A Dispersion Operator for Geometric Semantic Genetic Programming

Luiz Otavio V. B. Oliveira
Dep. Computer Science
UFMG
Belo Horizonte, Brazil
luizvbo@dcc.ufmg.br

Fernando E. B. Otero
School of Computing
University of Kent
Chatham Maritime, UK
F.E.B.Otero@kent.ac.uk

Gisele L. Pappa
Dep. Computer Science
UFMG
Belo Horizonte, Brazil
glpappa@dcc.ufmg.br

ABSTRACT

Recent advances in geometric semantic genetic programming (GSGP) have shown that the results obtained by these methods can outperform those obtained by classical genetic programming algorithms, in particular in the context of symbolic regression. However, there are still many open issues on how to improve their search mechanism. One of these issues is how to get around the fact that the GSGP crossover operator cannot generate solutions that are placed outside the convex hull formed by the individuals of the current population. Although the mutation operator alleviates this problem, we cannot guarantee it will find promising regions of the search space within feasible computational time. In this direction, this paper proposes a new geometric dispersion operator that uses multiplicative factors to move individuals to less dense areas of the search space around the target solution before applying semantic genetic operators. Experiments in sixteen datasets show that the results obtained by the proposed operator are statistically significantly better than those produced by GSGP and that the operator does indeed spread the solutions around the target solution.

CCS Concepts

•Computing methodologies → Heuristic function construction; Genetic programming;

Keywords

geometric semantic genetic programming; geometric semantic crossover; convex hull

1. INTRODUCTION

In recent years, several studies have attempted to include semantic knowledge into the search mechanism of genetic programming algorithms, producing positive impacts on their search performance [19]. In supervised learning problems, including symbolic regression, the semantics of an individual is defined as the vector of output values produced when applying the function the GP individual represents to the set of training cases [19].

One of the seminal works in the field of semantic genetic programming proposed geometric versions of the crossover and mutation operators, namely geometric semantic crossover (GSX) and mutation (GSM) operators [14]. These operators guarantee the semantic fitness landscape explored by GP is conic, a property with positive effects on the search process. There is formal evidence that indicates evolutionary algorithms with geometric operators can optimise cone landscapes with good results for virtually any metric [13]. However, geometric semantic genetic programming (GSGP) still presents drawbacks.

One of them comes from the fact that, when using only the crossover operator—which produces offspring by performing a convex combination of its parents—the set of possible individuals generated during evolution is delimited by the convex hull of the semantics of the current population [16]. The *convex hull* of a set of points is given by the set of all convex combinations of these points [17]. Hence, if the target output is not within the convex hull, the algorithm will never be able to find it.

The mutation operator appears as the solution to this problem, as it can expand the semantic search space and, consequently, the population convex hull. However, depending on how far from the target solution the individuals in the initial population are, GSGP may take a prohibitive amount of time to reach the relevant region of the search space where the target output is.

In this context, this paper proposes a new geometric semantic operator, named geometric dispersion (GD). GD takes as input a single individual and moves it in the semantic space aiming to better distribute the population around the target output. By doing that GD increases the chances of generating a convex hull from the population semantics that contains the target output vector (*out*) (see Figure 1). In contrast with GSM, which randomly moves the individual within a hypersphere around it, the geometric dispersion operator is deterministic, and moves the individual along a line crossing the origin of the semantic space to sparse regions around the target output.

We performed experiments in 16 symbolic regression datasets, and compared the results with the GSGP without using the operator and a canonical GP. Results of root mean square error showed that the proposed operator leads GSGP to obtain statistically significantly better results than the other two methods. An analysis of the distribution of the solutions along the dimensions of the semantic space

GECCO '16, July 20 - 24, 2016, Denver, CO, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4206-3/16/07...$15.00

DOI: http://dx.doi.org/10.1145/2908812.2908923

also showed the operator does spread the solutions in the semantic space.

The remainder of this paper is organized as follows. Section 2 presents background on GSGP, and Section 3 discusses different approaches proposed to increase the convex hull of the population. Section 4 introduces the geometric dispersion operator, while Section 5 shows results and empirical comparisons with GSGP and GP. Finally, Section 6 draws conclusions and presents directions for future work.

2. GEOMETRIC SEMANTIC GENETIC PROGRAMMING

This paper focuses on geometric semantic genetic programming for symbolic regression problems. Given a finite set of input-output pairs representing the training cases, defined as $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$—where $(\mathbf{x_i}, y_i) \in \mathbb{R}^d \times \mathbb{R}$ $(i = 1, 2, \ldots, n)$—symbolic regression consists in inducing a model $p : \mathbb{R}^d \to \mathbb{R}$ that maps inputs to outputs, such that $\forall(\mathbf{x_i}, y_i) \in T : p(\mathbf{x_i}) = y_i$.

In this scenario, let $in = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}]$ and $out = [y_1, y_2, \ldots, y_n]$ be the input and the output vectors, respectively, associated to the training cases. The semantics of a program p represented by an individual evolved by GSGP, denoted by $s(p)$, is the vector of outputs it produces when applied to the set of inputs in, i.e., $s(p) = p(in) = [p(\mathbf{x_1}), p(\mathbf{x_2}), \ldots, p(\mathbf{x_n})]$. This notation is extended to the semantics of a population of programs $P = \{p_1, p_2, \ldots, p_k\}$, i.e., $s(P) = \{s(p_1), s(p_2), \ldots, s(p_k)\}$. The semantics of any program can be represented as a point in a n-dimensional topological space $\mathcal{S}$ (called semantic space), where n is the size of the training set. Notice that the target output (out) can also be represented in the semantic space.

