A Generic Framework for Building Dispersion Operators in the Semantic Space

Luiz Otavio V. B. Oliveira, Fernando E. B. Otero, Gisele L. Pappa

Abstract This chapter proposes a generic framework to build geometric dispersion (GD) operators for Geometric Semantic Genetic Programming in the context of symbolic regression, followed by two concrete instantiations of the framework: a multiplicative geometric dispersion operator and an additive geometric dispersion operator. These operators move individuals in the semantic space in order to balance the population around the target output in each dimension, with the objective of expanding the convex hull defined by the population to include the desired output vector. An experimental analysis was conducted in a testbed composed of sixteen datasets showing that dispersion operators can improve GSGP search and that the multiplicative version of the operator is overall better than the additive version.

Key words: geometric semantic genetic programming, dispersion operators, diversity

1 Introduction

The role of the crossover operator in tree-based genetic programming has been a discussion point for a long time [2], as many researchers believed the lack of context associated with the tree nodes makes crossover to resemble a macro mutation. In semantic genetic programming algorithms, in particular their geometric counterparts, this is mitigated by making syntactic modifications more semantically-aware—i.e., focusing on how syntactic modifications reflect on the semantics of the individuals.

Luiz Otavio V. B. Oliveira
DCC, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Fernando E. B. Otero
School of Computing, University of Kent, Chatham Maritime, UK

Gisele L. Pappa
DCC, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

This chapter deals with the problem of symbolic regression, where the semantics of an individual is defined as a point in a n-dimensional space, called semantic space, and n is the number of examples in the training set. In geometric semantic genetic programming (GSGP), the geometric semantic crossover and mutation operators [13] guarantee that the semantic fitness landscape explored by the GP is conic, which has a positive impact in the search process. The problem is then how long GSGP might take to find the optimum.

The challenge of finding the optimal solution or not is then dependent on other components of GSGP. For example, as the GSGP crossover operator produces offspring by performing a convex combination of its parents, the set of candidate individuals generated during evolution is delimited by the *convex hull*[1] of the semantics of the current population [16]. Hence, if the target output is not within the convex hull, the algorithm will never be able to find it using crossover alone. The mutation operator deals with this problem by expanding the convex hull. However, GGSP might take a prohibitive amount of time to get to the relevant regions of the search space depending on the distribution of the individuals in the initial generation.

In this context, [15] presented a heuristic operator to move individuals through the semantic space in order to, hopefully, include the target output inside the convex hull defined by the current population. The operator, called geometric dispersion (GD), applies multiplicative constants to the individuals aiming to balance the proportion of the population on the left and right side of the target output in each dimension of the semantic space.

In this same direction, this chapter proposes a generic framework for geometric dispersion operators allowing different mathematical operations to redistribute the population. The operation used to add the constant to the individual has direct impact on the way it is moved through the space. Thus, other operations, besides the multiplication used in the original GD, allow the resulting individual to reach other regions of the semantic space with different effects on the search. The framework is used to build a geometric dispersion operator based on the addition operation and evaluates the impact of the new operator on the evolution. We performed an experimental analysis in a test bed composed of sixteen datasets. We compared the results obtained by GSGP with the multiplicative and the additive versions of the geometric dispersion, tested separately, and with the GSGP without the dispersion operators. Results indicate dispersion operators have a positive impact on the search, improving the root mean square error in relation to the GSGP without this operator.

The remaining of this chapter is organised as follows. Section 2 provides an overview to GSGP for symbolic regression problems and the crossover limitation regarding the population's convex hull. Section 3 reviews previous works involving the convex hull described by the population in GSGP and Section 4 presents a framework for GD operators along with two particular implementations. Section 5 presents the experimental analysis in sixteen different datasets followed by conclusions and research directions in Section 6.

[1] The *convex hull* of a set of points is given by the set of all possible convex combinations of these points [18].

2 Background

Most of genetic programming algorithms employ *traditional* genetic operators that perform syntactic modification on individuals in order to change their behaviour—the behaviour of an individual is referred to as its semantics. One particular drawback of traditional genetic operators is that there is no guarantee that syntactic modifications will lead to different behaviour. Therefore, they represent an indirect way of changing the semantics of an individual. Geometric semantic genetic programming (GSGP) [13], on the other hand, employ semantic genetic operators to introduce syntactic modification on individuals that guarantee to change their semantics.

