Genetic Programming and Evolvable Machines manuscript No.
(will be inserted by the editor)

Semantically-based Crossover in Genetic Programming: Application to Real-valued Symbolic Regression

Nguyen Quang Uy · Nguyen Xuan Hoai · Michael O'Neill · R.I. McKay · Edgar Galván-López.

Received: date / Accepted: date

Abstract We investigate the effects of semantically-based crossover operators in Genetic Programming, applied to real-valued symbolic regression problems. We propose two new relations derived from the semantic distance between subtrees, known as Semantic Equivalence and Semantic Similarity. These relations are used to guide variants of the crossover operator, resulting in two new crossover operators – Semantics Aware Crossover (SAC) and Semantic Similarity-based Crossover (SSC). SAC, was introduced and previously studied, is added here for the purpose of comparison and analysis. SSC extends SAC by more closely controlling the semantic distance between subtrees to which crossover may be applied. The new operators were tested on some real-valued symbolic regression problems and compared with Standard Crossover (SC), Context Aware Crossover (CAC), Soft Brood Selection (SBS), and No Same Mate (NSM) selection. The experimental results show on the problems examined that, with computational effort measured by the number of function node evaluations, only SSC and SBS were significantly better than SC, and SSC was often better than SBS. Further experiments were also conducted to analyse the perfomance sensitivity to the parameter settings for SSC. This analysis leads to a conclusion that SSC is more constructive and has higher locality than SAC, NSM and SC; we believe these are the main reasons for the improved performance of SSC.

Keywords Genetic Programming · Semantics · Crossover · Symbolic Regression · Locality.

Nguyen Quang Uy and Michael O'Neill and Edgar Galván-López
Natural Computing Research & Applications Group, University College Dublin, Ireland
Tel.: ++353-86-236-6125
Fax: ++353-1-7165396
E-mail: quanguyhn@gmail.com, m.oneill@ucd.ie, edgar.galvan@ucd.ie

Nguyen Xuan Hoai and R.I. (Bob) McKay
School of Computer Science and Engineering, Seoul National University, Korea
E-mail: nxhoai@gmail.com, rimsnucse@gmail.com

1 Introduction

Genetic Programming (GP) [36, 40, 49]) researchers have only recently paid much attention to semantic information, which has resulted in a dramatic increase in the number of realated publications (e.g. [4, 9, 26, 28, 29, 31, 32, 43, 57]). Previously, work in GP representation had focused mainly on syntactic aspects. From a programmer's perspective, however, maintaining syntactic correctness is only part of program construction: programs must be not only syntactically correct, but also semantically correct. Thus incorporating semantic awareness in the GP process could improve performance, extending the applicability of GP to problems that are difficult with purely syntactic approaches. So far, semantics has been incorporated into different phases of GP including fitness measurement [26, 29], operators execution [4, 57], valid checking [33, 58] and so forth. In this work, we investigate one method to incorporate semantic information into GP crossover operators for real-valued symbolic regression problems.

Previous Evolutionary Computation research has shown that characteristics of evolutionary operators such as their constructiveness, locality (small changes in genotype resulting in small changes in phenotype), and effect on population diversity strongly affect the performance of the resulting algorithms [7, 42, 51, 53]. However designing GP operators with these desirable characteristics can be very difficult. We aim to incorporate semantics into the design of new crossover operators so as to maintain greater semantic diversity, and provide greater constructiveness as well as higher locality than standard crossover (SC). We investigate the effects of these semantically-based operators on the performance of GP.

This paper addresses two main objectives. The first and narrower is to propose a new semantically-based schema for implementing crossover in GP that extends Semantics Aware Crossover (SAC) in our previous work [57]. The second and broader objective is to encourage GP researchers to pay greater attention to the use of semantics to improve the efficiency of GP search. It extends [57] in a number of ways. First, we change the way the semantics is used to constrain the crossover, resulting in a new crossover that we call Semantic Similarity-based Crossover (SSC). SSC extends SAC by not only encouraging exchange of semantically different material between parents, but also limiting this to small and controllable changes. SSC and SAC are compared with a broader class of related crossover operators in the literature and the results are positive.

Experiments to investigate the impact of crucial parameters on SSC's performance are also presented, providing the basis on which to choose appropriate values for these parameters. Subsequently, we conduct a more comprehensive analysis to investigate the possible reasons behind the effectiveness of SSC, and in particular, why SSC works so much better than SC, SAC and NSM. Finally, we extend the previous work by comparison on a much broader range of target functions. All of these extensions will be presented in detail in the following sections.

The remainder of the paper is organised as follows. In the next section we review the literature on GP with semantic information and on GP crossover operators. Section 3 contains the detailed descriptions of our new crossovers. The experimental settings are described in Section 4. A comparison of the effectiveness of SSC and other related crossover operators are presented in Section 5. An analysis of parameter sensitivity for SSC and an investigation of some of the characteristics of SSC follow in the next two

sections. The conclusions are presented in Section 8, leading to suggestions for future research in Section 9.

2 Background

In this section, we briefly review previous work on semantics in GP and on variants of GP crossover operators.

2.1 Semantics in Genetic Programming

Semantics is a broad concept that has been studied in a number of fields including Natural Language [1], Psychology [10] and Computer Science [46] among others. While the precise meaning varies from field to field, semantics is generally contrasted with syntax: the syntax refers to the surface form of an expression, while the semantics refers to its deeper meaning in some external World. In computer science, this external World is generally provided by the computational model.

In Computer Science, semantics can be informally defined as the meaning of syntactically correct programs or functions. Two programs that are syntactically the same must have the same semantics, but the converse may not be true.

As a simple example, consider two small programs shown in Equations 1 and 2. Syntactically, the first statement of each is identical, but the second statements differ. Semantically, however, they are identical: both programs compute the same result.

$$x = 1;\ y = x + x; \tag{1}$$

$$x = 1;\ \ y = 2 * x; \tag{2}$$

In GP, semantics has generally been used to provide additional guidance to the GP search. The necessary additional information is either added to or extracted from the GP individual's representation. Thus the available possibilities depend on the problem domain (Boolean or real-value,...), the GP individual representation (Grammar-, Tree- or Graph-based), and the search algorithm characteristics (fitness measure, genetic operators,. . .).

To date, there have been three main approaches to representing and extracting semantics and using it to guide the evolutionary process:

1. grammar-based [8, 9, 12, 50, 58, 59]
2. formal methods [26, 28, 29, 31–33]
3. GP s-tree representation [4–6, 37, 38, 43, 57]

The most popular form of the first uses attribute grammars. Attribute grammars are extensions of context-free grammars, in which a finite set of attributes provide context sensitivity [34]. GP individuals expressed in the form of attribute grammar trees can incorporate semantic information, which can be used to eliminate bad individuals (i.e., less fit individuals) from the population [12] or to prevent generating semantically invalid individuals as in [8, 9, 50, 58, 59]. The attributes used to present semantics are generally problem-dependent, and it is not always obvious how to determine the attributes for a given problem.

Recently, Johnson has advocated formal methods as a means to incorporate semantic information into the GP evolutionary process [26, 28, 29]. Formal methods are a class of mathematically-based techniques for the specification, development and verification of software and hardware systems [45]. They support the extraction and approximation of mathematical statements useful for system design and verification. In Johnson's work, semantic information extracted through formal methods, such as abstract interpretation or model checking, is used to quantify the fitness of individuals on some problems for which traditional sample-point-based fitness measure are unavailable or misleading. In [26, 28], Johnson used interval analysis (a form of abstract interpretation) to measure the fitness of individuals in solving a rectangle replacement problem and in robot control. By contrast, Keijzer [33] used interval analysis to check whether an individual is defined over the whole range of input values – if an individual is undefined anywhere, that individual can be assigned minimal fitness or simply deleted from the population. This allowed Keijzer to avoid discontinuities arising from protected operators, improving the evolvability of the system. Johnson [29] used model checking to measure individual fitness in evolving vending machine controllers. A controller is specified by a number of computation tree logic formulas [3]. Fitness of an individual is the number of formulas it satisfies. Subsequently, Katz and Peled [31, 32] also used model checking to define fitness in a GP system for the mutual exclusion problem. The advantage of formal methods lies in their strict mathematical background, potentially helping GP to evolve computer programs. However they are also high in complexity and difficult to implement, possibly explaining the limited research despite the advocacy of Johnson [27]. Their main application to date has lain in evolving control strategies.