GSGP introduces geometric semantic operators for GP that acts on the syntax of the programs, inducing a geometric behaviour on the semantic level [14]. Let P' be the solution set comprising all the possible candidate solutions to a problem in the arithmetic domain, the geometric semantic crossover and mutation operators are defined as follows:

Definition 1. Given two parent functions $p_1, p_2 \in P'$, the geometric semantic arithmetic crossover $GSX : P' \times P' \to P'$ returns the offspring arithmetic function

$$o = r \cdot p_1 + (1 - r) \cdot p_2 \ , \tag{1}$$

where r is a random real constant in $[0, 1]$ (for fitness function based on Euclidean distance) or a random real function with codomain $[0, 1]$ (for fitness function based on Manhattan distance).

Definition 2. Given a parent function $p \in P'$, the geometric semantic arithmetic mutation $GSM : P' \times \mathbb{R}^+ \to P'$ with mutation step ε returns the real function

$$o = p + \varepsilon \cdot (r_1 - r_2) \ , \tag{2}$$

where r_1 and r_2 are random real functions.

Despite the unimodal fitness landscape, the randomness present in these operators, available in the form of random real functions or constants, has been shown to be a better way to explore the space, in terms of generalisation, when compared to modifications of these operators where the randomness is replaced by decisions based on the training error [1, 6, 9].

Definition 3. The *convex hull* of a set S of points in $\mathbb{R}^n$, denoted as $\mathcal{C}(S)$, is the set of all convex combinations of points in S [17].

GSX is, by definition, a geometric crossover operator [14]. This property lead us to the following theorem regarding the convex hull of the population[1]:

THEOREM 1. *Let P_g be the population at generation g. For a GSGP, where the GSX operator is the only search operator available, we have $\mathcal{C}(s(P_{g+1})) \subseteq \mathcal{C}(s(P_g)) \subseteq \ldots \subseteq \mathcal{C}(s(P_1)) \subseteq \mathcal{C}(s(P_0))$.*

Theorem 1 is a particular case of the Theorem 3 defined and proved in [13], and has an important implication regarding the GSX operator. Given a population P and a semantic vector q in $\mathcal{S}$, the offspring resulting from the application of GSX to any pair of individuals in P can reach q if and only if $q \in \mathcal{C}(s(P))$. Consequently, if GSGP has no other search operators (only the GSX), a semantic q is reachable only if $q \in \mathcal{C}(s(P_0))$, i.e., if q is located inside the convex hull of the initial population.

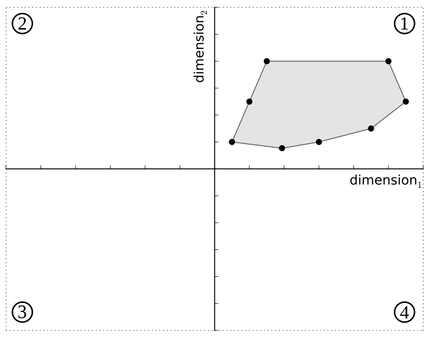
Figure 1 illustrates this situation for a bi-dimensional semantic space. Let $out = [0, 0]$ be the desired output vector defined by the training cases. Now consider two different populations P_c and P_s, where the individuals from P_c are concentrated in the first quadrant and, consequently, $\mathcal{C}(s(P_c))$ cannot reach the origin (out). On the other hand, the set $s(P_s)$ is distributed along the four quadrants and $\mathcal{C}(s(P_s))$ embraces the desired vector. In the first scenario, GSGP needs the mutation operator to expand the convex hull to reach the solution. In the second scenario, as it is already inside the convex hull, the desired vector can be found using the crossover operator alone or it can be calculated analytically with no need to use GSGP.

3. RELATED WORK

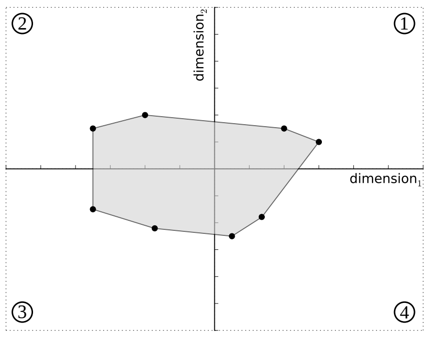
Previous work on GSGP have proposed different approaches to take advantage of the properties of the geometric semantic space to improve search. However, to the best of our knowledge, so far only one has investigated ways to increase the area covered by the population convex hull, specially focusing on the coverage of the initial population, as discussed in this section.

Regarding operators that take advantage of the conic shape of the geometric semantic space, in [18] the authors explore the geometry of the semantic space through the concept of error vector. An error vector is represented by a point in the n-dimensional space, called error space, given by the translation $t_e(p) = s(p) - out$. This notion is used to introduce the concept of optimally aligned individuals in the error space, i.e., given a number of dimensions $\mu = 1, 2, ..., n$, where n is the size of the training set, μ individuals are optimally aligned in the error space if they belong to the same μ-dimensional hyperplane intersecting the origin of the error space. The authors show that if μ individuals are optimally aligned, we can analytically obtain an equation to express the target output vector out. In this context, they present GP-based methods to find optimally aligned individuals in two and three dimensions, called ESAGP-1 (Error Space

[1]Let P be a population of individuals, we adopt the notation $\mathcal{C}(s(P))$ to denote the convex hull of the set composed by the semantics of the individuals of P, i.e., $s(P)$.



(a) P_c: population concentrated into a single quadrant.



(b) P_s: population encompasses solutions in all quadrants.

Figure 1: Different distributions of a population in a bi-dimensional semantic space. The desired output *out* is located in the origin of the space and the circled numbers indicate the quadrants.