In this chapter, we focus on GSGP applied to symbolic regression problems. Symbolic regression problems can be seen as a supervised learning procedure: given a finite set of input-output pairs representing the fitness 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 $\forall (\mathbf{x}_i, y_i) \in T : p(\mathbf{x}_i) = y_i$.

Let $I = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ and $O = [y_1, y_2, \ldots, y_n]$ be the input and the output vectors[2], respectively, associated to the fitness cases. The semantics of a program p represented by an individual evolved by GSGP, denoted as $s(p)$, is the vector of outputs it produces when applied to the set of inputs I, i.e., $s(p) = p(I) = [p(\mathbf{x}_1), p(\mathbf{x}_2), \ldots, p(\mathbf{x}_n)]$. 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 space S, referred to as the semantic space, where n is the number of fitness cases. Note that the desired output vector O can also be represented in the semantic space.

GSGP employs semantic geometric operators to evolve the individuals in a population. Let P' be the solution set comprising all the possible candidate solutions to a problem in the real domain, the geometric semantic crossover and mutation operators are defined as follows:

Definition 1. Given two parent programs $p_1, p_2 \in P'$, the geometric semantic crossover for the space of real functions $GSX : P' \times P' \to P'$ returns the real function

$$p_3 = r \cdot p_1 + (1 - r) \cdot p_2 \quad , \tag{1}$$

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 program $p \in P'$, the geometric semantic mutation for the space of real functions $GSM : P' \times \mathbb{R}^+ \to P'$ with mutation step ε returns the real function

[2] Note that when $\mathbf{x}_i \in \mathbb{R}^d$ with $d > 1$ the vector I becomes a matrix with dimensions $d \times n$. We allow an abuse of notation by representing the matrix as a vector with dimension n, where each element corresponds to a vector of dimension d.

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

where r_1 and r_2 are random real functions.

An interesting characteristic of GSGP is that the fitness of an individual p is the distance of its output vector $s(p)$ to the desired output vector O. Therefore, the fitness landscape induced by semantic genetic operators is unimodal by construction [13]. Despite the unimodal fitness landscape, the stochastic nature of these operators—as a result of using random real functions and constants—has been shown to be a more suitable way to explore the space in terms of generalisation, when compared to modifications of these operators where decisions are based on fitness cases error [1, 6, 10]. The area defined by the set of individuals (points in the semantic space) define the convex hull of the population:

Definition 3. The *convex hull* of a set H of points in $\mathbb{R}^n$, denoted as $C(H)$, is the set of all convex combinations of points in H [18].

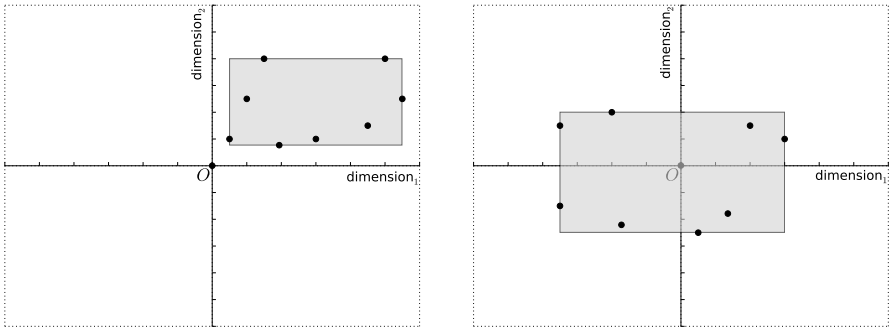
Let P be a population of individuals, we adopt the notation $C(s(P))$ to denote the convex hull of the set composed by the semantics of the individuals of P, i.e., $s(P)$. Since GSX is, by definition, a geometric crossover operator [13], we have the following theorem regarding the convex hull of the population:

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 $C(s(P_{g+1})) \subseteq C(s(P_g)) \subseteq \ldots \subseteq C(s(P_1)) \subseteq C(s(P_0))$.*

Theorem 1 is a particular case of the Theorem 3 defined and proved by [12], and it has an important implication regarding the GSX operator. Given a population P and a semantic vector q in S, the offspring resulting from the application of GSX to any pair of individuals in P can reach q if and only if $q \in C(s(P))$. Consequently, if GSGP has no other search operators (only GSX), a semantic vector q is reachable only if $q \in C(s(P_0))$—i.e., if q is located inside the convex hull of the initial population.

