Semantic Similarity based Crossover in GP for Real-valued Function Regression

Nguyen Quang Uy[1], Michael O'Neill[1], Nguyen Xuan Hoai[2], Bob Mckay[2], and Edgar Galván-López[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

Abstract. In this paper we propose a new method for implementing the crossover operator in Genetic Programming (GP) called Semantic Similarity based Crossover (SSC). This new operator is inspired by Semantic Aware Crossover (SAC) [20]. SSC extends SAC by adding semantics to control the change of the semantics of the individuals during the evolutionary process. The new crossover operator is then tested on a family of symbolic regression problems and compared with SAC as well as Standard Crossover (SC). The results from the experiments show that the change of the semantics (fitness) in the new SSC is smoother compared to SAC and SC. This leads to performance improvement in terms of percentage of succesful runs and mean best fitness.

1 Introduction

Genetic Programming (GP) is an evolutionary algorithm inspired by biological evolution to find the solution for an user-defined task. The program is usually presented in a language of syntactic formalisms such as s-expression trees [14], a linear sequence of instructions, grammars, or graphs [18]. The genetic operators in such GP systems are usually designed to ensure the syntactic closure property, i.e., to produce syntactically valid children from any syntactically valid parent(s). Using such purely syntactical genetic operators, GP evolutionary search is conducted on the syntactical space of programs with the only semantic guidance from an individual's fitness.

Although GP has shown to be effective in evolving programs for solving different problems using such (finite) behavior-based semantic guidance and pure syntactical genetic operators, this practice is somewhat unusual from a real programmers' perspective. Computer programs are not just constrained by syntax but also by semantics. As a normal practice, any change to a program should pay heavy attention to the change in semantics of the program. To amend this deficiency in GP, resulting from the lack of semantic guidance on genetic operators, Uy et al. [20] proposed a semantic-based crossover operator for GP, called Semantic Aware Crossover (SAC). The results reported in [20] show that using semantic information helps to improve performance of GP in terms of the number of successful runs in solving real-valued symbolic regression problems.

This paper extends the ideas presented in [20]. Our new operator, called Semantic Similarity based Crossover (SSC) is an improvement over SAC through the inclusion

of additional semantic information to control the change of semantics of individuals by only allowing swapping of two subtrees which are semantically similar while also being semantically different. Effectively an upper and lower bound on semantic difference is used to determine whether or not subtrees can be exchanged during a crossover event. By doing this, we expect that the change of fitness of an individual will be less destructive. This property has been proved to be very important in GP [19].

The paper is organised as follows. In the next section, we give a review of related work on semantic based crossovers in GP. In Section 3 we describe our new crossover operator and explain how it differs from the crossover operator proposed in [20]. The experiments on the new crossover operator 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 Previous Works

There has been a number of work in the literature on how to incorporate semantic information into GP. There are at least three ways in which semantics can be represented, extracted and used to guide GP: (a) using grammars [21, 2, 3], (b) using formal methods [9–11, 13, 12], and (c) based on GP tree-like structures [1, 16, 20].

In the first category, 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 obtained from individuals during the evlutionary process can be obtained. This information then can be used to remove bad individuals from the population as reported in [3] or to prevent generating semantically invalid individuals as in [21, 2]. The attributes used to present semantics are problem dependent and it is not always easy to design the attributes for each problem.

Within the second category, Johnson has advocated for using formal methods as a way of adding semantic information in GP [9–11]. In these methods, the semantic information extracted by using formal methods (e.g., Abstract Interpretation and Model Checking) is used to measure individuals' fitnesses 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 [13, 12]. In these works, semantics is also used to calculate the fitness of individuals.

Finally, with expression trees, semantic information has been incorporated mainly by modifying the crossover operator. Early work focused on the syntax and structure of individuals. In [8], the authors modified the crossover operator to take into account the depth of trees. Other work modified crossover taking into account the shape of the individuals [17]. More recently, context has been used as extra information for determining GP crossover points [6, 15]. However, all these methods have to pay extra time costing for evaluating the context of all subtrees within each individual in the population.

