Introducing Crossover in SLIM-GSGP

Gloria Pietropolli[1](✉), Davide Farinati[2](✉), Luca Manzoni[1], Mauro Castelli[2], Sara Silva[3], and Leonardo Vanneschi[1]

[1] Department of Mathematics, Informatics, and Geosciences, University of Trieste, Trieste, Italy
`{gloria.pietropolli,lmanzoni}@units.it, lvanneschi@novaims.unl.pt`

[2] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Lisboa, Portugal
`{dfarinati,mcastelli}@novaims.unl.pt`

[3] LASIGE, Faculty of Sciences, University of Lisbon, Lisbon, Portugal
`sgsilva@fc.ul.pt`

Abstract. The Semantic Learning algorithm based on Inflate and deflate Mutations (SLIM-GSGP, or simply SLIM) is a variant of Geometric Semantic Genetic Programming (GSGP) designed to generate compact and interpretable models while maintaining the beneficial characteristic of GSGP of inducing an error surface without local optima. To date, no crossover operator has been defined for SLIM and the existing SLIM framework relies solely on two mutation operators: inflate and deflate mutation. This paper introduces two novel crossover operators for SLIM: Swap Crossover (XOSw) and Donor Crossover (XODn). These crossovers capitalize on SLIM's linked-list representation to facilitate genetic exchange while controlling program size. Experimental results on five symbolic regression problems demonstrate that the new crossover operators often improve fitness and reduce model size when compared to standard SLIM and to GSGP. Our findings establish these operators as solid improvements of traditional GSGP crossover.

Keywords: SLIM · Geometric Semantic Genetic Programming · Geometric Semantic Crossover · Inflate Mutation · Deflate Mutation

1 Introduction

In traditional Genetic Programming (GP) [17], genetic operators act on the syntax of the individuals without any knowledge of how they affect semantics, often leading to drastic and unpredictable effects during the evolution [24]. To

G. Pietropolli and D. Farinati—These authors contributed equally to this work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-89991-1_7.

address this problem, some researchers have focused on studying semantic-aware operators [32,39], both acting directly [20,26,27] and indirectly [3,4,7,18,29].

Notably, Geometric Semantic Genetic Programming (GSGP), introduced in [26], is a GP variant where traditional crossover and mutation are replaced by Geometric Semantic Crossover (GSC) and Geometric Semantic Mutation (GSM), collectively known as Geometric Semantic Operators (GSOs). GSOs, although acting on the syntax of the programs, have known effects on their semantics. The main strength of GSOs lies in their ability to induce a unimodal fitness landscape on any supervised learning problem, which facilitates the search for good solutions, compared to standard GP [9]. Since its inception, various improvements have been made to GSGP [5,13,28,33–35,38], leading to several achievements across a range of applications [10,21,42]. Most improvements have focused on refining GSOs, with a particular emphasis on GSC [2,12,16,37]. However, a remaining limitation of GSGP is that, by design, GSOs always generate offspring larger than their parents, causing program sizes to grow rapidly. To avoid building and storing massive programs, which would make GSGP impractical for real-world applications, implementations were introduced [8,25,38] that store only the semantic vectors of newly created individuals, along with all individuals of the initial population and the random programs generated across generations. While this solution made GSGP fast enough to be used in practice, it is a workaround to the problem, since it does not limit code growth. Although other efforts have been made to solve this problem [22], no major breakthroughs were achieved. Moreover, despite its theoretical advantages, GSC in GSGP shows limited effectiveness, as it cannot generate individuals outside the convex hull of the current population.

Recently, the Semantic Learning algorithm based on Inflate and deflate Mutation (SLIM), a new variant of GSGP [36,40] (also referred to as SLIM-GSGP), was shown to preserve the ability to induce a unimodal error surface while evolving much smaller models than GSGP and standard GP. SLIM evolves individuals that can be represented as linked lists of expressions and uses two different mutation operators: Inflate Geometric Semantic Mutation (IGSM), analogous to the GSM of GSGP, which generates larger offspring by appending one or more items at the end of the list, and Deflate Geometric Semantic Mutation (DGSM), which produces smaller offspring by deleting one of these items from the list.

To date, no crossover has been defined for SLIM. This is not surprising, since mutation is a more effective operator than crossover for traditional GSGP [9,27]. However, improving the search ability of SLIM with a semantic-aware crossover is an open possibility. Leveraging the SLIM linked-list implementation, we can define crossovers that address both limitations of GSC, enabling offspring to extend beyond the convex hull of the population and avoiding unnecessary increases in offspring size. To answer this call, we introduce two different crossover operators for SLIM, providing their mathematical formulations and integrating them in different variants of SLIM, with and without using the deflate mutation. We study their effects on fitness and size of the resulting models on a set of real-world problems.

2 Semantic Learning Algorithm Based on Inflate and Deflate Mutation

This section describes how SLIM-GSGP differs from traditional GSGP. More specifically, after a brief introduction to GSGP (Sect. 2.1), it presents the two mutation operators, IGSM and DGSM, it introduces the different studied SLIM variants and it discusses the linked-list representation of the individuals (Sect. 2.2).

2.1 Geometric Semantic Genetic Programming

In GP, the term *semantics* refers to the vector of the output values of a program on a set of observations. GSGP [26] is a variant of GP that replaces traditional syntax-based genetic operators with GSOs. These operators have a known effect on the semantics of the individuals, inducing geometric properties in the semantic space. Specifically, given two parent functions $\mathcal{T}_1, \mathcal{T}_2 : \mathbb{R}^n \to \mathbb{R}$, the GSC produces an offspring function defined as: $\mathcal{T}_{\text{XO}} = \mathcal{T}_1 \cdot T_R + (1 - T_R) \cdot \mathcal{T}_2$, where T_R is a random real function with outputs in $[0, 1]$. Similarly, the GSM generates a mutated function from a parent $\mathcal{T} : \mathbb{R}^n \to \mathbb{R}$ as: $\mathcal{T}_M = \mathcal{T} + ms \cdot (T_{R1} - T_{R2})$ where T_{R1} and T_{R2} are random real functions with outputs in $[0, 1]$ and ms is a parameter called *mutation step*. GSC corresponds to a geometric crossover in the semantic space. In other words, it produces offspring whose semantics lie on the segment connecting the semantics of the two parents. As a consequence, the offspring cannot be worse than the worst of its parents on the training set. GSM, on the other hand, corresponds to a ball mutation in the semantic space, generating offspring within a sphere of radius ms centered in the semantics of the parent. One of the principal advantages of GSGP is that these operators induce a unimodal error surface, simplifying the search process by eliminating local optima. The interested reader is referred to [41] for a detailed explanation of the reason why GSOs induce a unimodal error surface. However, these operators also have a significant limitation: by design, they always produce offspring larger than their parents, leading to a fast growth in the size of the individuals during the evolution.