Figure 1 illustrates this situation for a two-dimensional semantic space. Without loss of generality let $O = [0,0]$ be the desired output vector defined by the training cases. Now consider two different populations P_a and P_b, where the individuals from P_a are concentrated in the upper-right side of O and, consequently, $C(s(P_a))$ cannot reach the origin O. On the other hand, the set $s(P_b)$ is distributed around the desired output such that $O \in C(s(P_b))$. In the first scenario, GSGP needs a mutation operator to expand the convex hull to reach O. In the second scenario, the desired vector O can be reached using a crossover operator alone, as it is already inside the convex hull, or it can be calculated analytically[3] with no need to use GSGP.

[3] The coefficients of convex combinations can be found by means of Gaussian elimination [9].



(a) P_a: population concentrated into a single quadrant

(b) P_b: population encompasses solutions in all quadrants

Fig. 1 Example of different distributions of a population in a two-dimensional semantic space. The desired output O is located in the origin of the space and the shaded area corresponds to the convex hull under the Manhattan distance.

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 two have investigated ways to increase the area covered by the population convex hull—in particular 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, Ruberto et al. [19] 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 $t_e(p) = s(p) - O$. 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 O. In this context, they present GP-based methods to find optimally aligned individuals in two and three dimensions, called ESAGP-1 (Error Space 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.

Castelli et al. [8], in contrast, extend ESAGP-1 to what they called 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 that consider the area covered by the convex hull, Pawlak [16] proposed 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 randomly picks a node from the function set to combine individuals already in the population. If the resulting program has semantics different from other individuals 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, following 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 both SDI and CI methods is the possible waste of resources, since individuals are randomly created, evaluated and discarded when they are semantically similar to an existing individual of the population or when it is already in the population's convex hull.

The Semantic Geometric Initialization (SGI) [17], on the other hand, generates a set S of semantics, such that the desired output is guaranteed to belong to the convex hull of S. These semantics are generated by adding or subtracting an offset to O in different combinations of the semantic space dimensions. Then, for each semantics $s_i \in S$, the method generates an individual whose semantics is equal to s_i. The synthesis of these individuals is domain dependent and authors presented methods to generate individuals for symbolic regression domain—by polynomial interpolation—and for boolean domain—by a boolean formula. The experimental analysis indicates that SGI can achieve training error significantly smaller than the ramped half-and-half method in symbolic regression and boolean problems. However, the test error achieved by SGI is significantly higher than the error achieved by ramped half-and-half [11], which indicates that SGI is very susceptible to overfitting.

Although not taking advantage of the geometric properties of the search space, Castelli et al. [7] proposed 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.

4 Geometric Dispersion Operators

In this section we present a generic geometric dispersion (GD) framework along with two implementations, the multiplicative geometric dispersion (MGD) [15] and the additive geometric dispersion (AGD) operators.

4.1 A Framework for Geometric Dispersion Operators

This section presents a general framework for geometric dispersion (GD)[4] operators aiming to redistribute the population around the desired output vector O in the semantic space. These operators move a given individual to the region of the semantic space around O with the lowest concentration of individuals in order to, hopefully, modify the convex hull of the population to contain the desired output.

GD operators adopt a greedy strategy to redistribute the population around O by examining each dimension of S separately. For each dimension of the semantic space, GD computes the proportion of individuals whose semantics is greater than and less than O for that dimension. The method uses this information to move the individuals through the semantic space—by means of mathematical operations applied to the individual's program—in order to balance each one of the dimensions of S.

When we know the region of the semantic space around O where we want to have individuals shifted to, different methods can be used to move individual p. GD operators do that by applying a constant m to p through a mathematical operation $\oplus$, in the form $m \oplus p$. The movement performed by the GD operator depends directly of the chosen operation for $\oplus$. Thus, the value of m must be chosen such that the displacement of p benefits the largest number of dimensions.

The process of finding this value is equivalent to find m that solves the inequality system:

$$\begin{cases} m \oplus s(p)[1] \lesseqgtr_1 O[1] \\ m \oplus s(p)[2] \lesseqgtr_2 O[2] \\ \dots \\ m \oplus s(p)[k] \lesseqgtr_k O[k] \end{cases} \tag{3}$$

where '$\lesseqgtr$' is a inequality operator ('$<$' or '$>$') chosen according to the asymmetry of the population.