In [1], the authors investigated the effects of directly using semantic information to guide GP crossover on Boolean domains. The main idea 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 in-

formation if and only if they both are reduced to the same ROBDD. The semantic equivalence 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 so, the authors argue that there is an increase in the semantic diversity of the evolving population and as a consequence an improvement in the GP performance.

Uy et al. [20] proposed a new crossover operator (SAC), based on the semantic equivalence checking of subtrees. The approach was tested on a family of real-value symbolic regression problems (e.g., polynomial functions). The 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 no longer possible. Secondly, the crossover operator is guided not by the semantics of the whole program tree, but by subtrees. This is inspired by recent work presented in [16] for calculating subtree semantics.

3 Semantic Similarity based Crossover

Semantic Similarity based Crossover (SSC) presented in this section is an extension of SAC [20]. In SAC, the semantic equivalence of two subtrees 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. This information is then used to guide crossover by preventing the swap of two equivalent subtrees in each crossover operation.

The method proposed in this paper differs from SAC in two ways. Firstly, the concept of semantically equivalent subtrees is replaced by the concept of semantically similar subtrees. As in [20], the 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. Assuming that we need to check if two subtrees St_1 and St_2 are similar or not, then the pseudo-code for doing it is as follows:

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

where *Abs* is the absolute function and α and β are two predefined values. α is the *lower bound semantic sensitivity*, β is the *upper bound semantic sensitivity*. Perhaps, the best value 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 different values for both the *lower bound semantic sensitivity* and the *upper bound semantic sensitivity* to see how our new crossover performs.

The second difference between SSA and SAC is the way in which the semantic similarity is used to guide crossover operation. In [20], the equivalent semantics is used to guide crossover by trying to select other subtrees to do crossover only one time when two subtrees are semantically equivalent. In SSC, when two subtrees are considered

Algorithm 1: Semantic Similarity based Crossover

```
select Parent 1 P1;
select Parent 2 P2;
Count=0;
while count<Max_Attempt do
    choose at random crossover points at Subtree1 in P1;
    choose at random crossover points at Subtree2 in P2;
    if Subtree1 is not similar with Subtree2 then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
if Count=Max_Attempt then
    choose at random crossover points at Subtree1 in P1;
    choose at random crossover points at Subtree2 in P2;
    execute crossover;
    return true;
```

as not similar, we try a number of times to pick up two other subtrees. The reason is that picking up two similar subtrees is more difficult than selecting two non-equivalent subtrees in [20]. Algorithm 1 shows how SSC works.

In our experiments, Max_Atempt is set at 3. The motivation for SSC is that while encouraging GP individuals to exchange semantically different subtrees as in SAC [20], it is also desirable to prevent exchanging subtrees where the semantic difference is too large since it might cause substantial or even unbounded change in the semantics of the individuals after being crossed over. In other words, while forcing a change in the semantics 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. For instance, consider an individual with the root node as the arithmetic multiplication (*) and its left and right subtrees have return values (semantics) of 10 and 3 respectively. Then, this individual has the return value of 30 (=10*3). If the right subtree is replaced by a semantically similar subtree such as the return value is 3.2, the return value of the individual is only slightly changed to 10*3.2=32. However, if it is replaced by a semantically different subtree, with the return value of 100 for example, the semantics of the individual will change dramatically to 10*100=1000. This likely causes a big change in the fitness of the individual.

4 Experimental Setup

To investigate the possible effects of SSC and to compare it with SAC and SC, we used four real-valued symbolic regression problems of increasing difficulty. The underlying

Table 1. Symbolic Regression Functions

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

Table 2. Run and Evolutionary Parameter Values

Parameter	Value	Parameter	Value
Generations	50	Population size	500
Selection	Tournament	Tournament size	3
Crossover probability	0.9	Mutation probability	0.1
Initial Max depth	6	Max depth	15
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	$0.02, 0.04, 0.06, 0.08$		
Higher semantic sensitivities	$8, 10, 12$		
Trials per treatment	100 independent runs for each value.		

functions, from [7], are shown in Table 1, and the parameters used for our experiments are shown in Table 2. The reason for choosing the *lower bound semantic sensitivities* of SSC as the *semantic sensitivities* of SAC is these values helped to improve the performance of SAC over SC as shown in [20]. The reason for setting the *upper bound semantic sensitivities* values is inspired from the results of our experiments. These values are sufficient to demonstrate the performance of SSC.

