Semantic Crossover Operator for GP based on the Second Partial Derivative of the Error Function

Ranyart R. Suárez[1,2], Mario Graff[2], Juan J. Flores[1]

[1] Division de Estudios de Posgrado,
Facultad de Ingenieria Eléctrica,
Universidad Michoacana de San Nicolás de Hidalgo, Mexico

[2] INFOTEC - Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación,
Cátedras CONACyT,

ranyart@dep.fie.umich.mx, juanf@umich.mx, mario.graff@infotec.com.mx

Abstract. In recent years, a variety of semantic operators have been successfully developed to improve the performance of GP. This work presents a new semantic operator based on the *semantic crossover based on the partial derivative error*. The operator presented here uses the information of the second partial derivative to choose a crossover point in the second parent. The results show an improvement with respect to previous semantic operator.

Keywords: genetic programming, semantic, partial derivative, back-propagation, symbolic regression.

1 Introduction

In the community there has been an increasing interest in the use of semantic operators in Genetic Programming (GP) because these kind of operators use the information provided by the behavior of individuals to produce offprings, i.e. these use the semantics (phenotype) of individuals. On the other hand, traditional genetic operators work by manipulating the syntactic representations (i.e., genotype) of the individuals assuring that the offspring generated from the parents will be syntactically different from their them. Meanwhile, semantic operators assure that the offspring generated is semantically different from them.

From all the approaches proposed in the literature, we can distinguish two classes. The first class is called *geometric operators* (or semi-geometric), these operators are called geometric because the crossover operator, under a metric $d : \mathbb{R}$ two parents p_1 and p_2 produce offspring that lie in the d-segment between the parents. Geometric operators have certain characteristics, for example, they assure that the offspring cannot be less fitted than the less fitted of the parents (*i.e.*, they can not be worse than the worse of the parents). Examples of this

class of operators are described in [4,5,7]. A detailed survey and review of this class of operators can be found in [11,8].

The other class of operators is more general and is simply called semantic operators because the operations to generate new individuals are driven by certain rules or conditions constrained to the semantics of the generated individuals. For example, in [1,2] the offspring is incorporated to the next generation only if it is semantically different from its parents (forcing to have different semantics among the population). These approaches were later extended to other domains [6,10].

This work is based in a previous semantic crossover operator presented by Graff *et al.* [3]. The main idea of their proposal is to compute the partial derivative of the fitness function with respect to the node selected as the crossing point in the first parent, and, with this information, chose the second crossing point in the second parent. This can be accomplished by performing the backpropagation algorithm (see [9]) in GP. The backpropagation algorithm is typically used for training Artificial Neural Networks (ANNs), but its application to GP is straightforward because GP and ANNs have some similarities. In GP, the individuals are represented as trees; the output of the nodes are fed to other nodes, just like the nodes of an ANN.

In [3] the first partial derivative of fitness function *w.r.t.* the crossing point in the first parent is computed in order to know whether the output of the subtree at that particular crossing point has to increase or decrease. With this information, a search is performed in all the subtrees of the second parent; the subtree with the closest output *w.r.t.* the first derivative is selected. For the rest of this paper this methodology will be called GPPDE (Genetic Programming with Partial Derivative Error).

The work presented in this paper is an extension of GPPDE. We propose to compute the second partial derivative of the fitness function *w.r.t.* the crossing point. The idea is that the information of the second derivative complement the information given by the first derivative, consequently, the performance obtained by a system using both derivatives will outperform the results achieved by GPPDE.

The rest of the paper is organized as follows: Section 2 explains how to compute the first and second partial derivatives in GP. Section 3 explains how to interpret the information of partial derivatives in the construction of a crossover operator. Section 4 presents the comparison of different crossover operators, and Section 5 presents the conclusions.

2 Partial Derivatives in GP

GPPDE chooses two parents and one crossing point in one of the parents. Let us assume that this crossing point in the first parent is node v, and the error function is E. The semantic operator requieres to compute $\frac{\partial E}{\partial v}$, which is the partial derivative of the error function with respect to the node selected as crossing point. Using backpropagation, the error can be propagated (using a

supervised learning approach, where target values are known), until it reaches the desired node (v).

