An 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

Abstract. It is well-known that the crossover operator plays an important role in Genetic Programming (GP). In Standard Crossover (SC), semantics are not used to guide the selection of the crossover points, which are generated randomly. This lack of semantic information is the main cause of destructive effects from SC (e.g., children having lower fitness than their parents). Recently, we proposed a new semantic based crossover known GP called *Semantic Aware Crossover* (SAC) [25]. We show that SAC outperforms SC in solving a class of real-value symbolic regression problems. We clarify the effect of SAC on GP search in increasing the semantic diversity of the population, thus helping to reduce the destructive effects of crossover in GP.

Keywords: Semantic Aware Crossover, Semantic, Constructive Effect, Bloat.

1 Introduction

Genetic Programming (GP) is an evolutionary algorithm, inspired by biological evolution, which finds solutions in the form of computer programs for a user-defined task [16]. The program is usually represented in the language of a syntactic formalism such as S-expression trees [16], a linear sequence of instructions, grammar derivation trees, or graphs [23]. The genetic operators in GP are usually designed to guarantee the syntactic closure property, i.e., to produce syntactically valid children from syntactically valid parent(s). In this process, GP evolutionary search is conducted on the syntactic space of programs, with the only semantic guidance coming from the fitnesses of programs. Although GP has shown its effectiveness in evolving programs for solving different problems, this practice is somewhat unusual from the perspective of real programmers. Computer programs are constrained not only by syntax, but also by semantics. In normal practice, any attempt to change to a program should pay heavy attention to the change in semantics. To see how this would play out in GP, we have recently proposed a semantic-based crossover operator called *Semantic Aware Crossover* (SAC) [25]. Our previous results showed that using semantic guidance for the crossover operator in GP improves performance, measured in terms of successful runs in solving a particular problem (in our work, we used real-valued symbolic regression problems). In this paper we extend our previous work by performing a deeper analysis of how SAC affects GP search and reduces the destructive effects of crossover.

The paper is organised as follows. In the following section, we describe previous work on semantic based operators. In Section 3, we briefly describe our approach (i.e.,

Z. Cai et al. (Eds.): ISICA 2009, CCIS 51, pp. 56–65, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Semantic Aware Crossover). Section 4 provides details on the experimental setup used to conduct our analysis. The results presented in this paper are discussed in Section 5. Finally, conclusions are drawn in Section 6.

2 Previous Work

Using semantic information in GP is not new – there have been a number of related studies in recent years. The use of semantic information in GP can be grouped into three categories, on the basis of their use of: grammars [26,3,4]; formal methods [10,11,12,14,13]; and GP representation [1,19,25].

Attribute grammars have been one of the most popular methods to incorporate semantic information into GP. By using an attribute grammar, and adding attributes to individuals, we can check useful semantic information about individuals during the evolutionary process. This information can be used to remove bad (i.e., less fit) individuals [4] or to prevent generation of invalid individuals [26,3]. However, the attributes are problem dependent, and it may be difficult to design attributes for some problems.

Recently, Johnson has advocated formal methods as a means to incorporate semantic information in the evolutionary process [10,11,12]. In [11], Johnson proposed a number of possible ways to incorporate semantics of programs. In these methods, the semantic information that is extracted by using formal methods, mostly based on Abstract Interpretation and Model Checking, is used as a way of measuring the fitness of individuals for some problems where sampling techniques are difficult to use. Katz and co-workers used model checking to solve a Mutual Exclusion problem [14,13]. In these works, semantics are extracted/calculated and then incorporated into the fitness of an individual.

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 [9], the authors modified crossover to take into account the depth of trees. Other work modified crossover to take into account the shape of the individual [24]. More recently, context has been used as extra information for determining GP crossover points [6,17]. However these methods all have to pay the huge time cost of evaluating the context of all subtrees of all individuals.

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 information if and only if they reduce 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 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 [25], 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 [19] for calculating subtree semantics.