Methods for extracting semantics from expression trees depend strongly on the problem domain. The finite inputs of Boolean domains mean that semantics can be accurately estimated in a variety of ways. Beadle and Johnson [4] investigated the effects of directly using semantic information to guide GP crossover on Boolean problem domains. They checked semantic equivalence between offspring and parents by transforming them to Reduced Ordered Binary Decision Diagrams (ROBDDs) [14]. Two trees are semantically equivalent if and only if they reduce to the same ROBDD. This is used to determine which participating individuals are copied to the next generation. If the offspring are semantically equivalent to their parents, the children are discarded and the crossover is restarted. This process is repeated until semantically new children are found. The authors argue that this results in increased semantic diversity in the evolving population, and a consequent improvement in the GP performance. This method of semantic equivalence checking is also applied to drive mutation [6] and guide the initialisation phase of GP [5], where the authors show that it benefits for GP in both phases. By contrast, McPhee et al. [43] extract semantic information from a Boolean expression tree by enumerating all possible inputs. They consider the semantics of two components in each tree: semantics of subtrees and semantics of context (the remainder of an individual after removing a subtree). They experimentally measured the variation of these semantic components throughout the GP evolutionary process. They paid special attention to fixed-semantic subtrees: subtrees where the semantics of the tree does not change when this subtree is replaced by another subtree. They showed that there may be very many such fixed semantic subtrees when the tree size increases during evolution; thus it becomes very difficult to change the semantics of trees with crossover and mutation once the trees have become large.

To the best of our knowledge, there has been no previous research on semantic guidance in real-valued problems before our own previous study [57]. There, we pro-

posed a new crossover operator, semantics aware crossover (SAC), based on checking the semantic equivalence of subtrees. SAC was tested on a family of real-valued symbolic regression problems, and was empirically shown to improve GP performance. SAC differs from Beadle and Johnson's approach [4] in two ways. First, the test domain is real-valued rather than Boolean. For real domains, it is not generally feasible to check semantic equivalence by reduction to a canonical form like a ROBDD. Second, 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 [43] for calculating subtree semantics. However, for real domains, measuring semantics by enumerating all possible inputs as in [43] is also infeasible, so that the semantics must be approximated.

Recently, Krawiec and Lichocki proposed a way to measure the semantics of an individual based on fitness cases [37]. In this work, the semantics of an individual is defined as a vector in which each element is the output of the individual at the corresponding input fitness case. This semantics is used to guide crossover in a method similar to SBS, known as *Approximating Geometric Crossover* (AGC). In AGC, a number of children are generated at each crossover, the children most similar to their parents – in terms of semantics – being added to the next generation. The experiment is conducted on both real-valued and boolean regression problems. The results show that AGC is no better than SC in real-value problems, and only slightly superior to SC in boolean problems. The same kind of semantics is then used to build functional modulation for GP [38], for which the experimental results show that it may be useful in characterising the compositionality and difficulty of a problem, potentially leading to performance improvements for GP.

2.2 Alternative Crossovers in Genetic Programming

It is well-known that crossover is the primary operator in GP [35]. In the standard crossover (SC) two parents are selected, and then one subtree is randomly selected in each parent. A procedure is called to check if these two subtrees are legal for crossover (syntactic closure properties, depth of resulting children,...). If so, the crossover is executed by simply swapping the two chosen subtrees, and the resulting offspring are added to the next generation. Figure 1 shows how SC works.

Much research has concentrated on the efficiency of crossover, resulting in new and improved operators which can be classified into three categories. These are:

1. crossovers based on syntax (structure)
2. crossovers based on context
3. crossovers based on semantics

Most of the early modifications to SC were based on syntax [24, 35, 39, 47, 48]. Koza [35] proposed a crossover that is 90% biased to function nodes and 10% bias to terminal nodes as crossover points. Although this method encourages the exchange of more genetic material (bigger subtrees) between the two participating individuals, it risks exacerbating bloat and thus making it more difficult to refine solutions in later generations [4]. O'Reilly and Oppacher [47] introduced height-fair crossover, in which all subtree heights in the two parents are recorded, and one subtree height is randomly selected. The crossover sites in both parents are then restricted to that particular height. Ito et al. [24] presented a similar depth-dependent crossover, aiming to preserve building blocks. In this method, the probability of selecting a node is biased towards the

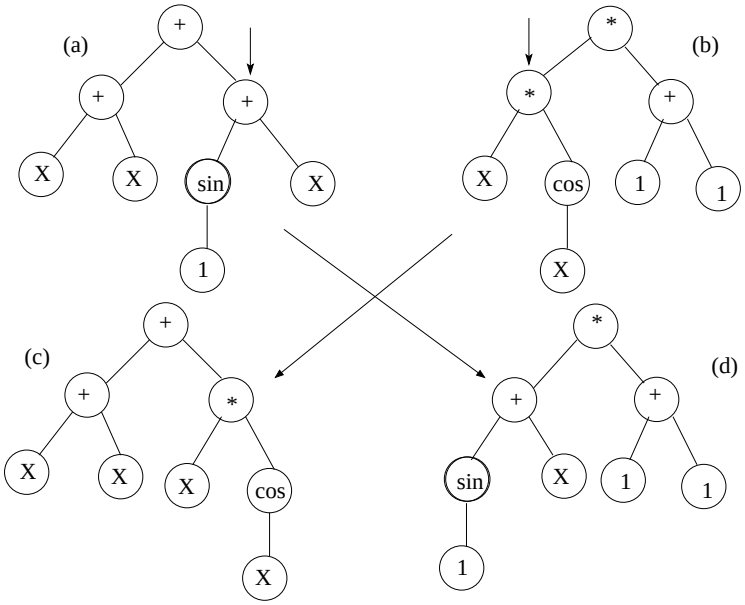

Fig. 1 Standard Crossover: parents (top) and their resulting offspring after applying crossover (bottom).

root – nodes near the root have a greater probability to be selected for crossover. The bias of the selection probability is set by the user, and it is left unchanged during the search process. However it is not robust: if it is not carefully set for a particular problem, the performance can be very poor [25]. In [39, 48], Poli and Langdon introduced one-point crossover and uniform crossover. In these methods, when two parents are selected for crossover, they are aligned based on their shapes. By aligning two parents, the common shape of these parents (starting from the roots) can be determined. The crossover points are then randomly selected from the nodes that lie in the common shape. This kind of crossover has been shown especially effective on Boolean problems as it causes a bigger genetic material exchange in earlier generations (in these generations the common shape is often very small) and yet can tune the solutions in later generations (when the common shapes are bigger).

More recently, context has been used as extra information for the selection of crossover points [2, 19, 41, 55, 56]. This class of crossover is perhaps closest to semantic based crossovers. Altenberg [2] proposed a new crossover inspired by the observation that in most animal species, breeding occurs more often than the number of surviving offspring might suggest. In other words, viable offspring are not always produced as a consequence of breeding. This crossover is called a *Soft Brood Selection* (SBS). Two parents are selected for crossover, then N random crossovers are performed to generate a *Brood* of $2N$ children. The children are evaluated and sorted based on their fitness. The two best children are copied to the next generation, the rest being discarded. This crossover was then developed by Tackett [55, 56] by using a subset of fitness cases to figure out which children in the *Brood* are added to the next generation. Hengpraprohm and Chongstitvatana [19] proposed *Selective Crossover*, in which each subtree is assigned an impact value, reflecting how well (or badly) the subtree affects

the containing tree. The impact of a subtree is determined by removing that subtree and replacing it with a random terminal node. The change in resulting fitness is the impact value. The crossover is performed by replacing the worst subtree of each parent with the best subtree of the other. Majeed and Ryan [41] proposed *Context Aware Crossover* (CAC); after two parents have been selected for crossover, one subtree is randomly chosen in the first. This subtree is then crossed over into all possible locations in the second, all generated children being evaluated. The best child (based on fitness) is selected as the result, and copied to the next generation. The advantage of these context-based crossovers is increased probability of producing better children. On the other hand, it can be very time consuming to evaluate the context of each subtree.

To the best of our knowledge, the only previous use of semantics in crossover are those previously discussed in Subsection 2.1. They include Beadle and Johnson's [4] Semantics Driven Crossover for Boolean problems, Krawiec and Lichocki's [37] Approximating Geometric Crossover, and our previous work [57] on Semantics Aware Crossover (SAC).

3 Methods

In this section we give a detailed discussion of our two crossovers. We start by briefly describing how we measure semantics in real-valued problems. This allows us to define a concept of semantic distance, on the basis of which we propose two semantic relationships,which we then use to define two new crossover operators.

3.1 Measuring Semantics

As discussed in Section 2, the appropriate definition of semantics for GP is far from obvious. The semantics of an individual is often understood as the behavior of that individual with respect to a set of input values. However the possibilities for computing such semantics depends on the domain. For real-valued problems, both canonical-form methods corresponding to Beadle and Johnson's [4] Boolean ROBDDs, and complete enumeration as in McPhee's approach [43], are infeasible. Instead, we propose a simple way to estimate the semantics of subtrees, in which the semantics is approximated by evaluating the subtree on a pre-specified set of points in the problem domain. We call this *Sampling Semantics*. Formally, the *Sampling Semantics* of any tree (subtree) is defined as follows:

Let F be a function expressed by a tree (subtree) T on a domain D. Let P be a set of points from domain D, $P = \{p_1, p_2, ...p_N\}$. Then the *Sampling Semantics* of T on P in domain D is the set $S = \{s_1, s_2, ..., s_N\}$ where $s_i = F(p_i), i = 1, 2, .., N$.