Alignment GP) and ESAGP-2, respectively. Experimental results suggest that searching for optimally aligned individuals (in two and three dimensions) is easier than directly searching for a globally optimal solution.

The authors in [8], in contrast, extend ESAGP-1 to what they call Pair Optimization GP (POGP). Unlike the original method, which represents individuals as simple expressions and computes the fitness by the angle between the error vector of an individual and a particular point called attractor, POGP represents individuals as pairs of expressions, and calculates the fitness as the angle between the error vectors of these two expressions. POGP experimental results indicate that the method deserves attention in future studies.

Concerning methods taking the area covered by the convex hull, in [16] the authors propose the Competent Initialization (CI) method, which aims to increase the convex hull of the initial population. The algorithm adopts a generalized version of the Semantically Driven Initialization (SDI) method [3], initially proposed for non-geometric spaces, to generate individuals semantically distinct. SDI picks a node randomly from the function set to combine individuals already in the population. If the resulting program has semantics different from the other members of the population, it is accepted. Otherwise, the method makes a new attempt of generating an individual. The process continues until a semantically distinct individual is created, in a trial-and-error strategy. CI, on the other hand, accepts the semantically distinct individual only if it is not in the current convex hull. The main drawback of this method is the possible waste of resources, since individuals are randomly created, evaluated and discarded when they are semantically similar to an existing member of the population or when it is already in the population's convex hull.

Although not taking advantage of the geometric properties of the search space, in [7] the authors propose a semantic-based algorithm that keeps a distribution of different semantics during the evolution to drive GP to search in areas of the semantic space where previous good solutions were found. The method outperformed standard GP and bacterial GP [4] in the test bed adopted. However, the individuals generated presented statistically bigger sizes than the individuals generated by the other two GP variants.

In contrast with the aforementioned works, the geometric dispersion operator proposed here has a different rationale: to improve the distribution of the points around the target point, extending the convex hull of the current population.

4. THE GEOMETRIC DISPERSION OPERATOR

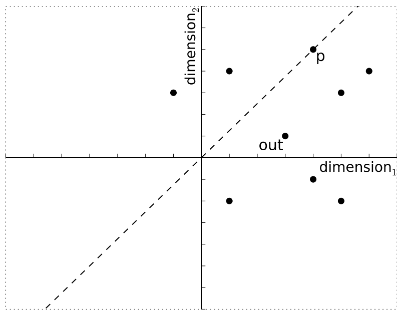
This section introduces the geometric dispersion (GD) operator. It takes as input a single individual p and moves it in the semantic space in order to better distribute the population around the desired output in each dimension of $\mathcal{S}$. By distributing the individuals around the desired output we increase the chances of generating a convex hull from the population semantics space that contains the desired output vector *out* (see Figure 1b).

The aim of GD is to move p to the region of the semantic space around *out* with the lowest concentration of individuals. In order to find this region, the population distribution in the semantic space is measured. The operator calculates the proportion of individuals in each side of *out* for each dimension of $\mathcal{S}$, i.e., the proportion of individuals greater and smaller than *out* in each dimension of the semantic space.

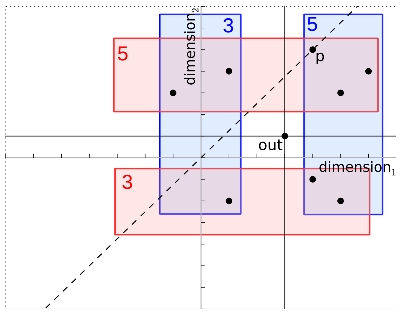
Knowing the region of the semantic space around *out* where we want to have individuals shifted to, different methods can be used to move individual p. GD does that by adding a multiplicative factor (m) to p, in the form $m \cdot p$. Note that, by definition, the movement performed by GD is limited to the line crossing both the origin and individual p in the semantic space (see Figure 2a). The problem is then to find the value of m that maximizes the number of dimensions benefited by the displacement of p.

Let $count_{gt}$ and $count_{lt}$ be n-dimensional arrays, where the value in position i corresponds to the number of individuals from P greater and smaller than *out* in dimension i, respectively. The GD operator moves p in a way that if $count_{lt}[i] < count_{gt}[i]$—there is a concentration of individuals on the right side of *out* in dimension i—GD moves p in dimension i to the left side of *out*. When the concentration of individuals in dimension i is on the left side of *out*, GD has the opposite effect—it moves p in dimension i to the right side of *out*. However, given that the operator can only move p in a straight line, there is no guarantee of achieving this behaviour for all dimensions.

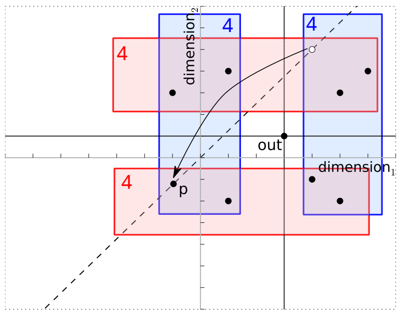
Given this limitation, the GD operator builds a system of linear inequalities that reflects the desired behaviour of the operator and finds the value of m that maximizes the number of inequalities satisfied, i.e., the number of dimensions



(a) GD line in the semantic space crossing $s(p)$ and the origin.



(b) Distribution of individuals around *out* regarding $dimension_1$ (blue) and $dimension_2$ (red). The numbers indicate the frequency of individuals on each side of *out* for each dimension.



(c) GD moves individual p along the line to a new position, balancing individual distribution in both dimensions.

