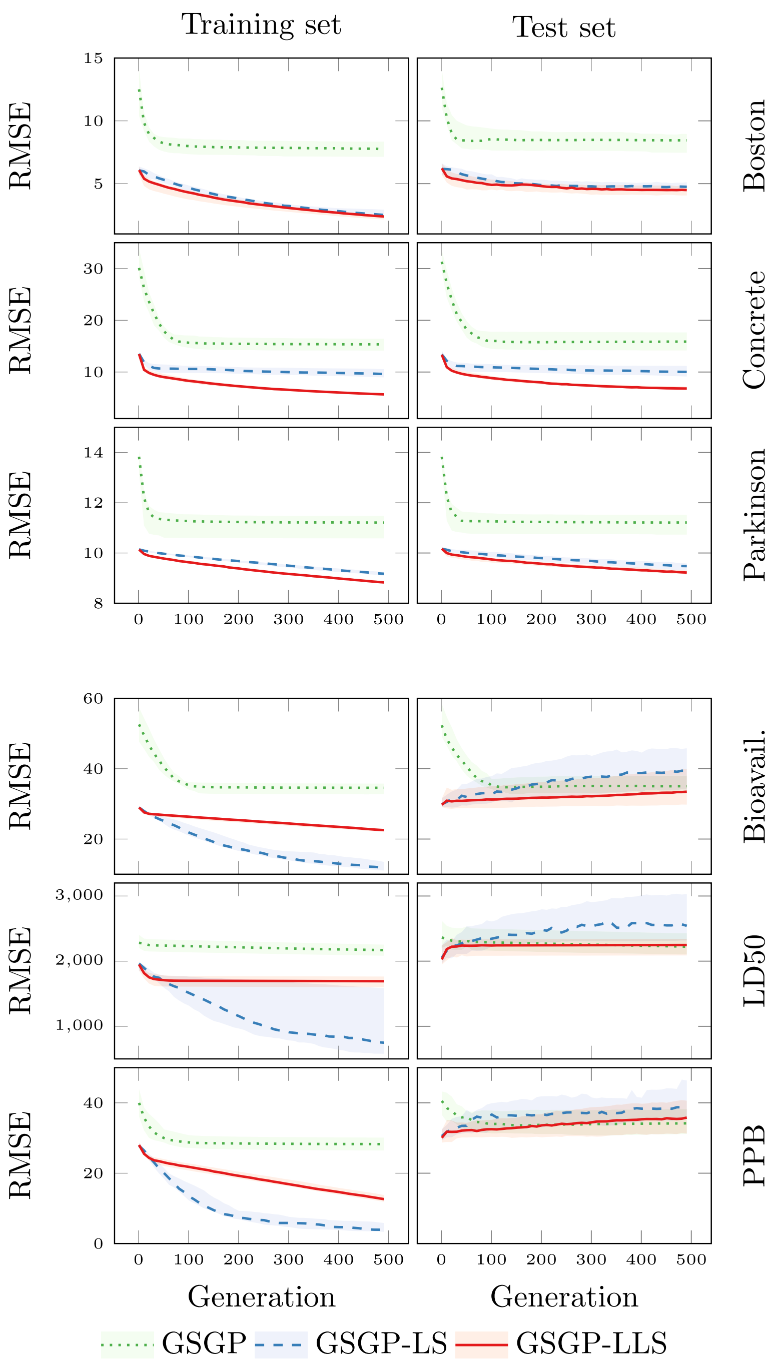

Fig. 3 Comparison between GSGP, GSGP-LS, and GSGP-LLS on the real-life problems. Evolution of median fitness and interquartile range (in 30 independent runs) of the best individual on the training set, evaluated on the training and test sets. Easier problems are on top (above the horizontal dashed line) and harder problems are on the bottom (below the dashed line)

