SEMANTICS BASED MUTATION IN GENETIC PROGRAMMING: THE CASE FOR REAL-VALUED SYMBOLIC REGRESSION

Nguyen Quang Uy[1], Nguyen Xuan Hoai[2], Michael O'Neill[1]
[1] Natural Computing Research & Applications Group, University College Dublin, Ireland
[2]School of Computer Science and Engineering, Seoul National University, Korea
quanguyhn@yahoo.com, nxhoai@gmail.com, m.oneill@ucd.ie

Abstract

In this paper we propose two new methods for implementing the mutation operator in Genetic Programming called Semantic Aware Mutation (SAM) and Semantic Similarity based Mutation (SSM). SAM is inspired by our previous work on a semantics based crossover called Semantic Aware Crossover (SAC) [19] and SSM is an extension of SAM by adding more control on the change of semantics of the subtrees involved in mutation operation. We apply these two new mutation operators to a class of real-valued symbolic regression problems and compare them with the Standard Mutation (SM) of Koza [13]. The results from the experiments show that while SAM does not help to improve the performance of Genetic Programming, SSM helps to significantly enhance Genetic Programming performance on the problems tried. The experiment results also show that the change of the semantics (fitness) in SSM is smoother than ones of both SAM and SM. This, we argue that is the main reason to the significant performance improvement of SSM over SAM and SC.

Keywords: *Genetic Programming, Semantics, Mutation Operator, Symbolic Regression*

1 Introduction

Semantics is currently an emerging hot topic in the field of Genetic Programming with a number of papers being published on this area in recent years (e.g.,[7, 8, 9, 11, 10, 1, 16, 19, 2]). Prior to this, a great deal of research in GP has focused on syntactical issues of the GP representation, which has brought valuable insights and contributions on the behaviour of GP. In particular grammar-based approaches to GP have been the focus of a large number of studies within the field. As every programmer is aware, syntax is only a small part of the story when it comes to problem solving, rather they are more concerned with the semantics, or meaning, of the syntax. In this paper we demonstrate that GP researchers should be increasingly aware of semantics and how they might be employed to improve the efficiency of GP search. In terms of research into representations in Evolutionary Computation, it is recognised that search operators such as mutation should exhibit the property of locality [17, 18]. For example, small changes made to a solution (e.g., a GP tree) should result in a correspondingly small fitness change, and similarly large changes to a solution should amount to a fitness change of an equivalent magnitude. In this paper we demonstrate that through the adoption of semantics we can design a mutation operator with improved locality and in turn improved performance on a set of symbolic regression problem instances. We introduce two new mutation operators in GP called *Semantic Aware Mutation* (SAM) and *Semantic Similarity based Mutation* (SSM). These operators extend standard mutation through the addition of semantic information to control the change of semantics of individuals during the evolutionary process by only allowing to replace a subtree by a semantically similar subtree when performing mutation. In so doing we expect that the change of fitness of individual will be less abrupt, exhibiting better behaviour in terms of locality.

The remainder of this paper is organized as follows. In the next section, we give a review of related work on some semantic based operators in GP. Section 3 contains the detailed descriptions of our new mutation operators. The experiments is described in section 4 of the paper. The results of the experiments are then given and discussed in section 5. Section 6 concludes the paper and highlights some potential future extensions of this work.

2 Semantics in Genetic Programming

There is a growing literature on the use of semantics in Genetic Programming, and there are at least three ways in which semantics can be represented, extracted and used to guide GP evolutionary process: use grammars [21, 2, 3], use formal methods [7, 8, 9, 11, 10], and is based on GP s-tree representation [1, 16, 19]. In the first way, the most popular formalism used to incorporate semantic information into GP is Attribute Grammars. By using an attribute grammar and adding some attributes to individuals, some useful semantic information of individuals during the evolutionary process can be checked.

This information then can be used to eliminate bad individuals from the population as in [3] or to prevent generating semantically invalid individuals as in [21, 2]. The attributes used to present semantics are usually problem dependent and it is not always obvious to determine the attributes for each problem.

Recently, Johnson has advocated for using formal methods as a way of adding semantic information in the evolutionary process of GP [7, 8, 9]. In these methods, the semantic information extracted by using formal methods (such as by Abstract Interpretation and Model Checking) is used to quantify the fitness of individuals in some problems which are difficult to use a traditional sample point based fitness measure. Katz and his co-workers used a Model Checking with GP to solve the Mutual Exclusion problem [11, 10]. In these works, semantics is also used to calculate the fitness of individuals.

