On the Analysis of Semantic Aware Crossover

Nguyen Quang Uy[1], Nguyen Xuan Hoai[2], Michael O'Neill[1],
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
nxhoai@gmail.com

Abstract. It is well-known that the crossover operator plays a very important role in genetic programming (GP). It is also widely admitted that standard crossover is made mostly randomly without semantic information. The lack of semantic information is the main reason that causes destructive effect, generally producing children worse than parents, of standard crossover. Recently, we have proposed a new semantic based crossover for GP, that is called Semantic Aware Crossover (SAC) [26]. It was shown in [26] that SAC outperforms standard crossover (SC) in solving a class of real-value symbolic regression problems. This paper extends [26] by giving some deeper analyses to understand why SAC helps to improve the performance of GP in solving these problems. The analyses show that SAC can increase the semantic diversity of population and this helps to reduce the crossover destructive effect in GP. The results also show that although SAC requires more time for checking semantics, this extra time is negligible.

Key words: Semantic aware crossover, semantic, constructive effect, bloat

1 Introduction

Genetic programming (GP) is an evolutionary algorithm inspired by biological evolution to find the solution as computer programs for an user-defined task [17]. The program is usually represented in a language of syntactic formalism such as s-expression trees [17], a linear sequence of instructions, grammar derivation trees, or graphs [24]. 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 the fitness of program measured by the difference of behavior of evolving programs and the target programs (usually on a finite input-output set called fitness cases).

Although GP has been shown to be effective in evolving programs for solving different problems using such (finite) behavior-based semantic guidance and pure syntactical genetic operator, this practice is somewhat unusual from 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 and not just those changes that guarantee to maintain the program syntactical validity. To amend this deficiency in GP resulting from the lack

of semantic guidance on genetic operators, recently, we have proposed a semantic-based crossover operator for genetic programming [26] that is called Semantic Aware Crossover (SAC). The experimental results in [26] show that using semantic guidance on the crossover operator helps to improve GP in terms of the number of successful runs in solving a class of real-value symbolic regression problems.

In this paper we extend work in [26] by giving some deeper analyses to understand why SAC helps to improve the performance of GP on the problems tried. The analyses show that SAC can increase the semantic diversity of population and this helps to reduce the crossover destructive effect in GP. The results also show that although SAC requires more time for checking semantics, this extra time is negligible.

The paper is organized as follows. In the next section, we give a review of related works on semantic based operations and semantics based crossover in GP. Section 3 describes briefly our crossover (SAC) proposed in [26]. The experiment setting 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 Related works

Using semantic information in genetic programming is not new, there has been a number of related research over the years. The use of semantic information in the literature of GP could be seen in three ways: Using grammars [27, 3, 4], using formal methods [11–13, 15, 14] and based directly on GP expression tree representation [1, 20, 26]. In the first way, Attribute Grammars have been the most popular formalisms used to incorporate semantic information into GP. By using an attribute grammar and adding some attributes to individuals, we can check some useful semantic information of individuals during the evolutionary process. This information can subsequently be used to remove bad individuals from the population [4] or can be used to prevent generating invalid individuals [27, 3]. However, the attributes that are used to present semantics are problem dependent. Moreover, it might not always be easy to design 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 [11–13]. In [12], he proposed a number of possible ways for incorporating program semantics extracted by formal method techniques into GP. In these methods, the semantic information that is extracted by using formal methods, mostly based on Abstract Interpretation and Model Checking, is mainly used as a way of measuring the fitness of individuals in some problems that are difficult to use a sample points based traditional fitness measure. Katz and coworkers used a model checking to solve Mutual Exclusion problem [15, 14]. In these works, semantics are extracted/calculated and then incorporated into the fitness of individuals.