5 Results and Discussion

We present the results of two experiments undertaken to understand the behaviour of Semantic Similarity based Crossover (SSC), the earlier Semantic Aware Crossover (SAC), with both benchmarked against standard crossover (SC). In the first instance we examine the classic performance metrics of mean best fitness and the number of successful runs, followed by an analysis of the locality of each operator.

5.1 Mean Fitness and Success Rates

Table 3 shows the percentage of successful runs. Figure 1 depicts the cumulative frequency using three different crossover operators, namely, SC, SAC, SSC (with *lower bound semantic sensitivity* and *upper bound semantic sensitivity* as 0.02 and 8, respectively). It can be seen from Table 3 that in almost all cases, SSC outperforms both SAC and SC. The exceptions mostly happen when solving the problem with target function F_1, which may simply be far too easy a problem to benefit from semantic information. Figure 1 shows that SSC usually find the perfect solutions faster than SAC and SC.

Table 3. Comparison of the percentage of successful runs.

sensitivities		F_1			F_2			F_3			F_4		
low	high	SC	SAC	SSC	SC	SAC	SSC	SC	SAC	SSC	SC	SAC	SSC
0.02	8	62	**70**	65	28	33	**42**	15	22	**34**	10	14	**26**
	10	62	70	**75**	28	33	**47**	15	22	**27**	10	14	**23**
	12	62	**70**	67	28	33	**37**	15	22	**29**	10	14	**17**
0.04	8	62	**70**	64	28	34	**38**	15	20	**32**	10	19	**23**
	10	62	70	**73**	28	34	**47**	15	20	**25**	10	19	**27**
	12	62	**70**	62	28	34	**36**	15	20	**31**	10	19	**20**
0.06	8	62	**71**	63	28	32	**39**	15	20	**32**	10	17	**27**
	10	62	71	**73**	28	32	**45**	15	20	**29**	10	17	**25**
	12	62	**71**	59	28	32	**33**	15	20	**32**	10	17	**19**
0.08	8	62	**70**	**70**	28	35	**42**	15	20	**26**	10	**17**	16
	10	62	**70**	**70**	28	**35**	**35**	15	20	**25**	10	17	**23**
	12	62	**70**	**70**	28	35	**37**	15	20	**24**	10	**17**	**17**

Table 4 gives the average of best solutions found in all runs of all GP systems. In this table, we use the shorhand *sen* for *sensitivity*. In Figure 2 we also show 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 sensititities* as 0.02 and 10 respectively. It is noted that, in these figure, we only show the statistics from the 10^{th} generation onwards. The reason is that at some 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 systems regardless of which crossover operator is used.

The results in Table 4 are consistent with those in Table 3 in that SSC is also superior than both SAC and SC finding solutions with better quality. Moreover, it can be observed from this table that the more difficult problem, the better performance achieved by SSC in comparison with SAC and SC. The results in Table 4 is also very solid as there is no exception in it. It expresses that SSC is also better than SAC and SC when they are compared by the average best fitness with the above *lower bound* and *upper bound semantic sensitivities*. Figure 2 shows that SSC not only outperforms than SAC and SC in terms of best fitness of runs but also better in terms of the mean best fitness at each generations.

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 SSC is significant. The t-test results of SAC in comparison with SC done in [20] is also transfered to this paper for the ease of comparison. The result of t-test (p-values) of both SSC and SAC in comparison with SC is shown in Table 5. In this table, if the improvement is remarkable, p-value is less than 0.05, and that value is bold faced as in the previous tables.