2.2 SLIM-GSGP

SLIM-GSGP [36,40] is an extension of GSGP that, while maintaining the property of inducing a unimodal error surface, generates smaller models. As the name suggests, it incorporates two types of mutation: the IGSM which, as traditional GSM, generates an offspring larger than its parent; and DGSM, which produces offspring that are smaller than their parents.

The IGSM shares the same definition of GSM. The definition and validity of DGSM are based on two key observations: first, GSM can be rewritten as $\text{GSM}(T) = T + ms \cdot (T_{R1} - T_{R2}) = T - ms \cdot (T_{R2} - T_{R1})$; second, the random trees T_{R1} and T_{R2} are interchangeable, as they are independently sampled from the same distribution. Thus, the GSM can be written as $\text{GSM}(T) = T - ms \cdot (T_{R1} - T_{R2})$. To provide some intuition of why this reasoning can be used to generate offspring that are smaller than their parent, consider for instance an individual T to which GSM is applied, say, three consecutive times. The result of this operation is:

$$T_M = \mathrm{GSM}^3(T) = T + ms \cdot (T_{R1} - T_{R2}) + ms \cdot (T_{R3} - T_{R4}) + ms \cdot (T_{R5} - T_{R6})$$

If we now apply again GSM, but this time using subtraction instead of addition, the generated individual is:

$$T_{M'} = T + ms \cdot (T_{R1} - T_{R2}) + ms \cdot (T_{R3} - T_{R4}) + ms \cdot (T_{R5} - T_{R6}) - ms \cdot (T_{R7} - T_{R8})$$

Before proceeding, it is important to recall that a widely adopted approach in the literature is to reuse random trees instead of generating new ones [38]. For example, we could use T_{R3} and T_{R4} instead of T_{R7} and T_{R8}, which would result in the following individual:

$$T' = T + ms \cdot (T_{R1} - T_{R2}) \cancel{+ms \cdot (T_{R3} - T_{R4})} + ms \cdot (T_{R5} - T_{R6}) \cancel{-ms \cdot (T_{R3} - T_{R4})}$$

After this simplification, the individual has a smaller genotype. Using this idea, DGSM simply works by removing one of the terms that IGSM had previosuly added to an individual, effectively performing a geometric semantic mutation. Both IGSM and DGSM perturb the semantics of the individual within the same range, specifically $[-ms, ms]$, where ms is the mutation step parameter. This allows us to reduce the size of the individuals while maintaining the semantic properties of the search.

The IGSM and DGSM are the only operators defined so far within the SLIM framework. To date, no crossover operator has been introduced for SLIM.

Different SLIM Variants. In [40], the authors considered three different methods (SIG2, ABS, SIG1) for defining a function that returns values within the range $[-ms, ms]$, and two different approaches for perturbing an individual to achieve ball mutation in the semantic space.

We provide the definition for two of the three functions that return values within $[-ms, ms]$, specifically those used in this paper, as follows:

1. Using two random trees T_{R1} and T_{R2}, each passed through the sigmoid function S (which maps outputs to $[0, 1]$) as follows:

   $$\mathrm{SIG2} = ms \cdot (S(T_{R1}) - S(T_{R2})),$$

   which is the same as traditional GSM.
2. Using a single random tree T_R passed through the sigmoid function S:

   $$\mathrm{SIG1} = ms \cdot (2 \cdot S(T_R) - 1).$$

On the other hand, a small perturbation of a numeric value can be achieved either by adding a small positive or negative quantity close to zero, or by multiplying by a factor slightly smaller or larger than one. Accordingly, there are two methods for perturbing an individual within the SLIM framework:

1. IGSM adds the function, while DGSM subtracts it.
2. IGSM multiplies by $(1 + \text{function})$, while DGSM divides by it.

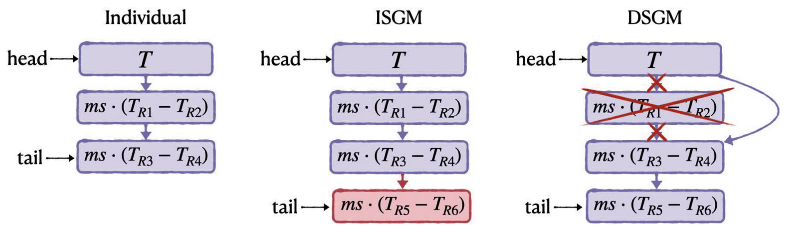
Individual
head
T
$ms \cdot (T_{R1} - T_{R2})$
tail
$ms \cdot (T_{R3} - T_{R4})$
ISGM
head
T
$ms \cdot (T_{R1} - T_{R2})$
$ms \cdot (T_{R3} - T_{R4})$
tail
$ms \cdot (T_{R5} - T_{R6})$
DSGM
head
T
$ms \cdot (T_{R1} - T_{R2})$
$ms \cdot (T_{R3} - T_{R4})$
tail
$ms \cdot (T_{R5} - T_{R6})$

Fig. 1. An example of a SLIM+SIG2 individual, with a visual representation of the effect of the IGSM and DGSM on its genotype.

By combining these two methods of perturbing an individual with the three functions defined for mutation, we obtain six SLIM variants. These are denoted as SLIM+ or SLIM∗, depending on whether they use addition or multiplication, followed by SIG2, ABS, or SIG1, based on the function applied. For instance, the variant called SLIM+SIG2 uses the sum of a small positive expression close to zero to perform ball mutation and the SIG2 function to define a perturbation within $[-ms, ms]$.