The use of semantic information on expression trees has been realized in the modification of the crossover operator. Some first modifications of the standard subtree crossover in GP focused on syntax and structure of individuals. In [10], the crossover is implemented based on the depth of the trees or as in [25] based on the shape of trees. More recently, context has been considered as extra information for determining

crossover points in GP [7, 18] which is perhaps most close to exploiting semantic information for modifying the standard crossover. The weakness of these context based methods is that it is rather time consuming to evaluate the context of all subtrees of an individual as required by these approaches. In [1], the authors investigate the effect of directly using semantic information to guide the crossover operator in GP on Boolean domains. Their main idea is to check the semantic equivalence between the newly born children with their parents. The semantic equivalence checking of two Boolean expression trees is done by transforming the trees to reduced ordered binary decision diagrams (ROBDDs), and that they have the same semantic if and only if they are reduced to the same ROBDD. The semantic equivalence checking is then used to determine which of the individual participating in crossover operation will be copied to the next generation. If the children born as the result of crossover are semantically equivalent with their parents, they are not copied to the next generation, their parents are copied instead. By doing this, the authors argued that it helps to increase the semantic diversity of evolving population of programs that helps to improve the performance of GP in these problems.

In our previous work [26], we proposed a new crossover operation, called Semantic Aware Crossover (SAC), based on the semantic equivalence checking of subtrees. GP with SAC was applied to a family of real-value symbolic regression problems and the experiment results show that SAC is really effective. Our work in [26] is different from [1] in two ways. Firstly, the domain for testing semantically driven crossovers is real-valued rather than Boolean. For real-valued domains, the idea of checking semantic equivalence by reducing to common ROBDDs is no longer possible. Secondly, the semantic guidance of the crossover operator is not from the whole program tree behavior but from subtrees. This is inspired by recent work in [20] for calculating subtree semantics. However, the subtree semantic calculated in [26] is for real-valued domains but not Boolean domains as in [20].

3 Semantic Aware Crossover

The aim of the study in [26] is to extend the earlier work [1, 20] to real-valued domains. For such problems it is not easy to compute the semantics or semantic equivalence of two expression trees by reducing them to a common structure as for Boolean domain as in [1]. Similarly, complete enumeration and comparison of subtree fitness as in [20] is also impossible on real domains. In fact, the problem of determining semantic equivalence between two real-valued expressions is known to be complete NP-hard [6]. Therefore, we have to calculate the approximate semantics. In [26], a simple method for measuring and comparing the semantics of two expressions is used. To determine the semantic equivalence of two expressions, we measure them against a random set of points sampled from the domain. If the output of the two trees on the random sample set are close enough (subject to a parameter called semantic sensitivity) then they are designated as semantically equivalent. It can be written in pseudo-code as follows:

```
If Abs(Value_On_Random_Set(P1)-Value_On_Random_Set(P2))<ε then
          Return P1 is semantically equivalent to P2.
```

Where Abs is the absolute function and ε is a predefined constant called the *semantic sensitivity*. This method is inspired by the simple technique for simplifying expression

trees proposed in [22] called equivalence decision simplification (EDS), where complicated subtrees could be replaced by much simpler and templated subtrees if they are semantically equivalent.

The semantic equivalence of two subtrees could be used to control the crossover operation by constraining the operator in such a way that if the two subtree under the crossover point are semantically equivalent, the operator is forced to be executed on two new crossover points. The algorithms for SAC given in [26] is as follow:

```
1.1.Select two parents: P1, P2
1.2.Choose at random crossover points at Subtree1 in P1
    Choose at random crossover points at Subtree2 in P2
    if (Subtree1 is not equivalent with Subtree2){
    Execute crossover
    Add the children to the new population
    Return TRUE }
    else{
    Choose at random crossover points at Subtree1 in P1
    Choose at random crossover points at Subtree2 in P2
    Execute crossover
    Return TRUE}
```

The motivation for doing SAC is to encourage GP individual trees to exchange subtrees that have different semantics, which is expected to encourage the change in semantics of the whole trees after each crossover. In [26], GP with SAC was proven to have the best performance on a family of real-valued regression problems in comparison with GP coupled with standard crossover (SC) and some other semantic checking based operation. However, the root of this success and the proof of achieving the semantic diversity in SAC motivation have not been clearly demonstrated. The following sections will give further analyses of the GP runs in [26], to gain further insight into the succint cause of the success of SAC.