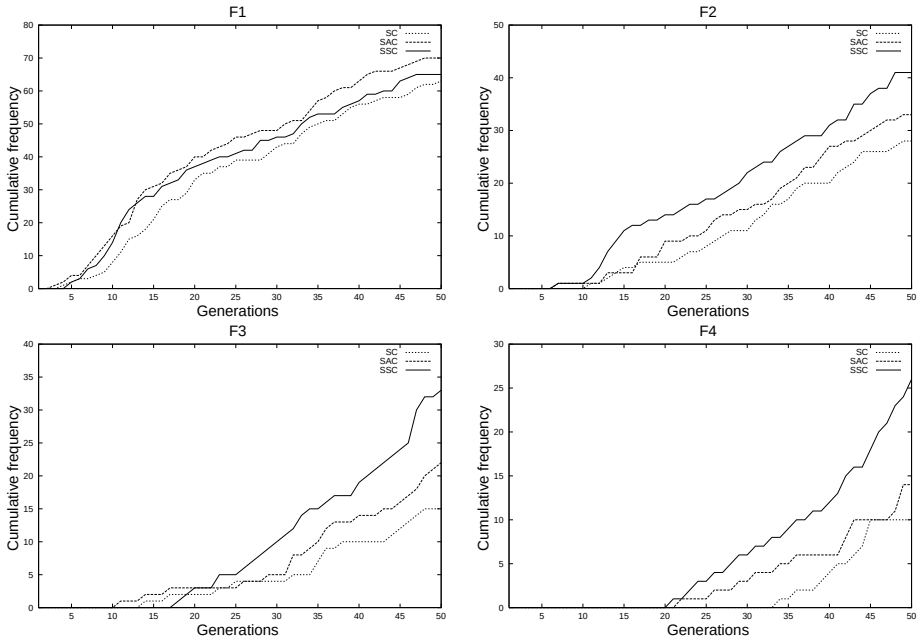

Fig. 1. Cumulative frequency α=0.02, β=8.

It can be seen from Table 5 that while the improvement in terms of the average best fitness of runs of SAC over standard crossover is either not signficant or borderline marginal, regardless of the upper and lower bounds explored on F_2, F_3 and F_4 there is almost always a significant difference is observed with SSC. The exceptions mostly lie in the easy-to-learn function F_1. On the contrary, in the most complicated target functions, F_3, F_4, there is no exception, whereas there are two exceptions in F_2. The results support the confirmation that the more difficult problem, the better GP performance is gained by using SSC.

5.2 Operator Locality

The next set of experimental results are for the investigation of the locality property of SSC. 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 an 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 crossover operator is different with other crossover operators in the literature in that it attempts to achieve high-locality.

Table 4. Comparison of the average best fitness over 100 runs.

sen		F_1			F_2			F_3			F_4		
low	high	SC	SAC	SSC	SC	SAC	SSC	SC	SAC	SSC	SC	SAC	SSC
0.02	8	0.13	0.13	**0.11**	0.26	0.24	**0.16**	0.30	0.28	**0.20**	0.40	0.33	**0.24**
	10	0.13	0.13	**0.09**	0.26	0.24	**0.14**	0.30	0.28	**0.21**	0.40	0.33	**0.23**
	12	0.13	0.13	**0.09**	0.26	0.24	**0.18**	0.30	0.28	**0.19**	0.40	0.33	**0.27**
0.04	8	0.13	0.13	**0.11**	0.26	0.23	**0.16**	0.30	0.27	**0.21**	0.40	0.33	**0.25**
	10	0.13	0.13	**0.09**	0.26	0.23	**0.13**	0.30	0.27	**0.21**	0.40	0.33	**0.23**
	12	0.13	0.13	**0.10**	0.26	0.23	**0.19**	0.30	0.27	**0.19**	0.40	0.33	**0.26**
0.06	8	0.13	0.10	**0.11**	0.26	0.23	**0.15**	0.30	0.27	**0.20**	0.40	0.32	**0.24**
	10	0.13	0.12	**0.08**	0.26	0.23	**0.16**	0.30	0.27	**0.21**	0.40	0.32	**0.23**
	12	0.13	0.12	**0.10**	0.26	0.23	**0.19**	0.30	0.27	**0.20**	0.40	0.32	**0.27**
0.08	8	0.13	0.14	**0.08**	0.26	0.22	**0.17**	0.30	0.28	**0.21**	0.40	0.33	**0.26**
	10	0.13	0.14	**0.09**	0.26	0.22	**0.17**	0.30	0.28	**0.21**	0.40	0.33	**0.26**
	12	0.13	0.14	**0.11**	0.26	0.22	**0.16**	0.30	0.28	**0.23**	0.40	0.33	**0.26**

