

# Solving the Exponential Growth of Symbolic Regression Trees in Geometric Semantic Genetic Programming

Joao Francisco B. S. Martins, Luiz Otavio V. B. Oliveira, Luis F. Miranda, Felipe Casadei, Gisele L. Pappa  
 Universidade Federal de Minas Gerais, Department of Computer Science  
 Belo Horizonte, Brazil  
 [joaofbsm,luizvbo,luisfmiranda,casadei,gppappa]@dcc.ufmg.br

## ABSTRACT

Advances in Geometric Semantic Genetic Programming (GSGP) have shown that this variant of Genetic Programming (GP) reaches better results than its predecessor for supervised machine learning problems, particularly in the task of symbolic regression. However, by construction, the geometric semantic crossover operator generates individuals that grow exponentially with the number of generations, resulting in solutions with limited use. This paper presents a new method for individual simplification named GSGP with Reduced trees (GSGP-Red). GSGP-Red works by expanding the functions generated by the geometric semantic operators. The resulting expanded function is guaranteed to be a linear combination that, in a second step, has its repeated structures and respective coefficients aggregated. Experiments in 12 real-world datasets show that it is not only possible to create smaller and completely equivalent individuals in competitive computational time, but also to reduce the number of nodes composing them by 58 orders of magnitude, on average.

## CCS CONCEPTS

•Computing methodologies → Genetic programming; Supervised learning by regression;

## KEYWORDS

Genetic Programming; Geometric Semantic Genetic Programming; Symbolic Regression; Solution Size; Function Simplification

## ACM Reference format:

Joao Francisco B. S. Martins, Luiz Otavio V. B. Oliveira, Luis F. Miranda, Felipe Casadei, Gisele L. Pappa. 2018. Solving the Exponential Growth of Symbolic Regression Trees in Geometric Semantic Genetic Programming. In *Proceedings of Genetic and Evolutionary Computation Conference, Kyoto, Japan, July 15–19, 2018 (GECCO '18)*, 8 pages.  
 DOI: 10.1145/3205455.3205593

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
 GECCO '18, Kyoto, Japan

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
 978-1-4503-5618-3/18/07...\$15.00  
 DOI: 10.1145/3205455.3205593

## 1 INTRODUCTION

The Geometric Semantic Genetic Programming (GSGP) [15] framework introduces geometric semantic operators to Genetic Programming (GP). These operators are capable of inducing a semantic effect through syntactic operations, and allow GSGP to explore a conic semantic fitness landscape, which can be efficiently optimized using evolutionary search [13]. GSGP has shown to outperform GP in different scenarios, specially in symbolic regression [8, 9, 26].

However, GSGP suffers from a problem that limits its use in real-world applications. By definition, the geometric semantic operators generate offspring composed by the complete representation of their parents, plus some additional structures, leading to an exponential growth of the solution size in the number of generations [24]. This extreme growth leads to excessive usage of memory and computational power, and also results in non-interpretable solutions [24].

Two different approaches have been followed in the literature to deal with the problem of exponential growth of GSGP individuals, although they were not able to effectively solve the problem. The first approach simply focuses on making the algorithm more efficient in terms of memory and computational resources [14, 25]. The second proposes new versions of the semantic crossover operators since they are the ones responsible for the excessive growth of the solutions [16, 22].

Most works based on GSGP follow the first approach, using an implementation presented by Vanneschi et al. [25] that stores pointers to the trees representing the individuals, instead of keeping the whole individual in memory. This implementation computes the semantics—and fitness—of new individuals from the values of their parents [6]. Although the implementation is very fast and reduces the memory needed during the evolution, the individuals are not explicitly built during the search. Thus, if we want to access the final individual or if new data is presented after the training stage, an assembling step is needed to generate the complete individual from the pointers, which still presents an exponential size.

This work presents a new method, called Geometric Semantic Genetic Programming with Reduced trees (GSGP-Red), to solve the problem of excessive growth of GSGP solutions for symbolic regression. GSGP-Red works by expanding the functions generated by the geometric semantic operators. The resulting expanded function is guaranteed to be a linear combination that, in a second step, has its repeated structures and respective coefficients aggregated. These expansion and aggregation operations ensure that only one copy of each function composing the solution is kept in the simplified individual, leading to a massive reduction of the size of the resulting solutions while guaranteeing the exact same results of GSGP.

An experimental analysis comparing the proposed method with GSGP and GP in 12 real-world datasets showed that GSGP-Red is capable of finding solutions equivalent to the ones generated by GSGP while resulting in individuals up to 64 orders of magnitude smaller than those generated by previous GSGP versions, with a practicable additional overhead in computational time.

The remainder of this paper is organized as follows. Section 2 introduces the main concepts of GSGP. Section 3 reviews related work, while Section 4 introduces the proposed method. Section 5 reports computational results, and Section 6 draws conclusions and points out direction of future work.

## 2 GEOMETRIC SEMANTIC GENETIC PROGRAMMING

Genetic Programming (GP) [11] manipulates individuals during the evolution by applying operators that modify the structure of their trees, i.e., their syntax. Although some restrictions are respected by GP operators—e.g., the arity of the function nodes—they do not consider the behaviour—i.e., the semantics—of the individuals. GSGP [13], on the other hand, employs operators that act on the syntax of the population with a defined semantic outcome.

In the context of symbolic regression, the semantics of a given individual  $p$ , representing a symbolic expression—usually stored as a tree—can be represented as the output vector it generates when applied to the training set  $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ —with  $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$  for  $i = 1, 2, \dots, n$ —given by  $s(p) = [p(\mathbf{x}_1), p(\mathbf{x}_2), \dots, p(\mathbf{x}_n)]$ . This representation allows us to describe the semantics of any individual in an  $n$ -dimensional semantic space [19].