Linked-List Implementation of SLIM. In SLIM, it is useful to represent the individuals as linked lists, where each item contains a subprogram. In this way, an IGSM can be implemented by appending one or more items at the end of the list, while DGSM can be implemented by deleting one of these items from the list. To evaluate an individual, it is sufficient to accumulate either the sum (for the SLIM+ variants) or the product (for the SLIM* variants) of the terms of the list. A visual representation of the linked-list implementation and the application of the IGSM and DGSM is shown in Fig. 1 for the specific case of the SLIM+SIG2 variant.

Thus, using a simple notation, a SLIM individual can be represented as:

$$\mathcal{T}_j = (T_j, \Delta T_1^{(j)}, \ldots, \Delta T_n^{(j)}),$$

where T_j is the head of the list (the initial individual before any mutation), n is the number of elements that belong to the linked list, excluding the head, and each $\Delta T_i^{(j)}$ denotes the i-th block belonging to individual j. To minimize memory occupation, these blocks can consist exclusively of the single tree used for mutation in the ABS and SIG1 cases, or they can represent the difference between two random trees in the SIG2 case.

3 Crossover in SLIM

In SLIM, the linked-list representation not only facilitates the implementation of the deflate operator but also allows for new approaches to recombine genetic

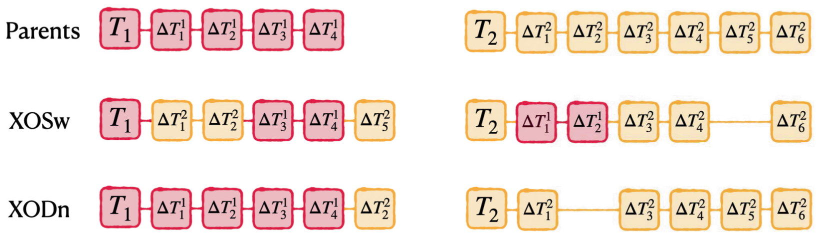

Fig. 2. An intuitive representation of the XOSw and XODn crossovers.

material within the population. Standard GSGP faces well-recognized limitations: (1) Even more than GSM, GSC generates offspring larger than the parents, leading to exponential growth in population size [26], and (2) GSC shows restricted search ability as, despite its theoretical advantages, it cannot generate new individuals outside the convex hull of the current population [9,27]. To address these issues, we propose leveraging the linked-list structure to define new crossover methods aimed at overcoming both limitations of GSC. For (1), we will design these crossover methods to control individual size, avoiding the rapid growth characteristic of GSC. For (2), we will move beyond standard geometricity, meaning these crossovers will not generate offspring to the segment between the parents, as GSC. However we can still define crossover operators with a clear geometrical meaning.

Firstly, we define Swap Crossover (XOSw), which swaps items of the linked list between parents, allowing recombination of genetic material while keeping offspring the same size as the parents. Next, we introduce Donor Geometric Crossover (XODn), where one parent (donor) donates a block to the other parent (receiver). This crossover mimics the SLIM mutation operators, essentially performing IGSM on the receiver (which gains a block) and DGSM on the donor (which loses a block). Unlike mutation, however, this operator uses genetic material already present in the population rather than a random block. A visual intuition of these crossovers is provided in Fig. 2.

As discussed in Sect. 2, SLIM can be implemented in two ways: additive (the SLIM+ variants) and multiplicative (the SLIM* variants). While the crossovers we introduce modify blocks in the same way in both forms, the subsequent evaluation of the offspring differs. Thus, we provide two variants for each crossover: one for the additive case (denoted by **A**) and one for the multiplicative case (denoted by **M**).

***Swap Crossover* (XOSw):** Given two parent functions
$\mathcal{T}_1 = (T_1, \Delta T_1^{(1)}, \ldots, \Delta T_{n_1}^{(1)})$ and $\mathcal{T}_2 = (T_2, \Delta T_1^{(2)}, \ldots, \Delta T_{n_2}^{(2)})$, where $n_1 < n_2$

without loss of generality, XOSw produces two offspring functions:

A. $$\mathcal{T}_{\text{XOSw}_1} = \text{sel}(T_1, T_2) + \textstyle\sum_{i=1}^{n_1} \text{sel}(\Delta T_i^{(1)}, \Delta T_i^{(2)}) + \sum_{i=n_1+1}^{n_2} \text{sel}(\varnothing, \Delta T_i^{(2)}),$$
$$\mathcal{T}_{\text{XOSw}_2} = \overline{\text{sel}}(T_1, T_2) + \textstyle\sum_{i=1}^{n_1} \overline{\text{sel}}(\Delta T_i^{(1)}, \Delta T_i^{(2)}) + \sum_{i=n_1+1}^{n_2} \overline{\text{sel}}(\varnothing, \Delta T_i^{(2)})$$

M. $$\mathcal{T}_{\text{XOSw}_1} = \text{sel}(T_1, T_2) \cdot \textstyle\prod_{i=1}^{n_1} \text{sel}(\Delta T_i^{(1)}, \Delta T_i^{(2)}) \cdot \prod_{i=n_1+1}^{n_2} \text{sel}(1, \Delta T_i^{(2)}),$$
$$\mathcal{T}_{\text{XOSw}_2} = \overline{\text{sel}}(T_1, T_2) \cdot \textstyle\prod_{i=1}^{n_1} \overline{\text{sel}}(\Delta T_i^{(1)}, \Delta T_i^{(2)}) \cdot \prod_{i=n_1+1}^{n_2} \overline{\text{sel}}(1, \Delta T_i^{(2)})$$

where $\text{sel}(a, b)$ selects a or b with equal probability, with $\varnothing$ indicating no additive term and 1 serving as the neutral multiplicative element. The operator $\overline{\text{sel}}(a, b)$ returns the complement of $\text{sel}(a, b)$, such that if $\text{sel}(a, b) = a$, then $\overline{\text{sel}}(a, b) = b$, and vice versa.

This crossover generates two distinct individuals. For the first offspring, each block is randomly selected from the corresponding position in either parent with equal probability. The second offspring is then constructed by taking all blocks not selected by the first; for each position, if the first offspring inherits a block from one parent, the second offspring inherits the corresponding block from the other parent. When parents differ in length, any extra blocks from the longer parent (for $i = n_1 + 1, \ldots, n_2$) may or not be assigned to the first offspring at random. The second offspring then inherits any remaining unassigned blocks, ensuring distinct contributions without overlap. As shown in Fig. 2, the two offspring contain blocks randomly selected from each parent. Being the second parent longer than the first, the two extra blocks are randomly assigned between the offspring: specifically, ΔT_5^2 goes to the first, while ΔT_6^2 goes to the second.