For example, suppose that we are considering the interval [0,1] and using a set of three points, $P = \{0, 0.5, 1\}$, for evaluating semantics. Then the *Sampling Semantics* of subtree St in Figure 2 on P is the set of three values $S = \{sin(1) - 0, sin(1) - 0.5, sin(1) - 1\}$=$\{084, 0.34, -0.16\}$. The value of N depends on the problems. If it is too small, the approximate semantics might be too coarse-grained and not sufficiently accurate. If N is too big, the approximate semantics might be more accurate, but more time consuming to measure. The choice of P is also important. If the members of P are too closely related to the GP function set (for example, π for exponential/trigonometric functions, or e for logarithmic functions), then the semantics might be misleading. For

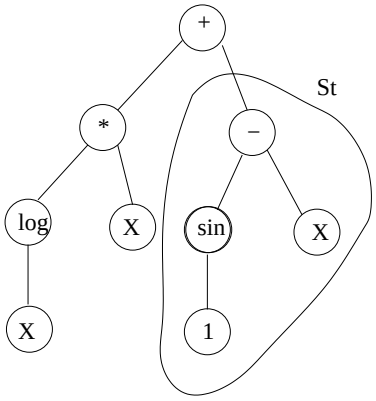

Fig. 2 Tree with subtree (for illustrating *Sampling Semantics*)

this reason, choosing them randomly may be the best solution. In this paper, the number of points for evaluating *Sampling Semantics* is set as the number of fitness cases for the problem (20 points for single variable functions and 100 points for bivariate functions, see Section 4), and we choose the set of points P uniformly randomly from the problem domain. [1]

Based on the *Sampling Semantics* (SS), we define a *Sampling Semantics Distance* between two subtrees. In our previous work [57], we defined the *Sampling Semantics Distance* as the sum of the absolute differences for all values of SS. That is, let $P = \{p_1, p_2, ..., p_N\}$ and $Q = \{q_1, q_2, ..., q_N\}$ be the SS of $Subtree_1(St_1)$ and $Subtree_2(St_2)$ on the same set of sample points, then the *Sampling Semantics Distance* (SSD) between St_1 and St_2 was defined as:

$$SSD(St_1, St_2) = |p_1 - q_1| + |p_2 - q_2| + ... + |p_N - q_N|$$

While the experiments in [57] showed that this SSD is beneficial, it has the undoubted weakness that the value of the SSD depends on the number of SS points (N). To reduce this drawback, we now use the mean of the absolute differences as the SSD between subtrees. In other word, the SSD between St_1 and St_2 is defined as:

$$SSD(St_1, St_2) = (|p_1 - q_1| + |p_2 - q_2| + ... + |p_N - q_N|)/N$$

3.2 Semantic Relationships

Based on *Sampling Semantics Distance*, we can define two semantic relationships between subtrees.Two subtrees are *Semantically Equivalent* (SE) on a domain if their SSD on the sample set of points is sufficiently similar (subject to a parameter called *semantic sensitivity*) – formally:

$$SE(St_1, St_2) = \textbf{if } \mathrm{SSD}(St_1, St_2) < \epsilon \textbf{ then true}$$
$$\textbf{else false}$$

[1] Since *Sampling Semantics* is defined for any subtree, it can be used in particular to estimate the semantics of the whole tree. We will use it in this way in the examples in later sections.

ϵ is the predefined *semantic sensitivity*. This subtree semantic relationship is similar to the metric we used in [57], and was inspired by the work of Mori et al. [44] on GP simplification. The experimental results in [44, 57] show that this semantic relationship benefits the GP search process.

The second relationship is known as *Semantic Similarity*.[2] The intuition behind semantic similarity is that exchange of subtrees is most likely to be beneficial if the two subtrees are not semantically identical, but also not too semantically dissimilar. Two subtrees St_1 and St_2 are semantically similar on a domain if their SSD on the sample set lies within a positive interval – formally:

$$SSi(St_1, St_2) = \textbf{if } \alpha < \mathrm{SSD}(St_1, St_2) < \beta \textbf{ then true}$$
$$\textbf{else false}$$

here α and β are two predefined constants, known as the lower and upper bounds for semantic sensitivity, respectively. Conceivably, the best values for *lower* and *upper bound semantic sensitivity* might be problem dependent. However we suspect that for most symbolic regression problems, there is a wide range of appropriate values (see Section 6, where we study various ranges of both *lower* and *upper bound semantic sensitivity*).

We note that there is some biological motivation for this approach. In mammals, the Major Histocompatibility Complex (MHC) genes (on chromosome 6 in humans) play a major role in the immune response, and thus are a key part of our defences against disease, and subject to strong and rapidly-changing evolutionary pressures. However they also play an important role both in mate selection (partners in the same species, but with dissimilar MHC genes, are preferred), and in speciation, because differences in MHC that are too big may cause an immune response from the mother to the foetus. Thus in this case at least, biology also appears to favour crossovers with semantic similarity lying in a specific range.

We conclude this section by highlighting some important differences between our semantic relations and fitness. First, for fitness calculation we need to know the fitness cases, and fitness reflects how good (close to the target function) an individual is. In measuring SS, we do not need to know the fitness cases (of course semantics can be measured using the fitness cases, but different cases can also be used). Second, fitness is measured for the whole individual, while SS is mainly used to encapsulate the semantics of subtrees. The last and most important difference is the objective: fitness is used for individual selection while SS is used to guide crossover. It is also noted that the semantic definition in Krawiec and Lichocki [37] is a particular case of *Sampling Semantics*, in which the set of sample points is the the same as the set of fitness cases, and the semantics of the whole tree (a particular subtree) is used in crossover.

3.3 Semantics Aware Crossover

A semantics aware crossover (SAC) was first proposed in [57]. SAC is motivated by the observation that GP crossover may exchange semantically equivalent subtrees, resulting

[2] We are using similarity here in its ordinary English meaning, where A is similar to B implies that A is not the same as B, as opposed to a common mathematical convention in which similarity includes equivalence.

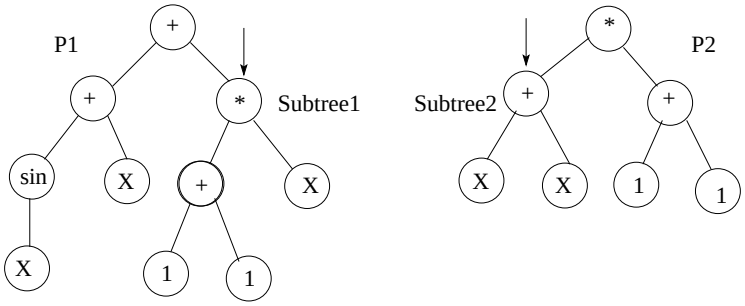

Fig. 3 Semantics equivalent subtrees are selected

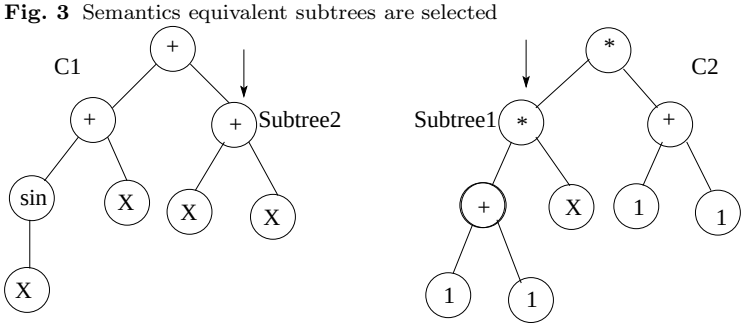

Fig. 4 The generated children from semantic equivalent subtree crossover.

in children that are identical to their parents. Consider the two selected parents P_1 and P_2 shown at the top of Figure 3. P_1 has the semantics $sin(X) + 3X$ and P_2 has the semantics 4X. $Subtree_1$ of P_1 and $Subtree_2$ of P_2 are semantically equivalent subtrees, both having semantics $2X$, although their structures are totally different. When these two subtrees are selected for crossover, the children are as shown in Figure 4. Obviously, these two children have different syntax (structure) from, but identical semantics to, their parents. C_1 has semantics of $sin(X) + 3X$ and C_2 has semantics of $4X$. This leaves the fitness of the children unchanged after crossover.