GSGP defines geometric semantic operators that generate offspring with a given behaviour in the semantic space w.r.t. a given metric. Given a parent individual  $p$ , the Geometric Semantic Mutation (*GSM*) operator applies a semantic perturbation to the individual, generating an offspring placed inside a ball centred on the parent individual in the semantic space, with radius  $\varepsilon \in \mathbb{R}$  proportional to the mutation step. The operator is defined as

$$GSM(p(\mathbf{x}), \delta) = p(\mathbf{x}) + \delta \times (r_m(\mathbf{x}) - r_n(\mathbf{x})), \quad (1)$$

where the parameter  $\delta$  is the mutation step and  $r_m$  and  $r_n$  are functions randomly built.

The Geometric Semantic Crossover (*GSX*) operator, on the other hand, combines two individuals  $p_1$  and  $p_2$ , resulting in a single offspring placed in the metric segment connecting the parents in the semantic space. The *GSX* operator defined w.r.t. the Euclidean distance is given by

$$GSX_E(p_1(\mathbf{x}), p_2(\mathbf{x})) = k \times p_1(\mathbf{x}) + (1 - k) \times p_2(\mathbf{x}), \quad (2)$$

where  $k \in \mathbb{R}$  is a constant uniformly sampled from  $[0, 1]$ . Similarly, the *GSX* operator defined w.r.t. the Manhattan distance is given by

$$GSX_M(p_1(\mathbf{x}), p_2(\mathbf{x})) = r_f(\mathbf{x}) \times p_1(\mathbf{x}) + (1 - r_f(\mathbf{x})) \times p_2(\mathbf{x}), \quad (3)$$

where  $r_f$  is a function randomly generated with codomain  $[0, 1]$ .

By construction, the geometric semantic mutation and crossover operators induce, respectively, linear and exponential growth of the individuals with the number of generations. Equations 4, 5 and 6 present the expected number of nodes of an individual of

the generation  $g > 0$ , generated by GSGP using only one of the geometric semantic operators—*GSM*, *GSX<sub>E</sub>* or *GSX<sub>M</sub>*, respectively [18, 20].  $E[P_0]$  is the expected number of nodes in the individuals of the initial population,  $E[r]$  is the expected number of nodes in the random functions generated by the operators and  $a$ ,  $b$  and  $c$  are the number of additional nodes (constant) used by *GSM*, *GSX<sub>E</sub>* and *GSX<sub>M</sub>*, respectively.

$$E[GSM, g] = E[P_0] + g \times (2 \times E[r] + a) \quad (4)$$

$$E[GSX_E, g] = 2^g \times E[P_0] + (2^g - 1) \times b \quad (5)$$

$$E[GSX_M, g] = 2^g \times E[P_0] + (2^g - 1) \times (E[r] + c) \quad (6)$$

This characteristic is pointed out as the main drawback of GSGP—after a few generations the population becomes unmanageable in terms of memory and computational time spent to compute the fitness [24]. In addition, the excessive size of the individuals makes the functions they represent very hard to understand and interpret [7]. Since one of the main advantages of GP over other black box learning approaches is the ability to find solutions in the form of comprehensible structures, the exponential size of GSGP solutions limits its usage in practice [16].

## 3 RELATED WORK

The exponential growth of GSGP individuals was identified by Moraglio et al. [15] in their seminal work. The authors propose to simplify the offspring during the evolution in order to keep the size of the individuals manageable. However, given the complexity of the process, they suggest simplifying the functions only sufficiently—i.e., partially instead of optimally—using, for example, a computer algebra system in order to avoid increasing the computational cost of GSGP excessively.

There are a few works in the literature that try to deal with the problem of tree exponential growth, but none of them actually solve it. These methods follow two main directions. The first focuses on more efficient implementations of GSGP, while the second proposes different modifications to the genetic operators aiming to reduce the size of the produced offspring—resulting in operators that are only approximately geometric.

Among works in the first group are the implementation of *geometric semantic operators for symbolic regression* proposed by Vanneschi and colleagues [6, 25], conceived to reduce memory consumption and computational time. The trees representing the individuals in the initial population and the functions used by geometric semantic operators are stored in memory, such that the subsequent individuals are composed of pointers to these structures. The semantics of the individuals is also stored in memory and used to compute the semantics of the offspring, reducing computational effort to calculate the fitness. However, the function represented by the individual is never truly built during the evolution. In order to obtain the symbolic function defined by the individual, we need to reconstruct it from the pointers and trees stored in memory, resulting in a function with size exponentially proportional to the number of generations. Instead of using pointers and data structures, Moraglio [14] uses higher-order functions and memoization

in his Python implementation, delegating the control to the compiler. The functions evolved are represented directly as compiled Python and still present exponential size when decompiled.

In the second group of works is the Subtree Semantic Geometric Crossover (SSGX) operator, an approximation for the  $GSX_M$  that generates smaller individuals proposed by Nguyen et al. [16]. Given two parent individuals,  $p_1, p_2$ , SSGX generates offspring by applying the  $GSX_M$  operator to the subtrees of  $p_1$  and  $p_2$  with semantics more similar to their respective parents, resulting in smaller functions. In addition, the method intercalates SSGX with the conventional subtree-swapping crossover [11] during crossover operations, resulting in solutions around 29 orders of magnitude smaller than those generated by  $GSX_M$ . However, although SSGX outperformed  $GSX_M$  in terms of test error in their experimental analysis, the configuration of the experiments can make the results inconclusive. This is because SSGX and  $GSX_M$  are tested using different mutation operators, which can be the responsible for the difference in the performance. In addition, SSGX is around three times more time consuming than  $GSX_M$ .