3 Semantic Aware Crossover

The aim of our previous work [25] was to extend the ideas in [1,19] to real-value domains. For real domains it is not feasible to compute the semantics by reduction to canonical form, as it was for the Boolean domain in [1]. Complete enumeration and comparison of subtree fitness as in [19] is also impossible on real domains. In fact, the problem of determining semantic equivalence between two real-value expressions is known to be complete NP-hard [5]. Therefore, we have to calculate the approximate semantics. In [25], 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 Sentivity*) then they are semantically equivalent, or in pseudo-code:

```
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 threshold for semantic sensitivity. This method is inspired by the simple technique for simplifying expression trees proposed in [21] known as Equivalence Decision Simplification (EDS), where complicated subtrees are replaced by simpler subtrees if they are semantically equivalent. The semantic equivalence of two subtrees can be used to control the crossover operation by constraining it so that if the corresponding subtrees beneath the crossover point are semantically equivalent, the selection is repeated with two new crossover points. Algorithm 1 shows how SAC works. The motivation behind SAC is to promote GP

Algorithm 1. Semantic Aware Crossover

select Parent 1 P_1;
select Parent 2 P_2;
choose at random crossover points at $Subtree_1$ in P_1;
choose at random crossover points at $Subtree_2$ in P_2;
if $Subtree_1$ *is not equivalent with* $Subtree_2$ **then**
  execute crossover;
  add the children to the new population;
  return true;
else
  choose at random crossover points at $Subtree_1$ in P_1;
  choose at random crossover points at $Subtree_2$ in P_2;
  execute crossover;
  return true;

to swap subtrees that have different semantics. In [25], SAC had the best performance (compared with SC and other semantic checking operators) on a family of real-value regression problems. The reasons for these good results were not clearly understood. The following sections will aid in understanding how this mechanism affects GP search.

4 Experimental Setup

We used four real-valued symbolic regression problems, of increasing difficulty, to analyse SAC. The underlying functions, from [8], are shown in Table 1, and evolutionary parameters in Table 2.

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		
Sensitivities	$0.01, 0.02, 0.04, 0.05, 0.06, 0.08, 0.1$		
Trials per treatment	100 independent runs for each value.		

5 Results and Discussion

5.1 Equivalent Crossovers

Because our crossover operator works by analysing semantics of subtrees, we would like to know how frequently SAC and SC swap equivalent subtrees. We collected statistics of the percentage of such crossover events. Figure 1 shows the percentage of semantically equivalent crossover events with $sensitivity = 0.01$, averaged over 100 runs, for each of the four functions, while Table 3 shows the same further averaged over all generations. We can see that the overall average for SC is around 14.5% whereas for SAC it is around 8 times smaller at about 1.8%. It is clear that SAC is more semantically exploratory than SC on these problems. We also conducted an experiment to test how crossover affects the relative fitness of the offspring compared to their parent when it swaps two semantically equivalent subtrees. The results indicate that in nearly all cases (about 98%), such crossover will result in an unchanged fitness.