Let GT (greater than) and LT (less than) be n-element arrays, where the i-th element corresponds to the number of individuals p_k in the current population P where $s(p_k)[i] > O[i]$ and $s(p_k)[i] < O[i]$, respectively. If $GT[i] > LT[i]$, the population is unbalanced with more individuals in the right side of the desired output vector in the i-th dimension of the semantic space, and the individual should be moved to the left side of O—the symbol '$\lesseqgtr_i$' is replaced by '$<$'. Otherwise, if $GT[i] < LT[i]$, the imbalance occurs in the opposite side, i.e., the population is concentrated on the left side of O in the dimension i and the individual should be moved to the opposite side of O—the symbol '$\lesseqgtr_i$' is replaced by '$>$'. Note that when $GT[i] = LT[i]$, no inequalities are added to the system. Therefore the number of inequalities in Eq. 3 is less or equal to the number of dimensions, i.e., $k \leq n$.

[4] [15] presents the first geometric dispersion operator. However, this operator is a particular case of the framework presented in this paper. Hence, hereafter their operator is referred as multiplicative geometric dispersion (MGD) operator in contrast to the GD framework.

Algorithm 1 GD procedure to build the system of inequalities

Require: Individual program (p), desired output (O), population distribution (GT, LT)
1: $B \leftarrow \{\}$
2: **for** $i \leftarrow 1$ **to** $|s(p)|$ **do** ▷ Calculate the bounds
3: **if** $GT[i] \neq LT[i]$ **then**
4: **if** $GT[i] > LT[i]$ **then**
5: *inqSign* $\leftarrow$ '*lessThan*' ▷ '$\lesseqgtr_i$' is replaced by '$<$'
6: **else**
7: *inqSign* $\leftarrow$ '*greaterThan*' ▷ '$\lesseqgtr_i$' is replaced by '$>$'
8: **end if**
9: Isolate m in the left side of the inequality ▷ $m \lesseqgtr_i O[i] \ominus s(p)[i]$
10: *bound* $\leftarrow$ right side value ▷ *bound* $\leftarrow O[i] \ominus s(p)[i]$
11: **if** *inqSign* $=$ '*lessThan*' **then**
12: Add *bound* to B as upper bound
13: **else**
14: Add *bound* to B as lower bound
15: **end if**
16: **end if**
17: **end for**
18: **return** B

However, due to the large number of inequalities in the system, usually it does not admit feasible solutions. Thus, instead of finding a value for m that satisfies all inequalities, the operator finds one that maximizes the number of satisfied inequalities. [15] present algorithms to both construct the system of inequalities and find the value of m that satisfies the largest number of inequalities when the mathematical operation adopted is the multiplication, i.e., $\oplus$ is $\times$ (times). We generalise these algorithms and present a framework, called geometric dispersion (GD), which moves individuals through the semantic space in order to distribute the population around O.

GD is independent of the arithmetic operation adopted in the inequalities. The only requirements are that the operation is binary and allows inverse. Let $\oplus$ and $\ominus$ be a binary operation and its inverse, respectively. The variable m can be isolated in the left side of the system of inequalities of Eq. 3 as showed in Eq. 4, such that the operator can find the value that satisfies the largest number of inequalities.

$$\begin{cases} m \lesseqgtr_1 O[1] \ominus s(p)[1] \\ m \lesseqgtr_2 O[2] \ominus s(p)[2] \\ \ldots \\ m \lesseqgtr_k O[k] \ominus s(p)[k] \end{cases} \tag{4}$$

Algorithm 1 introduces the procedure to define the system of inequalities. Given the arrays GT and LT, it checks each dimension i for an unbalanced distribution, i.e., where $GT[i] \neq LT[i]$. When these values differ, the method adds a new inequality to the system, represented by a bound value that should be satisfied.

The sign '$\lessgtr$' of the inequalities is defined according to the distribution of the population in the verified dimension (lines 4-8). If $GT[i] > LT[i]$, then '$\lessgtr_i$' is replaced by '$<$'. Otherwise, if $GT[i] < LT[i]$, it is replaced by '$>$'.

The next step of the method is to isolate m in the left side of the inequality and store the value of the right side in *bound* (lines 9-10). There are a few considerations in this step, according to the arithmetic operation used in the inequalities. E.g., if GD uses multiplication ($\oplus$ is $\times$), as presented by [15], the method must check for division by zero and negative value on the left side of the inequality. When a division by zero is found, the algorithm ignores the inequality. When the left side is negative, both sides of the inequality are multiplied by -1, inverting the inequality sign.

