# Semantic Crossover Based on the Partial Derivative Error

Mario Graff[1], Ariel Graff-Guerrero[2], and Jaime Cerda-Jacobo[1]

[1] Division de Estudios de Posgrado
Facultad de Ingenieria Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo, México
mgraffg@dep.fie.umich.mx, jcerda@umich.mx
[2] PET Centre, Centre for Addiction and Mental Health, Toronto, ON, Canada
Ariel_graff@yahoo.com.mx

**Abstract.** There is great interest for the development of semantic genetic operators to improve the performance of genetic programming. Semantic genetic operators have traditionally been developed employing experimentally or theoretically-based approaches. Our current work proposes a novel semantic crossover developed amid the two traditional approaches. Our proposed semantic crossover operator is based on the use of the derivative of the error propagated through the tree. This process decides the crossing point of the second parent. The results show that our procedure improves the performance of genetic programming on rational symbolic regression problems.

**Keywords:** Semantic Crossover, Symbolic Regression.

## 1 Introduction

Semantic genetic operators have been proposed to improve the performance of genetic programming. Semantic operators use the information of the phenotype to create new individuals. There are two approaches to develop semantic operators, the first is experimentally-based, and the second is theoretically-based.

The experimentally-based approach produces a semantic crossover based on either the fitness function or the behaviour of the trees. Blickle *et al.* [1] propose to select as crossing points only those nodes that have an impact in the fitness function, this is implemented using a flag on the nodes, that is set during the evaluation of the tree. Nguyen *et al.* [2, 3] produce offspring that are semantically different from its parents; this difference is measured by evaluating the individuals in a set of random inputs.

On the other hand, following a theoretically-based approach, Beadle *et al.* [4, 5] propose a semantic operator that only accepts an offspring if it is semantically non-equivalent to its parents. The non-equivalent property is verified by using a reduced ordered binary decisions diagram, which is also employed to develop a semantically different initial population (see [6]). In addition, Krawiec *et al.* [7] develop an approximation of a geometric semantic crossover that is important

M. Nicolau et al. (Eds.): EuroGP 2014, LNCS 8599, pp. 37–47, 2014.

because has the potential to convert the landscape into a cone. Furthermore, Moraglio *et al.* [8] have shown the feasibility of creating a geometric semantic crossover and mutation. The procedure proposed by Moraglio *et al.* to generate the offspring is clean and easy to implement; however, it has the drawback that the constructed offspring is always bigger than the sum of the lengths of its parents which imposes a limitation for applicability. Nonetheless, Vanneschi *et al.* [9] overcome this original limitation allowing the algorithm to be executed with the traditional parameters used in GP.

Our current contribution proposes a semantic crossover for tree-based genetic programming (GP) that is in-between the experimentally-based and the theoretically-based approaches. The proposed semantic crossover is based on the derivative of the error, i.e., the derivative of the fitness function $f(p)$. In symbolic regression problems, it is common to compute the fitness function as: $f(p) = \sum_{(x,y)\in\mathcal{T}}(y - p(x))^2$, where $\mathcal{T} = \{(x_i, y_i) \ : i = 1 \dots N\}$ is the training set, and $p(x)$ represents the output of the individual $p$ on input $x$.

The semantic crossover proposed here works by:

1. Computing $\frac{\partial f}{\partial v}$ equivalently to what backpropagation algorithm [10] does to update the weights of an Artificial Neural Network.[1] However, in our procedure, $v$ is a node randomly selected from the first parent, whereas in backpropagation $v$ is always a constants, i.e., a weight.
2. The result of $\frac{\partial f}{\partial v}$ is used to select the crossing point of the second parent.
3. The two points are used to perform a traditional subtree crossover.

The results illustrate, that GP enhanced with the proposed semantic crossover statistically outperforms a GP with traditional crossover on 1,100 rational functions used as testbed.

The current paper is organized as follows. Section 2 presents our novel semantic crossover. Section 3 describes the procedure used to generate the symbolic regression problems and the GP systems used to illustrate the effectiveness of our approach. Section 4, conclusions and future directions are presented.