Table 3. Average Percentage of Equivalent Crossovers.

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	14.9%	14.9%	14.9%	14.9%	15.1%	15.1%	15.1%
	SAC	**1.9%**	**1.9%**	**1.9%**	**1.9%**	**1.9%**	**2.0%**	**2.0%**
F_2	SC	14.1%	14.1%	14.1%	14.1%	14.1%	14.2%	14.2%
	SAC	**1.7%**	**1.7%**	**1.7%**	**1.8%**	**1.9%**	**1.9%**	**1.8%**
F_3	SC	14.4%	14.4%	14.4%	14.4%	14.5%	14.5%	14.5%
	SAC	**1.8%**	**1.8%**	**1.8%**	**1.8%**	**1.9%**	**1.9%**	**1.9%**
F_4	SC	14.1%	14.1%	14.1%	14.1%	14.3%	14.3%	14.3%
	SAC	**1.7%**	**1.7%**	**1.8%**	**1.8%**	**1.8%**	**1.8%**	**1.8%**

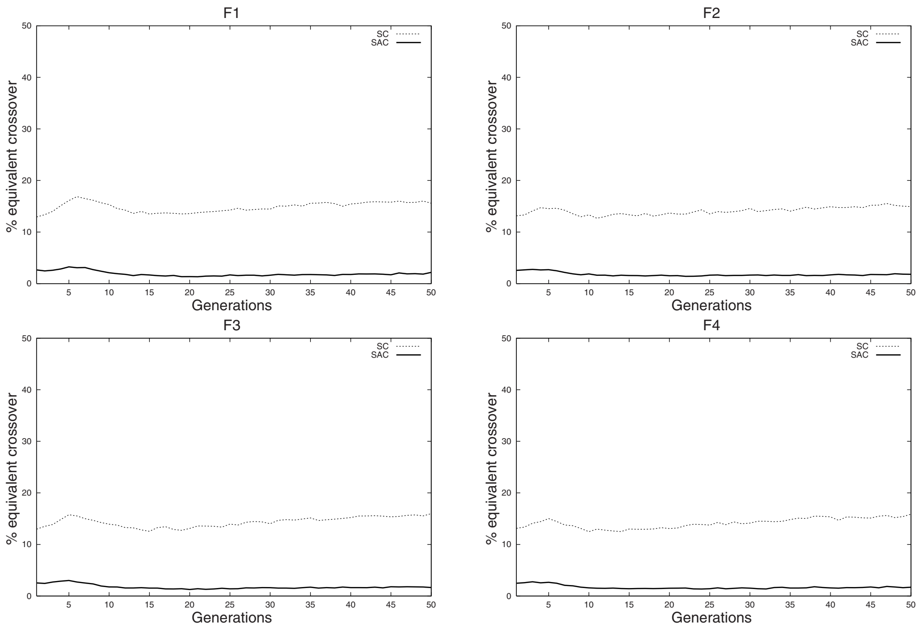

Fig. 1. Average Percentage of Equivalent Crossovers with $sensitivity = 0.01$

5.2 Semantic Diversity

In the previous section, we saw that SAC promotes swapping of semantically different subtrees, encouraging a change in semantics relative to their parents. How well does SAC promote diversity. Population diversity has been long seen as a crucial factor in GP [2]. The search process will be more effective if more diversity in the population is maintained. Two kinds of metrics have been used to measure and control the diversity of a population: genotypic and phenotypic [7]. The former is based on the syntax (i.e., structure) of an individual [22] and the latter on the behavior (i.e., fitness) of an individual [20]. In this paper, we propose a new measure to measure semantic diversity called *Semantic Crossover Diversity* (SCD). SCD works by measuring the difference between

Table 4. Percentage of Children with Different Fitness from their Parents

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	65.8%	65.8%	65.8%	65.8%	65.8%	65.8%	65.8%
	SAC	**73.4%**	**73.0%**	**72.5%**	**72.1%**	**72.4%**	**72.3%**	**73.1%**
F_2	SC	70.5%	70.5%	70.5%	70.5%	70.5%	70.5%	70.5%
	SAC	**79.0%**	**79.0%**	**78.5%**	**78.3%**	**78.8%**	**78.4%**	**79.1%**
F_3	SC	70.6%	70.6%	70.6%	70.6%	70.6%	70.6%	70.6%
	SAC	**80.6%**	**81.2%**	**81.6%**	**82.0%**	**82.0%**	**81.7%**	**82.2%**
F_4	SC	71.0%	71.0%	71.0%	71.0%	71.0%	71.0%	71.0%
	SAC	**80.2%**	**80.0%**	**80.0%**	**79.8%**	**80.0%**	**80.1%**	**79.7%**

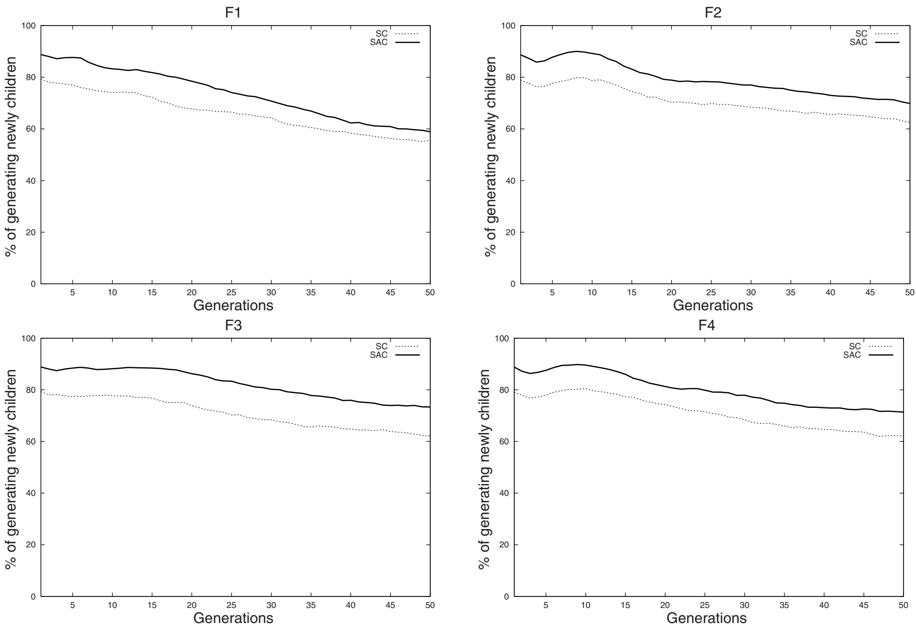

Fig. 2. The percentage of generating newly children with $sensitivity = 0.05$

individuals before and after applying crossover. We used SCD to measure the semantic diversity of SC and SAC by counting the percentage of these crossover events that generated semantically different offspring from their parents. These results are shown in Table 4 and figure 2. We can see that at the beginning of the run, SC generates around 80% of different offspring, while SAC generates about 90%. This is important: the early phase of evolution, is the primary exploration stage, when it is necessary to create semantically new individuals. After a few generations, the percentage decreases in both cases. However SAC still consistently produces about 10% more difference than SC. We note that SAC doest not guarantee to generating semantically new offspring. We believe that there are some fixed semantic subtrees as in the Boolean domain [19].