With expression trees, semantic information has been incorporated mainly by modifying the crossover operator. In [1], the authors investigated the effects of directly using semantic information to guide GP crossover on Boolean domains. The main idea proposed in [1] was to check the semantic equivalence between offspring and parents by transforming the trees to Reduced Ordered Binary Decision Diagrams (ROBDDs). Two trees have the same semantic if and only if they reduce to the same ROBDD. The checking is then used to determine which of the individuals participating in crossover will be copied to the next generation. If the offspring are semantically equivalent to their parents, then the parents are copied into the new population. By doing this, the authors argue, there is an increase in the semantic diversity of the evolving population and a consequent improvement in the GP performance.

In our previous work [19], we proposed a new crossover operator (SAC), based on the semantic equivalence checking of subtrees. Our approach was tested on a family of real-value symbolic regression problems (e.g., polynomial functions). Our empirical results showed that SAC improves GP performance. SAC differs from [1] in two ways. Firstly, the test domain is real-valued rather than Boolean. For real-value domains, checking semantic equivalence by reduction to common ROBDDs is not possible. Secondly, the crossover operator is guided not by the semantics of the whole program tree, but by that of subtrees. This is inspired by recent work presented in [16] for calculating subtree semantics.

While using semantics in GP has attracted a number of studies in recent years and some of this research has investigated ways to use semantics to guide the crossover operator, to the best of our knowledge, there has been no research in the literature on using semantics to guide the mutation operator. Although, most of GP researchers and users follow Koza's practice in [13] to use the crossover operator as the main search operator for GP, the comparative importance of GP crossover and mutation operators is still the subject of much debate. An early argument of this debate was given in [15] when the authors did an exclusive comparison between crossover and mutation and found that there was very little different performance between these two operators. In a more recent work [20], White and Poulding have done an experiment to compare the crossover and mutation in GP with the optimal conditions of the experiment settings for each operator. Their results showed that there were only 2 out of 6 tried problems, GP with crossover was better than GP with mutation. These research suggest that mutation operator is also important in GP and any improvement of the mutation operator could potentially lead the improvement of GP performance.

So far, most of the modifications to GP standard subtree mutation is purely based on syntax with the usual objective as reducing code bloat. An example of such modifications is to restrict the standard subtree mutation so that the increase of the program depth after being mutated is no more than 15% of its depth [12], or must be size-fair [14]. By contract, our new approach is to constraint the mutation operator by semantics in that it forces the change of the semantics of individuals after mutation as in SAM and try to keep this change smaller and smoother as in SSM.

3 Methodology

The first new mutation operator, SAM, is similar to SAC in [19], but adapted to constrain the semantics in mutation rather than in crossover. As in SAC, the semantic equivalence of the two subtrees (replaced and replacing subtrees in the mutation operation) is determined by comparing them on a set of random points in the domain. If the outputs of the two subtrees on the set is close enough (subject to a parameter called *semantic sensitivity*) then they are designated as semantically equivalent. The pseudo-code for semantically equivalence checking of subtrees St_1 and St_2 as follows:

```
If Abs(Value_On_Random_Set(St1)-Value_On_Random_Set(St2)< ε) then
        Return St1 is semantically equivalent to St2.
```

Here Abs is the absolute function and ϵ is a predefined constant called semantic sensitivity. This information is then used to guide the mutation operator by preventing replacing a subtree with a semantically equivalent subtree. Notice that here we also denote a randomly generated tree to replace a subtree in mutation operation as a subtree as it will become a subtree of the individual after mutation. Algorithm **??** shows how SAM works.