## 2 Semantic Crossover Based on Partial Derivative Error

The semantic crossover proposed is computed as follows: a) let $v$ be a node randomly selected from the first parent; b) given $v$, $\frac{\partial f}{\partial v}$ is computed; c) a node $u$ is selected from the second parent using the information of the partial derivative with respect to $v$; and d) finally, it is performed a subtree crossover using as crossing points $v$ and $u$. The rest of the section contains a detailed description of this procedure. Subsection 2.1 shows the procedure used to compute $\frac{\partial f}{\partial v}$, i.e., it presents the backpropagation algorithm implemented in a tree structure; and Subsection 2.2 describes the process to select $u$.

[1] The use of backpropagation in GP has been previously proposed in [11–14].

### 2.1 Backpropagation

The first step, in order to describe the semantic crossover proposed, is to show the process used to compute $\frac{\partial f}{\partial v}$. As we mentioned, the backpropagation algorithm can be used to obtain $\frac{\partial f}{\partial v}$. Backpropagation can be easily explained using the ideas presented on [10]. R. Rojas used a graphical representation to explain it and this representation can be easily codified in a tree-based GP.

Let us describe backpropagation by computing the chain rule, i.e., $\frac{\partial g(h(x))}{\partial x}$. The first step is to compute $g(h(x))$, see upper part of Figure 1 and, note, that each inner node is split in two part; the right part corresponds to the node's output and the left part stores the output of the operation shown in each node. The flow of information is indicated by the arrows. The second and, final step, is to traverse the tree backwards, see lower part of Figure 1. This backward step is performed by supplying a constant to the root, in this case 1 and then multiplying this constant by the value stored on the left side of each node. This process recursively continues until a leaf is reached. It is observed, in the lower part of the figure, that $\frac{\partial g(h(x))}{\partial x}$ is obtained at the end of this process.

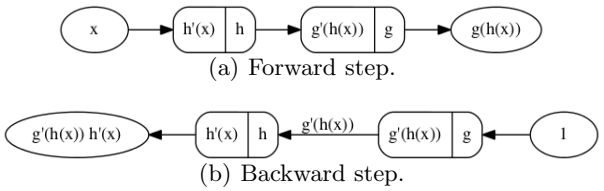
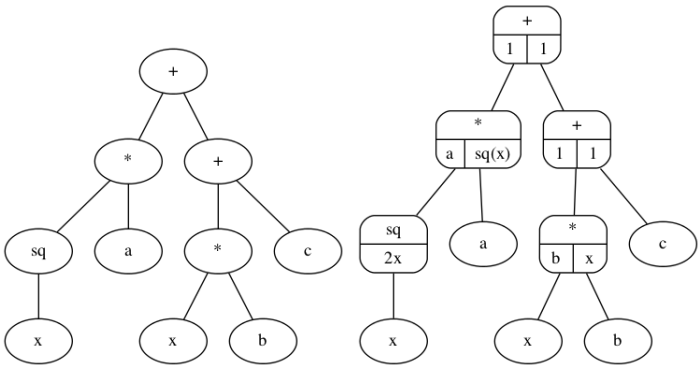



**Fig. 1.** Computing the derivative in a graph

Continuing with the description of backpropagation, let us suppose that a leaf node is a constant $w$, then, the process mentioned above computes $\frac{\partial f}{\partial w}$. This can be used to update $w$ following the rule $w = w - \nu\frac{\partial f}{\partial w}$, where $\nu$ is the learning factor. This update can be performed per training case, or for the whole training set, the former is known as *incremental learning* and the later is *batch learning*. In batch learning $w$ is updated using the rule $w = w - \nu\sum_{i=1}^{|\mathcal{T}|}\frac{\partial f_i}{\partial w}$, where $f_i$ be the error in the $i^{th}$ case of the training set. In addition to this, the resilient backpropagation (RPROP) [15] can be implemented using only the sign and a different $\nu$ for the increase and decrease values, i.e., $w = w - \nu_x \cdot \text{sign}(\sum_i^{|\mathcal{T}|}\frac{\partial f_i}{\partial w})$, where $x$ stands for decrease or increase learning rate.