Table 5. 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	11.8%	11.8%	11.8%	11.8%	11.8%	11.8%	11.8%
	SAC	**15.2%**	**15.0%**	**14.7%**	**15.4%**	**15.3%**	**15.5%**	**15.7%**
F_2	SC	13.0%	13.0%	13.0%	13.0%	13.0%	13.0%	13.0%
	SAC	**17.2%**	**16.7%**	**16.9%**	**17.3%**	**17.4%**	**17.3%**	**17.4%**
F_3	SC	12.9%	12.9%	12.9%	12.9%	12.9%	12.9%	12.9%
	SAC	**16.9%**	**16.8%**	**17.0%**	**17.1%**	**17.0%**	**17.2%**	**17.0%**
F_4	SC	12.8%	11.4%	11.4%	11.4%	11.4%	11.4%	11.4%
	SAC	**16.8%**	**16.8%**	**16.7%**	**17.0%**	**17.1%**	**17.0%**	**17.0%**

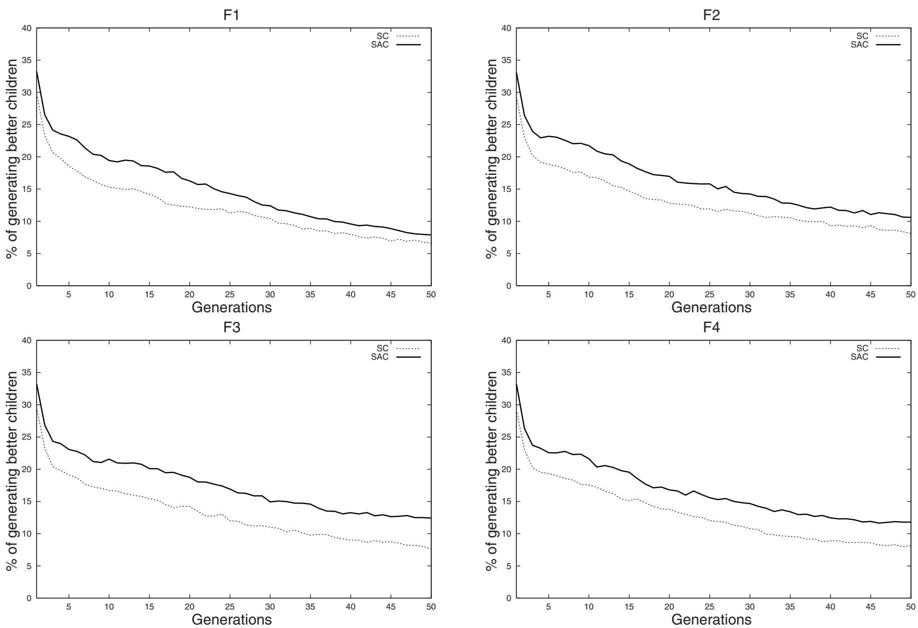

Fig. 3. The percentage of constructive crossover with *sensitivity*= 0.04

5.3 Constructive Effect

If SAC is more semantically productive than SC, then it is interesting to ask whether this helps the crossover operators to breed better children than their parents (more constructive crossover). To answer this, we measured the constructive effect of SAC and SC, using a method similar to that in [18]. The constructive effect is measured by calculating the percentage of events that generate a better child from its parents through crossover. The results are shown in Table 5, figure 3 showing the change over the course of evolution with *sensitivity* 0.04. We can see that SAC is more fitness constructive,

Table 6. The average size of individuals

Sensitivity		0.01	0.02	0.04	0.05	0.06	0.08	0.1
F_1	SC	44.2	44.2	44.2	44.2	44.2	44.2	44.2
	SAC	46.8	46.2	46.7	46.5	46.7	46.9	46.9
F_2	SC	46.1	46.1	46.1	46.1	46.1	46.1	46.1
	SAC	50.0	49.3	49.1	49.8	49.9	49.9	46.7
F_3	SC	44.8	44.8	44.8	44.8	44.8	44.8	44.8
	SAC	49.9	48.7	48.3	48.2	48.3	48.4	48.7
F_4	SC	48.0	48.0	48.0	48.0	48.0	48.0	48.0
	SAC	52.2	51.9	52.4	51.6	51.5	52.0	51.8

Table 7. 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

often 5% better, than SC. This strongly suggests why the performance of SAC was better than SC in terms of number of successful runs.

5.4 Code Bloat

The better performance of SAC comes at a cost: it takes time to calculate the subtree semantics. This is reflected in the slightly higher running time of SAC, as compared with SC. How much more expensive are these extra calculations? And is this the sole cause of the extra computational cost, or might code bloat play a role? To determine which is the main source we collected two statistics from the runs. The first is the average size of individuals (number of nodes) over 50 generations, averaged over 100 runs, for SAC versus SC. The second is the average size of subtrees which are semantic equivalence tested in SAC, averaged in the same way. The two statistics are shown in Table 6, and Table 7 respectively.