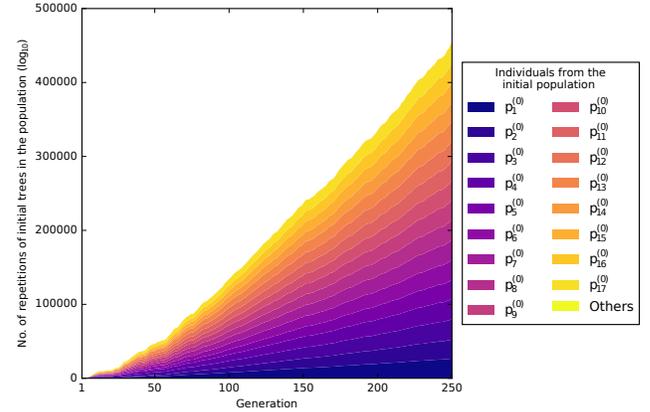
Pawlak and Krawiec [22], in turn, analyse a wide range of mutation and crossover operators under different metrics, including the ratio between the sizes of the offspring and their parents. The experimental analysis conducted with populations initialized using the ramped half-and-half method [11] showed that  $GSM$  generates offspring 5.16 times larger than their parents, on average, while the Tree Mutation (TM) [11], Competent Mutation (CM) [20] and Semantically Driven Mutation (SDM) [5] result in offspring around 2.5 times larger than their parents. A similar experiment involving crossover operators showed that, on average, the Subtree-Swapping Crossover (SSX) [11] and Semantically Driven Crossover (SDX) [4] generate offspring of the same size of their parents, and the Competent Crossover (CX) and  $GSX_E$  operators generate offspring 1.78 and 2.35 times larger than their parents, respectively. Notice, however, that TM and SSX are non-semantic operators, SDM and SDX are semantic but not geometric and CM and CX are approximately geometric semantic operators.

The method proposed here does not fit in any of the approaches listed before. It does not change the way  $GSM$  or  $GSX$  work and is able to generate exactly the same results that GSGP generates. It performs on-the-fly simplification of trees just after crossover and mutation operators are applied by taking advantage of the fact that the individuals are always linear combinations of trees. At the same time, its implementation does allow for efficient use of memory and computational time.

## 4 METHODOLOGY

By construction, GSGP operators keep the structure of the individuals being manipulated untouched, only adding other subtrees to them. Consequently, the structure of the individuals generated in the initial population is perpetuated through the whole evolution process, usually with multiple repetitions within the same individual. For instance, Figure 1 presents the frequency of the 1,000 individuals from the initial population composing the GSGP population throughout 250 generations.<sup>1</sup> Notice that only 17 individuals

<sup>1</sup>This experiment was carried out with the same parameters adopted in Section 5 on the CCN dataset.



**Figure 1: Frequency of appearance of initial population trees in the structure of individuals throughout generations.**

generated in the initial population, represented as  $p_1^{(0)}, p_2^{(0)}, \dots, p_{17}^{(0)}$  in the legend, compose individuals in the later generations and all of them have a high frequency of appearance.

Given the large concentration of duplicated trees from the initial population within an individual, the size of the solutions evolved by GSGP can be drastically reduced by combining these repeated structures. The approach we propose, named Geometric Semantic Genetic Programming with Reduced trees (GSGP-Red), takes advantage of the repetition of functions to reduce the computational cost involved in the GSGP evolution. However, contrary to related methods from the literature, our approach combines repeated trees, effectively simplifying the individuals generated and reducing the size of the solutions dramatically.

### 4.1 GSGP-Red

GSGP-Red works by exploring the linear combinations of individuals performed by the geometric semantic mutation and crossover operators. Looking at Equation 1, observe that the  $GSM$  operator generates a linear combination of the input parent and randomly generated trees. The  $GSX_E$  operator, in turn, generates a convex combination of the input parents. Thus, the evolution process performed by GSGP with these operators corresponds to recursively applying linear combinations to linear combinations of trees. GSGP-Red rewrites this recursion by expanding the terms and combining repeated structures.

Let  $P_0 = \{p_1^{(0)}, p_2^{(0)}, \dots, p_{|P_0|}^{(0)}\}$  be the initial GSGP population,  $R = \{r_1, r_2, \dots, r_{|R|}\}$  be the set of functions (trees) randomly generated by the  $GSM$  operator during the evolutionary process. An individual  $p_i$ , from generation  $g$ , can be represented as a linear combination, defined by the dot product  $C_i \cdot F_i$ , where the first operand,  $C_i = [c_{i,1}, c_{i,2}, \dots, c_{i,s_i}] \in \mathbb{R}^{s_i}$ , is the set of coefficients multiplying the terms; the second operand,  $F_i = [f_{i,1}, f_{i,2}, \dots, f_{i,s_i}]^T \in \{P_0 \cup R\}^{s_i}$ , is the set of functions (trees) composing the individual, perpetuated from the initial population or generated by the  $GSM$  operator; and  $s_i$  is the number of distinct functions composing  $p_i$ . For  $g = 0$ , the individual  $p_i$  from the GSGP initial population can be

described as in Equation 7. For  $g > 0$ , the individuals are recursively combined, as described in Equations 8 and 9.

$$\begin{aligned} p_i^{(0)} &= 1 \times p_i^{(0)} \\ &= c_{i,1} \times f_{i,1} \\ &= C_i \cdot F_i \end{aligned} \quad (7)$$

GSGP-Red applies two new steps to every new offspring generated by  $GSM$  or  $GSX_E$ , namely expansion and aggregation. During the expansion, GSGP-Red multiplies the coefficients from the geometric semantic operator by the randomly generated functions in  $GSM$  offspring and adds all the terms, as presented in Equation 8<sup>2</sup>. The same is done for the  $GSX_E$  offspring, but, in this case, the coefficients from the geometric semantic operator are multiplied by the coefficient vectors  $C_i$  and  $C_j$  from the parent individuals  $p_i$  and  $p_j$ , as presented in Equation 9. At the end of the expansion stage, the offspring resulting from the  $GSM$  and  $GSX_E$  operators—denoted as  $p_{OM}$  and  $p_{OX}$ , respectively—consist of linear combinations, which can be rewritten as the dot products  $C_{OM} \cdot F_{OM}$  and  $C_{OX} \cdot F_{OX}$ , respectively.