The advantage of computing the derivative of the error function is that the first derivative gives information about the error surface. For example, if $\frac{\partial E}{\partial v}$ has positive sign, it means that output in node v is greater than the value needed to minimize the error function, E. With this information, the crossing point in the second father is selected by comparing the sign of $\frac{\partial E}{\partial v}$ with the output of all subtrees in the parent, choosing the subtree that has similar sign output.

The main idea of this work is similar to GPPDE, that is, to propose a semantic crossover method for GP, based on the derivative of the error function. The semantic crossover method presented in this work uses the information provided by the second derivative of the error function to select the crossing point in the second parent (GPPDE uses the first derivative). We have called this method GPPDE2 (Genetic Programming with 2nd. order Partial Derivative error).

There are some conditions that need to be satisfied in order to compute the second derivative: 1) the error function has to be derivable and the second derivative of this function must exist and 2) the functions contained in the function set have to be derivable. The error function used in this work is the quadratic error (Equation (1)), where y is the desired output of the program and $\hat{y}$ is the current output. Note that E has first and second order derivatives with respect to the output of the program these can be seen in Equations (2) and (3).

$$E = (y - \hat{y})^2 \tag{1}$$

$$\frac{\partial E}{\partial \hat{y}} = -2(y - \hat{y}) \tag{2}$$

$$\frac{\partial^2 E}{\partial \hat{y}^2} = 2 \tag{3}$$

Before describing how the information of the first and second derivatives of E is used, we need to describe how the derivatives are computed in GP using the backpropagation algorithm.

2.1 Computing Partial Derivatives with the Backpropagation Algorithm

Backpropagation algorithm consists of two steps: 1) the forward step and 2) the backpropagation step. Consider the small program shown in Figure 1, and let us compute the first partial derivative of the programs' output with respect to node x, i.e., $\frac{\partial g(f(x))}{\partial x}$. To do so, we need to compute the output of each node and the first derivative of the function contained in the node as well; this is the forward step. Figure 2 shows the information stored in each node in the program.

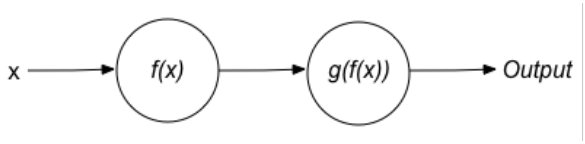

Fig. 1. GP program

Mathematically, the first and second derivatives of $g(f(x))$ with respect to x are: $\frac{\partial g(f(x))}{\partial x} = g'(f(x))f'(x)$ and $\frac{\partial^2 g(f(x))}{\partial x^2} = g''(f(x))f'(x)^2 + g'(f(x))f''(x)$. Note that all the terms needed to compute the first and second derivatives are stored in the nodes after applying the forward step. The backpropagation step consists in traversing the program backwards from the root to the selected node (in this case node x) and computing the product between the information stored in the nodes.

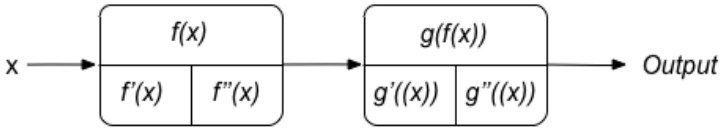

Fig. 2. Information stored in each node

Figure 3 shows how the backpropagation step is performed in order to compute the terms $\frac{\partial g(f(x))}{\partial x}$ and $\frac{\partial^2 g(f(x))}{\partial x^2}$. In Figure 3 (a), the backpropagation step is shown to compute the first derivative. The backpropagation step computes the products of the derivatives stored in the nodes; in this example, the products between input 1 and $g'(f(x))$, and $g'(f(x))$ and $f'(x)$ are computed. The result is $g'(f(x))f'(x)$, which is indeed the term $\frac{\partial g(f(x))}{\partial x}$. This is basically the process used by GPPDE to compute the first derivative of a function with respect to some node.