It is clear that the higher running time of SAC resulted from both causes: the individuals were larger. However it also seems that the overheads are acceptable. From Table 7, we see that the average size of subtrees in SAC is very small in comparison with the average size of individuals. Therefore, the time needed to calculate and compare subtree fitnes is small (in fact, we could reduce this further by using caching to improve the efficiency of subtree semantic calculation as in [15]). The average individual size in SAC was bigger than that of SC but not by a large margin. Nevertheless, we are interested to incorporate the size of subtrees into the selection of crossover points in future work, potentially avoiding this issue.

6 Conclusions

In this paper we have compared Semantic Aware Crossover (SAC) and Standard Crossover (SC) from a number of perspectives. We have shown that about 15% of SC operations exchange semantically equivalent subtrees. As a consequence SC tends to create offspring that are semantically similar to their parents. This weakness can be amended by preventing the swapping of equivalent subtrees. Our approach, Semantic Aware Crossover (SAC), adopts this idea. Secondly, we have shown that SAC helps to promote semantic diversity, so it generates more offspring that are semantically different from their parents. The results also show that SAC is more constructive than SC. Furthermore, we have seen that the additional computation cost of SAC is almost negligible.

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

References

1. Beadle, L., Johnson, C.: Semantically driven crossover in genetic programming. In: Proc. IEEE WCCI 2008, pp. 111–116. IEEE Press, Los Alamitos (2008)
2. Burke, E.K., Gustafson, S., Kendall, G.: Diversity in genetic programming: An analysis of measures and correlation with fitness. IEEE Transactions on Evolutionary Computation 8(1), 47–62 (2004)
3. Cleary, R., O'Neill, M.: An attribute grammar decoder for the 01 multi-constrained knapsack problem. In: Raidl, G.R., Gottlieb, J. (eds.) EvoCOP 2005. LNCS, vol. 3448, pp. 34–45. Springer, Heidelberg (2005)
4. de la Cruz Echeand'a, M., de la Puente, A.O., Alfonseca, M.: Attribute grammar evolution. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2005. LNCS, vol. 3562, pp. 182–191. Springer, Heidelberg (2005)
5. Ghodrat, M.A., Givargis, T., Nicolau, A.: Equivalence checking of arithmetic expressions using fast evaluation. In: Proc. of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems – CASES 2005. ACM, New York (2005)
6. Hengpraprohm, S., Chongstitvatana, P.: Selective crossover in genetic programming. In: Proc. of ISCIT International Symposium on Communications and Information Technologies, November 2001, pp. 14–16 (2001)
7. Hien, N.T., Hoai, N.X.: A brief overview of population diversity measures in genetic programming. In: Proc. of 11th Asia-Pacific Workshop on Intelligent and Evolutionary Systems, October 2006, pp. 128–139. Vietnamese Military Technical Academy (2006)
8. Hoai, N.X., McKay, R., Essam, D.: Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. In: Proc. of CEC 2002, pp. 1326–1331. IEEE Press, Los Alamitos (2002)
9. Ito, T., Iba, H., Sato, S.: Depth-dependent crossover for genetic programming. In: Proc. of IEEE WCCI 1998, pp. 775–780. IEEE Press, Los Alamitos (1998)