When defining a crossover in SLIM, we need to ensure that offspring blocks remain within the range $[-ms, ms]$, as this is appropriate for applying DGSM in the next evolutionary step. This crossover generates two offspring that maintain block values within this range, as they are composed of preexisting blocks, that already had this property.

The geometrical properties of XOSw are quite peculiar. In fact, while the crossover itself is not a geometric crossover, it *can* be defined (under some additional assumptions) as the crossover between two different individuals that derive from the original parents. In fact, let us define two trees, $\mathcal{T}_{\min}$ and $\mathcal{T}_{\max}$, representing the element-by-element minimum (resp., maximum) of each of the components of the trees $\mathcal{T}_1$ and $\mathcal{T}_2$. These trees are a (not necessarily strict) upper bound and lower bound to what can be achieved by crossover, and each individual resulting from the crossover between $\mathcal{T}_1$ and $\mathcal{T}_2$ is in the segment joining $\mathcal{T}_{\min}$ and $\mathcal{T}_{\max}$. Hence, while the crossover is not geometric in the classical sense, it is geometric on an *extension* of the original parents' semantic vectors, in a way similar to line crossovers. Thus, an "escape" from the convex hull of the population via crossover is possible, but in a tightly controlled way. A formal investigation of this fact, while interesting, is outside the scope of this paper.

A key property of this crossover is that the offspring match the size of their parents, ensuring that applying this operator does not lead to an overall growth in individual size; the total number of blocks between the two parents is maintained in the two offspring.

***Donor Crossover* (XODn):** Given two parent functions $\mathcal{T}_1 = (T_1, \Delta T_1^{(1)}, \ldots, \Delta T_{n_1}^{(1)})$ and $\mathcal{T}_2 = (T_2, \Delta T_1^{(2)}, \ldots, \Delta T_{n_2}^{(2)})$, with $\mathcal{T}_1$ as the donor and $\mathcal{T}_2$ as the receiver, XODn produces two offspring:

A. $\mathcal{T}_{\text{XODn}_1} = T_1^{(1)} + \sum_{i \neq j} \Delta T_i^{(1)}, \quad \mathcal{T}_{\text{XODn}_2} = T_1^{(2)} + \Delta T_j^{(1)} + \sum_{i=1}^{n_2} \Delta T_i^{(2)}$

M. $\mathcal{T}_{\text{XODn}_1} = T_1^{(1)} \cdot \prod_{i \neq j} \Delta T_i^{(1)}, \quad \mathcal{T}_{\text{XODn}_2} = T_1^{(2)} \cdot \Delta T_j^{(1)} \cdot \prod_{i=1}^{n_2} \Delta T_i^{(2)}$

where j represents the donated block index.

XODn leverages the SLIM linked-list structure by transferring a block from a donor to a receiver. An example can be found in Fig. 2, where the first parent is randomly chosen as the receiver and the second as the donor. A block is then randomly selected from the donor (in this example ΔT_2^2) and appended to the receiver.

Intuitively, it acts as DGSM on the donor and IGSM on the receiver, generating offspring within a radius ms around the parents. This approach preserves the geometricity property, thus maintaining the GSGP trait of inducing a unimodal error surface, while also enabling genetic material exchange without the GSC limitation of confining offspring within the initial solution space. Unlike mutation, which introduces a random block, XODn transfers a block from an existing individual, justifying its classification as a crossover due to the genetic material exchange. Moreover, offspring remain close in size to the parents (within one block more or less), preventing size growth.

4 Experimental Study

We now examine the impact of both XOSw and XODn crossovers on the fitness and size of the evolved individuals. We design two experimental setups: one where the crossover is added to the regular SLIM framework and one where the crossover is added to a modified version of SLIM from which the deflate mutation is removed.

We perform the experiments using two configurations—(SLIM+2SIG) (additive) and (SLIM*1SIG) (multiplicative)—as they were shown to be the best performing variants in [36]. Notice that, in SLIM+2SIG, the second setup (that uses no deflate) is equivalent to running GSGP replacing its traditional GSC by the new crossover, since the IGSM in SLIM+2SIG is equivalent to the standard GSM in GSGP. We will compare the results with two baselines: the standard SLIM (that uses only the two mutations IGSM and DGSM) and the standard GSGP (that uses GSC and GSM=IGSM).

4.1 Dataset and Experimental Settings

The test problems in this study are five widely recognized symbolic regression tasks frequently used as benchmarks in the GP literature (e.g., [9,19,31,36]): Istanbul (7 features, 536 samples [1]), Airfoil (5 features, 1052 samples [6]), Yacht (6 features, 307 samples [30]), Concrete Strength (8 features, 1030 samples [11]), and Concrete Slump (9 features, 102 samples [44]). Further details can be found in [23,43]. We performed Monte Carlo cross-validation [14] with 30 random data partitions into training (70%) and test (30%) sets. All the experiments were performed 30 times, each using one of the 30 partitions.

Regarding parameter settings, trees are initialized with a maximum depth of 6 using the ramped half-and-half initialization. Selection uses tournaments of size 2, with elitism that ensures the survival of the best individual from one generation to the next. The function set includes the four binary arithmetic operators (+, −, *, and /), with/protected as in [17]. The terminal set comprises only the variables of the problem, without any constants. The population size is set to 100 individuals, allowed to evolve for 1000 generations. All the methods use the same number of fitness evaluations, ensuring a fair comparison. Fitness is assessed by the Root Mean Squared Error (RMSE) between predicted and expected outputs.

For GSGP, we use standard probabilities of 0.8 for crossover and 0.2 for mutation. For the SLIM algorithm, the probability of applying DGSM is always 0.7. At every mutation event, the mutation step is randomly generated according to a uniform distribution in the range $[0, 1]$. Regarding XODn and XOSw, the probabilities are also set according to each problem. Then, the IGSM probability is equal to 1 minus the crossover probability. When XOSw is applied, a probability of 0.5 is used across all datasets but Istanbul, which is set to 0.8. Meanwhile, when using XODn a probability of 0.8 is used for both SLIM*1SIG and SLIM+2SIG on the Yacht and Istanbul datasets; 0.5 for both variants on Airfoil; and 0.5 for SLIM*1SIG and 0.8 for SLIM+2SIG on the Strength and Slump datasets.