In order to have a complete picture of how the process, Figure 2 presents an example. In the left of the figure (a), it is illustrated a function represented as a tree. In the right (b), we have the same tree is presented with the additional information required to compute the derivatives. Note that two slots are needed for the functions having two arguments. For example, in the product the first part stores the second input, given that $\frac{\partial x \cdot y}{\partial x} = y$ and the second part contains the first input. We have illustrated that the sum nodes store ones which is the

partial derivative with respect to each input. However, there is not need to store information on the sum nodes and has been included only to ensure a clear description of the example.




(a) Tradition tree on GP representing $y(x) = ax^2 + bx + c$.

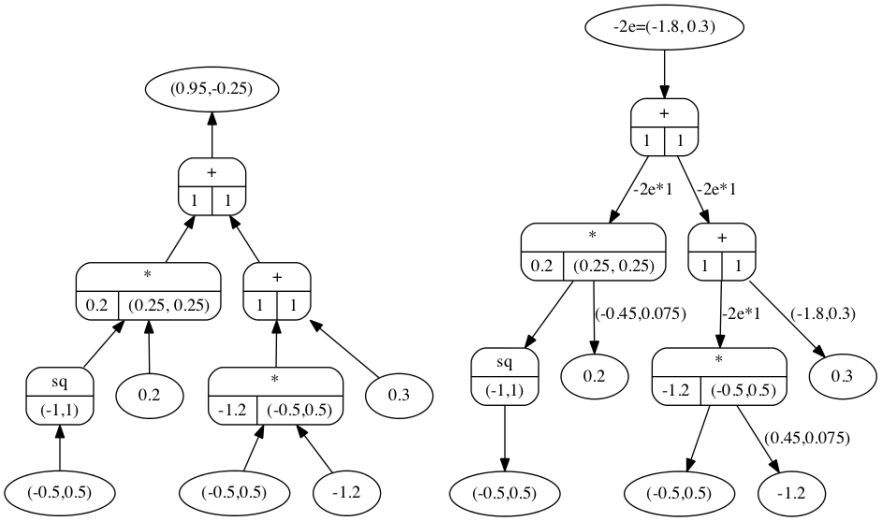
(b) Tree with additional storage to save the derivatives.

**Fig. 2.** Tree having an extra storage to compute the derivative

Figure 3 presents a complete example, in the left (a), it is shown that all the constants have values, e.g., $a = 0.2$, $b = -1.2$ and $c = 0.3$, and that there are two inputs $x_1 = -0.5$ and $x_2 = 0.5$. Furthermore, the sought function is $y(x) = 0.5x^2 - 2.25x + 0.6$ which leads to a training set $\mathcal{F} = \{(-0.5, 1.85), (0.5, -0.4)\}$. Under these circumstances the tree's output is $(0.95, -0.25)$. In the evaluation process, i.e., the forward step, all the partial derivatives are computed and stored in their respective nodes; this is shown in the lower part of each node (left of the figure). The backward step requires to feed the node with a value, this value is the derivative of the error, which depends on the function used to compute the error. Let $f(p) = (y - p)^2$ be the error function, then its derivative with respect to $p$ is $-2(y-p)$. Given that $y = (1.85, -0.4)$ is the desired output and $p = (0.95, -0.25)$ is the tree's output, hence the value fed is $-2(y - p) = (-1.8, 0.3)$. This value is propagated through the tree until a constant is reached. Finally, this constant is updated using the sign of all the values reach to that specific node. The constants are decreased or increased depending on whether the sign is positive or negative, respectively. In the case the value received by a constant is zero, then that constant is not updated.

### 2.2 Selecting the Crossing Points

So far, we have described the backpropagation algorithm implemented in a tree. This algorithm is recursive and in the backward step it stops when a leaf is reached; however, nothing forbids to stop it at any particular node. Let us suppose that the procedure is stopped at node $v$ ($v$ is randomly selected from the




(a) Individual evaluated on $x = -0.5$ and $x = 0.5$. The root node is the result of the evaluation.

(b) Error propagation on the two fitness cases. The root contains the error between $y(x)$ and the individual's output.