Table 5. T-test result (p-values).

sen		F_1		F_2		F_3		F_4	
low	high	SAC	SSC	SAC	SSC	SAC	SSC	SAC	SSC
0.02	8	0.68	0.86	0.46	**0.00**	0.41	**0.00**	0.12	**0.00**
	10	0.68	0.44	0.46	**0.00**	0.41	**0.00**	0.12	**0.00**
	12	0.68	0.36	0.46	**0.02**	0.41	**0.00**	0.12	**0.00**
0.04	8	0.93	0.94	0.36	**0.00**	0.30	**0.00**	0.12	**0.00**
	10	0.93	0.29	0.36	**0.00**	0.30	**0.00**	0.12	**0.00**
	12	0.93	0.58	0.36	0.05	0.30	**0.00**	0.12	**0.00**
0.06	8	0.98	0.96	0.22	**0.00**	0.26	**0.00**	0.08	**0.00**
	10	0.98	0.21	0.22	**0.00**	0.26	**0.01**	0.08	**0.00**
	12	0.98	0.77	0.22	0.05	0.26	**0.00**	0.08	**0.00**
0.08	8	0.60	0.08	0.25	**0.00**	0.49	**0.01**	0.15	**0.00**
	10	0.60	0.40	0.25	**0.00**	0.49	**0.00**	0.15	**0.00**
	12	0.60	0.94	0.25	**0.00**	0.49	**0.02**	0.15	**0.00**

To compare the locality property of SSC with SAC and SC, an experiment was conducted where the fitness change of individuals before and after crossover is measured. For example, if two individuals having fitness of 10 and 15 are selected for crossover, and that after the crossover operation their children have fitness of 17 and 9. Then, the change of fitness of these individuals is $Abs(17-10)+Abs(9-15)=13$. 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 crossover is shown in Table 6. Again, in this table, the best results (the smallest values) are bold faced.

In the Figure 3 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 0.04 and 10. Table 6 and Figure 3 show that the step of the fitness change of our

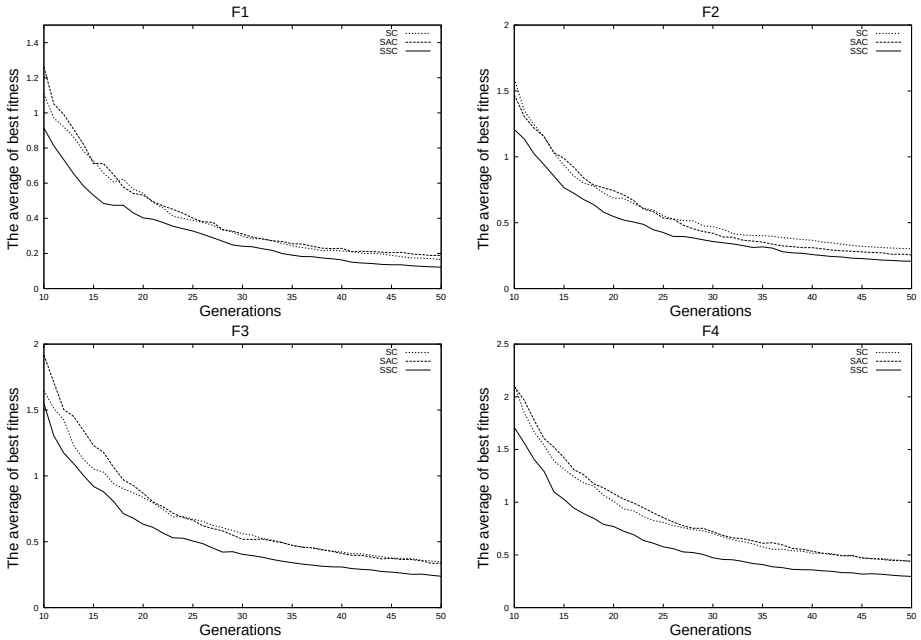

Fig. 2. Average of best fitness with α=0.02, β=10.