The sign of the inequalities is used to define the type of bound (lines 11-15). If the sign is '$<$', the value of m should be smaller than *bound* (it is an upper bound). Otherwise, m should be greater than *bound* (it is a lower bound). The bounds and their types are used to compute the value of m in Algorithm 2.

Algorithm 2 follows the method presented in [15]. It 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 until $(B[i], \infty)$ (lines 5-16). 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 17-30). If the best interval corresponds to $(-\infty, B[1])$ or to $(B[|B|], \infty)$, m takes the output of *getLeftExtreme* and *getRightExtreme*, respectively. Otherwise, the method selects a random value in the interval $(B[index], B[index+1])$.

Algorithms 3 and 4 present the procedures *getRightExtreme* and *getLeftExtreme*, respectively. The control variable *shiftOne* indicates if the methods should use the same strategy adopt by [15], i.e., shift values in the extreme of the interval by one. Otherwise, the algorithms shift the values by a random value proportional to the closest interval defined in B.

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). The probability of applying a GD operator—individual-wise—pgd, as proposed by [15], is given by:

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

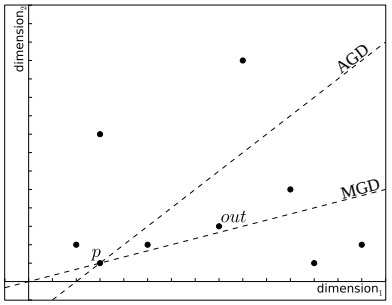
Algorithm 2 Finds m

Require: Set of bounds for m (B), shift control variable (*shiftOne*)
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 the best interval for m
6: $bound \leftarrow B[i]$
7: **if** *bound* is a lower bound **then**
8: $cSatisfied \leftarrow cSatisfied + 1$
9: **if** $cSatisfied > maxSatisfied$ **then**
10: $maxSatisfied \leftarrow cSatisfied$
11: $index \leftarrow i$
12: **end if**
13: **else**
14: $cSatisfied \leftarrow cSatisfied - 1$
15: **end if**
16: **end for**
17: **if** $index = 0$ **then** ▷ Calculate m
18: **if** B is empty **then** ▷ No need to move p
19: $m \leftarrow 1$
20: **else**
21: $m \leftarrow getLeftExtreme(B, shifOne)$
22: **end if**
23: **else**
24: **if** $index = |B|$ **then**
25: $m \leftarrow getRightExtreme(B, shifOne)$
26: **else**
27: $\delta \leftarrow B[index+1].value - B[index].value$
28: $m \leftarrow B[index].value + \delta \cdot rnd()$ ▷ $rnd()$ returns a random value in $(0,1)$
29: **end if**
30: **end if**
31: **return** m

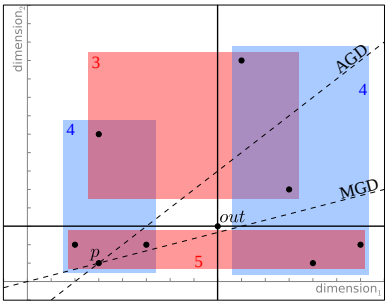
where pgd_0 is the base probability, α is the decay rate, g is current generation index and g_{max} is total number of generations. Equation 5 ensures the probability of applying the operator decays exponentially with the generations.

4.2 Multiplicative Geometric Dispersion

The geometric dispersion operator proposed in [15], here called multiplicative geometric dispersion (MGD), is an implementation of the GD framework where the constant m is multiplied by the semantics of the individual p. MGD manipulates inequality systems as given by Eq. 3—in the form $m \cdot s(p)[i] \leqslant_i O[i]$—and isolates m in the left side as presented by Eq. 4—in the form $m \leqslant_i O[i]/s(p)[i]$—, where $\oplus$ and $\ominus$ are replaced by $\times$ and $\div$, respectively. The multiplicative operation applied



(a) AGD and MGD lines in the semantic space.



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

Fig. 2 Lines described by the AGD and MGD operators applied to an individual p.

to the individual p is geometrically equivalent to moving it through the line crossing both $s(p)$ and the origin of S.

As discussed above, when isolating m in the i-th dimension, MGD must consider two special cases. First, if $s(p)[i] = 0$, isolating m implies in division by zero and the operator ignores the inequality. Second, if $s(p)[i] < 0$, the inequality is multiplied by -1 and the inequality sign is inverted. For instance, let $m \cdot (-2) > 4$ be one of the inequalities. Thus, as $s(p)[i] = -2 < 0$, the inequality is multiplied by -1 before isolating m in the left side, leading to $m \cdot 2 < -4$.