The code, for the complete reproducibility of the proposed experiments, is available at slim.

5 Results

Figure 3 shows the distribution of test fitness of the best individual at the final generation over 30 runs, while Fig. 4 shows its evolution across generations, median of the 30 runs. Figure 5 shows the distribution of the size of the best individual at the final generation over 30 runs.

Each row corresponds to a different problem. The left column shows the results for SLIM*1SIG, and the right one for SLIM+2SIG. Both compare the standard version (black) with the variants using the two crossovers introduced in Sect. 3. For each variant, we present results without DGSM, shown with dashed contours in the boxplot and dashed lines in the line plot, and with DGSM, drawn with solid contours and solid lines. We also include a boxplot in Fig. 3 and a line plot in Fig. 4 with the results of GSGP (in dashed contours, since it also does not include DGSM). In the boxplots, a G and/or S under or above the boxes means a statistically significant difference (according to the Mann-Whitney U test with $\alpha = 0.01$) when comparing to GSGP (G) or the respective SLIM (S). A character under/above a box means the respective variant is better/worse than the baseline (GSGP or SLIM). The supplementary material contains tables with the p-values of all pairwise comparisons.

5.1 Test Fitness

These plots reveal the effectiveness of the crossover operators. With the addition of crossover—whether DGSM is included or not—the test fitness in the last generation (Fig. 3) is always statistically equal to or better than traditional SLIM, except for a single case (top right plot, XODn without DGSM) where SLIM is significantly better. Looking at the evolution of test fitness along the generations (Fig. 4) we realize that this exception is one of the few cases where overfitting occurred, precisely with most crossover variants but not with SLIM. Had the evolution been stopped earlier, there would be no exception to the successful performance of the crossover operators when compared with SLIM. Regarding the comparison with GSGP, the crossover variants achieve statistically equal or better test fitness in 28 out of 40 comparisons (whereas SLIM only achieves this in 4 out of 10 comparisons). Looking again at the evolution of test fitness we realize that in some cases (e.g., on the Istanbul dataset) early stopping could have benefited the SLIM variants when compared to GSGP.

5.2 Model Size

The improvement of test fitness brought by the crossover operators often comes at the cost of model size (Fig. 5), which is generally equal to or significantly larger than in traditional SLIM, with the notable exception of XOSw with DGSM that only once (in 40 comparisons) produced models significantly larger than SLIM. Regarding GSGP, the distribution of model size is not shown because it exceeds the plot limits, except on the Istanbul dataset. The median size of GSGP individuals is 2628 for the Istanbul dataset and reaches as high as 10^{60} for Airfoil, 10^{68} for Yacht, 10^{70} for Strength, and 10^{86} for Slump. All SLIM variants produce significantly smaller models than GSGP, except for a few cases on the Istanbul dataset, where the differences are not significant. This difference substantially impacts computational efficiency and, even more importantly, model interpretability.

5.3 To Deflate or Not To Deflate?

As expected, variants that apply crossover with DGSM consistently produce smaller individuals compared to those without DGSM (Fig. 5), but the use of DGSM tends to decrease the performance in terms of fitness (Fig. 3). In XOSw fitness becomes significantly worse in 4/10 cases, while in XODn this happens in 2/10 cases. We hypothesize that this happens because crossover arranges blocks in useful combinations and then DGSM may discard valuable components of these combinations, diminishing their usefulness. However, an interesting observation emerges from the fitness evolution plots (Fig. 4). In the SLIM*1SIG Istanbul plot, the crossover variants without DGSM tend to overfit, as illustrated

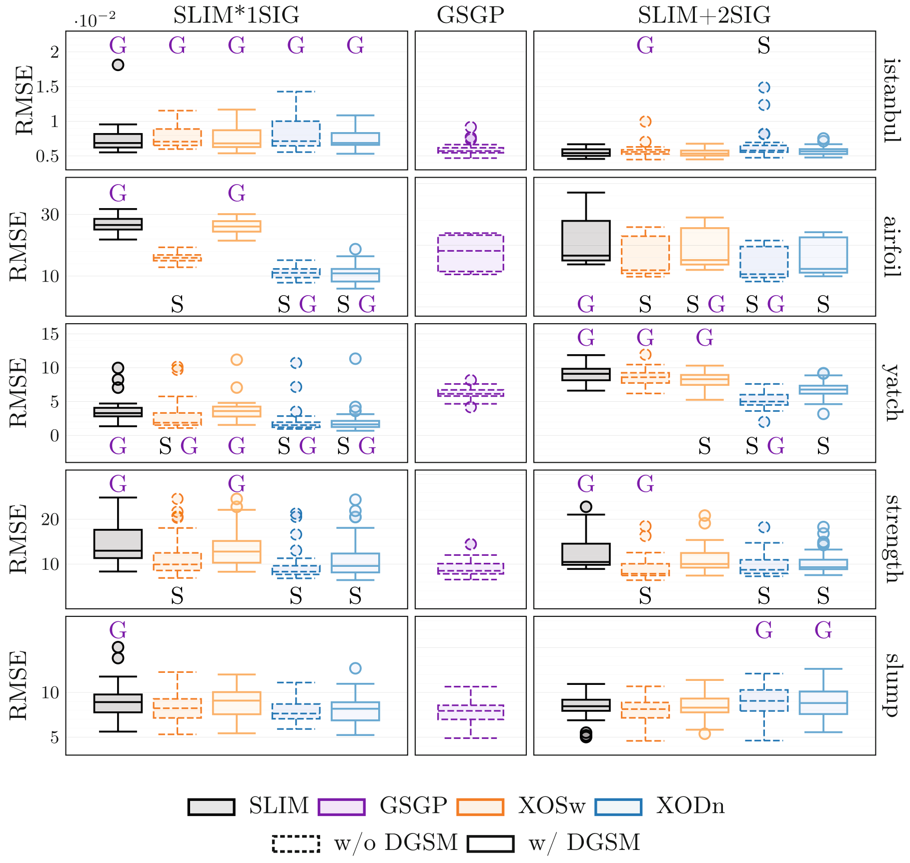

Fig. 3. Test fitness distribution of the best individual across 30 runs. Dashed and solid box contours indicate versions without and with DGSM, respectively. A G/S below or above a box indicates a statistically significant difference (Mann-Whitney U test with $\alpha = 0.01$) from GSGP (G) or SLIM (S), with the position indicating whether the variant is better (under) or worse (above).

by an increase in test fitness over time. A similar trend is observed in standard SLIM. However, when DGSM is included alongside crossover, overfitting appears to be reduced, likely due to the ability of DGSM to simplify the structure of the individuals.