**Fig. 3.** Example of the use of back-propagation in a tree, when the sought function is $y(x) = 0.5x^2 - 2.25x + 0.6$. The sum of the error on constant 0.2 is $-0.375$, on constant $-1.2$ is 1.05 and on 0.3 is $-1.5$, indicating that the value of the constants must be incremented, decremented and incremented, respectively.

first parent), then, at this point, it is obtained $\frac{\partial f}{\partial v}$, which indicates whether the values returned by $v$ must be either decreased or increased depending on whether $\frac{\partial f}{\partial v} > 0$ or $\frac{\partial f}{\partial v} < 0$, respectively.

In order to select the crossing point of the second parent, the following procedure is performed. Let $e$ be $\text{sign}(\frac{\partial f}{\partial v})$, $e_i$ corresponds the sign of the partial derivative error in the $i^{th}$ training case and $v$ correspond to the crossing point of the first parent. In a tree-based GP $v$ can be seen as a complete tree, i.e., one can remove $v$ from the parent and treat it as new individual. In this context let $p^v$ be the output of $v$ when it is evaluated with the inputs of $\mathcal{T}$ and $p_i^v$ represents the output of $v$ in the $i^{th}$ training case. Equivalently, for the second parent, we can compute the output of all the nodes, let $s^j$ represent the output of node $j$ of the second parent. Using $e$, $p$ and $s$, we can find the crossing point of the second parent, i.e., $u$, as:

$$u = \arg\max_j \sum_i \text{sign}(p_i^v - s_i^j) \cdot e_i. \tag{1}$$

In order to clarify Equation (1), let us analyse the two possible scenarios found in the maximum of Equation (1). Firstly, in the case $e_i > 0$, then $s_i^u$ is higher than $p_i^v$ consequently $p_i^n - s_i^m$ is positive. On the other hand, $e_i < 0$ implies that $p_i^n - s_i^m$ is negative; however, $e_i < 0$ is also negative and the result is positive.

Under this circumstances, $\sum_i \text{sign}(p_i^v - s_i^u) \cdot e_i$ equals $|\mathcal{T}|$, given that $p_i^v - s_i^u$ and $e_i$ have an equivalent sign.

## 3 Results

Our novel semantic crossover operator has been compared to a steady-state GP system with tournament selection, henceforth referred as *Standard*[2]. This *Standard* base system was enhanced with a simplification procedure, referred as, *Simplification*. We also compared additional GP systems where RPROP was applied to all the individuals generated (namely RPROP *always*) and another one that RPROP was only applied to any new best individual (namely RPROP *only on best*). The novel semantic crossover was incorporated in this latter system referred as GP with partial derivative error (GPPDE). The details of the parameters used in the GP systems are shown on Table 1 and the parameters used in RPROP are shown in Table 2. The RPROP parameters were taken from the Fast Artificial Neural Network Library (FANN) [17]

In our implementation, each epoch spent in RPROP was counted as an individual generated and the maximum number of generated individuals was $50,000$. In addition, only an individual that has been optimized with RPROP was kept only if the individual was better than the original.

**Table 1.** Genetic Programming Parameters

| Parameter | Value |
|---|---|
| Function Set | $\mathcal{F} = \{+, -, \times, /\}$ |
| Terminal set ($\mathbb{T}$) | $\mathbb{T} = \{x,\ \Re\}$ |
| Random constants (i.e., $\Re$) | 100 real value constants $\in [-10, 10]$ |
| Max length | 262143 |
| Crossover rate | 90% |
| Mutation rate | 10% |
| Population size | 1000 |
| Number of Generations | 50 |
| Mutation depth | random $\in [1, 5]$ |
| Tournament size | 2 |
| Max. number of epochs without improvement | 5 |

The simplification mechanism was used in all the GP systems except the Standard GP. This simplification was applied to all the elements of the initial population and to every new offspring. This procedure was very simple, it only incorporated rules to reduce the tree in the following cases, where $s$ stands for any subtree, $\Re$ is a constant and *op* represents any operation.

[2] This is very similar to TinyGP [16], it is only modified to allow the evolution of constants for each individual in the population.