Figure 2: The GD operator finds a multiplier m that moves the individual along a line connecting it to the origin to a different point in space.

that will be redistributed. The system of inequalities is of the form

$$\begin{cases} m \cdot s(p)[1] \lessgtr_1 out[1] \\ m \cdot s(p)[2] \lessgtr_2 out[2] \\ \ldots \\ m \cdot s(p)[k] \lessgtr_k out[k] \end{cases} , \qquad (3)$$

where '$\lessgtr_i$' is replaced by '<' or '>' depending on which side of *out* the individuals are concentrated for dimension i.

Given the system, we can isolate m in each inequality in the form

$$m \lessgtr_i \frac{out[i]}{s(p)[i]} , \qquad (4)$$

and find the value of m that maximizes the number of inequalities satisfied. The value in the right side defines an upper or lower bound for m if '$\lessgtr_i$' is a '<' or '>', respectively. Figure 2 presents the GD operator in action. Initially the individuals are distributed unequally in both dimensions (showed by the number in the coloured rectangles in Figure 2b). After the operator is applied to individual p, it is moved to a position that balances the distribution of individuals in both dimensions (Figure 2c).

Algorithm 1 presents the procedure to build the system of inequalities. Given the arrays $count_{gt}$ and $count_{lt}$, it iterates through the dimensions calculating the inequalities. The method isolates m (line 4) and stores the right side of the inequality along with its type (*'lb'* and *'ub'* for lower and upper bounds, respectively) in the set B (lines 5-14). Notice that when $s(p)[i] < 0$, we have to invert the inequality sign when m is isolated.

The next step corresponds to computing the value of m that satisfies the maximum number of inequalities, given its lower and upper bounds, presented in Algorithm 2. The procedure first sorts B by value in ascending order. The auxiliary variables *maxSatisfied*, *index* and *cSatisfied* store the number of inequalities satisfied by the best bound for m found so far, its index and the number of inequalities satisfied by the bound examined in the current iteration, respectively. The method starts by considering the interval before the first bound, i.e., $(-\infty, B[1].value)$. If a value from this interval is picked for m, all the upper bounds are satisfied, i.e., $maxSatisfied = n_{ub}$ (line 2). It then iterates over B counting the number of upper and lower bounds satisfied by each interval $(B[i], \infty)$ (lines 5-13). If the examined value corresponds to an upper bound, we decrement the *cSatisfied* counter, since the interval in the right side of the bound does not satisfy it. On the other hand, if the examined value corresponds to a lower bound, the right side interval satisfies the bound and *cSatisfied* is incremented.

After finding the best interval for m, the procedure assigns an actual value for m (lines 14-24). If the best interval corresponds to $(-\infty, B[1])$, m takes the value of $B[1]$ subtracted by one[2]. If the best interval corresponds to $(B[|B|], \infty)$, m takes one added to the value of $B[|B|]$. Otherwise, the method selects a random value in the interval $(B[index], B[index+1])$. The value of m returned by Algorithm 2 is then used to move individual p in the semantic space. GD is applied during the evolution at every generation, right before other genetic semantic operators (crossover and mutation).

Notice that the process of finding the set of inequalities satisfied by m is deterministic. The only stochastic part of the method is the selection of a value between two bounds (line 24). This fact justifies the use of other operators (crossover and mutation) in order to further explore the search space. Although the location of the optimal solution is known, making decisions based only on the distance to the solution (equivalent to the training error) may lead to overfitting, as presented in other works [1, 6, 9].

The time complexity of the operator is $O(n \log n)$, where n is the number of the training cases, bounded by the sort method applied in line 4 of Algorithm 2 (we adopted merge sort). The construction of the arrays $count_{lt}$ and $count_{gt}$ occurs once per generation with time $O(n|P|)$, where $|P|$ is the population size.

5. EXPERIMENTAL ANALYSIS

In this section we present an experimental analysis of the geometric dispersion operator within the GSGP in a diver-

[2]Although we shift the value of m by one, any other constant could be adopted.

Algorithm 1: Build System of Inequalities

```
Input: Individual program ($p$), desired output ($out$),
       population distribution ($count_{gt}, count_{lt}$)
Output: Set of bounds for $m$ ($B$)
1  $B \leftarrow \{\}$;
2  for $i \leftarrow 1$ to $|s(p)|$ do          // Calculate the bounds
3      if $s(p)[i] \neq 0$ then
4          $value \leftarrow \frac{out[i]}{s(p)[i]}$;
5          if $count_{gt}[i] > count_{lt}[i]$ then
6              if $s(p)[i] > 0$ then
7                  $B \leftarrow B \cup (value, \text{'ub'})$;
8              else
9                  $B \leftarrow B \cup (value, \text{'lb'})$;
10         if $count_{gt}[i] < count_{lt}[i]$ then
11             if $s(p)[i] > 0$ then
12                 $B \leftarrow B \cup (value, \text{'lb'})$;
13             else
14                 $B \leftarrow B \cup (value, \text{'ub'})$;
15 return $B$;
```