5.4 Swap Crossover or Donor Crossover

XOSw performs well both with and without DGSM: it is never surpassed by standard SLIM and is equal to or significantly better than GSGP in 12/20 cases (half with DGSM and half without DGSM). Let us focus on XOSw with DGSM. Notably, in [36,40] the authors show that SLIM models are either smaller or of comparable size to standard GP models and that this compactness allows

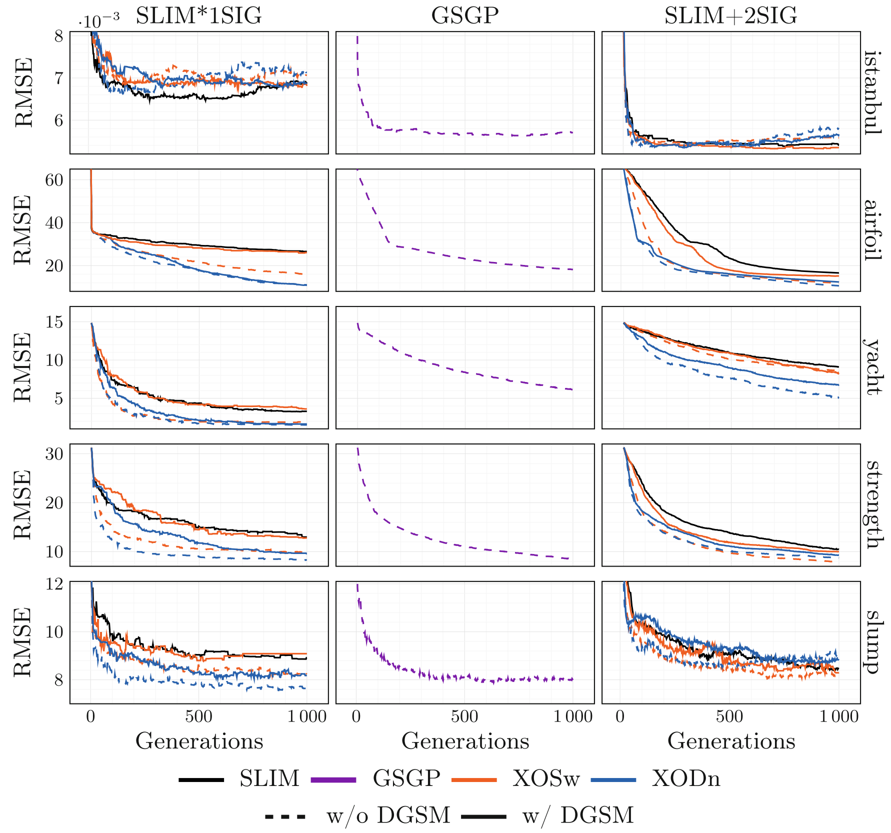

Fig. 4. Evolution of test fitness for the best individual in SLIM across 30 runs. Dashed lines represent versions without DGSM, while solid lines indicate versions with DGSM.

for human interpretation. XOSw with DGSM maintains (or even improves) this compactness while matching (or significantly outperforming) the fitness of standard SLIM in every case. This means that the exchange of genetic material is very beneficial when coupled with SLIM's deflate operator.

XODn produces larger models than both standard SLIM and XOSw. When compared to standard SLIM, the larger models of XODn with DGSM suggest that adding blocks via genetic material exchange from a donor—rather than from random trees—creates individuals with higher fitness, which increases their chances of survival and, over time, leads to growth in population size. In standard SLIM, when the DGSM operator randomly removes a block from an individual, the information in that block is lost. In contrast, with XODn, a block removed from one parent is always transferred to the other, allowing the information to survive across generations if it proves beneficial. These observations highlight the benefit of exchanging genetic material in evolution, which is confirmed by the fact that XODn often outperforms others in fitness: it is surpassed