**Table 2.** RPROP's parameters

| Parameter | Value |
|---|---|
| Increase Factor | 1.2 |
| Decrease Factor | 0.5 |
| Delta min | 0.0 |
| Delta max | 50.0 |
| Init previous step value | 0.0001 |

- Replace $(op\ \Re\ \Re)$ with the result of that operation.
- Replace $(+\ s\ 0)$ with $s$.
- Replace $(-\ s\ s)$ with 0.
- Replace $(*\ s\ 0)$ with 0.
- Replace $(*\ s\ 1)$ with $s$.
- Replace $(/\ 0\ s)$ with 0.
- Replace $(/\ s\ s)$ with 1.
- Replace $(\text{op op}^{-1}\ s)$ with $s$.

Figure 4 shows an example of the simplification mechanism. For example, the subtree $(*-3.01.2)$ is replaced by the constant $-3.6$ and the whole left subtree is replaced by 1.0. The rules were applied recursively and as consequence it might be a case in which an entire tree could be substituted by a constant.

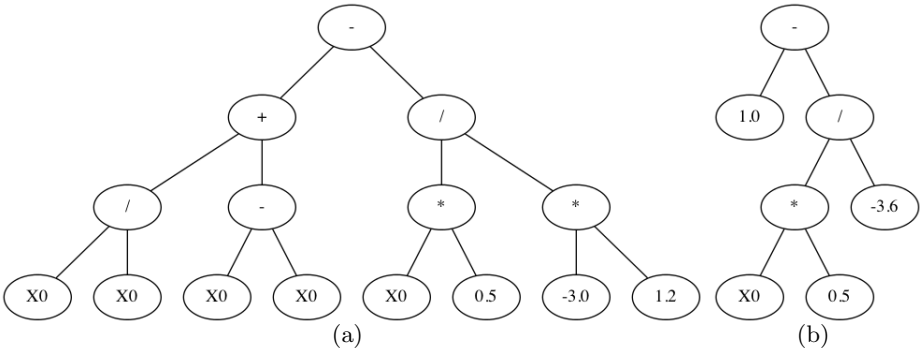



**Fig. 4.** Simplifying a tree, a) shows the original tree, and b) presents the tree after the simplification

The problem employed as benchmark, was a continuous symbolic regression problem on rational functions. We created 1,100 different rational functions using the following procedure. Two polynomials, $W(x)$ and $Q(x)$, were built by randomly choosing the degree of each in the range 2 to 8, and then choosing random real coefficients in the interval $[-10, 10]$ for the powers of $x$ up to the chosen degree. A rational function in our training set is then given by $y(x) = \frac{W(x)}{Q(x)}$. Each

of the rational functions in the set was then sampled at 21 points uniformly distributed in the interval $[-1, 1]$. This resulted in a target vector $\mathbf{y} \in \Re^{21}$. The aforementioned procedure was previously used in [18] to test their approach.

For each $\mathbf{y}$ vector 30 independent runs were performed recording the output of the best individual found, namely $\mathbf{p}$. With this value, it was computed $|\mathcal{T}|^{-1} \sum_i^{|\mathcal{T}|} \sqrt(|\hat{\mathbf{y}}_i - \hat{\mathbf{p}}_i|)$, where $\hat{\mathbf{y}} = \frac{\mathbf{y}-\mu}{\sigma}$ and $\hat{\mathbf{p}} = \frac{\mathbf{p}-\mu}{\sigma}$, $\mu$ is the mean of $\mathbf{y}$ and $\sigma$ is the standard deviation of $\mathbf{y}$. Finally, this value was recorded for each of the 30 independent runs and the performance was the average of these values defined as nBRF. Another measure of performance was used which is a normalize version of the mean absolute error (nMAE), i.e., $|\mathcal{T}|^{-1} \sum_i^{|\mathcal{T}|} |\hat{\mathbf{y}}_i - \hat{\mathbf{p}}_i|$, where $\hat{\mathbf{y}} = m\mathbf{y} + b$ and $\hat{\mathbf{p}} = m\mathbf{p} + b$. In this case, $m$ and $b$ were $[0, 1]$, zero stands for the minimum value of $\mathbf{y}$ and one for the maximum.