In this work we extend the procedure mentioned above to compute the second derivative. Whereas backpropagation in GPPDE *sees* one node at a time, GPPDE2 requires to see two nodes at a time (in this example, nodes f and g). With the information stored in these two nodes, two products are computed: 1) the product between the second derivative of the function in the first node and the square first derivative of the function in the second node, and, 2) the product between the first derivative of the function in the first node between the second derivative of the function in the second node. Figure 3 b) depicts the backpropagation step used by GPPDE2.

The following example allows to depict more clearly the process used to compute the second derivative. Consider the program shown in Figure 4, this program corresponds to $\hat{y} = 1.5x^2 - 0.7x + 1.2$. Suppose the program is chosen as

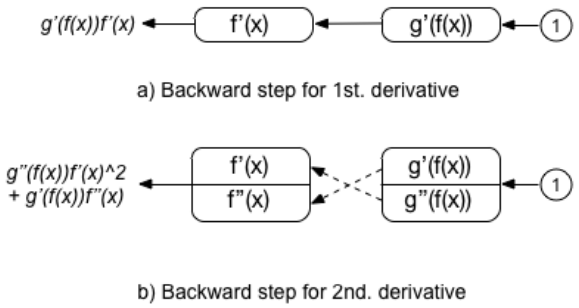

Fig. 3. Backpropagation step to compute the first and second derivatives

the first parent in a crossover operation and the crossing point is node v, i.e., the second constant -0.7. Let us assume that the inputs are $x = [-1, -0.5, 0, 0.5, 1]$, the backpropagation, in the forward step, computes the output of the program, i.e., $\hat{y}(x) = [3.4, 1.925, 1.2, 1.225, 2]$, and, also, stores the first and second derivatives of each function as in Figure 2. In this example the training set is $T = [\{-1, 1.9\}, \{-0.5, 1.175\}, \{0, 1.2\}, \{0.5, 1.975\}, \{1, 3.5\}]$, therefore the error is $e = (y(x) - \hat{y}(x)) = [-1.5, -0.75, 0, 0.75, 1.5]$.

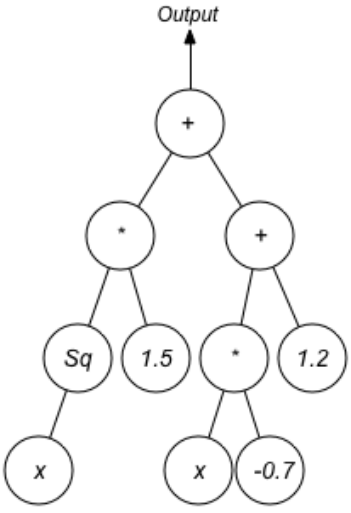

Fig. 4. Simple GP program

In the backward step the first elements that are needed are the first and second order derivatives of the fitness function. In this example, the fitness

function used is $E = (y(x) - \hat{y}(x))^2$, the first and second order derivative of E $w.r.t.$ the program's output are $\frac{\partial E}{\partial \hat{y}(x)} = -2(y(x) - \hat{y}(x)) = -2e$ and $\frac{\partial^2 E}{\partial \hat{y}(x)^2} = 2$, respectively. Figure 5 shows backward step used in GPPDE2. Note that in the figure, the root node is the function E, which takes two arguments y and $\hat{y}(x)$. Additionally, the target $y(x)$ and the part $1.5x^2$ of $\hat{y}(x)$ are simplified and the nodes involved in the process have been labeled $(E, 0, 1, 2)$.

The steps depicted in Figure 5 are the following:

(a) Node E correspond to the fitness function and node 0 is the root of the program. Following the example in Figure 3, the function in node E is g and the function in node 0 is f. GPPDE computes the term $\frac{\partial g(f(x))}{\partial x} = -2e * 1 = -2e$ and GPPDE2 computes the term $\frac{\partial^2 g(f(x))}{\partial x^2} = 2 * 1^2 + 0 * -2e = 2$, these terms are propagated to node 0.