The second mutation operator (SSM) differs from SAM in two ways. Firstly, the concept of semantically equivalent subtrees is replaced by the concept of semantically similar subtrees. Again, the semantic similarity of two subtrees is also checked by comparing them on a set of random points in the domain. If the output of these subtrees on the set is within an interval, then they are considered as semantically similar subtrees. the pseudo-code for semantically equivalence checking as follows:

```
If α<Abs(Value_On_Random_Set(St1)- Value_On_Random_Set(St2))<β then
        Return St1 is semantically similar to St2.
```

Algorithm 1: Semantic Aware Mutation	**Algorithm 2**: Semantic Similarity based Mutation

select Parent *P*;
choose at random mutation points at $Subtree_1$ in P;
generate at random a subtree $Subtree_2$;
If $Subtree_1$ is not equivalent with $Subtree_2$ Then
  execute mutation by replace $Subtree_1$ with $Subtree_2$;
  add the child to the new population ;
  return true;
Else
  choose at random mutation points at $Subtree_1$ in P;
  generate at random a subtree $Subtree_2$;
  execute mutation by replace $Subtree_1$ with $Subtree_2$;
  add the child to the new population ;
  return true;
End If
	select Parent P;
	Count=0;
	While *Count<Max_Attempt* Do
	  choose at random mutation points $Subtree_1$ in P;
	  generate at random a subtree $Subtree_1$ in P;
	  If $Subtree_1$ is similar to $Subtree_2$ Then
	  execute mutation by replace $Subtree_1$ with $Subtree_2$;
	  add the child to the new population ;
	  return true;
	  Else
	  Count=Count+1;
	  End If
	End While
	If *Count=Max_Attempt* Then
	  choose at random mutation points at $Subtree_1$ in P;
	  generate at random a subtree $Subtree_1$ in P;
	  execute mutation by replace $Subtree_1$ with $Subtree_2$;
	  add the child to the new population ;
	  return true;
	End If

Here Abs is also the absolute function and α and β are two predefined constants. We will call α as *lower bound semantic sensitivity*, β as *upper bound semantic sensitivity*. Perhaps, the best values for *lower bound semantic sensitivity* and *upper bound semantic sensitivity* are problem dependent. However, we strongly believe that there is a range of values that is good for almost every symbolic regression problems (see section 5). In this paper, we conducted the experiment with a range of both *lower* and *upper bound semantic sensitivity* to see how SSM performs with different values.

The second difference between SSM and SAM is the way in which the semantic similarity is used to guide the mutation operation. In SAM the equivalent semantics is used to guide mutation by trying to select other subtrees to do mutation only one time when two subtrees are semantically equivalent. In SSM, when two subtrees are considered as not similar, we try a number of times to pick up another subtree in the parent and generate a new subtree. The reason is that choosing two similar subtrees is more difficult than selecting not-equivalent two subtrees in SAM. Algorithm **??** shows how SSM works.

In this paper, Max_Attempt is chosen as 4. The motivation for SAM is to encourage the replacement of a subtree by a semantically different subtree that we hope will generate semantically different individuals in the next generation. SSM is inspired from the requirement of trying to cause a small change in term of semantics in the search process. In other words, while forcing a semantic change of the individuals in the population, we also want to keep this change bounded and small. It is expected that a smoother change in fitness of the individuals will be obtained.

4 Experimental Setup

Table 1: Symbolic Regression Functions

$F_1 = X^3 + X^2 + X$	$F_3 = X^5 + X^4 + X^3 + X^2 + X$
$F_2 = X^4 + X^3 + X^2 + X$	$F_4 = X^6 + X^5 + X^4 + X^3 + X^2 + X$

To investigate the effects of SSM, SAM and to compare them with standard subtree mutation (SM), we used four real-valued symbolic regression problems of increasing difficulty. These functions, from [6], are shown in table 1, and parameter setting is given in table 2. The reason for choosing the *lower bound semantic sensitivities* of SSM as the *semantic sensitivities* of SAM is that these values helped to improve the performance of SAC as shown in [19]. The reason for selecting the *upper bound semantic sensitivities* is inspired from our experiments that these values are sufficient to demonstrate the performance of SSM.

5 Results and discussion

To compare the effect of SSM and SAM with SM we recored a number of experimental results. The first one is to compare the number of successful runs over 100 runs of three mutation operators. The results are shown in Table 3. In this table,

Table 2: Run and Evolutionary Parameter Values

Parameter	Value	Parameter	Value
Generations	50	Population size	500
Selection	Tournament	Tournament size	3
Crossover probability	0.2	Mutation probability	0.8
Initial Max depth	6	Max depth	15
Max depth of mutation	5		
Non-terminals	+, -, *, /, sin, cos, exp, log (protected versions)		
Terminals	X, 1		
Number of samples	20 random points from $[-1 \ldots 1]$		
Successful run	sum of absolute error on all fitness cases < 0.1		
Termination	max generations exceeded		
Lower semantic sensitivities	$0.02, 0.04, 0.06$		
Higher semantic sensitivities	$8, 10, 12$		
Trials per treatment	100 independent runs for each value.		