new crossover operator (SSC) is smaller than both SAC and SC. This means that the change of fitness over generations of SSC is smoother than SAC and SC. The table and figure also show that the fitness change of SAC is only slightly smoother than SC. These results explain why SSC is much better than SAC and SC on the problems tried, while SAC is also better than SC but only slightly.

6 Conclusion and Future Work

In this paper, we have proposed a new semantic based crossover operator for GP, Semantic Similarity based Crossover (SSC). The new operator was tested and analysed on a class of real-valued symbolic regression problems and the results were compared using Semantic Aware Crossover (SAC) and standard GP crossover (SC). The experimental results show that SSC helps to improve the performance of GP in comparison with SAC and SC both in terms of the percentage of successful runs and the average of best fitness over a number of runs. The results from the experiments also show that this operator not only helps to encourage the exchange of subtrees with different semantics as in [20], but also makes a smaller change of fitness during the evolutionary process, by only allowing exchange of subtrees which have a controlled degree of similarity, ensuring a more well-behaved operator in terms of locality. We argue that this is the main reason why SSC outperformed SAC and SC on the problems tried.

In the near future, we are planning to extend the work presented in this paper in a number of ways. Firstly, we are aiming to apply SSC on more difficult symbolic

Table 6. The average individual fitness change before and after crossover operation

sen		F_1			F_2			F_3			F_4		
low	high	SC	SAC	SSC	SC	SAC	SSC	SC	SAC	SSC	SC	SAC	SSC
0.02	8	8.8	8.6	**6.9**	10.1	8.7	**6.0**	10.3	10.3	**6.8**	11.8	9.7	**7.5**
	10	8.8	8.6	**6.1**	10.1	8.7	**5.8**	10.3	10.3	**6.6**	11.8	9.7	**7.3**
	12	8.8	8.6	**6.5**	10.1	8.7	**5.9**	10.3	10.3	**7.2**	11.8	9.7	**7.5**
0.04	8	8.8	8.7	**6.7**	10.1	8.5	**6.1**	10.3	10.1	**7.4**	11.8	9.7	**8.0**
	10	8.8	8.7	**6.0**	10.1	8.5	**5.3**	10.3	10.1	**6.8**	11.8	9.7	**7.2**
	12	8.8	8.7	**6.0**	10.1	8.5	**5.8**	10.3	10.1	**7.3**	11.8	9.7	**7.4**
0.06	8	8.8	8.2	**6.8**	10.1	7.3	**5.9**	10.3	9.3	**7.2**	11.8	9.5	**7.8**
	10	8.8	8.2	**6.2**	10.1	7.3	**6.1**	10.3	9.3	**6.8**	11.8	9.5	**7.4**
	12	8.8	8.2	**5.7**	10.1	7.3	**5.8**	10.3	9.3	**7.4**	11.8	9.5	**7.3**
0.08	8	8.8	8.3	**6.6**	10.1	7.4	**5.3**	10.3	9.4	**7.3**	11.8	9.6	**7.9**
	10	8.8	8.3	**5.9**	10.1	7.4	**5.1**	10.3	9.4	**6.8**	11.8	9.6	**7.5**
	12	8.8	8.3	**5.4**	10.1	7.4	**6.1**	10.3	9.4	**7.4**	11.8	9.6	**6.9**

regression problems (the problems that are multi-variable and more complex in the structure of the solutions). For these problems, we predict that making small change in semantics is more difficult and also more important. Secondly, SSC could be used to enhance some previous proposed crossover operators that are purely based on the structure of trees such as crossover with bias on the depth of nodes [8] or one point crossover [17]. Another potential research direction is to apply SSC 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 from their parents in terms of semantics. Last but not least, we are planning to investigate the range of *lower bound semantic sensitivity* and *upper bound semantic sensitivity* values that are good for a class of problems. In this paper, these values are manually and experimentally specified, however, it may be possible to allow these values to self-adapt during the evolutionary process [4].