(b) Now, node 0 corresponds to g and node 1 corresponds to f. The terms computed are: $\frac{\partial g(f(x))}{\partial x} = -2e * 1 = -2e$ and $\frac{\partial^2 g(f(x))}{\partial x^2} = 2 * 1^2 + 0 * -2e = 2$, these terms are propagated to node 1.

(c) Node 1 is g and node 2 is f function, the terms computed are: $\frac{\partial g(f(x))}{\partial x} = -2e * x = -2ex$ and $\frac{\partial^2 g(f(x))}{\partial x^2} = 2 * x^2 + 0 * -2e = 2x^2$, these terms are propagated to node 2.

d) The process stops because the propagated terms have finally reached the parent node of v.

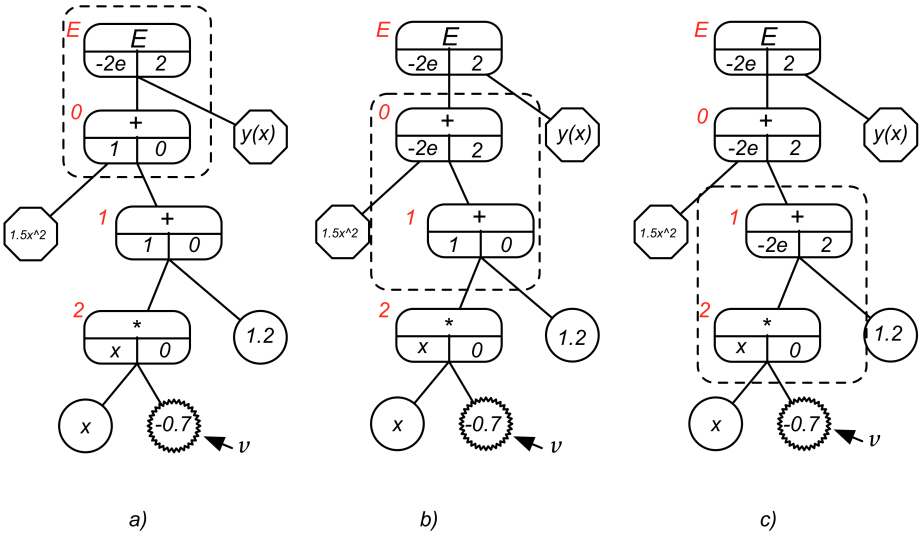

Fig. 5. Computing of partial second derivative of E function w.r.t. node v

When node v is reached, the propagated terms are: $-2ex$ and $2x^2$. These

values are indeed the first and second partial derivatives of $E = (y - ax^2 - vx - c)^2$ *w.r.t.* to v. Equations 4 and 5 show the derivatives.

$$\frac{\partial E}{\partial v} = 2(y - ax^2 - vx - c)(-x) = -2ex \tag{4}$$

$$\frac{\partial^2 E}{\partial v^2} = \frac{\partial}{\partial v} 2(y - ax^2 - vx - c)(-x) = \frac{\partial}{\partial v} 2vx^2 = 2x^2 \tag{5}$$

Note that the first and second derivatives stored in each node are vectors not scalars. Besides, every time that the values are propagated, these values replace the stored values in the next node.

3 Semantic Crossover Operator Using Partial Derivatives

GPPDE2 has computed the second partial derivative of the fitness function *w.r.t.* the node selected as the crossing point. Since the fitness function is quadratic, we decided to perform the newton method at the crossing point. The newton method (Equation 6) minimizes the error contribution to function E provided by the function that receives node v as an input.

$$X^{n+1} = X^n - f'(X^n) * [f''(X^n)]^{-1} \tag{6}$$

To demonstrate how the newton method works, let us perform the first iteration in the previous example with the information:

- The samples x = [-1, -0.5, 0, 0.5, 1]
- The error $e = [-1.5, -0.75, 0, 0.75, 1.5]$
- Output of node v, $X^n = [-0.7, -0.7, -0.7, -0.7, -0.7]$
- First partial derivative $\frac{\partial E}{\partial v} = -2ex = [-3, -0.75, 0, -0.75, -3]$
- Second partial derivative $\frac{\partial^2 E}{\partial v^2} = 2x^2 = [2, 0.5, 0, 0.5, 2]$

Since the output of node v is constant, let us take the sum of the first and second partial derivatives (-7.5 and 5). The first iteration of the newton method yields $X^1 = -0.7 - \frac{-7.5}{5} = -0.7 + 1.5 = 0.8$. This iteration of the newton method can be interpreted as the desired output from node v. So, the constant -0.7 has to be changed to 0.8. In this case, the new individual would be
$\hat{y}(x) = 1.5x^2 + 0.8 + 1.2$ which is indeed the target function for the example ($y(x)$).

In practice, the crossover point can be any node (function, variable or constant). Therefore, instead of taking the sum of partial derivatives, all the operations are done point to point between vectors, and, then, with X^1, i.e. the first iteration of the newton step, the crossover operator performs a search in all the possible sub-trees of the second parent choosing the node whose output has a minimum euclidean distance with X^1. Equation 7 shows the crossover operator, where X^1 is the first iteration of newton step, X_i^j represents the i^{th} element of X^1, S^j is the output of the j^{th} sub-tree in the second parent and S_i^j is the i^{th} element of S^j.

$$\arg\min_{j} \sum_{i}^{N} \sqrt{(X_i^1 - S_i^j)^2} \tag{7}$$

4 Results

One of the main applications of GP is symbolic regression. The problem of symbolic regression consists in finding a mathematical model that best fits certain data. In this contribution, the testbed consists in 1,100 symbolic regression problems used previously to test GPPDE in [3]. Each of the problems contained in this set was built in the following way: 1) Two polynomials $W(x)$ and $Q(x)$ were generated randomly choosing their degree in the range $[2, 8]$ with random real coefficients in the range $[-10, 10]$, 2) A rational function $y(x) = \frac{W(x)}{Q(x)}$ is used to create the data. Each function $y(x)$ was sampled 21 times uniformly distributed in the range $[-1, 1]$. The goal here is use GP to create a model that recreates the rational functions by fitting the sampled data.

Given that GPPDE2 uses more information than GPPDE, it is expected that GPPDE2 exhibits better performance than GPPDE. The first derivative used in GPPDE gives information about the direction in which the error will decrease, in other words, the first derivative tell GPPDE if the output in the subtree chosen for crossover has to be smaller or bigger. On the other hand, the second derivative computed in GPPDE2, provides the step size needed to minimize the error contribution for the subtree chosen for crossover.

Both GPPDE2 and GPPDE were run 20 times with the parameters shown in Table 1. This means that each of the 1100 problems were run 20 times and 20 different performances were obtained. Then, these measures were sorted and the median was chosen to be the final measure. We decided to choose the median because for both algorithms, in some problems, there was noise.

Table 1. GP Parameters

Parameter	Value
Population Size	1000
Number of Generations	50
Function Set ($\mathcal{F}$)	$\{+, -, \times, /\}$
Terminal Set ($\mathcal{T}$)	$\mathcal{T} = \{x \in \mathcal{R}\}$
Crossover rate	90%
Mutation rate	10%
Mutation depth	random $\in [1, 5]$
Selection	Tournament of size 2

Table 2 presents the results obtained by the GP systems tested. Additionally to GPPDE2 and GPPDE, we decided to add a third column that represents traditional Genetic Programming (GP). GP was added to the comparison in

order to demonstrate the differences in performance by performing traditional crossover and the crossover with partial derivatives.

All of the three GP systems see the problem as a minimization problem. This means that a performance closer to zero is better. The performance is given by the squared error between the target and the program's output. From Table 2 we can see that GPPDE2 won in 556 of 1100 problems (50.55%) whereas GPPDE won in 544 problems (49.45%) and GP did not have the best performance in any problem. There is no important difference between the number of problems won by GPPDE2 and GPPDE; however, there exists differences in performance. The mean of medians for GPPDE2 is 0.2905 and it was a better overall result than 0.5287 of GPPDE (a fitness closer to zero is better). Moreover, the standard deviation of GPPDE2 (1.9985) was smaller than the GPPDE's measure (5.9473) indicating more consistent results.