Table 3: The comparison of the percentage of successful runs

sensitivities		F_1			F_2			F_3			F_4		
low	high	SM	SAM	SSM	SM	SAM	SSM	SM	SAM	SSM	SM	SAM	SSM
0.02	8	23	18	**36**	5	3	**12**	3	3	**13**	0	1	**4**
	10	23	18	**46**	5	3	**14**	3	3	**6**	0	1	**7**
	12	23	18	**39**	5	3	**14**	3	3	**11**	0	1	**7**
0.04	8	23	15	**38**	5	8	**12**	3	3	**7**	0	0	**2**
	10	23	15	**39**	5	8	**16**	3	3	**8**	0	0	**5**
	12	23	15	**38**	5	8	**13**	3	3	**7**	0	0	**2**
0.06	8	23	17	**38**	5	5	**8**	3	2	**7**	0	0	**2**
	10	23	17	**41**	5	5	**15**	3	2	**7**	0	0	**5**
	12	23	17	**44**	5	5	**14**	3	2	**8**	0	0	**4**

the best results (the biggest values) are bold faced. It can be seen from Table 3 that the performance of SM and SAM on the tested problems are almost identical. For the easier target functions like F_1, SM was slightly better than SAM. However, on the more difficult target functions, F_2, F_3, and F_4, the difference in terms of the percentage of successful runs of two these mutation methods is almost diminished. Although, SAM did not help GP to improve its performance in solving these problems, SSM is by far better than both SM and SAM. For example, consider the case with *lower bound* and *higher bound semantic sensitivity* as 0.02 and 12, the number of successful runs of SM, SAM, and SSM for F_1, F_2, F_3, and F_4 were in turn (23, 18, 39), (5, 3, 14), (3, 3, 11), and (0, 1, 7). The most significant improvement achieved on the most difficult function, F_4. For this problem, SM could not find any solution (over 100 runs), SAM could only find 1 solution for all cases, while SSM found solutions more frequently (up to 7% of the runs).

The second line of results given in Table 4 is the average of best solutions found in all runs of all GP systems. In this table, *sen* is the shorthand for *sensitivity*. Figure 1 shows the average of best fitness and the average of average fitness (over 100 runs) in each of 50 generations with *lower bound* and *high upper bound semantic sensitivities* as 0.02 and 12 respectively. It is noted that, in this figure, we only show the statistics from the 10^{th} generation. The reason is that at some

Table 4: The comparison of the average best fitness over 100 runs

sen		F_1			F_2			F_3			F_4		
low	high	SM	SAM	SSM	SM	SAM	SSM	SM	SAM	SSM	SM	SAM	SSM
0.02	8	0.28	0.33	**0.19**	0.43	0.49	**0.27**	0.56	0.57	**0.36**	0.62	0.63	**0.45**
	10	0.28	0.33	**0.19**	0.43	0.49	**0.29**	0.56	0.57	**0.37**	0.62	0.63	**0.42**
	12	0.28	0.33	**0.19**	0.43	0.49	**0.28**	0.56	0.57	**0.33**	0.62	0.63	**0.41**
0.04	8	0.28	0.35	**0.18**	0.43	0.43	**0.27**	0.56	0.59	**0.36**	0.62	0.63	**0.44**
	10	0.28	0.35	**0.18**	0.43	0.43	**0.29**	0.56	0.59	**0.38**	0.62	0.63	**0.42**
	12	0.28	0.35	**0.19**	0.43	0.43	**0.32**	0.56	0.59	**0.31**	0.62	0.63	**0.40**
0.06	8	0.28	0.32	**0.19**	0.43	0.42	**0.27**	0.56	0.55	**0.35**	0.62	0.63	**0.42**
	10	0.28	0.32	**0.18**	0.43	0.42	**0.29**	0.56	0.55	**0.38**	0.62	0.63	**0.42**
	12	0.28	0.32	**0.18**	0.43	0.42	0.34	0.56	0.55	**0.33**	0.62	0.63	**0.38**