$$\begin{aligned} GSM(p_i, \delta) &= p_i + \delta \times (r_m - r_n) \\ &= C_i \cdot F_i + \delta \times r_m - \delta \times r_n \\ &= c_{i,1} \times f_{i,1} + \dots + c_{i,s_i} \times f_{i,s_i} \\ &\quad + \delta \times r_m - \delta \times r_n \\ &= [c_{i,1}, \dots, c_{i,s_i}, \delta, -\delta] \cdot [f_{i,1}, \dots, f_{i,s_i}, r_m, r_n]^T \\ &= C_{OM} \cdot F_{OM} \end{aligned} \quad (8)$$

$$\begin{aligned} GSX_E(p_i, p_j) &= k \times p_i + (1 - k) \times p_j \\ &= k \times (C_i \cdot F_i) + (1 - k) \times (C_j \cdot F_j) \\ &= k \times c_{i,1} \times f_{i,1} + \dots + k \times c_{i,s_i} \times f_{i,s_i} \\ &\quad + (1 - k) \times c_{j,1} \times f_{j,1} + \dots \\ &\quad + (1 - k) \times c_{j,s_j} \times f_{j,s_j} \\ &= [k \times c_{i,1}, \dots, k \times c_{i,s_i}, \\ &\quad (1 - k) \times c_{j,1}, \dots, (1 - k) \times c_{j,s_j}] \\ &\quad \cdot [f_{i,1}, \dots, f_{i,s_i}, f_{j,1}, \dots, f_{j,s_j}]^T \\ &= C_{OX} \cdot F_{OX} \end{aligned} \quad (9)$$

During the aggregation stage, GSGP-Red combines functions appearing more than once in the list of functions from the resulting individual. Let  $p_{new} = C_{new} \cdot F_{new}$  be an offspring resulting from the expansion step and  $f_{rep}$  be a function appearing  $l$  times in  $F_{new}$ , i.e.,  $f_{rep_1}, f_{rep_2}, \dots, f_{rep_l}$ , with the respective coefficients  $c_{rep_1}, c_{rep_2}, \dots, c_{rep_l}$  in  $C_{new}$ . The aggregation step keeps the first appearance of  $f_{rep}$  ( $f_{rep_1}$ ) in  $F_{new}$  and its respective coefficient ( $c_{rep_1}$ ) in  $C_{new}$ , removing all the other function occurrences and their respective coefficients— $f_{rep_2}, \dots, f_{rep_l}$  and  $c_{rep_2}, \dots, c_{rep_l}$ . For each coefficient removed, its value is added to the coefficient of the instance kept by the method, i.e., the value of  $c_{rep_1}$  is updated to  $\sum_{i=1}^l c_{rep_i}$ , as they all come from a linear combination. The size of the individual— $s_{new}$ —is also updated to reflect the removal of

the repeated functions and their respective coefficients from the tree representation.

Note that here we work with the  $GSX$  defined w.r.t. the Euclidean distance. The motivation for choosing  $GSX_E$  over  $GSX_M$  for GSGP-Red comes from the fact that  $GSX_M$  multiplies the parents by a randomly generated function—contrary to the linear combination performed by  $GSX_E$ —which would imply in additional complexity in time and space to store and manipulate a function instead of a constant. Notice that the usage of the  $GSX_E$  over the  $GSX_M$  or vice versa is an open discussion in the literature, with some works defending the usage of  $GSX_M$ , given empirical analysis [16], and others defending the usage of  $GSX_E$ , given its progression properties [21].

Next, we illustrate the expansion and aggregation operators. Consider a initial population with individuals  $P = \{p_1^{(0)}, p_2^{(0)}, p_3^{(0)}\}$ , where

$$p_1^{(0)} = x_1/x_2 = 1 \times (x_1/x_2) = c_{1,1} \times f_{1,1}, \quad (10)$$

$$p_2^{(0)} = x_2 + 0.4 = 1 \times (x_2 + 0.4) = c_{2,1} \times f_{2,1}, \quad (11)$$

$$p_3^{(0)} = x_1 - 0.6 = 1 \times (x_1 - 0.6) = c_{3,1} \times f_{3,1}. \quad (12)$$

Crossing over  $p_2^{(0)}$  with  $p_3^{(0)}$  (Equation 13) and mutating  $p_1^{(0)}$  (Equation 14) and  $p_3^{(0)}$  (Equation 15), with a mutation step arbitrarily set to 0.1, results in three new individuals,  $p_1^{(1)}$ ,  $p_2^{(1)}$  and  $p_3^{(1)}$ , respectively, composing the population of the next generation.

$$\begin{aligned} p_1^{(1)} &= 0.3 \times p_2^{(0)} + (1 - 0.3) \times p_3^{(0)} \\ &= 0.3 \cdot [1 \times (x_2 + 0.4)] + (1 - 0.3) \times [1 \times (x_1 - 0.6)] \\ &= [0.3 \times (x_2 + 0.4)] + [0.7 \times (x_1 - 0.6)] \\ &= c_{1,1} \times f_{1,1} + c_{1,2} \times f_{1,2} \end{aligned} \quad (13)$$

$$\begin{aligned} p_2^{(1)} &= p_1^{(0)} + 0.1 \times [(x_1) - (2 \times x_2)] \\ &= (x_1/x_2) + 0.1 \times [(x_1) - (2 \times x_2)] \\ &= [1 \times (x_1/x_2)] + [0.1 \times (x_1)] + [-0.1 \times (2 \times x_2)] \\ &= c_{2,1} \times f_{2,1} + c_{2,2} \times f_{2,2} + c_{2,3} \times f_{2,3} \end{aligned} \quad (14)$$

$$\begin{aligned} p_3^{(1)} &= p_3^{(0)} + 0.1 \times [(x_1 - 0.6) - (x_1 \times x_2)] \\ &= x_1 - 0.6 + 0.1 \times [(x_1 - 0.6) - (x_1 \times x_2)] \\ &= [1 \times (x_1 - 0.6)] + [0.1 \times (x_1 - 0.6)] \\ &\quad + [-0.1 \times (x_1 \times x_2)] \\ &= [(1 + 0.1) \times (x_1 - 0.6)] + [-0.1 \times (x_1 \times x_2)] \\ &= c_{3,1} \times f_{3,1} + c_{3,2} \times f_{3,2} \end{aligned} \quad (15)$$

In this example, the random constant used by crossover in Eq. 13 is equal to 0.3 and the two functions randomly generated by the mutation operator are  $x_1$  and  $2 \times x_2$  in Eq. 14 and  $x_1 - 0.6$  and  $x_1 \times x_2$  in Eq. 15. Notice that one of the random functions generated in Eq. 15 is equal to the function represented by the parent individual—both presented in bold. These functions are then combined in a single function and the coefficients are summed up, generating a smaller individual.