4.3 Additive Geometric Dispersion

Besides the MGD, we present a geometric dispersion operator based on addition. The additive GD (AGD) moves a given individual p through the line $L = \{s(p) + t : t \in \mathbb{R}\}$, with $L \subset S$, in order to redistribute the population around O. The inequalities used by AGD are in the form $m + s(p)[i] \lesseqgtr O[i]$, which result in $m \lesseqgtr O[i] - s(p)[i]$, where i is the dimension analysed.

The use of different mathematical operations within the GD operators allows them to explore different regions of S. Figure 2 presents an example in a two-dimensional semantic space. In order to keep $dimension_1$ balanced and balance $dimension_2$, it is necessary to move p to the upper-left side of O. However, in this example, only AGD can reach this region of the space.

Algorithm 3 *getRightExtreme* procedure

Require: Set of bounds for m (B), control variable *shiftOne*

1: **if** *shifOne*=TRUE **or** $|B| < 2$ **then**
2: **return** $B[|B|] + 1$
3: **else**
4: $\delta \leftarrow B[|B|].value - B[|B|-1].value$
5: **return** $B[|B|] + \delta \cdot rnd()$
6: **end if**

Algorithm 4 *getLeftExtreme* procedure

Require: Set of bounds for m (B), control variable *shiftOne*

1: **if** *shifOne*=TRUE **or** $|B| < 2$ **then**
2: **return** $B[1] - 1$
3: **else**
4: $\delta \leftarrow B[2].value - B[1].value$
5: **return** $B[1] - \delta \cdot rnd()$
6: **end if**

5 Experimental Analysis

This section presents an empirical analysis of the effect of different versions of the GD operator within GSGP. We compare the results obtained by GSGP with AGD (referred to as GSGP+A), GSGP with MGD [15] (referred to as GSGP+M) and GSGP without dispersion operators [5] in a test bed of sixteen symbolic regression datasets comprising both real-world and synthetic problems, as presented in Table 1. The test bed along with parameters adopted in the algorithms are the same from our previous work [15].

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 five different sets 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 1,000 individuals evolved for 2,000 generations with tournament selection of size 10. The same random seed is employed to initialize the pseudorandom number generator in all methods. 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) [14] as an alternative to the arithmetic division. The terminal set included the variables of the problem and constant values randomly picked from the interval $[-1, 1]$. 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 base probability (pgd_0) and the decay rate (α) values for all the GD variants are the ones leading to the smaller median training RMSE, as presented in our previous experiments [15]. The values vary in each dataset, as presented in the last two columns of Table 1. The two ways of setting the final value of m in the Algorithms 3 and 4, defined by the boolean variable *shiftOne*, were analysed in different

configurations. A 'R' in the end of the configuration name indicates that *shiftOne* is `FALSE`, i.e., m is calculated as a random value proportional to the interval nearest to the extreme.

Table 1 and 2 present the median training and test RMSE and respective IQR (Interquartile Range), according to 50 executions. Table 3 shows the number of datasets were the method in the row is statistically better than the method in the column regarding the test RMSE, according to Wilcoxon test with 95% confidence level. The results indicate the search performed by GSGP benefits from the dispersion provided by the operators, as pointed out by the score of GSGP in relation the GD configurations. Regarding the use of the shift one algorithm or the random method to compute the values of m in the extremes, there are no significant differences on the dispersion operators. Lastly the results indicate that overall the multiplicative version of the geometric dispersion operator performs better than the additive counterpart.

Table 1 Training RMSE (median and IQR) obtained by the algorithms for each dataset. The last two columns present the parameters used as input in GD operators.