SAC prevents the swapping of such semantically equivalent subtrees in crossover. Each time two subtrees are chosen for crossover, a semantic check (using *Semantic Equivalence*) is performed to determine if they are equivalent. If they are, the crossover is aborted and instead performed on two other randomly chosen subtrees. Further detail on SAC can be found in [57]. SAC was partly inspired by Gustafson's *No Same Mate* selection [18] in which no two individuals with the same fitness may be selected for crossover. It, in turn, was motivated by experiments, in which he found that two parents with the same fitness often produce children with unchanged fitness upon crossover.

3.4 Semantic Similarity-based Crossover

The new semantically based crossover, SSC, is an extension of SAC in two ways. First, when two subtrees are selected for crossover, their semantic similarity, rather than semantic equivalence, is checked. Second, semantic similarity is more difficult to satisfy than semantic equivalence, so repeated failures may occur. As a result, SSC uses mul-

Algorithm 1: Semantic Similarity based Crossover

```
select Parent 1 P1;
select Parent 2 P2;
Count=0;
while Count<Max_Trial do
    choose a random crossover point Subtree1 in P1;
    choose a random crossover point Subtree2 in P2;
    randomly generate a number of points (P) on the problem domain;
    measure SSD between Subtree2 and Subtree1 on P;
    if Subtree1 is similar to Subtree2 then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
if Count=Max_Attempt then
    choose a random crossover point Subtree1 in P1;
    choose a random crossover point Subtree2 in P2;
    execute crossover;
    return true;
```

tiple trials to find a semantically similar pair, only reverting to random selection after passing a bound on the number of trials. Algorithm 1 shows how SSC works in detail.

In our experiments, we test a range of values of Max_Trial to gain an understanding of its effect on SSC. The motivation for SSC is to encourage exchange of semantically different, but not wildly different, subtrees. While forcing a change in the semantics of the individuals in the population, we want to keep this change bounded and small. We anticipate that a smoother change in semantics of the individuals will result, and might lead to a smoother change in fitness of the individuals after crossover. For instance, consider two parents selected for crossover in Figure 5. Assume that we measure the SS of a tree on the 10 points, $P = \{1, 2, ..., 10\}$. Then the SS of parents P_1, P_2 and of Subtree1 (St_1), Subtree2 (St_2), and Subtree3 (St_3) are as shown in Table 1 and Table 2. It can be seen from these tables that St_1 and St_2 are semantically similar (using α=10^{-4}, β=0.4 as in this paper), with the SSD being only 0.09, while St_1 and St_3 are semantically dissimilar since the SSD is 4.5. If crossover is performed by swapping two semantically similar subtrees (St_1 and St_2), the generated children are show in Figure 6. The SS of the two children (C_1, C_2) are shown in Table 1. We can also measure the SSD between C_1 and P_1 and between C_2 and P_2 (as shown in columns $C_1 - P_1$ and $C_2 - P_2$ in Table 1). Evidently, the change of semantics through crossover is quite small (1.1 with C1 and 1.65 with C2). This, we hope, will make for a smoother change of fitness.

By contrast, if crossover is conducted by swapping two dissimilar subtrees (St_1 and St_3), the children are shown in Figure 7. The results of the calculation of the SS of the two children (C_3 and C_4) and the semantic distances between these children and their parents are shown in Table 2. It can be seen from this table that the change in semantics between parents and children is rather large (143 and 82.5 for $C3$ and $C4$, respectively). This, we anticipate, will lead to an abrupt change in fitness after crossover.

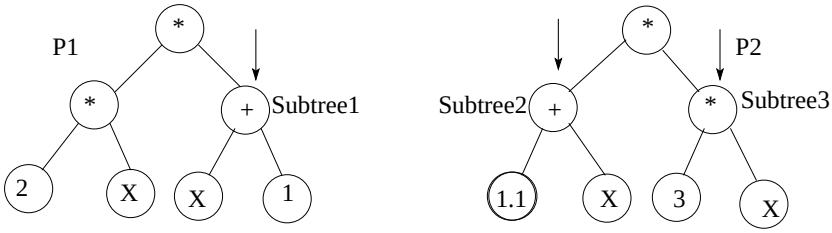

Fig. 5 Parents for crossover

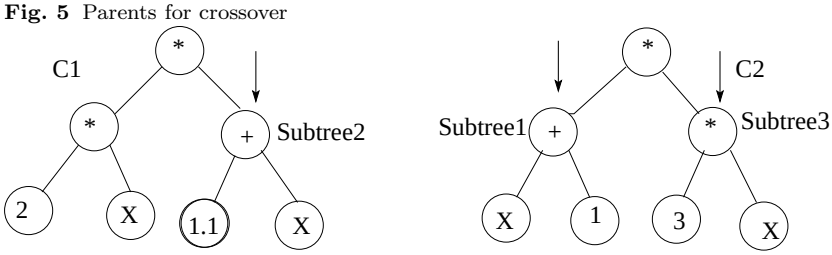

Fig. 6 Children generated by crossing over two semantically similar subtrees

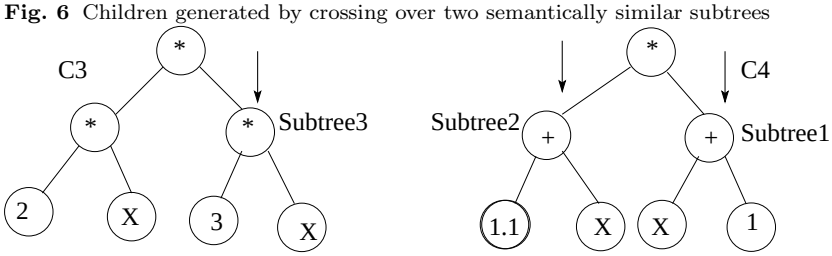

Fig. 7 Children generated by crossing over two semantically dissimilar subtrees

Table 1 Sampling semantics of parents, subtrees and children when swapping two similar subtrees.

Points	P_1	P_2	St_1	St_2	St_1-St_2	C_1	C_2	C_1-P_1	C_2-P_2
1	4	6.3	2	2.1	0.1	4.2	6	0.2	0.3
2	12	18.6	3	3.1	0.1	12.4	18	0.4	0.6
3	24	36.9	4	4.1	0.1	24.6	36	0.6	0.9
4	40	61.2	5	5.1	0.1	40.8	61	0.8	1.2
5	60	91.5	6	6.1	0.1	61.0	91	1.0	1.5
6	84	127.8	7	7.1	0.1	85.2	127	1.2	1.8
7	112	170.1	8	8.1	0.1	113.4	170	1.4	2.1
8	144	218.4	9	9.1	0.1	145.6	218	1.6	2.4
9	180	272.7	10	10.1	0.1	181.8	272	1.8	2.7
10	220	333.0	11	11.1	0.1	222.0	333	2.0	3.0
SSD					0.09			1.1	1.65

4 Experimental Settings

To experimentally investigate the possible effects of SSC in comparison with other crossover operators, we test them on ten real-valued symbolic regression problems. These problems are grouped into three categories: polynomial functions; trigonometric, logarithm and square-root functions; and bivariate functions. Most are taken from

Table 2 Sampling semantics of parents, subtrees and children when swapping two different subtrees.

Points	P_1	P_2	St_1	St_3	St_1-St_3	C_3	C_4	C_3-P_1	C_4-P_2
1	4	6.3	2	2	0	6	4.2	2	2.1
2	12	18.6	3	4	1	24	9.3	12	9.3
3	24	36.9	4	6	2	54	16.4	30	20.5
4	40	61.2	5	8	3	96	25.5	56	35.7
5	60	91.5	6	10	4	150	36.6	90	54.9
6	84	127.8	7	12	5	216	49.7	132	78.1
7	112	170.1	8	14	6	294	64.8	182	105.3
8	144	218.4	9	16	7	384	81.8	240	136.5
9	180	272.7	10	18	8	486	101.0	306	171.7
10	220	333.0	11	20	9	600	122.1	380	201.0
SSD					4.5			143	82.5

Table 3 Symbolic Regression Functions.

Functions	Fitcases
$F_1 = x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F_2 = x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	20 random points $\subseteq$ [-1,1]
$F_5 = sin(x^2)cos(x) - 1$	20 random points $\subseteq$ [-1,1]
$F_6 = sin(x) + sin(x + x^2)$	20 random points $\subseteq$ [-1,1]
$F_7 = log(x + 1) + log(x^2 + 1)$	20 random points $\subseteq$ [0,2]
$F_8 = \sqrt{x}$	20 random points $\subseteq$ [0,4]
$F_9 = sin(x) + sin(y^2)$	100 random points $\subseteq$ [-1,1]x[-1,1]
$F_{10} = 2sin(x)cos(y)$	100 random points $\subseteq$ [-1,1]x[-1,1]

Table 4 Run and Evolutionary Parameter Values.