Table 3 presents the performance of the different GP systems. It includes the performance of the standard GP, GP with the simplification procedure, GP with RPROP applied to all the individuals (RPROP always), GP with RPROP applied to every new best (RPROP only on best). GPPDE showed the best performance in the two performance measures. In order to test whether the differences in performance were statistically significant, a Wilcoxon signed-rank test [19] were performed, the $p$ values were well below 0.01 indicating that the performance of GPPDE was better than the second best with a confidence of 99%.

A particular result of note was the nMAE's performance of the GP with simplification. The comparison of the performance using a Wilcoxon signed-rank test showed a $p = 0.2948$ indicating not difference in performance. This result was contrary to our expectations because GP with RPROP should be better than GP with simplification because the former was used to optimize the constants.

**Table 3.** Performance (BRF and nMAE) of the different GP systems on the rational functions

| GP systems | nBRF | nMAE | Length |
|---|---|---|---|
| Standard | $0.2492 \pm 0.0857$ | $0.0741 \pm 0.0486$ | $958.6116 \pm 178.0225$ |
| Simplification | $0.2450 \pm 0.0853$ | $0.0719 \pm 0.0475$ | $990.1021 \pm 193.6305$ |
| RPROP always | $0.5370 \pm 0.1027$ | $0.3063 \pm 0.1042$ | $\mathbf{61.8987 \pm 11.7165}$ |
| RPROP only on best | $0.2394 \pm 0.0852$ | $0.0719 \pm 0.0480$ | $647.3761 \pm 154.5169$ |
| GPPDE | $\mathbf{0.1320 \pm 0.0535}$ | $\mathbf{0.0251 \pm 0.0263}$ | $6077.9015 \pm 2178.9033$ |

GPPDE showed the best performance; however, the average length of the trees was considerable longer than those obtained with others GP systems. GPPDE generates trees that were on average more than 6 times longer than those obtained with Simplification GP. On the other hand, when RPROP was applied to all the generated individuals, the trees were short and this might be the cause for the poor performance observed. The results indicate that RPROP should be

used more carefully in order to allow more exploration of the search space, or used a larger number of generations.

In order to complement the information presented in Table 3, the number of times each algorithm presented the best performance was counted, the performance measure used was nMAE. This procedure showed that Simplification GP presented the best performance in 5 of the 1,100 problems and GPPDE has the best performance in 1095 of the 1,100 problems.

Figure 5 presents the average output of Simplification GP and GPPDE on one of the five problems in which GPPDE did not exhibit the best performance. In this problem, Simplification GP had 0.0362 and GPPDE has 0.0464, these two values were computed using nMAE. GPPDE did not reach the peaks as closely as Simplification GP, although the behaviour of these two systems on this problem was qualitatively equivalent.

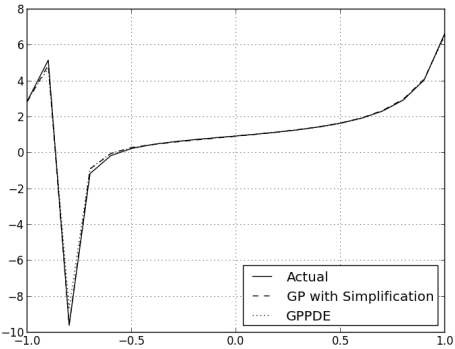



**Fig. 5.** Average output of GP with simplification and GPPDE on a problem where GPPDE does not have the best performance

The performance (nBRF) presented on Table 3 can be compared with the performance of the different GP algorithms previously presented by Graff and Poli [18]. The systems presented by Graff and Poli [18] include: generational GP systems with roulette and tournament selection, steady-state GP systems with tournament selection, generational gene expression programming (GEP) with roulette and tournament selection, steady-state GEP with tournament selection and different versions of stochastic iterated hill climber (SIHC). In total there are 20 different GP systems. First our standard GP system correspond to the steady-state GP system with tournament selection and 100% crossover. Our implementation has a performance of 0.2492 and the performance previously presented by our group [18] was 0.2535. These two values seems to be comparable and the difference might be due to the random number generators or any other minor modification.