10. Johnson, C.: Deriving genetic programming fitness properties by static analysis. In: Foster, J.A., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A.G.B. (eds.) EuroGP 2002. LNCS, vol. 2278, pp. 298–308. Springer, Heidelberg (2002)
11. Johnson, C.: What can automatic programming learn from theoretical computer science. In: Proc. of the UK Workshop on Computational Intelligence. University of Birmingham (2002)

12. Johnson, C.: Genetic programming with fitness based on model checking. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 114–124. Springer, Heidelberg (2007)
13. Katz, G., Peled, D.: Genetic programming and model checking: Synthesizing new mutual exclusion algorithms. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 33–47. Springer, Heidelberg (2008)
14. Katz, G., Peled, D.: Model checking-based genetic programming with an application to mutual exclusion. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 141–156. Springer, Heidelberg (2008)
15. Keijzer, M.: Alternatives in subtree caching for genetic programming. In: Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., Soule, T. (eds.) EuroGP 2004. LNCS, vol. 3003, pp. 328–337. Springer, Heidelberg (2004)
16. Koza, J.: Genetic Programming: On the Programming of Computers by Natural Selection. MIT Press, MA (1992)
17. Majeed, H., Ryan, C.: A less destructive, context-aware crossover operator for GP. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 36–48. Springer, Heidelberg (2006)
18. Majeed, H., Ryan, C.: On the constructiveness of context-aware crossover. In: Proc. of GECCO 2007, pp. 1659–1666. ACM Press, New York (2007)
19. McPhee, N., Ohs, B., Hutchison, T.: Semantic building blocks in genetic programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 134–145. Springer, Heidelberg (2008)
20. Mori, N.: A novel diversity measure of genetic programming. In: Proc. of Randomness and Computation: Joint Workshop "New Horizons in Computing" and "Statistical Mechanical Approach to Probabilistic Information, July 2005, pp. 18–21 (2005)
21. Mori, N., McKay, R., Hoai, N.X., Essam, D.: Equivalent decision simplification: A new method for simplifying algebraic expressions in genetic programming. In: Proc. of 11th Asia-Pacific Workshop on Intelligent and Evolutionary Systems (2007)
22. O'Reilly, U.M.: Using a distance metric on genetic programs to understand genetic operators. In: Proc. of IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, pp. 4092–4097. IEEE, Los Alamitos (1997)
23. Poli, R., Langdon, W., McPhee, N.: A Field Guide to Genetic Programming (2008), `http://lulu.com`
24. Poli, R., Langdon, W.B.: Genetic programming with one-point crossover. In: Proc. of Soft Computing in Engineering Design and Manufacturing Conference, pp. 180–189. Springer, Heidelberg (1997)
25. Uy, N.Q., Hoai, N.X., O'Neill, M.: Semantic aware crossover for genetic programming: the case for real-valued function regression. In: Vanneschi, L., et al. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 25–36. Springer, Heidelberg (2009)
26. Wong, M.L., Leung, K.S.: An induction system that learns programs in different programming languages using genetic programming and logic grammars. In: Proc. of the 7th IEEE International Conference on Tools with Artificial Intelligence (1995)