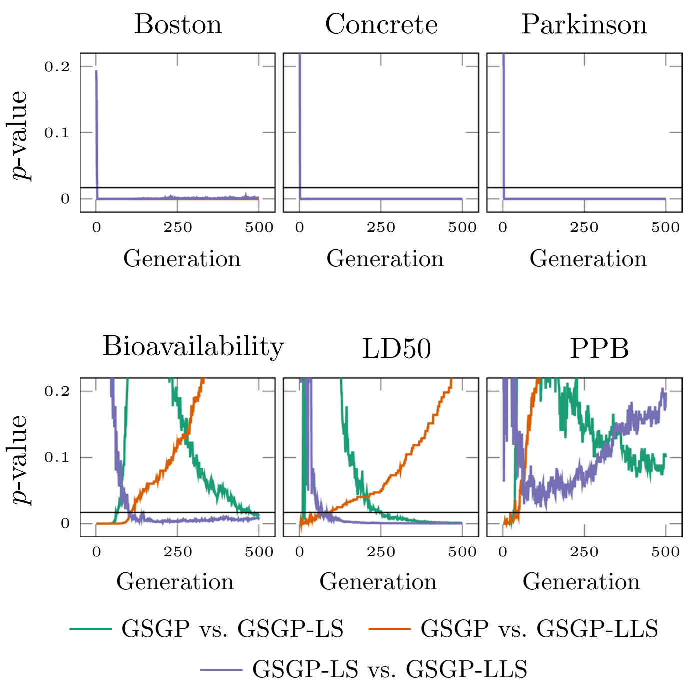

Fig. 4 Evolution of p-values of the Mann–Whitney U test from the pairwise comparisons of the three methods on the test data of the real-life problems, divided by difficulty

GSGP-LLS a greater degree of freedom in the search for a function with optimal shape, which clearly leads to an overall better fit of the training data.

Another observation is that, on the three hardest problems, GSGP-LS fits the training data much easier than GSGP-LLS. We speculate this descends from the fact that LLS acts as a regulariser, preventing individuals from making excessively large steps in the search space, as they are stabilised in their current location by the addition of the scaling coefficients in their genomes. This is closely related to how GSGP-LLS overfits less than GSGP-LS: we elaborate further on this in the following discussion.

Discussion Although the results obtained with GSGP-LS and GSGP-LLS appear generally promising, we have encountered some overfitting issues. Since GSGP has demonstrated its ability to control overfitting [13, 52], it is natural to wonder if the introduction of LS specifically disrupts the benefits of GSGP or if LS is in general more prone to overfitting on some datasets. We already addressed this question in our previous work [14], including an experiment on GP, where we found that LS can induce or worsen overfitting on some problems, regardless of the base evolutionary algorithm.

Interestingly, though, we observe that GSGP-LLS overfits less than GSGP-LS. To explain this phenomenon we studied how the two algorithms explore the semantic space. In fact, previous studies have highlighted that performing larger steps in the semantic space can lead to overfitting [36]. To test if this is the reason behind

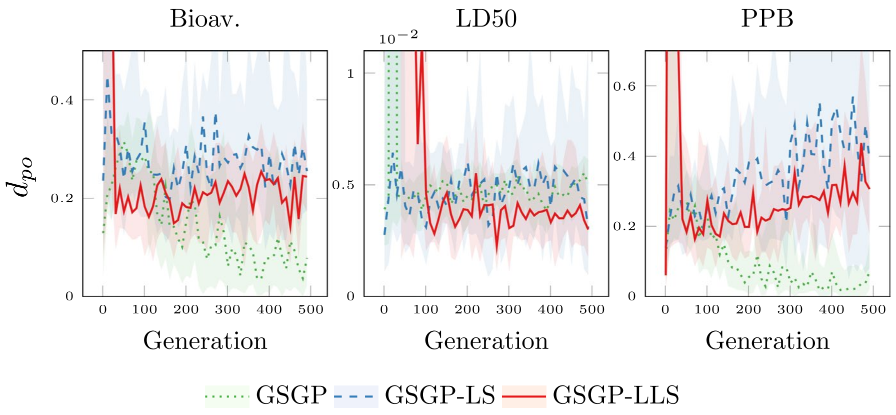

Fig. 5 Distance between parent and (final) offspring for the best individual in the population along all the generations

the larger overfitting of GSGP-LS with respect to GSGP-LLS, we measure the size of the step performed in the semantic space at each generation by the two variants. More in detail, we measure the normalised parent-offspring distance d_{po} defined as the Euclidean distance between the semantics of the parent and the one its offspring, divided by the fitness of the parent (for crossover we consider the distance between the first parent and the offspring). For GSGP-LLS, we compute d_{po} after the offspring has undergone the traits acquisition, i.e., after the inclusion of the scaling coefficients in its genome. In Fig. 5, we show the evolution of d_{po} for the best individual in the population, for the three datasets where we observed overfitting. For two of the problems, Bioavailability and PPB, it is quite obvious the relationship between the distances and the amount of overfitting observed in Fig. 3. The larger the distances, the more overfitting, GSGP-LS being the one with larger distances and higher overfitting, followed by GSGP-LLS and finally GSGP with the smaller distances and less overfitting. Also for the LD50 problem, it is clear that GSGP-LLS exhibits smaller distances than GSGP-LS, justifying why it overfits less than GSGP-LS. As for the distances observed in GSGP, it is not clear where they stand when compared to GSGP-LS. The distance measurements contain a large amount of noise, both in mutation (random mutation step) and crossover (random choice of the first parent), blurring any clear relationship that may be present. Still, and curiously enough, LD50 is precisely the problem where GSGP overfits the most.