Acknowledgements

This paper was funded under a Postgraduate Scholarship from the 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.

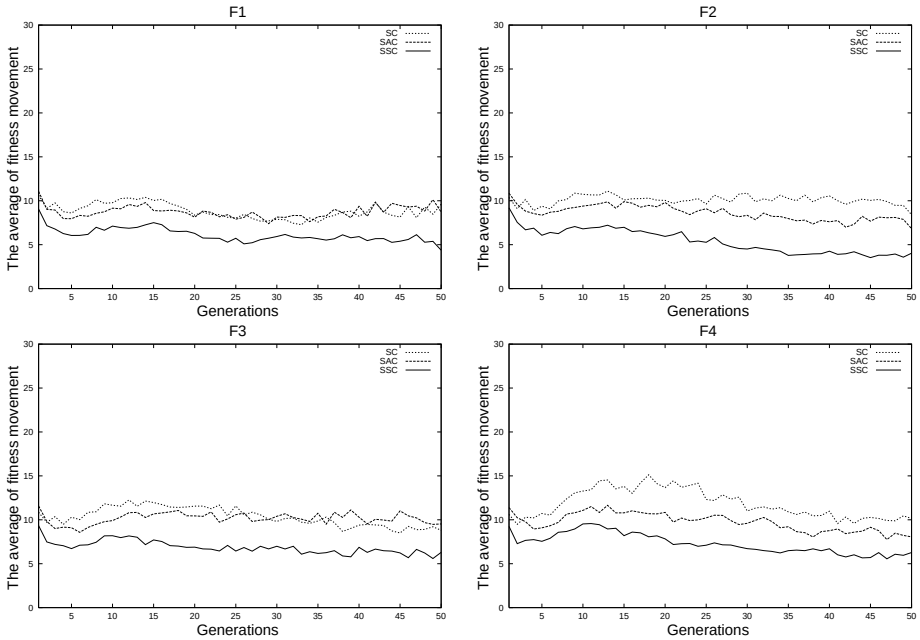

Fig. 3. The average fitness movement before and after crossover with α=0.04, β=10

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.
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. S. Hengpraprohm and P. Chongstitvatana. Selective crossover in genetic programming. In *Proceedings of ISCIT International Symposium on Communications and Information Technologies*, pages 14–16, November 2001.
7. 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.
8. T. Ito, H. Iba, and S. Sato. Depth-dependent crossover for genetic programming. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 775–780. IEEE Press, May 1998.
9. 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.
10. C. Johnson. What can automatic programming learn from theoretical computer science. In *Proceedings of the UK Workshop on Computational Intelligence*. University of Birmingham,

2002.
11. 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.
12. 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.
13. 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.
14. J. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MITPress, MA, 1992.
15. H. Majeed and C. Ryan. A less destructive, context-aware crossover operator for gp. In *Proceedings of the 9th European Conference on Genetic Programming*, pages 36–48. Lecture Notes in Computer Science, Springer, April 2006.
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. R. Poli and W. B. Langdon. Genetic programming with one-point crossover. In *Proceedings of Soft Computing in Engineering Design and Manufacturing Conference*, pages 180–189. Springer-Verlag, June 1997.
18. R. Poli and W. L. N. McPhee. *A Field Guide to Genetic Programming*. http://lulu.com, 2008.
19. 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.
20. N. Q. Uy, N. X. Hoai, and M. ONeill. Semantic aware crossover for genetic programming: the case for real-valued function regression. In *Proceedings of EuroGP09*. Springer.
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.