Parameter	Value
Selection	Tournament
Tournament size	3
Crossover probability	0.9
Mutation probability	0.05
Initial Max depth	6
Max depth	15
Max depth of mutation tree	5
Non-terminals	+, -, *, / (protected versions), sin, cos, exp, log (protected versions)
Terminals	X, 1 for single variable problems, and X,Y for bivariable problems
Raw fitness	sum of absolute error on all fitness cases
Hit	when an individual has an absolute error < 0.01 on a fitness case
Successful run	when an individual scores hits on all fitness cases
Trials per treatment	100 independent runs for each value

the works of Hoai et al. [20], Keijzer [33], and Johnson [30]. These functions are shown in Table 3 and the main parameters used for our experiments are given in Table 4. The parameter settings are similar to our previous work [57]. Although these experiments purely concern crossover, we have retained mutation at a low rate, because we aim to study crossover in the context of a normal GP run. Note that the number of generations and the population size are not specified in Table 4; they will be determined appropriately for each experiment. Note also that the *raw fitness function* is the sum of the absolute error over all fitness cases, and a run is considered as *successful* when some individual *hits* (i.e. absolute error <0.01) every fitness case.

We divided our experiments into three sets. The first set investigates the performance of SSC. SSC was compared with five other crossover operators: Standard Crossover (SC), Semantics Aware Crossover (SAC), Context Aware Crossover (CAC), Soft Brood Selection (SBS), and No Same Mate (NSM) selection. The second set analyses the sensitivity of SSC's parameters – including *lower* and *upper semantic sensitivities*, maximum number of trials (Max_Trial), and number of sample points. The last set of experiments investigate some characteristics of SSC, including the rate of semantically equivalent crossover events, semantic diversity, locality, and constructivity. These three sets of experiments are detailed in the following sections.

5 Comparative Results

This section presents our experimental results on the performance of SSC in comparison with SC, SAC, NSM, CAC, and SBS. When comparing different methods, one of the fundamental questions is how to compare their performance in a fair way. Traditionally, GP researchers often set up a predetermined population size and number of generations. Depending on the methods employed, the standard approach of comparing performance in terms of fitness at each generation may not be completely fair, due to possible differences in computational overhead.

An alternative, and often better, way is to run different GP systems (e.g., using different crossover methods) with the same predefined number of individual fitness evaluations. This would not, however, be fair in the context of this paper, because the semantic subtree checking in SSC and SAC may be performed on much smaller subtrees than the individuals (whole trees), and hence may cost much less. Moreover because of differences in bloat, the average size of the individuals in different methods may also differ [11, 22].

Here, we use a measure based on the number of function node evaluations to estimate the computational cost of each GP run. This kind of measurement has been adopted in a number of recent GP studies [23, 60]. By using the number of node evaluations, we can readily estimate the additional computational effort of non-standard crossovers used in the experiments in this paper (i.e SAC, SSC, CAC, SBS).[3]. In these experiments, the number of node evaluations is set to $15 * 10^6$. This value was experimentally determined as allowing our base comparator, SC, to easily find solutions in the easy problems (F_1), with about 50% successful runs, but not allowing it to readily find solutions to harder problems, with only about 5% success in F_4.

The experimental settings of these experiments were as follows. For all methods, the GP basic parameters were as in Table 4. The population size for SC, SAC, SSC

[3] We assume that the computation costs of all primitive functions are the same, or at least negligibly different when compared to the cost of individual fitness evaluation

Table 5 Number of successful runs out of 100 runs

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	48	22	7	4	20	35	35	16	7	18
NSM	48	16	4	4	19	36	40	28	4	17
SAC2	53	25	7	4	17	32	25	13	4	4
SAC3	56	19	6	2	21	23	25	12	3	8
SAC4	53	17	11	1	20	23	29	14	3	8
SAC5	53	17	11	1	19	27	30	12	3	8
CAC1	34	19	7	7	12	22	25	9	1	15
CAC2	34	20	7	7	13	23	25	9	2	16
CAC4	35	22	7	8	12	22	26	10	3	16
SBS31	43	15	9	6	31	28	31	17	13	33
SBS32	42	26	7	8	36	27	44	30	17	27
SBS34	51	21	10	9	34	33	46	25	26	33
SBS41	41	22	9	5	31	34	38	25	19	33
SBS42	50	22	17	10	41	32	51	24	24	33
SBS44	40	25	16	9	35	43	42	27	33	34
SSC 8	66	28	**22**	10	48	**56**	59	21	25	47
SSC12	**67**	33	14	**12**	47	47	66	**38**	37	51
SSC16	55	**39**	20	11	46	44	**67**	29	30	**59**
SSC20	58	27	10	9	**52**	48	63	26	**39**	51

and NSM were set to 500 as in [57]. For SAC, the semantic sensitivity was set to 10^{-X} with X=2, 3, 4, and 5.[4]. For SSC, the *lower* and *upper semantic sensitivity* were set to 10^{-4} and 0.4, respectively. The maximum number of attempts to form an SSC crossover Max_Trial was set to 8, 12, 16, and 20, forming four schemas of SSC[5]. These values were determined from the experiments in Section 6, where they were found to be suitable values for the performances of SSC.

The population sizes we use for CAC and SBS follow previous research, where they are set much smaller than the population size for SC. Here, we chose 200 as in [37, 42]. For CAC, we followed the Majeed and Ryan [42], in using CAC only after 80% of the node evaluations of a run. We also extended CAC with a scheme similar to Tackett's [56], checking child fitness not only by using all fitness cases, but also through a subset of fitness cases. Ratios of 1/X (X=1, 2, and 4)[6] were used in this experiment (i.e. only 1/X of the fitness cases were used to find the best of breed individual, reducing the overall cost).

For SBS [56], the original experiments used 4 brood sizes (2, 3, 4, 5). Here we used the best two (3, 4). To measure the fitness of the individuals in the brood, we used only a portion of the fitness cases, with a 1/X (X =1, 2, 4) ratio.[7]

To examine and compare the performance of these methods, we recorded two classic performance metrics, namely mean best fitness and the percentage of successful runs. The percentage of successful runs are recorded in Table 5, it should be noted here that a run is called successful run if it can find an individual that scores hits on all fitness cases, where a hit means that for that case, absolute error <0.01. In the result tables, Ms is a shorthand for Methods. In each setting, the best-performing schema is printed

[4] SACs with different X are denoted as SACX (with X=2, 3, 4, and 5)

[5] Denoted as SSCX, where X is 8, 12, 16, and 20

[6] Denoted as CACX with X=1, 2, and 4

[7] Denoted as SBSXY, with X=3, 4 and Y=1, 2, 4.

Table 6 The mean best fitness of 100 runs. A Wilcoxon signed-rank test was conducted; if a treatment is better than SC with a confidence level of 99%, the result is printed in *italic* face

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	0.18	0.26	0.39	0.41	0.21	0.22	0.13	0.26	5.54	2.26
NSM	0.16	0.29	0.34	0.40	0.19	0.17	0.11	*0.19*	5.44	2.16
SAC2	0.16	0.27	0.42	0.50	0.22	0.23	0.15	0.27	5.99	3.19
SAC3	0.13	0.27	0.42	0.48	0.18	0.23	0.15	0.27	5.77	3.13
SAC4	0.15	0.29	0.41	0.46	0.17	0.22	0.15	0.26	5.77	3.03
SAC5	0.15	0.29	0.40	0.46	0.17	0.21	0.15	0.26	5.77	2.98
CAC1	0.33	0.41	0.51	0.53	0.31	0.42	0.17	0.35	7.83	4.40
CAC2	0.32	0.41	0.52	0.53	0.31	0.42	0.17	0.35	7.38	4.30
CAC4	0.33	0.41	0.53	0.53	0.30	0.42	0.17	0.35	7.80	4.32
SBS31	0.18	0.29	0.30	0.36	*0.17*	0.30	0.15	*0.19*	4.78	2.75
SBS32	0.18	0.23	0.28	0.36	*0.13*	0.28	0.10	*0.18*	4.47	2.77
SBS34	0.16	0.23	*0.31*	*0.33*	*0.13*	0.21	0.11	*0.19*	*4.17*	2.90
SBS41	0.18	0.26	*0.27*	0.38	*0.12*	0.20	0.13	*0.20*	*4.40*	2.75
SBS42	0.12	0.24	*0.29*	*0.30*	*0.12*	0.18	0.10	*0.16*	*3.95*	2.76
SBS44	0.18	0.24	*0.33*	*0.35*	*0.15*	0.16	0.11	*0.19*	*2.85*	1.75
SSC8	*0.09*	*0.15*	*0.19*	*0.29*	*0.10*	*0.09*	*0.07*	*0.15*	*3.91*	*1.53*
SSC12	*0.07*	*0.17*	*0.18*	*0.28*	*0.10*	*0.12*	*0.07*	*0.13*	*3.54*	*1.45*
SSC16	*0.10*	*0.15*	*0.23*	*0.26*	*0.10*	*0.10*	*0.06*	*0.14*	*3.11*	*1.22*
SSC20	*0.08*	*0.18*	*0.23*	*0.30*	*0.09*	*0.10*	*0.06*	*0.14*	*2.64*	*1.23*

in bold face. We can see that only SBS and SSC are definitely better than SC; while the performances of NSM and SAC are very similarl to SC, CAC is often poorer. The reason might lie in the high cost of the method CAC uses to find the best crossover site, with the result that it quickly reaches the maximum function node evaluations, and the run terminates.