Going back to the results of Figs. 3 and 4, we wonder how to tackle the problem of overfitting. Since the test error starts rising from the very beginning of the evolution, it is not possible to rely on the most trivial solution that consists of early stopping. For this reason, in the second part of Sect. 7, we propose other strategies that should be explored in the future to limit overfitting.

7 Conclusions and future work

We investigated the impact of incorporating Linear Scaling (LS) in Geometric Semantic Genetic Programming (GSGP). Our exploration encompassed two distinct approaches: a conventional Darwinian strategy where LS exclusively contributed to the fitness computation (GSGP with LS, GSGP-LS), and a Lamarckian approach (GSGP with Lamarckian LS, GSGP-LLS), where the scaling coefficients are integrated into the individual's genetic material, becoming accessible for the computation of offspring. Our analysis involved an extensive experimental evaluation on the task of symbolic regression for five theoretical benchmarks and six real-life problems of varying complexities. We compared GSGP-LS and GSGP-LLS against standard GSGP, both in terms of efficiency, i.e., how fast evolution is able to achieve the desired goal, and in terms of generalisation, i.e., how well the induced model is able to generalise to unseen data. Our findings indicate a notable enhancement in standard GSGP performance through the incorporation of LS across various scenarios, evident in both training and test sets. Namely, on the theoretical benchmarks, GSGP-LS and GSGP-LLS both neatly outperform GSGP, with a competitive balance between Darwinian and Lamarckian evolution. Conversely, on the real-world problems GSGP-LLS consistently surpasses GSGP-LS, excelling both in convergence speed and generalisation capabilities. However, our observations reveal that the integration with LS renders GSGP more susceptible to overfitting, especially when dealing with challenging data. This trend echoes our earlier findings with standard GP under analogous circumstances [14], suggesting that LS may induce overfitting in particularly difficult cases. Notably, GSGP-LLS exhibits a reduced susceptibility to overfitting, attributed to its more localised search within the semantic space.

In the future, our main objective is to address the overfitting issue. In problems where overfitting does not occur immediately, early termination of the search process could be effective. A stopping condition based on the semantic neighbourhood, as suggested by [53], might be a valuable avenue to explore when the datasets are too small to split into training, test and validation. Beyond early stopping, other methods have recently demonstrated their effectiveness in mitigating overfitting. One conceivable strategy involves dynamic activation and deactivation of LS throughout the evolution. This approach, inspired by a recent contribution on local search within GSGP [54], would involve enabling LS at the onset of the run and subsequently disabling it to curb overfitting. This strategy could enable harnessing the benefits of LS in initial generations, followed by continued evolution using standard GSGP to limit overfitting. Other established methods for controlling overfitting in GSGP and GP include dynamic interleaving of training instances [55] and soft target regularisation [56]. These methods hold promise for enhancing the combination of GSGP with LS and LLS. Furthermore, exploring the potential of explicit feature selection in a pre-processing phase, building upon the implicit feature selection in GP, as exemplified by the approach proposed in [57], is a worthwhile research direction. Last, given the large size of the trees found by GSGP—here as many as 10^8 nodes, with an almost equal distribution among different symbols in all scenarios—it would be noteworthy to explore the application of LS and LLS on SLIM_GSGP [58], a recently introduced non-bloating variant of GSGP.

Acknowledgements We wish to thank the SPECIES Society and Anna Esparcia-Alcázar for organising the SPECIES Summer School 2022, which brought us together and gave us the chance to start this collaboration.

Author contributions GN: conceptualisation, methodology, investigation, software, data curation, visualisation, writing, review. BS: investigation, software, data curation, visualisation, writing. FG: investigation, software, data curation, writing, review. SS: conceptualisation, methodology, writing, review. LV: conceptualisation, methodology, writing, review.

Funding Open access funding provided by Università degli Studi di Trieste within the CRUI-CARE Agreement. 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).