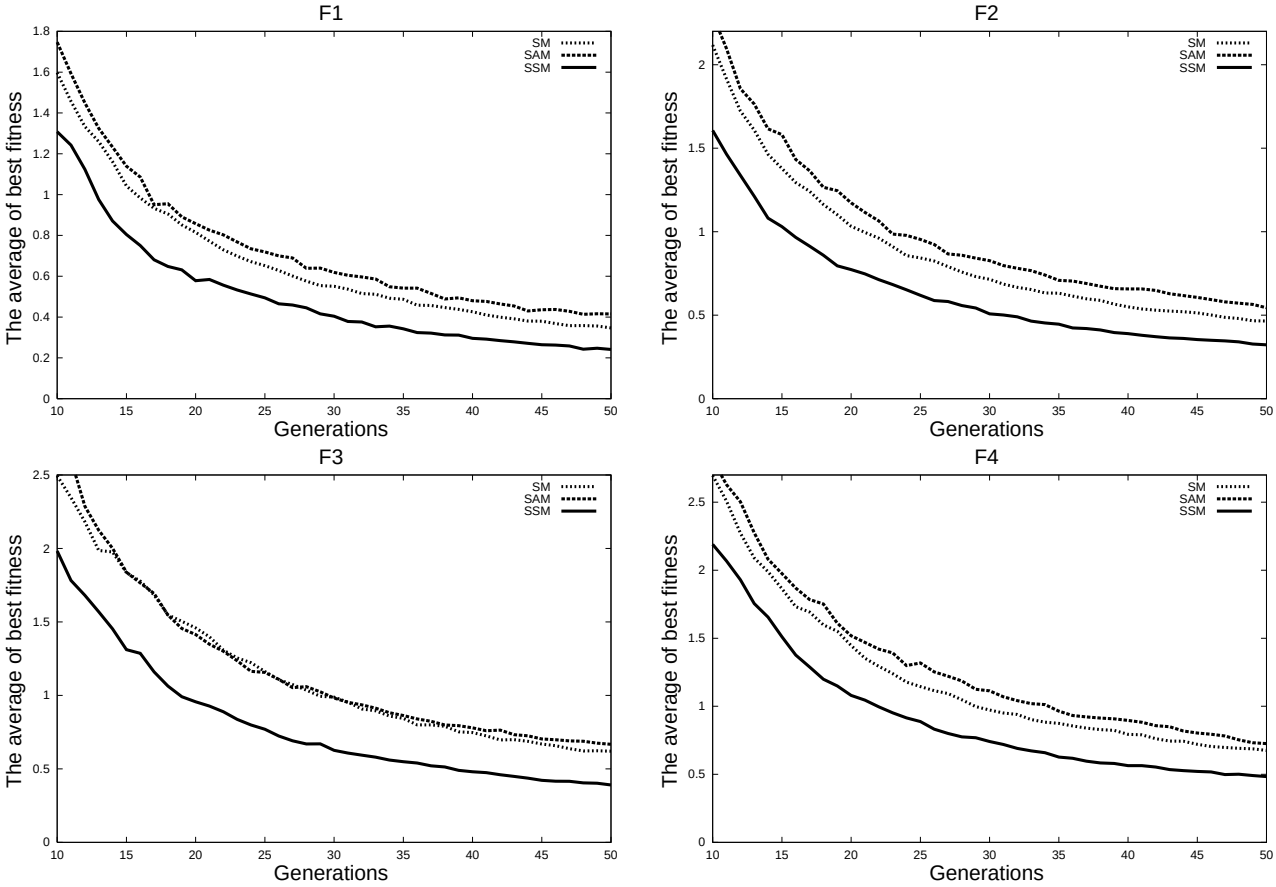

Figure 1: The average of best fitness with α=0.02, β=12

first generations, the values of those statistics are usually big (which is expected as the fitness of individuals at the early stage of evolution is usually very bad). Therefore, it is difficult to scale the graphs to highlight the difference. Moreover, at these early generations, the statistics on fitness values were almost similar in all of GP system runs regardless of which mutation operator is used.