4 Experiment settings

All parameters of the experiments in this papers are the same as in our privious work [26] meaning that the standard crossover and SAC are tested on a class real-valued of symbolic regression problems with target functions as a family of polynomials of increasing degree given in [9]:$F_1 = X^3 + X^2 + X$, $F_2 = X^4 + X^3 + X^2 + X$, $F_3 = X^5 + X^4 + X^3 + X^2 + X$, *and* $F_4 = X^6 + X^5 + X^4 + X^3 + X^2 + X$. The parameters setting for SC and SAC are the same and as follows:

- Population size: 500
- Number of generation: 50
  - Tournament selection size: 3
  - Crossover probability: 0.9
  - Mutation probability: 0.1
  - Max depth of program tree at the initial generation: 6
  - Max depth of program tree at all time: 15
  - Non-terminals: +, -, *, / (protected version), sin, cos, exp, log (protected)

- Terminals: X, 1
- Number of sample: 20 random points from [-1. . . 1].
- Hit: when an individual has an absolute error < 0.01 on a fitness case.
- Termination: A program scores 20 hits or maximum generation is exceeded.

The *semantic sensitivities* used in the experiment are: 0.01, 0.02, 0.04, 0.05, 0.06, 0.08, 0.1. The reason why we choose this semantic sensitivities is that they are the values that help to improve the performance of SAC versus SC as has been shown in [26]. For each kind of crossover (SC and SAC), each target problem, and semantic sensitivity, 100 runs are performed which makes the total number of runs 5600.

5 Results and Discussion

To analysis the behavior of SAC and to compare it with SC, we conducted rerun the experiments in [26] and collect stattistics on some aspects of them. These statistics and analyses are presented in the following subsections.

5.1 Equivalent crossovers

In the first experiment analysis, we investigate the question of frequency of semantically equivalent crossover event. It means how often SAC and SC exchange the semantically equivalent subtree. To answer this question we did collect statistics as the percentage of such crossovers events averaged over all generation and all runs in [26]. The result is shown in Table 1. we also graph the average percentage of semantically equivalent crossover event over 100 runs for each of 50 generations with *sensitivity* as 0.01 in Figure 1

Table 1. The average percentage of equivalent crossovers

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	24.2%	24.2%	24.2%	24.2%	24.2%	24.2%	24.2%
	SAC	**5.4%**	**5.3%**	**5.3%**	**5.3%**	**5.3%**	**5.3%**	**5.3%**
F_2	SC	22.1%	22.1%	22.1%	22.1%	22.1%	22.1%	22.1%
	SAC	**4.2%**	**4.2%**	**4.2%**	**4.2%**	**4.2%**	**4.2%**	**4.2%**
F_3	SC	22.5%	22.6%	22.5%	22.6%	22.6%	22.6%	22.6%
	SAC	**4.6%**	**4.5%**	**4.5%**	**4.5%**	**4.5%**	**4.5%**	**4.5%**
F_4	SC	21.8%	21.9%	21.9%	21.9%	21.9%	21.9%	21.9%
	SAC	**4.1%**	**4.1%**	**4.1%**	**4.1%**	**4.1%**	**4.1%**	**4.1%**

From Table 1 and Figure 1 it can be seen that overall the average percentage of semantically equivalent crossover events in SC (about 20%) is 5 folds bigger than SAC (about 4%). We also have conducted an experiment to test how crossover affect the relative fitness of the children to their parent when it swaps two semantically equivalent subtrees. The result is that in nearly all cases (about 98%), such crossover will produce