<sup>2</sup>For the sake of simplicity, we omit the input parameters of the functions.

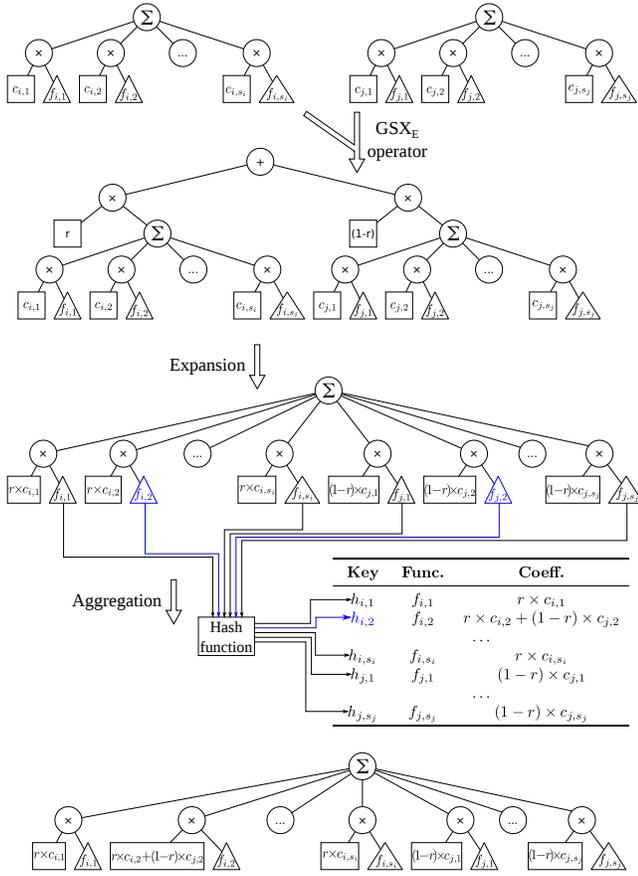


Figure 2: Expansion and aggregation after applying the  $GSX_E$  operator.

## 4.2 Implementation Details

The implementation employed in the experimental analysis of Section 5 iterates through the functions composing the new individual, performing the expansion and aggregation steps sequentially. In addition, it keeps a hash table for each individual to store the trees and their respective coefficients uniquely—indexed by the symbolic expression represented by the tree—speeding up the process of aggregating the functions composing an individual. This is the only form of representation that is necessary for each individual, with no direct pointers to its parents, who are implicitly stored amongst the expanded and then aggregated set of functions and their respective coefficients, unlike previous implementations [6, 25]. The code is available for download<sup>3</sup> or can also be directly executed from the Lemonade<sup>4</sup> data analysis platform.

Figures 2 and 3 depict the functioning of GSGP-Red with the hash tables used in our implementation. The hash function maps the functions composing the individual to a hash key, used to index the table. Figure 2 presents a  $GSX_E$  offspring transformed by GSGP-Red—the hash tables of the parents are omitted in the figure. The expansion step results in two repeated trees—represented

<sup>3</sup><https://github.com/laic-ufmg/GSGP-Red>

<sup>4</sup><https://demo.ctweb.inweb.org.br>

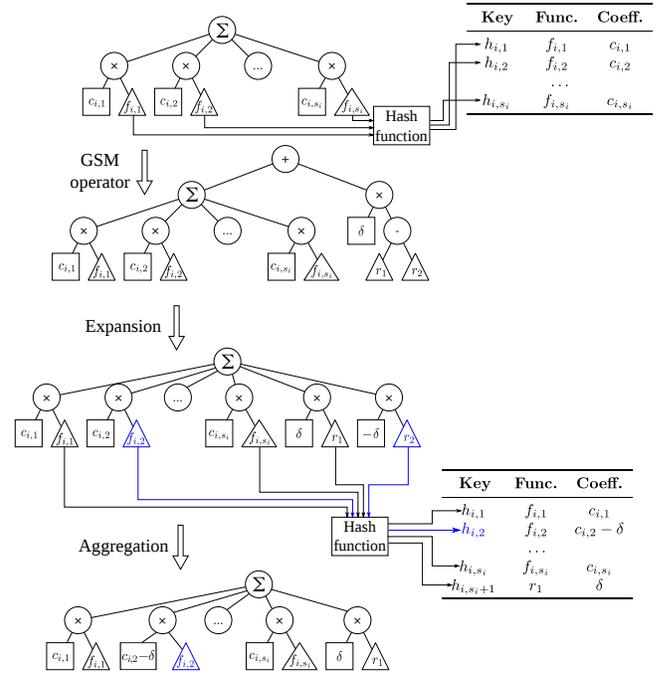


Figure 3: Expansion and aggregation after applying the GSM operator.

by  $f_{i,2}$  and  $f_{j,2}$ , in blue in the figure—which are combined in the aggregation step—notice the sum of the coefficients  $r \times c_{i,2}$  and  $(1-r) \times c_{j,2}$  and the hash function resulting in only one blue arrow.

Fig. 3, on the other hand, presents the GSGP-Red procedure applied to a GSM offspring. In our example, the mutation operator generates a tree— $r_2$ —equivalent to a function composing the parent individual— $f_{i,2}$ —resulting in the same hash index—presented in blue. During the aggregation, the coefficients of these functions are combined— $c_{i,2} + (-\delta)$ —when the hash table is updated. Notice that, although infrequent, it is possible for the GSM to generate a tree already generated somewhere else during the evolution, given the finite number of possible combinations of functions and terminals.