Dataset	**GSGP**		**GSGP+A**		**GSGP+AR**		**GSGP+M**		**GSGP+MR**		**GD param.**	
	Med.	**IQR**	**Med.**	**IQR**	**Med.**	**IQR**	**Med.**	**IQR**	**Med.**	**IQR**	α	pgd_0
airfoil*	7.885	0.527	1.873	0.057	1.900	0.052	1.886	0.041	1.890	0.051	5	0.2
bioavailability*	9.893	0.652	9.653	0.552	9.706	0.547	9.695	0.690	9.794	0.703	10	0.4
concrete*	3.647	0.138	3.659	0.136	3.644	0.150	3.654	0.118	3.634	0.109	10	0.6
cpu*	6.126	0.665	6.149	0.977	6.223	1.067	6.151	0.905	6.215	0.790	10	0.4
energyCooling*	1.257	0.070	1.282	0.065	1.267	0.057	1.271	0.066	1.255	0.052	10	0.4
energyHeating*	0.802	0.113	0.790	0.084	0.789	0.123	0.798	0.083	0.764	0.084	10	0.2
forestfires*	30.737	4.626	31.684	4.858	31.247	4.490	30.967	4.201	31.534	4.156	10	0.4
keijzer-5†	0.045	0.003	0.063	0.006	0.062	0.006	0.026	0.008	0.026	0.009	0	0.6
keijzer-6†	0.007	0.005	0.007	0.007	0.007	0.005	0.007	0.006	0.006	0.005	0	0.2
keijzer-7†	0.017	0.010	0.017	0.010	0.016	0.009	0.016	0.009	0.014	0.009	5	0.2
ppb*	0.917	0.266	0.930	0.241	0.924	0.274	0.954	0.305	0.937	0.202	5	0.2
towerData*	20.436	0.610	20.558	0.704	20.587	0.610	20.405	0.621	20.472	0.404	10	0.2
vladislavleva-1†	0.012	0.002	0.012	0.002	0.012	0.001	0.012	0.002	0.012	0.002	10	0.6
vladislavleva-4†	0.038	0.001	0.038	0.002	0.038	0.002	0.038	0.001	0.038	0.002	10	0.2
wineRed*	0.493	0.011	0.494	0.012	0.494	0.010	0.494	0.011	0.495	0.010	10	0.2
wineWhite*	0.641	0.003	0.642	0.004	0.642	0.004	0.642	0.003	0.641	0.003	10	0.2

* Real-world dataset † Synthetic dataset

6 Conclusion

This chapter presented a general framework to construct geometric dispersion (GD) operators for GSGP in the context of symbolic regression, followed by two concrete instantiations: the multiplicative geometric dispersion (GD) operator proposed in [15] and another derivation based on the addition operator. These operators move the individuals in order to balance the population around the target output in each dimension of the semantic space, with the objective of expanding the convex hull defined by the population to include the desired output vector.

Table 2 Test RMSE (median and IQR) obtained by the algorithms for each dataset.

Dataset	GSGP		GSGP+A		GSGP+AR		GSGP+M		GSGP+MR	
	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
airfoil	8.417	0.757	2.154	0.237	2.131	0.272	2.131	0.243	2.152	0.210
bioavailability	30.736	2.326	31.139	4.170	30.682	4.189	30.860	4.426	30.619	3.773
concrete	5.394	0.642	5.285	0.624	5.244	0.575	5.144	0.635	5.054	0.489
cpu	30.917	15.185	32.804	13.166	32.400	16.182	30.837	14.563	32.027	16.790
energyCooling	1.515	0.147	1.553	0.151	1.489	0.180	1.531	0.159	1.486	0.194
energyHeating	0.956	0.185	0.919	0.157	0.928	0.136	0.971	0.128	0.933	0.169
forestfires	51.632	48.166	52.026	48.626	51.483	50.444	50.227	48.373	50.590	48.762
keijzer-5	0.049	0.005	0.066	0.006	0.065	0.007	0.028	0.009	0.027	0.010
keijzer-6	0.398	0.339	0.275	0.228	0.293	0.203	0.281	0.282	0.250	0.166
keijzer-7	0.018	0.010	0.019	0.010	0.018	0.010	0.017	0.009	0.015	0.008
ppb	28.740	5.290	27.337	5.031	28.139	5.630	28.568	6.170	27.969	5.849
towerData	21.920	1.272	21.979	1.264	21.826	1.263	21.769	1.252	21.871	1.134
vladislavleva-1	0.044	0.030	0.041	0.030	0.046	0.022	0.044	0.030	0.039	0.025
vladislavleva-4	0.052	0.003	0.051	0.003	0.050	0.003	0.051	0.004	0.052	0.002
wineRed	0.620	0.040	0.614	0.041	0.610	0.042	0.615	0.046	0.619	0.049
wineWhite	0.696	0.014	0.695	0.015	0.696	0.014	0.696	0.015	0.696	0.013

Table 3 Number of datasets where the method in the row obtained statistically smaller test RMSE in relation to the method in the column. Results according to the Wilcoxon test with 95% confidence level.