Algorithm 2: Finds m

```
Input: Set of bounds for $m$ given by Alg. 1 ($B$)
Output: Individual multiplier ($m$)
1  $n_{ub} \leftarrow$ number of 'ub's in $B$;
2  $maxSatisfied \leftarrow cSatisfied \leftarrow n_{ub}$;
3  $index \leftarrow 0$;
4  Sort $B$ by value in ascending order;
5  for $i \leftarrow 1$ to $|B|$ do     // Find best interval for $m$
6      $bound \leftarrow B[i]$;
7      if $bound.type =$ 'lb' then
8          $cSatisfied \leftarrow cSatisfied + 1$ ;
9          if $cSatisfied > maxSatisfied$ then
10             $maxSatisfied \leftarrow cSatisfied$;
11             $index \leftarrow i$;
12     else
13         $cSatisfied \leftarrow cSatisfied - 1$ ;
14 if $index = 0$ then                    // Calculate $m$
15     if $B$ is empty then          // No need to move p
16         $m \leftarrow 1$;
17     else
18         $m \leftarrow B[1].value - 1$ ;            // Shift by one
19 else
20     if $index = |B|$ then
21         $m \leftarrow B[|B|].value + 1$ ;          // Shift by one
22     else
23         $\delta \leftarrow B[index+1].value - B[index].value$;
           // rnd() returns a random value in $(0, 1)$
24         $m \leftarrow B[index].value + \delta \cdot rnd()$;
25 return $m$;
```

sified collection of real-world and synthetic datasets. The results obtained by GSGP with the GD operator, from now on referred as GSGP$^+$, are compared with a canonical GP [2] and the GSGP without the use of the operator [5].

The experimental test bed, presented in Table 1, is composed of datasets selected from the UCI machine learning repository [12] and GP benchmarks [21]. For each real-world dataset, we performed a 5-fold cross-validation with 10 replications, resulting in 50 executions. For the synthetic datasets (except *keijzer-6* and *keijzer-7*), we generated 5 samples and, for each sample, applied the algorithms 10 times, resulting again in 50 executions. For keijzer-6 and keijzer-7, the test set is fixed, so we performed 50 executions. The categorical attributes, namely 'vendor name' and 'model name' from the *cpu* dataset and 'month' and 'day' from the *forestFires* dataset, were removed for compatibility purposes.

All executions used a population of 1000 individuals evolved for 2000 generations with tournament selection of size 10. The grow method [11] was adopted to generate the random functions inside the geometric semantic crossover and mutation operators, and the ramped half-and-half method [11] used to generate the initial population, both with maximum individual depth equals to 6. The function set included three binary arithmetic operators $(+, -, \times)$ and the analytic quotient (AQ) [15] as an alternative to the arithmetic division. The terminal set included the variables of the problem and constant values randomly picked from the interval $[-1, 1]$. The GP method employed the canonical crossover and mutation operators [11] with probabilities 0.9 and 0.1, respectively. GSGP employed the geometric semantic crossover for fitness function based on Manhattan distance and mutation operators, as presented in [5], both with probability 0.5.

The differences in the probabilities of applying the mutation and crossover operators in GP and GSGP/GSGP$^+$ are justified based on previous work. While GP typically applies mutation with probability much smaller than the crossover [2], GSGP adopts relatively high mutation rate in order to be able to effectively explore the search space [20].

5.1 Parameter tuning

As we are interested in understanding the impact of the GD operator, we fix the other GSGP parameters and focus on looking at the results as we varied the probability values of GD. Note that the proposed operator cannot be used in isolation or without the mutation operator (i.e. only with crossover). This is because GD works with the restriction that individuals have to be moved along the line that crosses the desired output and the individual to be moved. As individuals movements are restricted to this line, it can stagnate the search. Hence, mutation still plays an important role when GD is applied, while GD main advantage is its ability to move individuals in the semantic space to enhance the convex hull generated by the population. For this reason, all experiments reported in this paper use GD, crossover and mutation operators with their respective probabilities.

One thing we realized during preliminary experiments was that moving individuals around the semantic spaces in the initial generations has a high positive impact in search. However, this behaviour is not maintained during the whole evolutionary process, leading to negative or no effects in later generations. Results of these experiments are omitted due to space constraints. For this reason, we proposed to dynamically adjust the probability *pgd* of applying GD as follows:

Table 1: Median training RMSE of the GSGP$^+$ with different values of α and pgd_0 for the adopted test bed. The smallest RMSE for each dataset is presented in bold.

Dataset	$pgd_0 = 0.2$				$pgd_0 = 0.4$				$pgd_0 = 0.6$			
	$\alpha = 0$	$\alpha = 2$	$\alpha = 5$	$\alpha = 10$	$\alpha = 0$	$\alpha = 2$	$\alpha = 5$	$\alpha = 10$	$\alpha = 0$	$\alpha = 2$	$\alpha = 5$	$\alpha = 10$
airfoil*	1.5180	1.4754	**1.4517**	1.4600	1.6117	1.4912	1.4566	1.4566	1.8437	1.5306	1.4715	1.4654
bioavailability*	1.3029	1.1031	1.0323	1.0464	1.8538	1.1723	1.0593	**1.0241**	3.1383	1.3819	1.0892	1.0352
concrete*	2.8193	2.7705	2.7500	2.7460	2.9370	2.7824	2.7503	2.7462	3.0971	2.8285	2.7881	**2.7319**
cpu*	1.1506	0.9829	1.0197	1.0358	1.2904	1.1042	1.0083	**0.9685**	1.4872	1.2693	1.0930	1.0863
energyCooling*	1.0632	1.0203	0.9966	0.9995	1.1199	1.0520	1.0036	**0.9853**	1.2435	1.0769	1.0301	1.0086
energyHeating*	0.7577	0.7216	0.7329	**0.7025**	0.8191	0.7710	0.7440	0.7299	0.9325	0.8011	0.7448	0.7374
forestFires*	3.4611	3.4065	3.1822	3.2245	3.9684	3.5108	3.3149	**3.1620**	4.4722	3.5889	3.3112	3.2421
keijzer-5†	0.0481	0.0461	0.0447	0.0449	0.0430	0.0447	0.0443	0.0431	**0.0261**	0.0262	0.0311	0.0347
keijzer-6†	**0.0067**	0.0072	0.0069	0.0071	0.0092	0.0091	0.0097	0.0090	0.0095	0.0097	0.0086	0.0101
keijzer-7†	0.0166	0.0173	**0.0162**	0.0166	0.0195	0.0177	0.0192	0.0194	0.0178	0.0189	0.0173	0.0171
ppb*	0.0043	0.0033	**0.0028**	0.0029	0.0070	0.0041	0.0030	0.0028	0.0045	0.0055	0.0036	0.0030
towerData*	19.1502	18.6724	18.5735	**18.3659**	20.4999	19.1012	18.7814	18.6049	22.1271	19.7742	18.9411	18.6882
vladislavleva-1†	0.0122	0.0119	0.0118	0.0116	0.0128	0.0122	0.0120	0.0119	0.0148	0.0121	0.0118	**0.0116**
vladislavleva-4†	0.0401	0.0387	0.0383	**0.0376**	0.0428	0.0396	0.0386	0.0380	0.0476	0.0404	0.0389	0.0386
wineRed*	0.3397	0.3259	0.3217	**0.3192**	0.3731	0.3375	0.3265	0.3222	0.4225	0.3498	0.3296	0.3234
wineWhite*	0.5469	0.5388	0.5346	**0.5332**	0.5669	0.5447	0.5377	0.5350	0.5938	0.5533	0.5406	0.5363