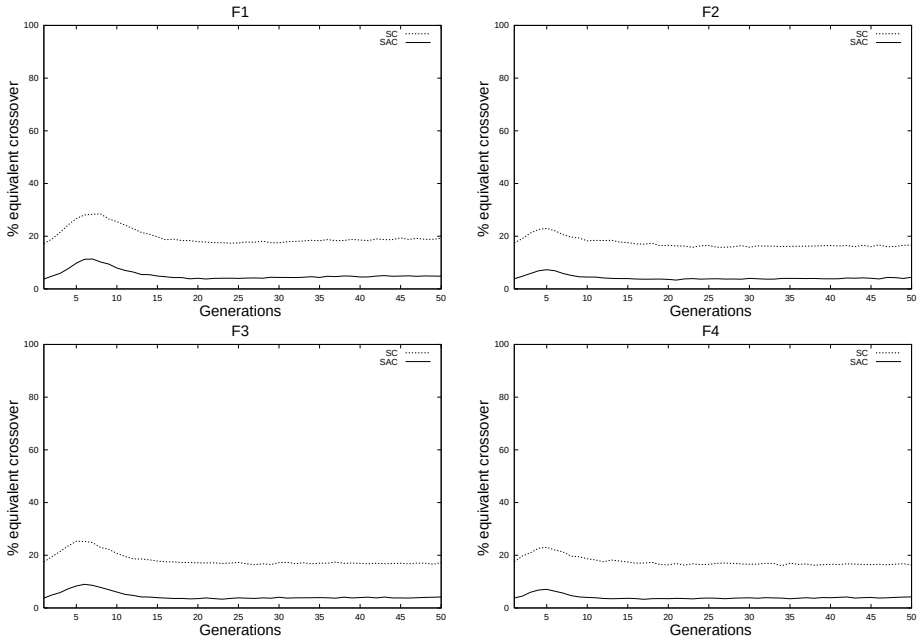

Fig. 1. The average percentage of equivalent crossovers with *sensitivity*=0.01

the two children that have identical fitness with their parents. This infers that about 20% of SC and much smaller with about 4% of SAC, does not produce new children during the evolutionary process. Therefore, we argue that SAC is more semantical exploratory than SC on the problems tried.

Table 2. The average percentage of different children from their parent in crossover

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	63.4%	63.4%	63.4%	63.4%	63.4%	63.4%	63.4%
	SAC	**73.4%**	**72.1%**	**71.7%**	**73.2%**	**73.0%**	**73.5%**	**73.7%**
F_2	SC	66.8%	66.8%	66.8%	66.8%	66.8%	66.8%	66.8%
	SAC	**80.3%**	**79.1%**	**80.0%**	**80.5%**	**80.7%**	**80.3%**	**80.4%**
F_3	SC	67.6%	67.6%	67.6%	67.6%	67.6%	67.6%	67.6%
	SAC	**77.7%**	**79.7%**	**80.4%**	**78.1%**	**78.3%**	**78.1%**	**78.5%**
F_4	SC	67.9%	67.9%	67.9%	67.9%	67.9%	67.9%	67.9%
	SAC	**80.2%**	**80.1%**	**80.0%**	**80.7%**	**80.8%**	**80.7%**	**80.4%**

5.2 Semantic diversity

In the previous section, statistics has shown that, on the problem tried SAC encourage more exchange of semantically different subtrees, which inevitably encourage the change in semantics of the children compared to their parent. It triggers the second experiment in this section on semantic diversity. Population diversity has been long seen as a crucial factor in genetic programming [2]. In general, the search process will be more robust if the more population diversity is maintained. There are two kind of metrics which have been used for measuring and controlling the diversity of population is genotypic diversity and phenotypic diversity [8]. While the first one concerts to syntax (structure) of individuals in the population [23], the second one is based on the behavior (fitness) [21] of individuals in the population. In this paper, we propose a new measure for semantic diversity of genetic operators called *semantic diversity of crossover* (SDC). SDC is different with other metrics in that it does not aim to measure the difference between the individuals in the same population but to measure the difference between the individuals of the two successive populations. In other word, SDC is used to measure how are individuals different before and after crossover. Here, the difference between individuals before and after crossover is again determined based on a set of random points drawn from the problem domain.

We use SDC to measure the semantic diversity of SC and SAC by counting the percentages of these crossover events in the runs that generated semantically new children from their parents. This value is then averaged over 50 generations and 100 runs and shown in table 2. We also show in Figure 2. the average percentage of different children over 100 runs for each of 50 generations with sensitivity as 0.05.

It is obvious from Table 2 and Figure 2 that there is a strong correlation with the statistics given in the previous subsection. In some first generations, about 70% of SC generated different children while this value of SAC was nearly 90%. It is important as in the early phase of evolutionary process, it is expected that GP would have high exploration capacity in creating (semantically) new individuals. During the evolutionary process, the percentage of different children of both crossover go down. However, SAC is always about 15% higher than one of SC. It should be noted that generally in SAC it is not guaranteed that SAC always generates two semantically new children even when the two semantically equivalent subtrees is prevented.From our point of view the reason might lie in the existence of some fixed semantic subtrees as in the boolean domains, which was shown in [20]. However, further analysis needs to be conducted to reach a more certain conclusion.