## 5 EXPERIMENTAL ANALYSIS

In this section we present an experimental analysis of the performance of GSGP-Red when applied to 12 real-world datasets. The datasets are described in Table 1, which shows their number of attributes (# of attrs) and number of instances (# of instances). GSGP-Red results are compared to the results obtained by GSGP and the canonical GP [3] in terms of training and test Root Mean Square Error (RMSE), size of the functions—given by the number of tree nodes—and computational time expended by the methods.

Given the non-deterministic nature of the methods, each experiment was repeated 30 times—6 times for each fold in a 5-fold cross-validation procedure. In order to validate the results, we performed Wilcoxon signed-rank tests [10, 23] with a confidence level of 95%, under the null hypothesis that GSGP-Red performance—in terms of test RMSE, solution size and computational time—is equal to the performance of the other methods for each dataset.

**Table 1: Datasets used in the experiments.**

Abbr.	Dataset	# of attrs	# of instances	Source
air	Airfoil	6	1503	[1, 12]
ccn	CCN	123	1994	[12]
ccun	CCUN	125	1994	[12]
con	Concrete	9	1030	[1, 12]
eneC	Energy Cooling	9	768	[1, 12]
eneH	Energy Heating	9	768	[1, 12]
par	Parkinsons	19	5875	[12]
ppb	PPB	627	131	[1]
tow	Tower	26	4999	[1]
wineR	Wine Red	12	1599	[1, 12]
wineW	Wine White	12	4898	[1, 12]
yac	Yacht	7	308	[1, 12]

### 5.1 GP and GSGP settings

GP and GSGP were run with a population of 1,000 individuals evolved for 250 generations with tournament selection of size 7 and 10, respectively. The terminal set comprised the variables of the problem and constant values uniformly picked from  $[-1, 1]$ , described by Koza [11] as *ephemeral random constants* (ERC). The function set included three binary arithmetic operators (+, -, ×) and the analytic quotient (AQ) [17], which has the general properties of division but without discontinuity, given by:

$$AQ(a, b) = \frac{a}{\sqrt{1 + b^2}} \quad (16)$$

The GP method employed the canonical crossover and mutation operators [11] with probabilities 0.9 and 0.1, respectively. GSGP employed  $G_{SX}_E$  and  $G_{SM}$  operators, both with probability 0.5, as presented in [6], but without the logistic function to bound the outputs of the randomly generated functions. The grow method [11] was adopted to generate the random functions within the geometric semantic crossover and mutation operators, and the ramped half-and-half method [11] to generate the initial population, both with a maximum individual depth equal to 6. Following the work from Albinati et al. [2], the mutation step adopted by the geometric semantic mutation operator was defined as 10% of the standard deviation of the target output vector given by the training data. Both methods used the RMSE calculated over the obtained and expected output values for the training set. The same parameters adopted for GSGP were used in GSGP-Red experiments.

### 5.2 Experimental Analysis

Table 2 presents the RMSE obtained by each method on the 12 datasets. The symbol  $\blacklozenge$  indicates the null hypothesis (GSGP-Red performance is equal to the performance of other methods) was not discarded and the symbol  $\blacktriangle(\blacktriangledown)$  indicates that the performance of GSGP-Red was better (worse) than the performance of the GP. Recall that the results of GSGP/GSGP-Red are the same (see Section 4 for details), as the proposed method generates a solution equivalent to the one produced by GSGP. According to the outcomes of the statistical tests regarding the test RMSE, GP is better than GSGP in the Yacht dataset, and the results have no statistical difference for datasets CCUN and PPB. In the other 9 cases, GSGP/GSGP-Red is

**Table 2: Median RMSE of the best individual for training and test sets for GP and GSGP/GSGP-Red. The symbol  $\blacktriangle(\blacktriangledown)$  indicates that the performance of GSGP-Red was better (worse) than the performance of GP.**

Dataset	RMSE	GSGP/GSGP-Red	GP
air	Training	11.783	17.353
	Test	11.280	17.612 $\blacktriangle$
ccn	Training	0.128	0.145
	Test	0.138	0.149 $\blacktriangle$
ccun	Training	377.616	386.203
	Test	405.463	396.308 $\blacklozenge$
con	Training	8.510	9.352
	Test	8.886	9.750 $\blacktriangle$
eneC	Training	3.114	3.364
	Test	3.129	3.455 $\blacktriangle$
eneH	Training	2.677	2.973
	Test	2.739	3.101 $\blacktriangle$
par	Training	9.812	9.955
	Test	9.868	9.995 $\blacktriangle$
ppb	Training	14.870	27.644
	Test	29.647	28.542 $\blacklozenge$
tow	Training	46.634	50.050
	Test	46.533	50.155 $\blacktriangle$
wineR	Training	0.632	0.657
	Test	0.636	0.652 $\blacktriangle$
wineW	Training	0.729	0.756
	Test	0.735	0.766 $\blacktriangle$
yac	Training	6.529	3.413
	Test	6.437	3.541 $\blacktriangledown$

better than GP, which motivates the construction of methods such as GSGP-Red.

The previous results confirm that the solutions generated by GSGP-Red are equivalent to those generated by GSGP and, in most cases, better than the solutions produced by the canonical GP. However, the results that show the main contribution of the proposed method are listed in Table 3, where we show the median number of nodes in the best individuals of GSGP-Red, GSGP and GP. Again, the symbol  $\blacklozenge$  indicates the null hypothesis (GSGP-Red size is equal to the size of other methods) was not discarded and the symbol  $\blacktriangle(\blacktriangledown)$  indicates that the performance of GSGP-Red was better (worse) than the performance of the method indicated by the column (GSGP or GP).