* Real-world dataset   † Synthetic dataset

$$pgd_g = pgd_0 \cdot exp\left(\frac{-\alpha \cdot g}{g_{max}}\right) \quad , \tag{5}$$

where pgd_0 is the base probability, α is the decay rate, g is current generation index and g_{max} is total number of generations. Equation 5 ensures the probability of applying the operator decays exponentially with the generations.

We tested the application of GD with different initial probabilities, and here we show the best results, which were obtained by setting pgd_0 to 0.2, 0.4 and 0.6. For these three values, we vary the parameter α from Equation 5, which defines the decay rate of pgd_0 along the generations. Note that when α equals to 0, the decay function is not used, and pgd_0 becomes constant along the generations.

Table 1 presents the results of root mean squared error (RMSE) for different combinations of pgd_0 and α in the training set. The results show that lower values for pgd_0 $(0.2, 0.4)$, allied with higher α values (10) tend to reduce the training RMSE. This fact indicates the application of the GD operator with low probability only in the early generations can reduce the training RMSE in relation to other combination of values.

5.2 Experimental Results

After parameter tuning, GSGP$^+$ was evaluated considering three metrics: (i) the RMSE when compared to GSGP and GP; (ii) the distribution of the individuals along the dimensions of the semantic space; and (iii) the execution time. We start by discussing the results of error. Table 2 presents the median RMSE and IQR (Interquartile Range) in the training and test sets, according to 50 executions. GSGP$^+$ results considered in this analysis were obtained with the values of α and pgd_0 generating the smallest median training RMSE in each of the datasets, presented in bold in Table 1. When analysing the differences between the median test and training RMSE's of GSGP$^+$ and GSGP, we observe that both present similar generalization power, with a small increase on the test error when compared to the training error.

We adopted the less conservative variant of the Friedman test proposed by Iman and Davenport [10], here called adjusted Friedman test, to analyse the statistical difference of the results. We performed an adjusted Friedman test under the null hypothesis that the performance of the methods—measured by their median test RMSE—are equal, obtaining the *p-value* 1.68×10^{-5}, which implies in discarding the null hypothesis with a confidence level of 95%.

As the null hypothesis was discarded, a Finner post hoc test was carried out with a confidence level of 95% in order to compute the adjusted *p-value* regarding the probability of the performance of GSGP$^+$ differs from each one of the other methods. The average ranks calculated for the statistical tests and the adjusted *p-values* (APV) resulting from the Finner test are presented in the last two rows of the Table 2, respectively. Given that the GSGP$^+$ obtained the smaller average rank among the three methods along with the APV's obtained by comparing it to the other methods, lead us to conclude that, with a confidence level of 95%, GSGP$^+$ performs better in terms of median test RMSE than the other methods.

The second aspect we evaluate is whether GD actually improves the distribution of the population around the desired output *out* or not. For that, we propose a new measure, called mean dimension distribution (MDD). For each dimension d of the training set, it is defined as

$$mdd(P, d) = abs\left(\frac{1}{|P|}\sum_{i=1}^{|P|} ge(s(p_i)[d], out[d]) - 0.5\right) \quad , \tag{6}$$

where P is the population, $|P|$ is the population size, p_i is its i-th individual, $ge(a, b)$ returns 1 if $a \geq b$ and 0 otherwise and $abs(a)$ returns the absolute value of a. MDD is defined in the $[0, 0.5]$ interval. A $mdd(P, d) = 0$ means that P is evenly distributed in dimension d, i.e., half the individuals are greater than or equal to and half are smaller than *out* in dimension d, while $mdd(P, d) = 0.5$ means the population is badly distributed, i.e., all individuals are greater than or equal to *out* and none is smaller than *out* in the dimension d or vice-versa.

Figure 3a shows an analysis of the MDD throughout the generations for dataset bioavailability using a heatmap. We reported the mean of the values resulting from ten replications of the first fold adopted in the test bed, using the best parameter combination from Section 5.1. The values in

Table 2: Training and test RMSE's (median and IQR) obtained by the algorithms for each dataset.