5.3 Constructive effect

As is shown in section 5.2, SAC is more semantically productive than SC generating more children that are different with their parents. However, it would be interesting to ask, at least on the problem tried, whether this helps the crossover operations in breeding better children from the parents (more constructive crossover). To answer this question we conducted an experiment on constructive effect of SAC and SC. The methodology to measure the constructive effect in here is similar to that [19]. It means that comparison the constructive effect of two crossovers is done by simply calculating the percentages

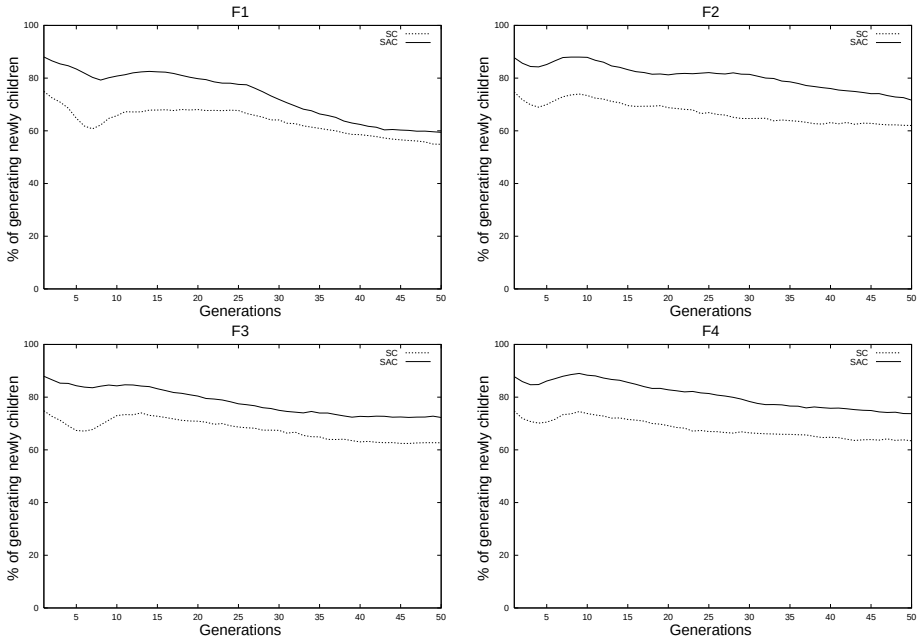

Fig. 2. The percentage of generating newly children of two crossover schemas

of the events of generating a better child from its parents through crossover. This value is then averaged over number of generations and number of runs. The results are given in Table 3, and in Figure 3 depicts the average percentage of generating a better child from its parent over 100 runs for each 50 generations with *sensitivity* is 0.08.

Table 3. The average percentage of better children than their parent in crossover

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	10.5%	10.5%	10.5%	10.5%	10.5%	10.5%	10.5%
	SAC	**15.5%**	**15.0%**	**14.7%**	**15.4%**	**15.3%**	**15.5%**	**15.7%**
F_2	SC	11.3%	11.3%	11.3%	11.3%	11.3%	11.3%	11.3%
	SAC	**17.2%**	**16.7%**	**16.9%**	**17.3%**	**17.4%**	**17.3%**	**17.4%**
F_3	SC	11.7%	11.7%	11.7%	11.7%	11.7%	11.7%	11.7%
	SAC	**15.9%**	**16.8%**	**17.0%**	**16.0%**	**16.1%**	**16.0%**	**16.1%**
F_4	SC	11.4%	11.4%	11.4%	11.4%	11.4%	11.4%	11.4%
	SAC	**16.7%**	**16.8%**	**16.7%**	**17.0%**	**17.1%**	**17.0%**	**17.0%**

The result from Table 3 and Figure 3 shows that SAC is more fitness constructive, often 5%-10% better, than SC. This result explains why the performance of SAC was better than SC in terms of number of successful runs as in [26].