Note that GSGP-Red individuals are always much smaller than those generated by GSGP. By calculating the reduction in size when comparing GSGP to GSGP-Red, solutions from the latter are, on average, 58 orders of magnitude smaller. The maximum reduction in size occurred in CCUN (64 order of magnitude) and the minimum reduction occurred in the Parkinsons dataset (45 orders of magnitude). It is important to point out that the function sizes are still substantially bigger than the ones generated by GP, but without forgetting that the RMSE results for GSGP are still, in general, superior. As discussed later, we believe that the functions generated

**Table 3: Median size of the best individual (in number of nodes) for GSGP-Red, GSGP and GP. The symbol ▲(▼) indicates that the performance of GSGP-Red was better (worse) than the performance of the method indicated in the column.**

Dataset	GSGP-Red	GSGP	GP
air	33,353	1.72e+50 ▲	86 ▼
ccn	22,819	7.25e+58 ▲	40 ▼
ccun	3,373	2.33e+67 ▲	45 ▼
con	6,007	1.23e+65 ▲	43 ▼
eneC	6,584	1.08e+65 ▲	64 ▼
eneH	6,881	1.19e+65 ▲	67 ▼
par	34,386	1.88e+49 ▲	81 ▼
ppb	12,185	2.29e+64 ▲	61 ▼
tow	4,843	7.34e+66 ▲	49 ▼
wineR	9,220	3.53e+64 ▲	49 ▼
wineW	9,983	2.20e+64 ▲	45 ▼
yac	13,706	1.23e+61 ▲	62 ▼

by GSGP-Red can be further reduced using other techniques, such as algebraic simplification.

### 5.3 Run-time Analysis

In order to analyse to what extent the application of the expansion and aggregation processes increase GSGP computational cost, we compare the median time spent by GSGP-Red and the canonical versions of GP and GSGP to generate regression models for our testbed, including both training and test stages. The results of this analysis, shown in Table 4, indicate that the running times for GSGP-Red are, in general, higher than those presented by GSGP, but there are exceptions. For some datasets, GSGP-Red was not only faster than GP but also faster than GSGP itself. This can be explained by the fact that, when running GSGP-Red, we do not need to calculate the test fitness for every created individual. This was mandatory in the GSGP implementation proposed by Vanneschi [25], for example, as the best individual could not be easily reconstructed at the end of the evolution process, and values of RMSE for training and test were computed during evolution. This is not the case for GSGP-Red, which can easily store the solution generated for later use in new data. Hence, GSGP-Red only evaluates the test fitness of the best overall individual. For some datasets, removing these operations make the total runtime decrease considerably, making GSGP-Red adoption even more appealing.

GSGP-Red was faster than GSGP in three datasets: CCUN, Tower and Wine White, being statistically worse in all the remaining. However, the fact that GSGP-Red takes longer to run does not indicate a lack of efficiency or some fundamental problem compromising its application since, in absolute terms, the difference between the two methods is still small. On the other hand, when compared to GP, GSGP-Red is faster in 10 out of 12 datasets, with execution times, on average, 35% faster. In conclusion, GSGP-Red is overall slower

**Table 4: Median of the execution time (in seconds) of GSGP-Red, GSGP and GP. The symbol ▲(▼) indicates that the performance of GSGP-Red was better (worse) than the performance of the method indicated in the column.**

Dataset	GSGP-Red	GSGP	GP
air	113.28	48.32 ▼	202.13 ▲
ccn	96.10	63.35 ▼	154.75 ▲
ccun	53.15	63.62 ▲	148.98 ▲
con	39.19	35.17 ▼	77.85 ▲
eneC	36.24	28.99 ▼	78.65 ▲
eneH	37.85	27.92 ▼	86.57 ▲
par	191.93	165.36 ▼	767.46 ▲
ppb	37.58	10.74 ▼	15.57 ▼
tow	110.60	136.90 ▲	486.54 ▲
wineR	55.03	48.32 ▼	128.04 ▲
wineW	127.24	140.66 ▲	378.36 ▲
yac	39.19	15.03 ▼	32.35 ▼

than GSGP but is still efficient and yet better than GP in terms of RMSE and execution time.

## 6 CONCLUSIONS AND FUTURE WORK

This paper presented Geometric Semantic Genetic Programming with Reduced trees (GSGP-Red), a new method that solves the exponential growth of GSGP solutions with the number of generations for symbolic regression problems. The method expands the functions representing the individuals into linear combinations, and then aggregates the repeated structures. This process results in functions many times smaller than those generated by GSGP.

An experimental analysis was performed in a testbed composed of 12 real-world datasets in order to compare GSGP-Red with its predecessor and with GP. Results showed that the new method is capable of generating solutions equivalent to those generated by GSGP in terms of error, but 58 orders of magnitude smaller, on average., in terms of size (number of tree nodes). In addition, an analysis of the execution time revealed that GSGP-Red, although slower than GSGP on average, can also perform the search faster than GSGP, depending on the dataset.

Potential future developments include simplifying the solutions using computer algebra systems [15] and integrating approximated geometric semantic operators—e.g., the competent mutation and crossover operators from Pawlak [20]—to GSGP-Red, in order to reduce even further the size of the solutions generated.

Compiling all these ideas into a single framework seems a promising direction to make the readability and degree of understanding of the GSGP solutions closer to those of models generated by GP.

## ACKNOWLEDGMENTS

This work was partially supported by the following Brazilian Research Support Agencies: CNPq, FAPEMIG, CAPES. The authors' work has also been partially funded by the EUBra-BIGSEA project

by the European Commission under the Cooperation Programme (MCTI/RNP 3rd Coordinated Call), Horizon 2020 grant agreement 690116. Finally, we would like to thank Gabriel Coutinho for his valuable insights on the mathematical aspects of the problem in question.