The results in Table 4 are consistent with those in Table 3 in that SSM is also superior than both SAM and SM finding solutions with better quality and also there is a very small difference between SAM and SM. To measure the statistical significance of the results in Table 4, we also conducted some statistical tests. Here the t-test was used to see if the improvement over the average best fitness of SSM over SM is significant. The t-test of SAM in comparison with SM is also calculated for the ease of comparison. The result of t-test is then used to decide if the improvement of SSM over Sm and the difference between SAM and SM are remarkable. if they are remarkable, the corresponding values in Table 4 are bold faced.

It can be seen from Table 4 that while the difference in terms of the average best fitness of runs of SAM and SM is not significant, the improvement of SSM over SM is considerably significant in most situations. The exceptions only lie in the quite easy-to-learn function F_2. On the contrary, in the most two complicated and hard to learn functions, F_3, F_4, there is no exception. The results support the confirmation that the SSM is better than SM in terms of the average of best solutions. Figure 1 shows that SSM not only outperforms than SAM and SM in terms of best fitness of runs but also better in terms of the average of best fitness and the average of average fitness in each generations.

The next set of experimental results are for the investigation of the locality property of SSM. It is well known that using a high-locality representation (small change in genotype corresponds to small change in phenotype) is important for efficient evolutionary search [5]. It is also widely admitted that designing a search operator for GP that could correspond a small change in syntax (genotype) to a small change in semantics (phenotype) is very difficult. Therefore, nearly all current GP representations and operators are low-locality, meaning that a small (syntactic) change in a parent can cause a big or even uncontrollable (semantical) change in their children. Our new mutation operator (SSM) is different with other mutation operators in the literature in that aspect, trying to achieve high-locality by controlling the small change in terms of semantics.

To compare the locality property of SSM with SAM and SM, an experiment was conducted where the fitness change of individuals before and after mutation is measured. For example, if an individual having fitness of 15 is selected to for being mutated, and that after mutation operation, its offspring has fitness of 9. Then, the change of fitness of this individual

Table 5: The average individual fitness change before and after crossover operation

sen		F_1			F_2			F_3			F_4		
low	high	SM	SAM	SSM	SM	SAM	SSM	SM	SAM	SSM	SM	SAM	SSM
0.02	8	11.0	11.5	**9.6**	12.0	12.4	**10.0**	13.9	12.7	**10.5**	13.9	12.8	**11.1**
	10	11.0	11.5	**9.4**	12.0	12.4	**9.2**	13.9	12.7	**10.0**	13.9	12.8	**10.3**
	12	11.0	11.5	**7.4**	12.0	12.4	**8.5**	13.9	12.7	**8.9**	13.9	12.8	**8.8**
0.04	8	11.0	11.6	**9.2**	12.0	12.6	**9.5**	13.9	12.7	**10.5**	13.9	13.0	**11.0**
	10	11.0	11.6	**8.9**	12.0	12.6	**9.0**	13.9	12.7	**9.5**	13.9	13.0	**9.9**
	12	11.0	11.6	**7.6**	12.0	12.6	**8.6**	13.9	12.7	**9.3**	13.9	13.0	**8.9**
0.06	8	11.0	11.1	**9.2**	12.0	12.8	**10.0**	13.9	12.4	**10.6**	13.9	13.1	**11.0**
	10	11.0	11.1	**9.0**	12.0	12.8	**9.6**	13.9	12.4	**9.8**	13.9	13.1	**9.9**
	12	11.0	11.1	**7.5**	12.0	12.8	**8.3**	13.9	12.4	**9.2**	13.9	13.1	**8.7**

is $Abs(15-9)=6$. Here Abs is again the absolute function. This value is then averaged over whole population and over 100 runs as well as for 50 generations. The results about the average of the fitness change of individuals before and after mutation is shown in Table 5. Again, in this table, the best results (the smallest values) are bold faced.

In the Figure 2 we show the change of the average of fitness movement of 100 runs for each of 50 generations with *lower bound* and *upper bound semantic sensitivities* as 0.02 and 12. Table 5 and Figure 2 show that the step of the fitness change of SSM is smaller than both SAM and SM. This means that the change of fitness over generations of SSM is smoother than SAM and SM. The table and figure also show that the fitness change of SM is only slightly smoother than SAM in two easy functions, F_1, F_2 and this value of SAM is slightly smoother than SM in two more difficult functions, F_3, F_4. These results explain why SSM is much better than SAM and SM on the problems tried, while SAM is not better than SM.