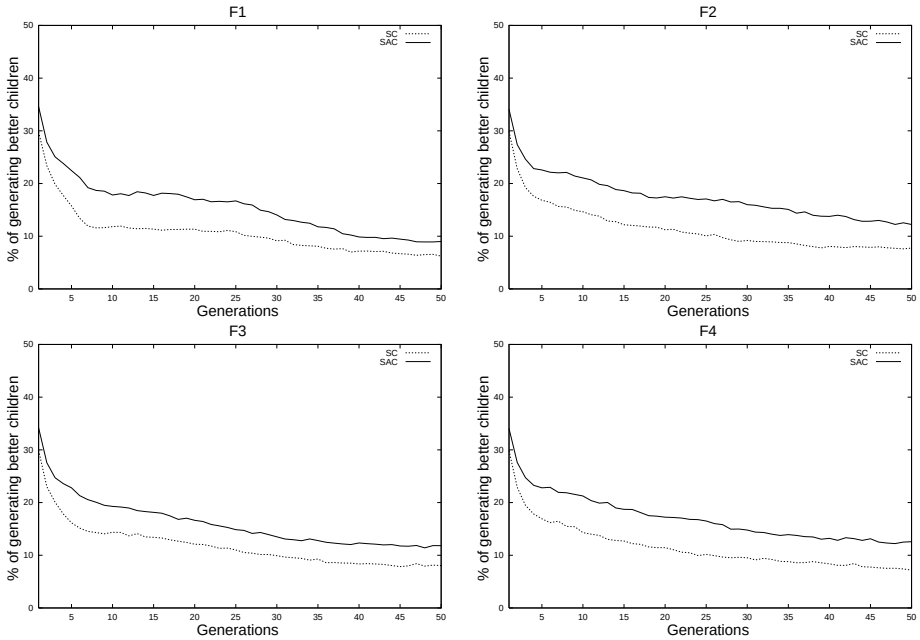

Fig. 3. The percentage of constructive crossover with *sensitivity* is 0.08

5.4 Code bloat

The better performance of SAC as observed in [26] goes with a cost as it takes more time to calculate the subtree semantics. It reflected in the slightly higher running time of SAC compared to SC. But how much really expensive the extra calculations are? In this paper, we do not want to compare the running time only and simply but to look deeper in the reasons that cause the extra computation time of SAC and investigate how much that expensiveness likely to be. To understand the root of possible extra computation time of SAC compared to SC, we conducted a code bloat analysis for the runs of the two operators. Here, it is supposed that the extra computation time of SAC might mainly come from two sources. The first source is that the individuals in SAC runs were more complicated than those of SC. Therefore, the time to evaluate individual fitness in SAC runs is higher than in SC runs. The second source is the time needed to compare the semantic equivalence of the two subtrees. To determine which is the main source we collected two statistics from the experiment runs. The first one is the average size of individuals (number of nodes) over 50 generations and averaged over 100 runs of SAC versus SC. The second one is the average size of subtrees which need to be checked for the semantical equivalence testing in SAC. This is averaged for each of 50 generation and over 100 runs. The two statistics are shown in Table 4, and Table 5 respectively.

It can be seen from Table 4 and Table 5 that the higher running time of SAC was caused not only by the calculation of subtree semantics but also by the increase of the size of individuals (bloat). However, these two time measure are almost negligible for

Table 4. The average size of individuals

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	36.8	36.8	36.8	36.8	36.8	36.8	36.8
	SAC	43.5	43.2	43.7	43.5	43.7	43.9	43.9
F_2	SC	42.5	42.5	42.5	42.5	42.5	42.5	11.3
	SAC	47.3	49.3	49.1	46.8	46.9	46.9	46.7
F_3	SC	43.5	43.5	43.5	43.5	43.5	43.5	43.5
	SAC	47.9	47.7	47.3	48.2	48.3	48.4	47.7
F_4	SC	45.2	45.2	45.2	45.2	45.2	45.2	45.2
	SAC	51.1	50.9	50.4	50.6	50.5	51.0	50.8