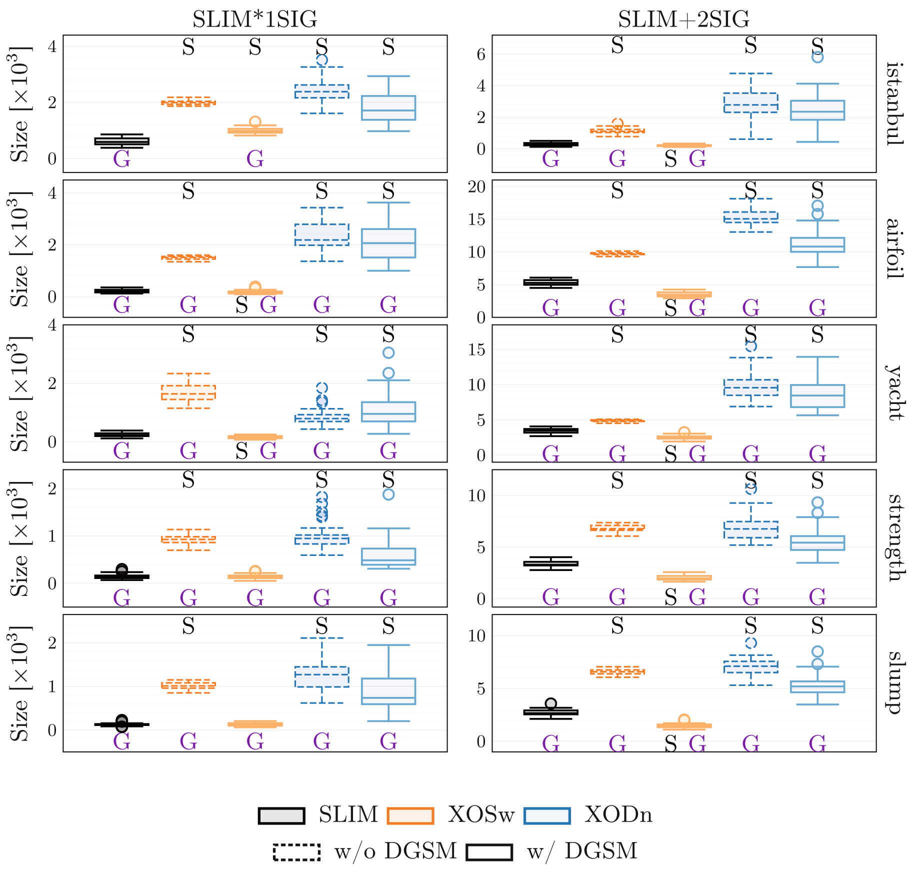

Fig. 5. Size distribution of the best individual in SLIM across 30 runs, following the codification detailed in Fig. 3.

by standard SLIM in only one case, and it is equal to or significantly better than GSGP in 16/20 cases (half with DGSM and half without DGSM), also achieving better results early in the evolution (Fig. 4).

5.5 SLIM*1SIG or SLIM+2SIG

Comparing the results between SLIM*1SIG and SLIM+2SIG, we observe that the best-performing alternative is generally XODn in SLIM*1SIG, which often outperforms GSGP. For maintaining smaller individual sizes, XOSw combined with DGSM in SLIM*1SIG proves to be the most effective.

In SLIM+2SIG, individuals tend to be larger than in SLIM*1SIG variants, as SLIM*1SIG mutations add only one extra tree, compared to two in SLIM+2SIG. Let us recall that SLIM+2SIG algorithms without DGSM essentially function as GSGP variants with an alternative crossover replacing GSC, since the IGSM in SLIM+2SIG is equivalent to the standard GSM in GSGP. Thus, the dashed results on the SLIM+2SIG plot represent GSGP variants with these alternative crossovers, enabled by the linked-list structure in both SLIM and GSGP. When comparing GSC with XODn or XOSw, we see not only a significant reduction in model size but also statistically equal or better performance than GSGP in most cases. This supports findings in [9,27] on the limited effectiveness of GSC in GSGP and suggests that the crossovers here defined offer a strong alternative, addressing both the limited search ability and size-growth issues inherent to GSC.

6 Conclusion

We introduced two novel crossover operators within the Semantic Learning algorithm based on Inflate and deflate Mutations (SLIM), designed to overcome the main limitations of traditional GSGP, namely the vast growth in program size and the restricted search ability of traditional geometric semantic crossover (GSC). Capitalizing on SLIM's linked-list representation, we proposed the Swap Crossover (XOSw) and the Donor Crossover (XODn), which facilitate effective genetic material exchange while maintaining controlled model sizes. XOSw works by exchanging linked-list components between two parents, while XODn transfers one block from a parent (donor) to the other parent (receiver). Our experimental results across five symbolic regression problems reveal that these crossovers often enhance the predictive ability of SLIM, achieving statistically significant improvements over traditional SLIM in terms of fitness on unseen data and model size, especially when combined with deflate mutation (DGSM). These findings establish XOSw and XODn as robust improvements of standard GSC.

Future work will focus on several fronts to further refine and expand the capabilities of SLIM. First, exploring adaptive crossover mechanisms [15] could allow SLIM to dynamically adjust the choice and application of crossover operators based on evolving population characteristics. Additionally, applying SLIM in a broader array of real-world domains will enable further evaluation of the versatility and effectiveness of these crossovers.

Acknowledgments. This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia), under the project - UIDB/04152/2020 (DOI: 10.54499/UIDB/04152/2020) - Centro de Investigação em Gestão de Informação (MagIC)/NOVA IMS),(https://doi.org/10.54499/UIDB/04152/2020) the project 2024.07277. IACDC, and the LASIGE Research Unit, ref. UID/000408/2025.

References

1. Akbilgic, O., Bozdogan, H., Balaban, M.E.: A novel hybrid RBF neural networks model as a forecaster. Stat. Comput. **24**, 365–375 (2014)
2. Albinati, J., Pappa, G.L., Otero, F.E., Oliveira, L.O.V.: The effect of distinct geometric semantic crossover operators in regression problems. In: Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, 8–10 Apr 2015, Proceedings 18, pp. 3–15. Springer (2015)
3. Beadle, L., Johnson, C.G.: Semantically driven crossover in genetic programming. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pp. 111–116. IEEE (2008)
4. Beadle, L., Johnson, C.G.: Semantically driven mutation in genetic programming. In: 2009 IEEE Congress on Evolutionary Computation, pp. 1336–1342. IEEE (2009)
5. Bonin, L., Rovito, L., De Lorenzo, A., Manzoni, L.: Cellular geometric semantic genetic programming. Genet. Program Evolvable Mach. **25**(1), 8 (2024)
6. Brooks, T.F., Pope, D.S., Marcolini, M.A.: Airfoil self-noise and prediction. Technical report (1989)
7. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. Comput. IEEE Trans. **100**(8), 677–691 (1986)
8. Castelli, M., Manzoni, L.: Gsgp-c++ 2.0: a geometric semantic genetic programming framework. SoftwareX **10**, 100313 (2019). https://doi.org/10.1016/j.softx.2019.100313, https://www.sciencedirect.com/science/article/pii/S2352711019301736
9. Castelli, M., Manzoni, L., Gonçalves, I., Vanneschi, L., Trujillo, L., Silva, S.: An analysis of geometric semantic crossover: a computational geometry approach. In: IJCCI (ECTA), pp. 201–208 (2016)
10. Castelli, M., Trujillo, L., Vanneschi, L.: Energy consumption forecasting using semantic-based genetic programming with local search optimizer. Comput. Intell. Neurosci. **2015**(1), 971908 (2015)
11. Castelli, M., Vanneschi, L., Silva, S.: Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. Expert Syst. Appl. **40**(17), 6856–6862 (2013)
12. Chen, Q., Zhang, M., Xue, B.: New geometric semantic operators in genetic programming: perpendicular crossover and random segment mutation. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 223–224 (2017)
13. Chen, X., Ong, Y.S., Lim, M.H., Tan, K.C.: A multi-facet survey on memetic computation. IEEE Trans. Evol. Comput. **15**(5), 591–607 (2011)
14. Coombes, K., Baggerly, K., Morris, J., Dubitzky, W., Granzow, M., Berrar, D.: Fundamentals of Data Mining in Genomics and Proteomics, pp. 79–99. Kluwer, Boston (2007)
15. Ferreira, J., Castelli, M., Manzoni, L., Pietropolli, G.: A self-adaptive approach to exploit topological properties of different gas' crossover operators. In: European Conference on Genetic Programming (Part of EvoStar), pp. 3–18. Springer (2023)
16. Hara, A., Kushida, J.I., Tanemura, R., Takahama, T.: Deterministic crossover based on target semantics in geometric semantic genetic programming. In: 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), pp. 197–202. IEEE (2016)

17. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**, 87–112 (1994)
18. Krawiec, K.: Medial crossovers for genetic programming. In: Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, 11–13 Apr 2012. Proceedings 15, pp. 61–72. Springer (2012)
19. La Cava, W., et al.: Contemporary symbolic regression methods and their relative performance. arXiv preprint arXiv:2107.14351 (2021)
20. Mambrini, A., Manzoni, L., Moraglio, A.: Theory-laden design of mutation-based geometric semantic genetic programming for learning classification trees. In: 2013 IEEE Congress on Evolutionary Computation, pp. 416–423. IEEE (2013)
21. Marchetti, F., Pietropolli, G., Camerota Verdù, F.J., Castelli, M., Minisci, E.: Control law automatic design through parametrized genetic programming with adjoint state method gradient evaluation (2023). Available at SSRN 4490005
22. Martins, J.F.B.S., Oliveira, L.O.V.B., Miranda, L.F., Casadei, F., Pappa, G.L.: Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, pp. 1151–1158. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3205455.3205593
23. McDermott, J., et al.: Genetic programming needs better benchmarks. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, pp. 791–798 (2012)
24. McPhee, N.F., Ohs, B., Hutchison, T.: Semantic building blocks in genetic programming. In: Genetic Programming: 11th European Conference, EuroGP 2008, Naples, Italy, 26–28 Mar 2008, Proceedings 11, pp. 134–145. Springer (2008)
25. Moraglio, A.: An efficient implementation of GSGP using higher-order functions and memoization. In: Johnson, C., Krawiec, K., Moraglio, A., O'Neill, M. (eds.) Semantic Methods in Genetic Programming, Ljubljana, Slovenia 13 Sep 2014. http://www.cs.put.poznan.pl/kkrawiec/smgp2014/uploads/Site/Moraglio2.pdf. workshop at Parallel Problem Solving from Nature 2014 conference
26. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Coello, C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32937-1_3
27. Moraglio, A., Mambrini, A.: 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, pp. 989–996 (2013)
28. Nadizar, G., Sakallioglu, B., Garrow, F., Silva, S., Vanneschi, L.: Geometric semantic GP with linear scaling: Darwinian versus Lamarckian evolution. Genet. Program Evolvable Mach. **25**(2), 1–24 (2024)
29. Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic aware crossover for genetic programming: the case for real-valued function regression. In: Genetic Programming: 12th European Conference, EuroGP 2009 Tübingen, Germany, 15–17 Apr 2009 Proceedings 12, pp. 292–302. Springer (2009)
30. Ortigosa, I., Lopez, R., Garcia, J.: A neural networks approach to residuary resistance of sailing yachts prediction. In: Proceedings of the International Conference on Marine Engineering Marine, vol. 2007, p. 250 (2007)
31. Orzechowski, P., La Cava, W., Moore, J.H.: Where are we now? A large benchmark study of recent symbolic regression methods. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1183–1190 (2018)
32. Pawlak, T.P., Wieloch, B., Krawiec, K.: Review and comparative analysis of geometric semantic crossovers. Genet. Program Evolvable Mach. **16**, 351–386 (2015)

33. Pietropolli, G., Manzoni, L., Paoletti, A., Castelli, M.: Combining geometric semantic GP with gradient-descent optimization. In: European Conference on Genetic Programming (Part of EvoStar), pp. 19–33. Springer (2022)
34. Pietropolli, G., Manzoni, L., Paoletti, A., Castelli, M.: On the hybridization of geometric semantic GP with gradient-based optimizers. Genet. Program Evolvable Mach. **24**(2), 16 (2023)
35. Trujillo, L., et al.: Local search is underused in genetic programming. In: Genetic Programming Theory and Practice XIV, pp. 119–137 (2018)
36. Vanneschi, L.: SLIM_GSGP: the non-bloating geometric semantic genetic programming. In: Giacobini, M., Xue, B., Manzoni, L. (eds.) Genetic Programming, pp. 125–141. Springer, Cham (2024)
37. Vanneschi, L., et al.: PSXO: population-wide semantic crossover. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 257–258 (2017)
38. Vanneschi, L., Castelli, M., Manzoni, L., Silva, S.: A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, 3–5 Apr 2013, Proceedings 16, pp. 205–216. Springer (2013)
39. Vanneschi, L., Castelli, M., Silva, S.: A survey of semantic methods in genetic programming. Genet. Program Evolvable Mach. **15**(2), 195–214 (2014). https://doi.org/10.1007/s10710-013-9210-0
40. Vanneschi, L., Farinati, D., Rasteiro, D., Rosenfeld, L., Pietropolli, G., Silva, S.: Exploring non-bloating geometric semantic genetic programming. In: Genetic Programming Theory and Practice. (to appear)
41. Vanneschi, L., Silva, S.: Lectures on Intelligent Systems. Springer, Cham (2023)
42. Vanneschi, L., Silva, S., Castelli, M., Manzoni, L.: Geometric semantic genetic programming for real life applications. In: Genetic Programming Theory and Practice XI, pp. 191–209 (2014)
43. White, D.R., et al.: Better GP benchmarks: community survey results and proposals. Genet. Program Evolvable Mach. **14**, 3–29 (2013)
44. Yeh, I.C.: Simulation of concrete slump using neural networks. Proc. Inst. Civ. Eng. Constr. Mater **162**(1), 11–18 (2009)