Data availability Data available on request from the authors.

Declarations

Ethical approval Non applicable.

References

1. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
2. D.A. Augusto, H.J.C. Barbosa, Symbolic regression via genetic programming. In: Proceedings. Vol.1. Sixth Brazilian Symposium on Neural Networks, pp. 173–178 (2000). https://doi.org/10.1109/SBRN.2000.889734
3. I. Icke, J.C. Bongard, Improving genetic programming based symbolic regression using deterministic machine learning. In: 2013 IEEE Congress on Evolutionary Computation, pp. 1763–1770 (2013). https://doi.org/10.1109/CEC.2013.6557774
4. M. Nicolau, J. McDermott, Genetic programming symbolic regression: what is the prior on the prediction? In: Banzhaf, W., Goodman, E., Sheneman, L., Trujillo, L., Worzel, B. (eds.) Genetic Programming Theory and Practice XVII, pp. 201–225. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-39958-0_11
5. M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling. In: C. Ryan *et al.* (ed.) Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003. LNCS, vol. 2610, pp. 71–83. Springer, Essex (2003)
6. F. Archetti, S. Lanzeni, E. Messina, L. Vanneschi, Genetic programming for computational pharmacokinetics in drug discovery and development. Genet. Program Evolvable Mach. **8**(4), 413–432 (2007)

7. A. Raja, R.M.A Azad, C. Flanagan, C. Ryan, Real-time, non-intrusive evaluation of voip. EuroGP'07, pp. 217–228. Springer, Berlin, Heidelberg (2007)
8. M. Virgolin, T. Alderliesten, A. Bel, C. Witteveen, P.A.N. Bosman, Symbolic regression and feature construction with gp-gomea applied to radiotherapy dose reconstruction of childhood cancer survivors. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '18, pp. 1395–1402. Association for Computing Machinery, New York, NY, USA (2018).https://doi.org/10.1145/3205455.3205604
9. S. Ruberto, V. Terragni, J.H. Moore, Sgp-dt: towards effective symbolic regression with a semantic gp approach based on dynamic targets. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. GECCO '20, pp. 25–26. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3377929.3397486
10. D. Costelloe, C. Ryan, On improving generalisation in genetic programming, in *Genetic Programming*. ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner (Springer, Berlin, 2009), pp.61–72
11. A. Moraglio, K. Krawiec, C. Johnson, Geometric semantic genetic programming. In: Coello, C.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) Parallel Problem Solving from Nature - PPSN XII. Lecture Notes in Computer Science, vol. **7491**, pp. 21–31. Springer (2012)
12. M. Castelli, S. Silva, L. Vanneschi, A c++ framework for geometric semantic genetic programming. Genet. Program Evolvable Mach. **16**(1), 73–81 (2015)
13. L. Vanneschi, S. Silva, M. Castelli, L. Manzoni, Geometric semantic genetic programming for real life applications. In: Riolo, R., Moore, J.H., Kotanchek, M. (eds.) Genetic Programming Theory and Practice XI, pp. 191–209. Springer, New York, NY (2014)
14. G. Nadizar, F. Garrow, B. Sakallioglu, L. Canonne, S. Silva, L. Vanneschi, An investigation of geometric semantic gp with linear scaling. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '23, pp. 1165–1174. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3583131.3590418
15. H. Iba, H. Garis, T. Sato, *Genetic Programming Using a Minimum Description Length Principle* (MIT Press, Cambridge, 1994)
16. H. Iba, N. Nikolaev, Genetic programming polynomial models of financial data series. In: Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512), vol. 2, pp. 1459–14662 (2000). https://doi.org/10.1109/CEC.2000.870826
17. N.Y. Nikolaev, H. Iba, Regularization approach to inductive genetic programming. IEEE Trans. Evol. Comput. **5**(4), 359–375 (2001). https://doi.org/10.1109/4235.942530
18. H.G. Hiden, M.J. Willis, M.T. Tham, P. Turner, G.A. Montague, Non-linear principal components analysis using genetic programming. In: Second International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications, pp. 302–307 (1997). https://doi.org/10.1049/cp:19971197
19. M. Keijzer, Scaled symbolic regression. Genet. Program Evolvable Mach. **5**(3), 259–269 (2004). https://doi.org/10.1023/B:GENP.0000030195.77571.f9
20. F. Archetti, I. Giordani, L. Vanneschi, Genetic programming for anticancer therapeutic response prediction using the nci-60 dataset. Comput. Oper. Res. **37**, 1395–1405 (2010). https://doi.org/10.1016/j.cor.2009.02.015
21. C. Pennachin, M. Looks, J.A. Vasconcelos, Robust symbolic regression with affine arithmetic. In: Genetic and Evolutionary Computation Conference (GECCO) (2010)
22. R.M.A. Azad, C. Ryan, A simple approach to lifetime learning in genetic programming-based symbolic regression. Evol. Comput. **22**(2), 287–317 (2014)
23. M. Virgolin, T. Alderliesten, P.A.N. Bosman, Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '19, pp. 1084–1092. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3321707.3321758
24. S. Ruberto, V. Terragni, J. Moore, A semantic genetic programming framework based on dynamic targets. Genet. Programm. Evolv. Mach. **22**, 1–31 (2021). https://doi.org/10.1007/s10710-021-09419-3
25. S. Ruberto, V. Terragni, J.H. Moore, Towards effective gp multi-class classification based on dynamic targets. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '21, pp. 812–821. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3449639.3459324