6 Conclusion and future works

In this paper, we have proposed two new semantics based mutation operators for GP (SAM and SSM). The new operators were tested and analysed on a class of real-valued symbolic regression problems and the results was compared with the standard mutation of Koza. The experimental results show that SSM helps to improve the performance of GP in comparison with SAM and SM while SAM is not better than SM. The experimental results also show that SSM makes a smaller change of fitness during the evolutionary process in comparison with SAM and SM. We argue that this is the main reason why SSM outperformed SAM and SM on the problems tried.

In the near future, we are planning to extend the work in a number of ways. Firstly, we are aiming to apply SSM on more difficult symbolic regression problems (the problems that are multi-variable and more complex in the structure of the solutions). For these problems, we predict that making a small change in semantics is more difficult and also more important. Another potential research direction is to apply SSM on other kind of problem domains such as on Boolean problems that have been investigated in [16]. It could be even more difficult to generate the children that are different with their parents in terms of semantics. Lastly, we are investigating the range of *lower bound semantic sensitivity* and *upper bound semantic sensitivity* that are good for a class of problems. In this paper, these values are manually and experimentally specified. However, these value might change adaptively during the evolutionary process as the way to self-adaptation the parameters of generic algorithm is done [4].

Acknowledgments

This paper was funded under a Postgraduate Scholarship from Irish Research Council for Science Engineering and Technology (IRCSET).

References

[1] L. Beadle and C. Johnson. Semantically driven crossover in genetic programming. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 111–116. IEEE Press, 2008.

[2] R. Cleary and M. O'Neill. An attribute grammar decoder for the 01 multi-constrained knapsack problem. In *Proceedings of the Evolutionary Computation in Combinatorial Optimization*, pages 34–45. Springer Verlag, April 2005.

[3] M. de la Cruz Echeanda, A. O. de la Puente, and M. Alfonseca. Attribute grammar evolution. In *Proceedings of the IWINAC 2005*, pages 182–191. Springer Verlag Berlin Heidelberg, 2005.