Table 2. Results of GP Systems

	GPPDE2	GPPDE	GP
# of problems won	556	544	0
Mean of Medians	0.2905	0.5287	4.1425
Std. Deviation	1.9985	5.9473	24.6912

From these results it can be inferred that GPPDE2 obtained better results than GPPDE. Given the number of problems that each algorithm won and the small difference in these results, is reasonably to question if GPPDE2 is indeed performing better than GPPDE. To answer this, we decided to measure the difference in performance when some algorithm outperforms the other. We measure the squared difference of performance between GPPDE2 and GPPDE when one of the algorithms won, Table 3 presents this comparison. From the table, it can be seen that when GPPDE2 has a better performance than GPPDE, the margin between the two algorithms is about 1.5670, this margin is three times bigger than the inverse case (when GPPDE has better performance than GPPDE2) with 0.5108.

Table 3. Difference in performance for GPPDE2 and GPPDE

Method	Difference in performance
GPPDE2	1.5670
GPPDE	0.5108

5 Conclusions

In this work, a new semantic crossover operator called GPPDE2 has been presented. This new method is and improved version of GPPDE, a previous crossover

semantic method. The idea behind both algorithms is to compute the partial derivatives of the error function with respect to some node (the node is selected by the cross point) in the first parent. With this information, choose the second cross point in the second parent. The main difference is that GPPDE computes the first derivative and GPPDE2 computes the second derivative. Results show that the crossover operator that uses the information of the second derivative of the error function in order to choose the crossing point in the second parent gives better results. Both semantic operators gave better results than using the traditional crossover operator (both crossing points randomly chosen) for the tested problems.

The results presented in this work open the possibility of further analysis in order to explore the capabilities of GPPDE2, for example, increasing the number of functions contained in the function set and analyze if this changes improve the current performance.

References

1. Beadle, L., Johnson, C.: Semantically driven crossover in genetic programming. In: IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). pp. 111–116 (2008)
2. Beadle, L., Johnson, C.: Semantically driven mutation in genetic programming. In: IEEE Congress on Evolutionary Computation, 2009. CEC '09. pp. 1336–1342 (2009)
3. Graff, M., Graff-Guerrero, A., Cerda-Jacobo, J.: Semantic crossover based on the partial derivative error. In: Genetic Programming, pp. 37–47. Springer (2014)
4. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. p. 987994. GECCO '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1569901.1570036
5. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Parallel Problem Solving from Nature-PPSN XII, pp. 21–31. Springer (2012)
6. Nguyen, Q.U., Nguyen, X.H., O?Neill, M.: Semantic aware crossover for genetic programming: the case for real-valued function regression. In: Genetic Programming, pp. 292–302. Springer (2009)
7. Pawlak, T., Wieloch, B., Krawiec, K.: Semantic Backpropagation for Designing Search Operators in Genetic Programming. IEEE Transactions on Evolutionary Computation Early Access Online (2014), 00000
8. Pawlak, T.P., Wieloch, B., Krawiec, K.: Review and comparative analysis of geometric semantic crossovers. Genetic Programming and Evolvable Machines pp. 1–36 (2014)
9. Rojas, R.: Neural Networks: A Systematic Introduction. Springer, 1 edn. (Jul 1996)
10. Uy, N.Q., Hoai, N.X., ONeill, M., McKay, R.I., Galvn-Lpez, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines 12(2), 91–119 (Jul 2010), http://www.springerlink.com/content/48411662081364h6/
11. Vanneschi, L., Castelli, M., Silva, S.: A survey of semantic methods in genetic programming. Genetic Programming and Evolvable Machines 15(2), 195–214 (Jun 2014), http://link.springer.com/article/10.1007/s10710-013-9210-0