## REFERENCES

- [1] Julio Albinati, Gisele L. Pappa, Fernando E. B. Otero, and Luiz Otavio V. B. Oliveira. 2015. The Effect of Distinct Geometric Semantic Crossover Operators in Regression Problems. In *18th European Conference, EuroGP 2015 (LNCS)*, Penousal Machado, Malcolm I. Heywood, James McDermott, Mauro Castelli, Pablo García-Sánchez, Paolo Burelli, Sebastian Risi, and Kevin Sim (Eds.), Vol. 9025. Springer International Publishing, 3–15. DOI: [http://dx.doi.org/10.1007/978-3-319-16501-1\\_1](http://dx.doi.org/10.1007/978-3-319-16501-1_1)
- [2] Julio Albinati, Gisele L. Pappa, Fernando E. B. Otero, and Luiz Otavio V. B. Oliveira. 2015. The Effect of Distinct Geometric Semantic Crossover Operators in Regression Problems. In *Genetic Programming*, Penousal Machado, Malcolm I. Heywood, James McDermott, Mauro Castelli, Pablo García-Sánchez, Paolo Burelli, Sebastian Risi, and Kevin Sim (Eds.). Springer International Publishing, Cham, 3–15.
- [3] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. 1998. *Genetic Programming – an Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers.
- [4] Lawrence Beadle and Colin G Johnson. 2008. Semantically driven crossover in genetic programming. In *Evolutionary Computation, 2008. IEEE Congress on. IEEE*, 111–116.
- [5] Lawrence Beadle and Colin G Johnson. 2009. Semantically driven mutation in genetic programming. In *Evolutionary Computation, 2009. IEEE Congress on. IEEE*, 1336–1342.
- [6] Mauro Castelli, Sara Silva, and Leonardo Vanneschi. 2015. A C++ framework for geometric semantic genetic programming. *Genetic Programming and Evolvable Machines* 16, 1 (Mar 2015), 73–81. DOI: <http://dx.doi.org/10.1007/s10710-014-9218-0>
- [7] Mauro Castelli, Leonardo Vanneschi, and Aleš Popovič. 2016. Controlling individuals growth in semantic genetic programming through elitist replacement. *Computational intelligence and neuroscience* 2016 (2016), 42.
- [8] Mauro Castelli, Leonardo Vanneschi, and Sara Silva. 2013. Prediction of high performance concrete strength using Genetic Programming with geometric semantic genetic operators. *Expert Systems with Applications* 40, 17 (2013), 6856–6862.
- [9] Mauro Castelli, Leonardo Vanneschi, and Sara Silva. 2014. Prediction of the Unified Parkinson’s Disease Rating Scale Assessment Using a Genetic Programming System with Geometric Semantic Genetic Operators. *Expert Systems with Applications* 41, 10 (2014), 4608 – 4616. DOI: <http://dx.doi.org/10.1016/j.eswa.2014.01.018>
- [10] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.
- [11] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Vol. 1. MIT Press, Cambridge, MA, USA.
- [12] M. Lichman. 2015. UCI Mach. Learning Repository. (2015). <http://archive.ics.uci.edu/ml>
- [13] Alberto Moraglio. 2011. Abstract convex evolutionary search. In *Proceedings of the 11th workshop on Foundations of genetic algorithms (FOGA '11)*. ACM, 151–162.
- [14] Alberto Moraglio. 2014. An efficient implementation of GSGP using higher-order functions and memoization. In: Johnson, C., et al. (eds.) *Semantic Methods in Genetic Programming*, Ljubljana, Slovenia, 13 Sept. 2014. Workshop at Parallel Problem Solving from Nature 2014 conference. (2014).
- [15] Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. 2012. Geometric Semantic Genetic Programming. In *Parallel Problem Solving from Nature, PPSN XII (part 1)*, Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone (Eds.). Lecture Notes in Computer Science, Vol. 7491. Springer, 21–31. DOI: [http://dx.doi.org/10.1007/978-3-642-32937-1\\_3](http://dx.doi.org/10.1007/978-3-642-32937-1_3)
- [16] Quang Uy Nguyen, Tuan Anh Pham, Xuan Hoai Nguyen, and James McDermott. 2016. Subtree semantic geometric crossover for genetic programming. *Genetic Programming and Evolvable Machines* 17, 1 (Mar 2016), 25–53.
- [17] J. Ni, R. H. Driberg, and P. I. Rockett. 2013. The use of an analytic quotient operator in genetic programming. *Evolutionary Computation, IEEE Trans. on* 17, 1 (Apr 2013), 146–152.
- [18] L. O. V. B. Oliveira. 2016. *Improving Search in Geometric Semantic Genetic Programming*. Ph.D. Dissertation. Universidade Federal de Minas Gerais.
- [19] Luiz Otavio V. B. Oliveira, Fernando E. B. Otero, and Gisele L. Pappa. 2016. A Dispersion Operator for Geometric Semantic Genetic Programming. In *Proc. of the Genetic and Evolutionary Computation Conference 2016*. ACM, 773–780. DOI: <http://dx.doi.org/10.1145/2908812.2908923>
- [20] Tomasz P. Pawlak. 2015. *Competent Algorithms for Geometric Semantic Genetic Programming*. Ph.D. Dissertation. Poznan University of Technology, Poznań, Poland.
- [21] Tomasz P Pawlak and Krzysztof Krawiec. 2015. Progress properties and fitness bounds for geometric semantic search operators. *Genetic Programming and Evolvable Machines* (2015), 1–19. DOI: <http://dx.doi.org/10.1007/s10710-015-9252-6>
- [22] Tomasz P Pawlak and Krzysztof Krawiec. 2017. Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. *Evolutionary computation* Early Access (2017), 1–36.
- [23] David J Sheskin. 2003. *Handbook of parametric and nonparametric statistical procedures* (3rd ed. ed.). Chapman & Hal/CRC.
- [24] Leonardo Vanneschi. 2017. An introduction to geometric semantic genetic programming. In *NEO 2015*. Springer, 3–42.
- [25] Leonardo Vanneschi, Mauro Castelli, Luca Manzoni, and Sara Silva. 2013. A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In *16th European Conference, EuroGP 2013 (LNCS)*, Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Şima Etaner-Uyar, and Bin Hu (Eds.), Vol. 7831. Springer Berlin Heidelberg, 205–216.
- [26] Leonardo Vanneschi, Sara Silva, Mauro Castelli, and Luca Manzoni. 2014. Geometric Semantic Genetic Programming for Real Life Applications. In *Genetic Programming Theory and Practice XI*, Rick Riolo, Jason H. Moore, and Mark Kotanchek (Eds.). Springer New York, 191–209.