26. L. Vanneschi, M. Castelli, E. Costa, A. Re, H. Vaz, V. Lobo, P. Urbano, Improving maritime awareness with semantic genetic programming and linear scaling: prediction of vessels position based on ais data, in *Applications of Evolutionary Computation*. ed. by A.M. Mora, G. Squillero (Springer, Cham, 2015), pp.732–744
27. F. Gruau, D. Whitley, Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evol. Comput. **1**(3), 213–233 (1993)
28. D. Whitley, V.S. Gordon, K. Mathias, Lamarckian evolution, the baldwin effect and function optimization. In: Parallel Problem Solving from Nature-PPSN III: International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel, October 9–14, 1994 Proceedings 3, pp. 5–15 (1994). Springer
29. B.J. Ross, A lamarckian evolution strategy for genetic algorithms. Pract. Handb. Genet. Algorithms Complex Coding Syst. **3**, 1–16 (1999)
30. J.M. Mingo, R. Aler, Grammatical evolution guided by reinforcement. In: 2007 IEEE Congress on Evolutionary Computation, pp. 1475–1482 (2007). IEEE
31. A. Topchy, W.F. Punch, et al. Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), vol. 155162 (2001). Morgan Kaufmann San Francisco, CA
32. W. La Cava, L. Spector, Inheritable epigenetics in genetic programming. In: Riolo, R., Worzel, W.P., Kotanchek, M. (eds.) Genetic Programming Theory and Practice XII, pp. 37–51. Springer, Cham (2015)
33. J. Merta, T. Brandejskỳ, Lifetime adaptation in genetic programming for the symbolic regression. In: Computational Statistics and Mathematical Modeling Methods in Intelligent Systems: Proceedings of 3rd Computational Methods in Systems and Software 2019, Vol. 2 3, pp. 339–346 (2019). Springer
34. L. Vanneschi, In: Schütze, O., Trujillo, L., Legrand, P., Maldonado, Y. (eds.) An Introduction to Geometric Semantic Genetic Programming, pp. 3–42. Springer, Cham (2017)
35. A. Moraglio, A. Mambrini, Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. GECCO '13, pp. 989–996. Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2463372.2463492
36. I. Gonçalves, S. Silva, C.M. Fonseca, On the generalization ability of geometric semantic genetic programming, in *Genetic Programming*. ed. by P. Machado, M.I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, K. Sim (Springer, Cham, 2015), pp.41–52
37. A. Moraglio, An efficient implementation of GSGP using higher-order functions and memoization. In: Semantic Methods in Genetic Programming, Workshop at Parallel Problem Solving from Nature (2014)
38. J.F.B.S. Martins, L.O.V.B. Oliveira, L.F. Miranda, F. Casadei, G.L. Pappa, Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '18, pp. 1151–1158. ACM, New York, NY, USA (2018)
39. I. Bakurov, M. Buzzelli, M. Castelli, L. Vanneschi, R. Schettini, General purpose optimization library (gpol): a flexible and efficient multi-purpose optimization library in python. Appl. Sci. (2021). https://doi.org/10.3390/app11114774
40. M. Kommenda, B. Burlacu, G. Kronberger, M. Affenzeller, Parameter identification for symbolic regression using nonlinear least squares. Genet. Program Evolvable Mach. **21**(3), 471–501 (2020)
41. D. Harrison, D.L. Rubinfeld, Hedonic housing prices and the demand for clean air. J. Environ. Econ. Manag. **5**(1), 81–102 (1978). https://doi.org/10.1016/0095-0696(78)90006-2
42. I.-C. Yeh, Modeling of strength of high-performance concrete using artificial neural networks. Cem. Concr. Res. **28**(12), 1797–1808 (1998). https://doi.org/10.1016/S0008-8846(98)00165-3
43. M.A. Little, P.E. McSharry, S.J. Roberts, D.A. Costello, I.M. Moroz, Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. Biomed. Eng. **6**(1), 23 (2007). https://doi.org/10.1186/1475-925X-6-23
44. G. Dick, A.P. Rimoni, P.A. Whigham, A re-examination of the use of genetic programming on the oral bioavailability problem. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 1015–1022 (2015)
45. J. McDermott, D.R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, U.-M. O'Reilly, Genetic programming needs better benchmarks.