Turning specifically to SSC and SBS, we find that SSC is often better, and more consistently so, than SBS. While SSC is consistently superior on all tested functions, SBS seems to perform similarly to SC on some functions, such as F_1, F_2 and F_6. For SBS, reducing the number of fitness cases used to choose individuals from the brood improves the performance. It is not clear, however, to what extent we can reduce the number of fitness cases to further enhance the performance. In some cases, reducing only 2 times performs better than 4 times. For SSC, it seems that the values of Max_Trial from 8 to 20 give consistently good performance. In general, SSC performs better than SBS, and is the best of all methods on the the tested problems.

Table 6 shows the best fitness found, averaged over all 100 runs of each GP system. The results are consistent with those in Table 5, in that SAC and NSM are mostly equal to SC, CAC is often worse than SC, and only SBS and SSC are better than SC. The table again shows the consistently superior performance of SSC where it is better than SC on all test functions, while SBS is less convincing on three problems: F_1, F_2, F_6, and F_{10}. It can also be seen that although both SSC and SBS are superior to SC, the margin of improvement is different: SBS is often only slightly better than SC while SSC is widely better than SC in all cases.

We tested the statistical significance of the results in Table 6 using a Wilcoxon signed-rank test with a confidence level of 99%. In Table 6, if a run is significantly better than SC, its result is printed in *italic* face. It can be seen that while NSM is only significantly better than SC on one function (F_8), SBS is regularly significantly

Table 7 The mean best fitness of 100 runs of SSC with different parameter values. SSCUX shows the effect of *upper bound semantic sensitivity*, SSCLX of *lower bound semantic sensitivity*, SSCMTX that of *Max_Trial* (X) and SSCNPX that of the number of sample points.

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SSCU01	0.12	0.18	0.24	0.35	0.12	0.14	0.10	0.19	2.65	0.98
SSCU02	0.09	0.17	0.20	0.30	0.09	0.10	0.08	0.15	1.95	0.83
SSCU04	0.08	0.14	0.21	0.27	0.11	0.07	0.07	0.11	1.00	0.70
SSCU06	0.06	0.16	0.22	0.28	0.11	0.07	0.08	0.12	2.01	0.68
SSCU08	0.06	0.19	0.21	0.29	0.14	0.12	0.08	0.13	2.43	0.96
SSCU1	0.09	0.19	0.26	0.31	0.16	0.15	0.10	0.15	2.53	1.26
SSCL1	0.10	0.16	0.24	0.31	0.15	0.11	0.10	0.15	1.75	1.06
SSCL2	0.06	0.14	0.22	0.27	0.10	0.07	0.07	0.13	1.32	0.66
SSCL3	0.09	0.15	0.22	0.26	0.11	0.08	0.07	0.12	0.99	0.73
SSCL4	0.08	0.14	0.21	0.27	0.12	0.07	0.07	0.11	1.00	0.70
SSCL5	0.09	0.15	0.22	0.29	0.13	0.07	0.07	0.11	1.01	0.73
SSCMT4	0.10	0.20	0.23	0.32	0.12	0.11	0.10	0.15	1.86	0.85
SSCMT8	0.08	0.15	0.21	0.24	0.11	0.09	0.07	0.13	1.12	0.71
SSCMT12	0.08	0.14	0.21	0.27	0.12	0.07	0.07	0.11	1.00	0.70
SSCMT16	0.09	0.16	0.19	0.26	0.10	0.09	0.07	0.11	0.98	0.78
SSCMT20	0.08	0.15	0.19	0.22	0.09	0.08	0.07	0.10	1.20	0.66
SSCNP05	0.07	0.16	0.20	0.25	0.10	0.08	0.11	0.16	1.28	0.70
SSCNP1	0.08	0.14	0.21	0.27	0.12	0.07	0.07	0.11	1.00	0.70
SSCNP2	0.07	0.14	0.21	0.26	0.11	0.07	0.09	0.13	1.11	0.79

better than SC, except on some specific functions, F_1, F_2, F_6, F_7, and F_{10}. SSC is always superior to SC in all cases and on all tested problems.

6 SSC Parameter Sensitivity Analysis

The experiments in this section investigate the effect of changing some parameters of SSC. The GP parameters were setup as in Table 4. The population size was set at 500. Four parameters of SSC were investigated, namely, *lower bound semantic sensitivity* (LBSS), *upper bound semantic sensitivity* (UBSS), Max_Trial (MT), and the number of sample points (NP) used for semantic checking. First, we examined the effect of the most important parameter, UBSS. We fixed the other parameters as follows: LBSS: 10^{-4}, MT: 12, and NP: 20 points for single variable functions and 100 for bivariate functions. The UBSS was set at 6 values: 0.1, 0.2, 0.4, 0.6, 0.8, and 1.[8]

The second experiment analysed the effect of LBSS. In this experiment, the other parameters were set as follows: UBSS= 0.4, MT= 12, and NP= number of fitness cases. Five values for LBSS were investigated, i.e. 10^{-X} where (X=1, 2, 3, 4, and 5).[9]

The third experiment tested sensitivity to the number of trials allowed in selecting similar subtrees in SSC (MT). For this experiment, LBSS= 10^{-4}, UBSS=0.4, and NP= number of fitness cases. MT was set at 4, 8, 12, 16, 20.[10]

The final experiment observed the effect of changing the number of sample points in semantic checking (NP). The experimental settings in this experiment were: LBSS=

[8] Denoted as SSCUX where X is 0.1, 0.2, 0.4, 0.6, 0.8, or 1

[9] Denoted as SSCLX with X=1, 2, 3, 4, and 5.

[10] Denoted as SSCMTX, with X=4, 8, 12, 16, and 20.

Table 8 The percentage of SSC that successfully exchange two semantically similar subtrees

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SSCU01	42.1	40.9	45.6	42.2	39.6	49.3	25.2	27.9	39.8	46.4
SSCU02	57.6	62.4	62.8	61.9	68.3	73.2	52.7	44.1	56.3	64.2
SSCU04	77.2	81.1	79.4	78.4	81.2	85.4	81.2	67.3	74.8	80.5
SSCU06	94.5	95.1	95.2	95.2	95.5	96.1	97.6	88.9	93.4	95.2
SSCU07	97.2	98.4	98.3	98.3	98.5	97.7	99.3	95.4	96.4	98.5
SSCU1	99.6	99.8	99.8	99.7	99.9	99.7	99.9	98.9	99.3	99.6
SSCNP4	42.4	41.6	42.1	41.2	44.0	48.5	47.6	29.4	38.2	44.6
SSCNP8	64.9	67.5	66.3	66.5	68.3	74.5	71.2	52.4	62.5	68.2
SSCNP12	77.2	81.1	79.4	78.4	81.2	85.4	81.2	67.3	74.8	80.5
SSCNP16	85.5	86.9	86.4	86.2	88.9	90.5	89.1	74.3	82.4	86.8
SSCNP20	90.4	91.5	90.8	90.7	93.4	93.9	92.8	83.5	93.8	96.4

10^{-4}, UBSS=0.4, and MT= 12. NP was set to a ratio of 1/2, 1 or 2 of the number of fitness cases.[11]

To estimate the effect of changing these parameters, we recorded the best fitness of a run. These values were averaged over 100 runs, the results being shown in Table 7. We can see that the value of UBSS has a remarkable effect on the performance of SSC. It seems that values from 0.2 to 0.8 are suitable for the problems under test, with values from 0.4 to 0.6 being the best. If UBSS is too small (0.1) or too big (1) the performance of SSC is poorer. This can be explained by recording the percentage of SSC that successfully selects two semantically similar subtrees, as shown in Table 8. [12] We can see that if UBSS is too small, only a few SSCs can succeed in exchanging semantically similar subtrees (from 30% to 40% when UBSS is 0.1), so that SSC underperforms.[13] By contrast, if UBSS is too large, it is almost trivial to find semantically similar subtrees (almost 100% for UBSS=1) because most subtrees are sufficiently semantically similar, so that SSC behaves like SC.