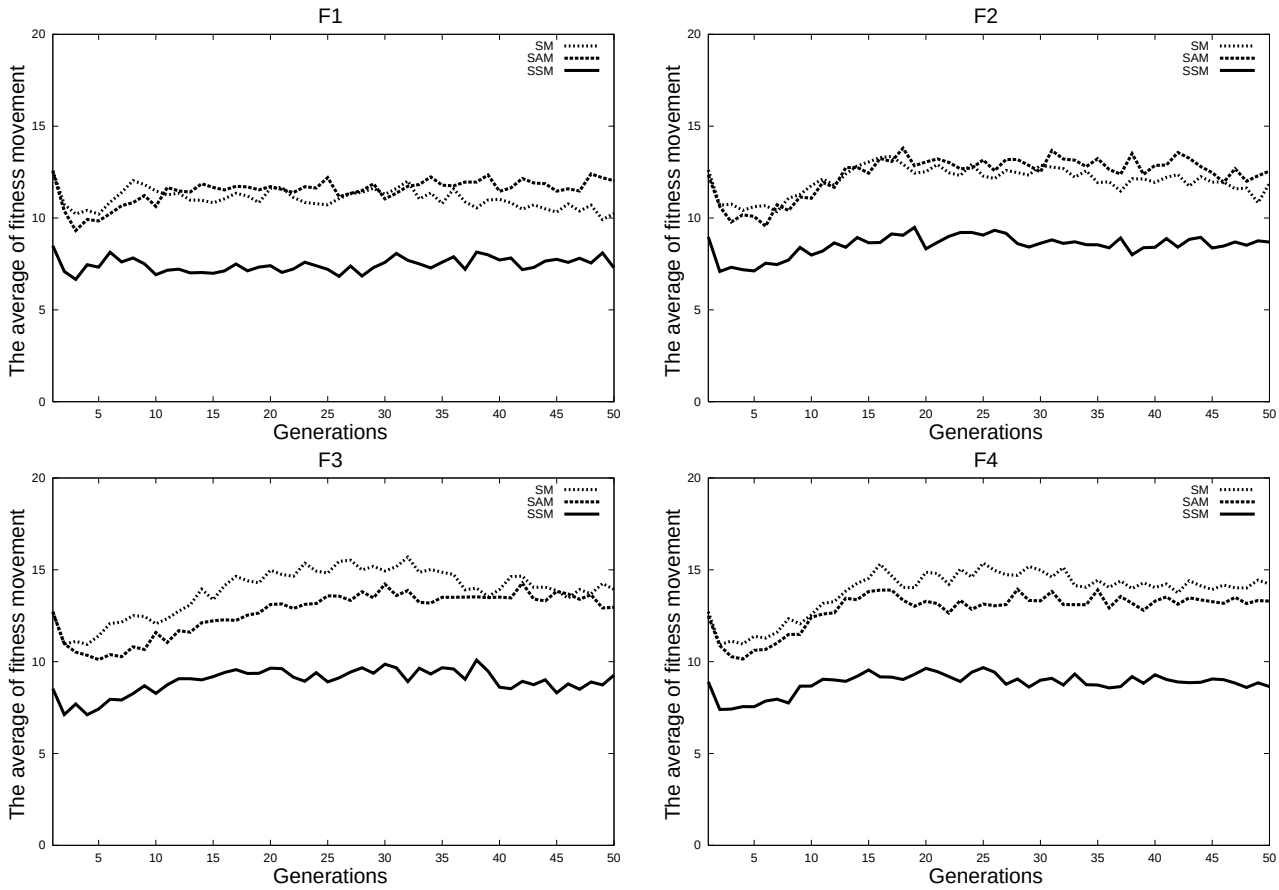

Figure 2: The average fitness movement before and after crossover with α=0.04, β=10

[4] K. Deb and H. G. Beyer. Self-adaptation in real-parameter genetic algorithms with simulated binary crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 172–179. Morgan Kaufmann, July 1999.

[5] J. Gottlieb and G. Raidl. The effects of locality on the dynamics of decoder-based evolutionary search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 283290. ACM, 2000.

[6] N. X. Hoai, R. McKay, and D. Essam. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)*, pages 1326–1331. IEEE Press.

[7] C. Johnson. Deriving genetic programming fitness properties by static analysis. In *Proceedings of the 4th European Conference on Genetic Programming (EuroGP2002)*, pages 299–308. Springer, 2002.

[8] C. Johnson. What can automatic programming learn from theoretical computer science. In *Proceedings of the UK Workshop on Computational Intelligence*. University of Birmingham, 2002.

[9] C. Johnson. Genetic programming with fitness based on model checking. In *Proceedings of the 10th European Conference on Genetic Programming (EuroGP2002)*, pages 114–124. Springer, 2007.

[10] G. Katz and D. Peled. Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. *Automated Technology for Verification and Analysis, Lecture Notes in Computer Science*, 5311:33–47, 2008.

[11] G. Katz and D. Peled. Model checking-based genetic programming with an application to mutual exclusion. *Tools and Algorithms for the Construction and Analysis of Systems*, 4963:141–156, 2008.

[12] K. E. Kinnear. A rigorous evaluation of crossover and mutation in genetic programming. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP*, pages 881–888. IEEE Press, July 1998.

[13] J. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MITPress, MA, 1992.

[14] W. B. Langdon. The evolution of size in variable length representations. In *Proceedings IEEE International Conference on Evolutionary Computation*, page 633638. IEEE Press, May 1998.

[15] S. Luke and L. Spector. A comparison of crossover and mutation in genetic programming. In *Proceedings of the Second Annual Conference on Genetic Programming*, pages 240–248. San Francisco, CA, USA, April 1997.

[16] N. McPhee, B. Ohs, and T. Hutchison. Semantic building blocks in genetic programming. In *Proceedings of 11th European Conference on Genetic Programming*, pages 134–145. Springer.

[17] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, 2nd edition edition, 2006.

[18] F. Rothlauf and M. Oetzel. On the locality of grammatical evolution. In *Proceedings of the 9th European Conference on Genetic Programming*, pages 320–330. Lecture Notes in Computer Science, Springer, April 2006.

[19] N. Q. Uy, N. X. Hoai, and M. O'Neill. Semantic aware crossover for genetic programming: the case for real-valued function regression. In *Proceedings of EuroGP09*. Springer.

[20] D. White and S. Poulding. Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks*, pages 57–69. LNCS, 5481, April 2009.

[21] M. L. Wong and K. S. Leung. An induction system that learns programs in different programming languages using genetic programming and logic grammars. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, 1995.