	GSGP	GSGP+A	GSGP+AR	GSGP+M	GSGP+MR	Total (wins)
GSGP	–	1	1	0	0	2
GSGP+A	3	–	1	2	0	6
GSGP+AR	5	1	–	1	4	11
GSGP+M	6	3	5	–	3	17
GSGP+MR	7	4	4	3	–	18
Total (losses)	21	9	11	6	7	

Experimental analysis was performed on a test bed composed by sixteen datasets to compare the effects of GD operators within GSGP: GSGP with additive GD (GSGP+A), multiplicative GD (GSGP+M) and without GD operators were compared regarding the test RMSE. The results showed that GD operators can improve the search performance in terms of test RMSE. Also, they showed that GSGP+M presents advantage over the GSGP+A regarding test RMSE.

Future works include proposing novel dispersion operators following the generic framework, exploring different algorithms to compute the value of m, analysing the impact of using different GD operators simultaneously and tuning the control parameters used by GD operators.

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

References

1. Albinati, J., Pappa, G.L., Otero, F.E.B., Oliveira, L.O.V.B.: The effect of distinct geometric semantic crossover operators in regression problems. In: Proc. of EuroGP, pp. 3–15 (2015)
2. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic Programming — an Introduction: on the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers (1998)
3. Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genetic Prog. and Evolvable Machines **10**(3), 307–337 (2009)
4. Botzheim, J., Cabrita, C., Kóczy, L.T., Ruano, A.E.: Genetic and bacterial programming for b-spline neural networks design. Journal of Advanced Computational Intelligence **11**(2), 220–231 (2007)
5. Castelli, M., Silva, S., Vanneschi, L.: A C++ framework for geometric semantic genetic programming. Genetic Prog. and Evolvable Machines **16**(1), 73–81 (2015)
6. Castelli, M., Trujillo, L., Vanneschi, L., Silva, S., Z-Flores, E., Legrand, P.: Geometric semantic genetic programming with local search. In: Proc. GECCO'15, pp. 999–1006. ACM (2015)
7. Castelli, M., Vanneschi, L., Silva, S.: Semantic search-based genetic programming and the effect of intron deletion. Cybernetics, IEEE Trans. on **44**(1), 103–113 (2014)
8. Castelli, M., Vanneschi, L., Silva, S., Ruberto, S.: How to exploit alignment in the error space: Two different GP models. In: R. Riolo, et al. (eds.) Genetic Programming Theory and Practice XII, Genetic and Evolutionary Computation, pp. 133–148. Springer International Publishing (2015)
9. Gentle, J.E.: Numerical Linear Algebra for Applications in Statistics. Statistics and Computing. Springer New York (1998)
10. Gonçalves, I., Silva, S., Fonseca, C.M.F.: On the generalization ability of geometric semantic genetic programming. In: Proc. of EuroGP, pp. 41–52 (2015)
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT Press (1992)
12. Moraglio, A.: Abstract convex evolutionary search. In: Proc. of the 11th FOGA, pp. 151–162 (2011)
13. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Proc. of PPSN XII, vol. 7491, pp. 21–31. Springer (2012)
14. Ni, J., Drieberg, R.H., Rockett, P.I.: The use of an analytic quotient operator in genetic programming. Evolutionary Computation, IEEE Trans. on **17**(1), 146–152 (2013)
15. Oliveira, L.O.V.B., Otero, F.E.B., Pappa, G.L.: A dispersion operator for geometric semantic genetic programming. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference (to appear), GECCO '16. ACM (2016)
16. Pawlak, T.P.: Competent algorithms for geometric semantic genetic programming. Ph.D. thesis, Poznan University of Technology, Poznan, Poland (2015)
17. Pawlak, T.P., Krawiec, K.: Semantic geometric initialization. In: I.M. Heywood, J. McDermott, M. Castelli, E. Costa, K. Sim (eds.) Proc. of the EuroGP'16, *LNCS*, vol. 9594, pp. 261–277. Springer International Publishing, Cham (2016)
18. Roman, S.: Advanced linear algebra, *Graduate Texts in Mathematics*, vol. 135, 2nd edn. Springer New York (2005)
19. Ruberto, S., Vanneschi, L., Castelli, M., Silva, S.: ESAGP - a semantic GP framework based on alignment in the error space. In: Proc. of EuroGP, pp. 150–161 (2014)