The GPPDE showed the best performance overall the systems tested. Our previous work [18] showed that SIHC system with subtree mutation and 25000 as the maximum number of mutations showed a performance of 0.2021 which

is considerable higher than the performance obtained by GPPDE (0.0464). Unfortunately, it is not available the lengths of the tree generated by SIHC and consequently the full performance cannot be compared.

## 4 Conclusions

Our work presents the development of a novel semantic crossover operator that is based on the derivative of the errors. The results showed that choosing the crossover point of the second parent according to Equation 1 improves considerably the learning abilities of the GP systems. GPPDE showed the best performance in almost all the problems tested (1095 out of 1100), and it also obtained on average the best performance in comparison to the systems presented here, and, also, to the 20 systems previously presented by our group [18].

The improvement presented needs plenty of computations performed on the trees. That is, one needs to keep the output, and the derivative of the output, for every node of every tree in the population. This may be a drawback because the technique may be limited to small populations and/or small trees. However, the full potential that this amount of information can provide has not been properly explored. For example, in order to evaluate an offspring, it is only needed to recalculate the nodes whose values have been changed, this would make the algorithm faster. On the other hand, one can measure the fitness of each node, and take decisions dynamically based on the fitness. Another limitation with the proposed semantic crossover operator is that all the functions in the function set need to be derivable. This may be a major drawback for some problems; however, for symbolic regression problems it seems reasonable to use only derivable functions.

**Acknowledgments**. This project would not be possible without the use of the CAMH Specialized Computing Cluster (SCC).

We would like to anonymous reviewers for their fair and useful comments and ideas. The paper has been considerably strengthened thanks to their feedback.

## References

1. Blickle, T., Thiele, L.: Genetic programming and redundancy. Choice 1000, 2 (1994)
2. Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic aware crossover for genetic programming: The case for real-valued function regression. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 292–302. Springer, Heidelberg (2009)
3. Uy, N.Q., Hoai, N.X., ONeill, M., McKay, R.I., Galvn-López, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines 12(2), 91–119 (2010)
4. Beadle, L., Johnson, C.: Semantically driven crossover in genetic programming. In: IEEE Congress on Evolutionary Computation, CEC 2008 (IEEE World Congress on Computational Intelligence), pp. 111–116 (2008)
5. Beadle, L., Johnson, C.: Semantically driven mutation in genetic programming. In: IEEE Congress on Evolutionary Computation, CEC 2009, pp. 1336–1342 (2009)

6. Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genetic Programming and Evolvable Machines 10(3), 307–337 (2009)
7. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO 2009, pp. 987–994. ACM, New York (2009)
8. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012)
9. Vanneschi, L., Castelli, M., Manzoni, L., Silva, S.: A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş., Hu, B. (eds.) EuroGP 2013. LNCS, vol. 7831, pp. 205–216. Springer, Heidelberg (2013)
10. Rojas, R.: Neural Networks: A Systematic Introduction, 1st edn. Springer (July 1996)
11. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), pp. 155–162 (2001)
12. Smart, W., Zhang, M.: Continuously evolving programs in genetic programming using gradient descent. In: Proceedings of 2004 Asia-Pacific Workshop on Genetic Programming (2004)
13. Zhang, M., Smart, W.: Genetic programming with gradient descent search for multiclass object classification. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 399–408. Springer, Heidelberg (2004)
14. Graff, M., Pena, R., Medina, A.: Wind speed forecasting using genetic programming. In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 408–415 (2013)
15. Igel, C., Hüsken, M.: Empirical evaluation of the improved rprop learning algorithms. Neurocomputing 50, 105–123 (2003)
16. Poli, R.: TinyGP. See Genetic and Evolutionary Computation Conference (GECCO 2004) (June 2004), competition at `http://cswww.essex.ac.uk/staff/sml/gecco/TinyGP.html`
17. Nissen, S.: Implementation of a fast artificial neural network library (fann). Technical report, Department of Computer Science University of Copenhagen, DIKU (2003), `http://fann.sf.net`
18. Graff, M., Poli, R.: Practical performance models of algorithms in evolutionary program induction and other domains. Artificial Intelligence 174(15), 1254–1276 (2010)
19. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1(6), 80 (1945)