Dataset	GSGP$^+$				GSGP				GP			
	Training		Test		Training		Test		Training		Test	
	Median	IQR	Median	IQR	Median	IQR	Median	IQR	Median	IQR	Median	IQR
airfoil	1.886	0.041	2.131	0.243	7.885	0.527	8.417	0.757	6.389	6.004	5.959	5.972
bioavailability	9.695	0.690	30.860	4.426	9.885	0.689	30.969	2.507	34.364	4.971	35.404	7.387
concrete	3.654	0.118	5.144	0.635	3.647	0.138	5.394	0.642	8.330	1.152	8.534	1.044
cpu	6.151	0.905	30.837	14.563	6.126	0.665	30.917	15.185	24.809	4.777	34.664	14.756
energyCooling	1.271	0.066	1.531	0.159	1.257	0.070	1.515	0.147	3.201	0.204	3.242	0.242
energyHeating	0.798	0.083	0.971	0.128	0.802	0.113	0.956	0.185	2.875	0.127	2.818	0.403
forestfires	30.967	4.201	50.227	48.373	30.737	4.626	51.632	48.166	36.671	13.406	59.572	59.438
keijzer-5	0.026	0.008	0.028	0.009	0.045	0.003	0.049	0.005	0.015	0.006	0.016	0.005
keijzer-6	0.007	0.006	0.281	0.282	0.007	0.005	0.398	0.339	0.025	0.021	0.271	0.217
keijzer-7	0.016	0.009	0.017	0.009	0.017	0.010	0.018	0.010	0.035	0.023	0.035	0.021
ppb	0.954	0.305	28.568	6.170	0.917	0.266	28.740	5.290	27.082	1.523	29.202	5.596
towerData	20.405	0.621	21.769	1.252	20.436	0.610	21.920	1.272	57.783	5.272	58.397	5.155
vladislavleva-1	0.012	0.002	0.044	0.030	0.012	0.002	0.044	0.030	0.026	0.012	0.073	0.023
vladislavleva-4	0.038	0.001	0.051	0.004	0.038	0.001	0.052	0.003	0.169	0.011	0.170	0.011
wineRed	0.494	0.011	0.615	0.046	0.493	0.011	0.620	0.040	0.659	0.016	0.653	0.048
wineWhite	0.642	0.003	0.696	0.015	0.641	0.003	0.696	0.014	0.750	0.008	0.754	0.022
Avg. rank*	1.250				2.063				2.688			
Finner APV*	—				0.0216				0.0001			

* Regarding the median test RMSE

Table 3: Median elapsed time (in seconds) of the GSGP and GSGP$^+$. All differences are statistically significant according to a Wilcoxon test with a confidence level of 95%.

Dataset	GSGP$^+$	GSGP	Difference (%)
airfoil	1708.3	1523.9	12.10
bioavailability	421.0	387.6	8.62
concrete	1229.7	1195.0	2.90
cpu	327.4	313.4	4.48
energyCooling	852.9	801.1	6.47
energyHeating	826.8	782.3	5.70
forestfires	585.5	555.8	5.34
keijzer-5	10579.3	9287.9	13.90
keijzer-6	234.0	224.5	4.23
keijzer-7	1021.2	995.1	2.62
ppb	215.4	203.4	5.92
towerData	5357.9	4997.3	7.22
vladislavleva-1	1847.4	1870.3	-1.23
vladislavleva-4	5344.2	5074.2	5.32
wineRed	1597.2	1447.7	10.33
wineWhite	4690.9	4339.7	8.09

each cell correspond to the difference between the MDD obtained by the GSGP$^+$ and GSGP, i.e., values close to -0.5 indicate the dimension is better distributed in the GSGP$^+$, values close to 0.5 indicate the opposite and values close to 0 indicate similar behaviour. The x axis corresponds to the number of generations (only values multiple of 10 are considered), and the y axis represents different training cases (i.e., all dimensions of the semantic space). As expected, GSGP$^+$ population is better distributed than the GSGP population in the initial generations (blue region in the left side). However, this behaviour does not persist along the evolution, given the decaying probability of the GD. After 250 generations, approximately, both methods present a similar MDD, denoted by the high concentration of green in the remainder of the heatmap.

Figures 3b and 3c show the evolution of the fitness of the best individual along the generations in the training and test sets for GSGP and GSGP$^+$, respectively. Note that, at the begining of the generations, when the probability of applying the GD operator is higher, the error in the training set drops quicker for GSGP$^+$ than GSGP. However, as the operator becomes less applied, the curve from GSGP approximates quickly. Something similar happens for the test set, but note that from generation 700 on the error starts to grow. However, GSGP errors are overall higher than GSGP$^+$.

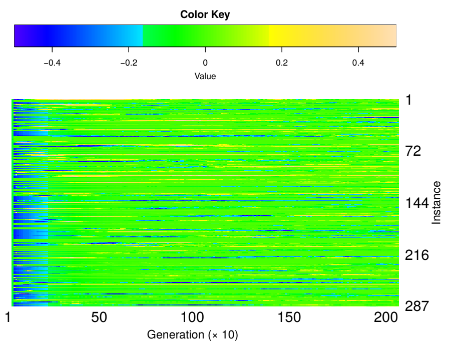
The third point to be analysed is the impact of the operator on the execution time of GSGP. Table 3 presents the time elapsed during the execution of GSGP$^+$ and GSGP, considering both training and test stages. Except for vladislavleva-1, the use of GD increases the execution time of the GSGP from 2.62% to 13.9%, depending on the dataset. The trade-off between the improvement on RMSE and the increase on computational time needs to be further analysed, and more efficient versions of the operator can be designed.