Table 5. The average size of subtrees in SAC

Sensitivity	0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	4.6	4.3	4.4	4.6	4.8	4.6	4.9
F_2	4.7	4.5	4.6	4.9	5.0	4.7	4.8
F_3	4.7	4.5	4.4	4.6	5.1	4.7	4.9
F_4	4.7	4.5	4.7	4.6	5.0	4.6	5.0

the problem tried. In Table 5, it can be seen that the average size of subtrees in SAC is very small in comparison with average size of individuals. Therefore, the time needed to calculate and compare subtree subtree fitnes is small. Moreover, there are also some methods which could be used to store semantics of these subtrees and that leads to more efficient subtree semantic calculation. One example of such method is the use of cache as in [16], which we aim to do in further extensions of SAC. The average individual size in SAC was bigger than that of SC but only with a very small margin. Moreover, there could also be some ways to reduce further the size of the individuals in SAC runs. For instance size of subtrees could be incorporated into the selection of crossover points apart from semantic information which might prefer smaller subtrees.

6 Conclusion and future works

In this paper we have compared SAC and SC on different aspects. Firstly, we have pointed out that there are about 20% of SC operations is the swap of semantically equivalent subtrees. This likely leads to the breed of children that are semantically similar to their parents. This weakness of SC can be amended in SAC by preventing the occurence of such swapping operations. Secondly, we have shown that, at least on the problem tried, SAC helps to promote better semantic diversity generating more semantically new children than SC. The results also show that SAC is more constructive than SC. This can be seen as a direct consequence of better semantic diversity obtained with SAC. Furthermore, we also show that the extra computation time of SAC compared to SC is almost negligible.

In future, we aim to apply SAC to various and more difficult symbolic regression problems (such multi-variate regression with more complex solution structure re-

quired). In these problems the promotion of semantic diversity might be more difficult. we are also planning to combine SAC idea with some of previous proposed crossovers in the literature that are based on the structure of trees such as crossover with bias on the depth of nodes as in [10] or one point crossover as in [25]. Another potential research is to apply SAC on problems of Boolean domain that as in [20], of which it is very difficult to generate the children that are different from their parents in terms of semantics. Finally, we are intending to investigate the suitable range of *semantic sensitivity* for each class of problems. In this papers, these values were determined mostly by hand tunning. However, these value can be incorporated into GP individual and get evolved in a way that is similar to the self-adaptation of genetic algorithm parameters as in [5].

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. E. K. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004.
3. 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.
4. 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.
5. 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.
6. M. A. Ghodrat, T. Givargis, and A. Nicolau. Equivalence checking of arithmetic expressions. In *Proceedings of the CASES05*. ACM, September 2005.
7. 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.
8. N. T. Hien and N. X. Hoai. A brief overview of population diversity measures in genetic programming. In *Proceedings of 11th Asia-Pacific Workshop on Intelligent and Evolutionary Systems*, pages 128–139. Vietnamese Military Technical Acadely.
9. 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.
10. 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.
11. 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.
12. C. Johnson. What can automatic programming learn from theoretical computer science. In *Proceedings of the UK Workshop on Computational Intelligence*. University of Birmingham, 2002.

13. 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.
14. 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.
15. 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.
16. M. Keijzer. Alternatives in subtree caching for genetic programming. In *Proceedings of the Genetic Programming 7th European Conference*, pages 328–337. Springer-Verlag, April 2004.
17. J. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MITPress, MA, 1992.
18. 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.
19. H. Majeed and C. Ryan. On the constructiveness of context-aware crossover. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO)*, pages 1659–1666. ACM Press, July 2007.
20. 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.
21. N. Mori. A novel diversity measure of genetic programming. In *Proceedings of Randomness and Computation: Joint Workshop "New Horizons in Computing" and "Statistical Mechanical Approach to Probabilistic Information*, pages 18–21.
22. N. Mori, R. McKay, N. X. Hoai, and D. Essam. Equivalent decision simplification: A new method for simplifying algebraic expressions in genetic programming. In *Proceedings of 11th Asia-Pacific Workshop on Intelligent and Evolutionary Systems*.
23. U. M. O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation*, pages 4092–4097. IEEE.
24. R. Poli, W. Langdon, and N. McPhee. *A Field Guide to Genetic Programming*. http://lulu.com, 2008.
25. 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.
26. 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.
27. 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.