While changing UBSS has a remarkable effect on SSC, LBSS has almost no effect on performance provided it is sufficiently small. Table 7 shows that while LBSS was set to small values (from 10^{-2} to 10^{-5}), the performance of SSC was almost unchanged. In order to understand this, we recorded the percentage of subtrees with SSD smaller than 10^{-2} that are actually semantically identical. In fact, 99% of such semantically equivalent subtrees actually have the same semantics. Thus 99% of these subtrees would have satisfied the equivalence condition regardless of the values of LBSS. Only in the case when LBSS gets too big, e.g. 0.1 , does SSC have poorer performance. In this case, SSC prevents swapping of subtrees with similar but unequal semantics. We recorded how many subtree checks found a nonzero SSD smaller than 0.1; this happened approximately 30% of the time, misleading SSC. In general, we can see that LBSS is only required to be sufficiently small, and perhaps any value under 10^{-2} would be suitable.

The third parameter investigated is the number of unsuccessful trials permitted in selecting semantically similar subtrees (MT). Values of MT from 8 to 20 keep the

[11] Denoted as SSCNPX with X=0.5, 1 or 2.

[12] The values for SSCLX and SSCPX are not shown in this table as they have little effect.

[13] We have tried increasing the Max_Trial to compensate for decreasing the upper bound. This was unsuccessful, as if UBSS is too small, the exchange of semantics between the two parents is also too small, so that SSC is more readily trapped in local optima.

performance of SSC roughly consistent. When MT is too small, e.g. MT = 4, the performance of SSC is worse. This can also be understood by observing the percentage of SSC events that successfully exchanged two semantically similar subtrees. For MT=4, only 30% to 40% of SSC events successfully exchanged subtrees, while this figure rises to about 90% for MT=20. Thus further increasing MT may not help, because nearly all crossover events have already successfully exchanged semantically similar subtrees.

The last parameter under investigation is the number of sample points (NP) on which the semantics was measured. Usually, this number is set equal to the number of fitness cases. The results in Table 7 show that there was little effect when this value was doubled, or when it was halved.

Overall, these results highlight some important issues in determining the values for SSC parameters. It seems that UBSS should lie in the range 0.2 to 0.8, LBSS should be less than 10^{-2}, MT in the range 8 to 20, and NP similar to the number of fitness cases so long as this number is not too big.

7 Some Characteristics of Semantic Similarity based Crossover

This section analyses some characteristics of SSC, namely the rate at which semantically equivalent crossover events occur, the semantic diversity resulting from such crossovers, the locality of the operator, and its constructive effect. The results were compared with SC, SAC and NSM. The GP parameter settings in this section are described in Table 4, with the population size being set to 500 and the number of generation to 50. Five configurations of SAC were used, with semantic sensitivities set to 10^{-X} with X=1, 2, 3, 4, and 5.[14] For SSC, LBSS was set to 10^{-4} and UBSS to 0.4. NP was set equal to the number of fitness cases. Five configurations of SSC were used, with MT varying through 4, 8, 12, 16, and 20.

7.1 Rates of Semantically Equivalent Crossover Events

The first set of results record the extent of semantically equivalent exchanges arising from the three crossover operators. Here we say that a crossover operation is an equivalent crossover if it is performed by exchanging two semantically equivalent subtrees. Since the new crossover operators (SAC and SSC) work by checking the semantics of subtrees and trying to prevent the exchange of semantically equivalent subtrees, it would be informative to see how frequently this actually happens. This information shows us how frequently SC fails to change the semantics of individuals (i.e. makes semantically unproductive crossovers), and the extent to which SAC, and especially SSC, can overcome this problem. The results are shown in Table 9.

It can be seen from Table 9 that the overall average for equivalent crossovers in SC was around 15%; NSM behaved similarly, only reducing the rate by about 1%. By contrast, these values for both SAC and SSC were substantially improved, ranging from 2% to 3% for SAC, and from 2% to 5% for SSC (except when MT is very small, e,g MT = 4). It is clear that SAC and SSC are more semantically exploratory than SC and NSM on these problems. It should also be noted here that 99% of pairs of semantically equivalent subtrees consist of subtrees with identical semantics. As a result, approximately 99% of such crossovers leave the fitnesses of the chidren unchanged.

[14] Denoted as SACX, for X=1, 2, 3, 4, or 5.

Table 9 The percentage of Semantically Equivalent Crossover for four crossover operators: SC, NSM, SAC and SSC.

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	15.5	14.4	14.7	14.1	13.2	15.7	14.3	14.4	12.4	14.1
NSM	14.9	14.1	14.5	13.8	12.5	14.8	14.2	14.1	11.5	12.7
SAC1	3.54	3.12	3.18	3.12	3.28	3.84	3.32	3.34	3.49	3.67
SAC2	2.18	1.88	1.93	1.86	1.53	2.24	1.80	1.89	1.49	1.99
SAC3	2.16	1.85	1.90	1.65	1.52	2.08	1.77	1.88	1.47	2.01
SAC4	2.15	1.83	1.88	1.84	1.52	2.04	1.75	1.86	1.46	1.97
SAC5	2.10	1.82	1.87	1.84	1.51	2.01	1.72	1.81	1.46	2.03
SSC4	8.87	8.33	8.35	8.06	6.83	7.81	7.27	9.80	7.63	6.56
SSC8	5.92	4.76	4.97	4.76	3.89	3.88	4.00	6.94	4.76	3.75
SSC12	4.11	2.76	3.10	3.04	2.25	2.38	2.70	5.08	3.24	2.84
SSC16	2.80	1.99	2.27	2.00	1.39	1.61	1.65	3.94	1.99	2.04
SSC20	2.29	1.49	1.57	1.40	0.96	1.19	1.24	2.82	1.49	1.76

Table 10 The percentage of generating new semantics for SC, NSM, SAC and SSC (i.e. differing in terms of the semantic equivalence measure).

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	62.9	67.9	67.6	68.1	61.5	67.0	66.4	61.0	74.8	74.9
NSM	66.9	70.4	70.1	72.2	63.2	70.9	71.1	66.6	78.3	77.6
SAC1	70.2	73.5	75.1	76.1	65.2	74.1	73.7	70.2	82.1	84.6
SAC2	71.3	74.5	77.4	77.6	66.5	75.9	76.7	71.7	83.8	86.7
SAC3	72.1	74.7	77.6	78.3	67.7	76.9	75.4	71.7	84.1	86.3
SAC4	71.6	75.1	77.4	77.5	66.1	76.7	75.2	71.5	84.1	86.5
SAC5	71.4	74.9	77.1	77.5	65.9	76.8	75.4	71.8	84.3	85.8
SSC4	72.1	76.3	77.4	79.1	69.2	76.3	75.6	75.5	78.7	82.8
SSC8	75.9	80.7	80.8	82.9	73.6	82.7	81.3	74.8	80.8	89.3
SSC12	78.9	84.1	84.5	84.3	78.3	83.8	82.8	76.9	85.8	90.2
SSC16	78.8	85.9	85.6	87.2	78.1	86.9	85.2	78.6	89.8	91.6
SSC20	77.9	85.3	86.7	87.4	79.1	84.5	83.9	78.7	88.9	91.0

The improved semantic exploratory capacity of SAC and SSC can potentially lead to more semantic diversity, in that they could generate more new semantics than SC and NSM. Here, crossover A is considered to generate more semantic diversity than crossover B if A generates semantically new children, differing from the semantics of the parents, at a higher rate than B. In Table 10 we measured this rate for each crossover configuration. In table 10 we see that while NSM was only slightly better than SC, SAC was better than both, while SSC was better than all other crossover operators in this respect. Interestingly, although SAC was often better than SSC in preventing equivalent crossovers, by keeping semantic changes small, SSC was generally better than SAC at producing semantically diverse crossovers. We note that SAC and SSC cannot guarantee the generation of semantically new offspring, despite trying to swap semantically different subtrees. We suspect this arises from the existence of fixed-semantic subtrees similar to those whose existence McPhee demonstrated in Boolean domains [43].

Table 11 The average change of fitness after crossover for SC, NSM, SAC, and SSC (averaged over the whole population and 100 runs.

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	9.74	9.21	10.5	10.6	7.30	7.44	8.13	9.36	17.8	20.1
NSM	7.64	8.30	9.21	10.1	6.00	6.34	7.40	7.84	14.7	18.5
SAC1	8.42	8.71	9.54	10.9	7.01	6.54	7.05	7.96	15.9	17.7
SAC2	8.38	8.69	9.42	10.8	6.93	6.48	6.96	7.85	15.8	17.5
SAC3	8.03	8.63	9.19	10.3	6.82	6.56	6.92	7.66	15.5	17.3
SAC4	7.88	8.64	9.03	10.4	7.14	6.65	7.30	7.60	15.5	17.1
SAC5	7.88	8.70	9.43	10.4	7.18	6.78	7.25	7.68	15.4	17.4
SSC4	6.83	6.41	6.72	7.11	5.06	4.60	5.38	6.50	13.1	13.5
SSC8	5.12	5.01	5.69	5.45	3.58	3.47	3.87	5.84	12.4	9.50
SSC12	4.07	4.00	4.90	4.97	3.09	2.70	3.32	5.26	11.3	8.76
SSC16	4.34	3.44	4.26	4.10	2.84	2.58	2.82	4.45	9.25	7.83
SSC20	4.19	3.22	3.55	3.90	2.56	2.26	2.97	3.64	7.32	9.15

7.2 Operator Locality

The next set of experments analyse the locality of SSC compared with SAC, SC and NSM. It is generally believed that using a representation with high locality (small change in genotype correspond to small change in phenotype) is important for efficient evolutionary search [15–17, 21, 52]. It is also generally agreed that designing a search operator for GP ensuring which achieves this is very difficult. Thus most current GP representations and operators are low-locality – a small (syntactic) change from parent to child can cause a large semantic change. Our new crossover operator (SSC) differs from others in directly controlling the scale of change in terms of semantics rather than syntax.