In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. GECCO ’12, pp. 791–798. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2330163.2330273
46. J. Woodward, S. Martin, J. Swan, Benchmarks that matter for genetic programming. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation. GECCO Comp ’14, pp. 1397–1404. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2598394.2609875
47. M. Nicolau, A. Agapitos, M. O’Neill, A. Brabazon, Guidelines for defining benchmark problems in genetic programming. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 1152–1159 (2015). https://doi.org/10.1109/CEC.2015.7257019
48. J. McDermott, G. Kronberger, P. Orzechowski, L. Vanneschi, L. Manzoni, R. Kalkreuth, M. Castelli, Genetic programming benchmarks: looking back and looking forward. ACM SIGEVOlution (2022). https://doi.org/10.1145/3578482.3578483
49. G. Pietropolli, L. Manzoni, A. Paoletti, M. Castelli, Combining geometric semantic gp with gradient-descent optimization. In: European Conference on Genetic Programming (Part of EvoStar), pp. 19–33 (2022). Springer
50. L. Vanneschi, S. Silva, *Lectures on Intelligent Systems* (Springer, Berlin, 2023)
51. W. Dubitzky, M. Granzow, D.P. Berrar, *Fundamentals of Data Mining in Genomics and Proteomics* (Springer, Cham, 2006)
52. L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, A new implementation of geometric semantic gp applied to predicting pharmacokinetic parameters. In: Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3–5, 2013. Proceedings, vol. 7831, pp. 205–216 (2013). Springer Berlin, Germany
53. I. Gonçalves, S. Silva, C.M. Fonseca, M. Castelli, Unsure when to stop? ask your semantic neighbors. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 929–936 (2017)
54. M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, P. Legrand, Geometric semantic genetic programming with local search. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO ’15, pp. 999–1006. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2739480.2754795
55. I. Gonçalves, S. Silva, Balancing learning and overfitting in genetic programming with interleaved sampling of training data, in *Genetic Programming*. ed. by K. Krawiec, A. Moraglio, T. Hu, A.Ş. Etaner-Uyar, B. Hu (Springer, Berlin, 2013), pp.73–84
56. L. Vanneschi, M. Castelli, Soft target and functional complexity reduction: a hybrid regularization method for genetic programming. Expert Syst. Appl. **177**, 114929 (2021). https://doi.org/10.1016/j.eswa.2021.114929
57. N.M. Rodrigues, J.E. Batista, W. La Cava, L. Vanneschi, S. Silva, Slug: Feature selection using genetic algorithms and genetic programming, in *Genetic Programming*. ed. by E. Medvet, G. Pappa, B. Xue (Springer, Cham, 2022), pp.68–84
58. L. Vanneschi, SLIM_GSGP: The non-bloating geometric semantic genetic programming. In: European Conference on Genetic Programming (Part of EvoStar), pp. 125–141 (2024). Springer

Authors and Affiliations

Giorgia Nadizar[1] · Berfin Sakallioglu[2] · Fraser Garrow[3] · Sara Silva[4] · Leonardo Vanneschi[2]

✉ Giorgia Nadizar
giorgia.nadizar@phd.units.it

Berfin Sakallioglu
bsakallioglu@novaims.unl.pt

Fraser Garrow
fg28@hw.ac.uk

Sara Silva
sara@fc.ul.pt

Leonardo Vanneschi
lvanneschi@novaims.unl.pt

[1] Department of Mathematics, Informatics, and Geosciences, University of Trieste, Trieste, Italy

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

[3] School of Informatics, University of Edinburgh, Edinburgh, UK

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