6. CONCLUSIONS

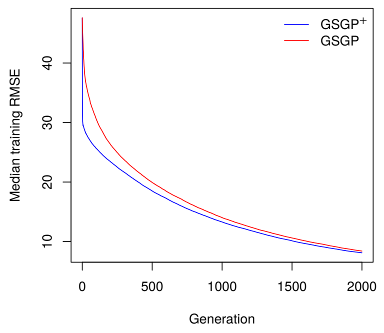
This paper presented a new operator designed to better distribute the population around the desired output vector during the evolutionary process performed by GSGP. The geometric dispersion (GD) operator moves a given individual along a line in the semantic space to regions with low concentration of individuals.

Experiments were performed in a test bed composed of sixteen datasets from synthetic and real domains. GSGP with the GD operator was compared to GSGP without it and a canonical GP in terms of median test RMSE. The reported results showed GSGP with the GD operator performs significantly better than both methods. We also analysed the GD impact on the population distribution around the target output. The analysis showed that the operator promoted a better distribution of the population around the desired output in relation to GSGP in the initial generations, where the probability of applying it was higher.

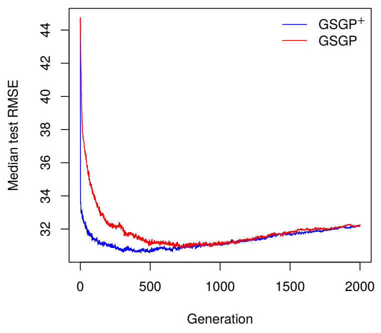
Potential future developments include investigating the use of other operations instead of multiplication to move individuals with the GD operator, and the development of more sophisticated implementations that might reduce differences of computational time when it is compared to GSGP. Furthermore, an analysis of the impact of including information about the distribution of the population in other stages of the evolution, such as in the selection phase, are worth investigating.



(a) Heatmap with the differences between GSGP$^+$ and GSGP according to MDD for each instance in generations numbers multiple of 10.



(b) Median RMSE in the training set.



(c) Median RMSE in the test set.

Figure 3: MME Heatmap and evolution of the RMSE over the generations for GSGP$^+$ and GSGP in the dataset bioavailability.

7. ACKNOWLEDGEMENTS

The authors would like to thank CAPES, CNPq (141985/2015-1) and Fapemig for their financial support.

8. REFERENCES

[1] J. Albinati, G. L. Pappa, F. E. B. Otero, and L. O. V. B. Oliveira. The effect of distinct geometric semantic crossover operators in regression problems. In *Proc. of EuroGP*, pages 3–15, 2015.

[2] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming — an Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, 1998.

[3] L. Beadle and C. G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Prog. and Evolvable Machines*, 10(3):307–337, Sep 2009.

[4] J. Botzheim, C. Cabrita, L. T. Kóczy, and A. E. Ruano. Genetic and bacterial programming for b-spline neural networks design. *Journal of Advanced Computational Intelligence*, 11(2):220–231, 2007.

[5] M. Castelli, S. Silva, and L. Vanneschi. A C++ framework for geometric semantic genetic programming. *Genetic Prog. and Evolvable Machines*, 16(1):73–81, Mar 2015.

[6] M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, and P. Legrand. Geometric semantic genetic programming with local search. In *Proc. GECCO'15*, pages 999–1006. ACM, 2015.

[7] M. Castelli, L. Vanneschi, and S. Silva. Semantic search-based genetic programming and the effect of intron deletion. *Cybernetics, IEEE Trans. on*, 44(1):103–113, Jan 2014.

[8] M. Castelli, L. Vanneschi, S. Silva, and S. Ruberto. How to exploit alignment in the error space: Two different GP models. In *Genetic Programming Theory and Practice XII*, pages 133–148. 2015.

[9] I. Gonçalves, S. Silva, and C. M. F. Fonseca. On the generalization ability of geometric semantic genetic programming. In *Proc. of EuroGP*, pages 41–52, 2015.

[10] R. L. Iman and J. M. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595, 1980.

[11] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, volume 1. MIT Press, 1992.

[12] M. Lichman. UCI machine learning repository, 2015. http://archive.ics.uci.edu/ml.

[13] A. Moraglio. Abstract convex evolutionary search. In *Proc. of the 11th FOGA*, pages 151–162, 2011.

[14] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In *Proc. of PPSN XII*, volume 7491, pages 21–31. 2012.

[15] J. Ni, R. H. Drieberg, and P. I. Rockett. The use of an analytic quotient operator in genetic programming. *Evolutionary Computation, IEEE Trans. on*, 17(1):146–152, Apr 2013.

[16] T. P. Pawlak. *Competent Algorithms for Geometric Semantic Genetic Programming*. PhD thesis, Poznan University of Technology, Poznan, Poland, 2015.

[17] S. Roman. *Advanced linear algebra*, volume 135 of *Graduate Texts in Mathematics*. Springer New York, 2nd edition, 2005.

[18] S. Ruberto, L. Vanneschi, M. Castelli, and S. Silva. ESAGP - a semantic GP framework based on alignment in the error space. In *Proc. of EuroGP*, pages 150–161, 2014.

[19] L. Vanneschi, M. Castelli, and S. Silva. A survey of semantic methods in genetic programming. *Genetic Prog. and Evolvable Machines*, 15(2):195–214, 2014.

[20] L. Vanneschi, S. Silva, M. Castelli, and L. Manzoni. Geometric semantic genetic programming for real life applications. In R. Riolo et al., editors, *Genetic Prog. Theory and Practice XI*, pages 191–209. 2014.

[21] D. White, J. McDermott, M. Castelli, L. Manzoni, B. Goldman, G. Kronberger, W. Jaśkowski, U. M. O'Reilly, and S. Luke. Better GP benchmarks: community survey results and proposals. *Genetic Prog. and Evolvable Machines*, 14(1):3–29, Mar 2013.