To compare locality, we measured the fitness change between parents and children in crossover. For example, suppose two individuals having fitness of 10 and 15 are selected for crossover, and their children have fitness of 17 and 9. The change of fitness is $Abs(17-10)+Abs(9-15)=13$ (for this purpose, we compare the fitness of a child with that of the parent in which it is rooted). This value was averaged over the whole population and over 100 runs. The average fitness change of individuals before and after crossover is shown in Table 11.

Table 11 shows that the step size of the fitness change for SSC was much smaller than for either SAC, SC or NSM. This leads to smoother fitness change over time for SSC than for the others. This is important, as it is not easy to ensure the locality property GP. The table also reveals that the fitness change in SAC and NSM were only slightly smoother than in SC.

7.3 Constructive Effects

The previous results show that SAC and SSC are more semantically productive than SC and NSM, and that SSC has higher locality than the others. Does this help SSC (and maybe SAC) to generate better children than their parents (more constructive crossover)? We measured the constructive effect of SAC, SSC, NSM and SC, using Majeed's [42] method. However we adapt the method slightly, only comparing the fitness of a child with that of the parent in which it is rooted.

Table 12 The percentage of semi-constructive crossovers of SC, NSM, SAC, and SSC (i.e. at least one child is better than the corresponding parent.

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	19.2	21.2	21.0	21.1	18.7	21.1	21.4	18.6	28.2	28.9
NSM	22.2	23.3	22.5	23.8	19.5	23.8	24.1	21.5	31.6	30.6
SAC1	24.7	26.3	26.5	26.7	22.3	26.1	27.2	24.2	34.1	34.2
SAC2	25.4	26.3	27.8	27.8	23.5	27.5	28.8	25.6	36.0	36.9
SAC3	25.9	26.5	27.8	28.3	23.8	28.2	27.6	25.5	36.2	36.7
SAC4	25.7	27.0	27.7	27.8	23.2	28.2	27.4	25.5	36.4	37.4
SAC5	25.7	26.9	27.6	27.9	22.9	28.6	27.5	25.6	36.2	37.1
SSC4	26.9	28.2	29.0	29.3	25.7	28.9	28.7	25.9	33.4	36.8
SSC8	30.0	32.1	32.0	32.7	29.9	33.9	33.2	29.0	36.8	41.2
SSC12	32.7	35.5	34.6	34.3	32.9	35.5	35.2	31.9	38.5	43.1
SSC16	33.3	37.1	35.9	37.1	33.6	37.2	36.3	34.1	41.5	43.3
SSC20	32.7	36.7	37.0	36.9	34.7	36.0	36.0	34.0	41.4	42.8

Table 13 The percentage of full-constructive crossovers of SC, NSM, SAC, and SSC (i.e. where both children are better than the corresponding parents.

Ms	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
SC	2.06	2.27	2.23	2.24	1.96	2.32	2.36	1.88	3.46	3.74
NSM	2.26	2.46	2.37	2.66	1.97	2.56	2.59	2.15	3.90	3.78
SAC1	2.82	3.04	3.05	3.02	2.46	3.11	3.25	2.65	4.51	4.67
SAC2	2.90	3.04	3.29	3.25	2.67	3.24	3.49	2.89	4.95	4.87
SAC3	3.05	3.10	3.29	3.36	2.65	3.35	3.32	2.81	4.94	4.91
SAC4	3.03	3.17	3.25	3.28	2.57	3.39	3.29	2.82	4.96	4.94
SAC5	3.01	3.18	3.20	3.31	2.52	3.43	3.28	2.82	4.92	4.92
SSC4	3.60	3.80	3.78	3.92	3.20	3.94	3.63	3.07	4.68	5.18
SSC8	4.25	4.58	4.54	4.65	3.96	5.11	4.53	3.89	5.17	6.04
SSC12	4.70	5.21	5.07	5.02	4.51	5.38	4.88	4.43	5.45	6.35
SSC16	4.76	5.55	5.26	5.49	4.65	5.73	5.12	4.85	6.00	6.34
SSC20	4.73	5.34	5.39	5.47	4.86	5.49	5.01	4.73	5.86	6.31

We can distinguish *semi-constructive crossovers* from *full-constructive crossovers*. Let us assume that two parents P_1 and P_2 are selected for crossover, generating two children C_1, C_2 (C_1 rooted in P_2 and C_2 rooted in P_2). Then, a crossover is called semi-constructive if it generates at least one child that is better than its parents. In other words, the condition (C_1 *is better than* P_1 *OR* C_2 *is better than* P_2) is used to count semi-constructive crossovers. When the condition is more strict – both children are better than their parents (C_1 *is better than* P_1 *AND* C_2 *is better than* P_2) – the crossover is called full-constructive. A crossover that is not semi-constructive nor full-constructive is called destructive.

The semi-constructive and full-constructive crossovers' results for SSC, SAC, NSM and SC are shown in Table 12 and Table 13, respectively. It can be seen from Table 12 that while NSM is only slightly more semi-constructive than SC, both SSC and SAC were more semi-constructive than SC and NSM. This is a consequence of the greater semantic diversity of SAC and SSC relative to SC and NSM. Usually, the semi-constructive crossover rate of SAC is from 5% to 7% higher than SC, and of SSC from 12% to 18% higher. These increases are particularly important because the semi-constructive rate for SC was fairly small (about 20%).

Table 13 shows how difficult it is for GP standard crossover to generate improved solutions. The percentage of fully constructive crossovers for SC and NSM were roughly the same, at only 2% for one-variable functions and 3% for bivariate functions. By adding semantics to control the crossover operator, far more full-constructive behaviour is obtained. SAC often scored 1.5 times higher than SC and NSM in frequency of full-constructive events, and SSC around 2 to 3 times higher. SSC generated more full-constructive events than SAC (up to 1.5 times) on both univariate and bivariate functions.

8 Conclusions

In this paper, we have proposed a new method for measuring semantics of real-valued symbolic regression problems, which we called *Sampling Semantics* (SS). Using it, we can define the semantic distance between two subtrees; we then proposed two semantic relations (Semantic Equivalence and Semantic Similarity) for determining the semantic acceptability of exchanging two subtrees. These semantic relations are used to guide crossover, resulting in two new semantically based crossover operators for GP: Semantics Aware Crossover (SAC) and Semantic Similarity-based Crossover (SSC). The new operators were tested on a class of real-valued symbolic regression problems and compared with some similar schemas including No Same Mate (NSM), Soft Brood Selection (SBS) and Context Aware Crossover (CAC), as well as standard GP crossover (SC). On a wide range of problems, only SSC and SBS were consistently better than SC, and of them SSC is the most effective crossover operator.

We also investigated the effect of various parameters on SSC to determine ranges of suitable values. Some characteristics of SSC were analysed, showing that both SAC and SSC improve the resulting semantic diversity. We showed that SSC achieves higher locality than either SAC, NSM or SC. We argue that this is the main reason for its better constructive effect compared to SAC, NSM and SC. This results in a substantial, and statistically significant, improvement in performance from SSC, while SAC and NSM generate almost equivalent performance to SC.

8.1 Assumptions and Limitations

Although this paper has shown that many benefits are to be gained from incorporating semantics into the design of crossover operators for GP, there are some limitations. First, the paper focuses on the domain of real-valued problems, leaving other domains an open question. [15] Second, the semantic sensitivities were experimentally determined and might not be the best choices either for these problems, and/or for others. Adaptive mechanisms to determine these values are currently under investigation.

We hypothesised that fixed semantics might occur in real-valued trees as in Boolean trees, but further studies need to be conducted to understand whether fixed semantics really occurs, and if so, what form it takes in real-valued problem domains.

[15] In fact one simple way to use our method is to transform boolean function learning problems to real-valued ones as in [54].

9 Future work

In the near future, we plan to extend this work in a number of ways. First, we will apply SSC to some more difficult symbolic regression problems (problems that have more complex-structured solutions). For these problems, we predict that making small changes in semantics will be both more difficult, but also more important. Second, SSC could be used to enhance some previously proposed crossover operators, which are based purely on the structure of trees – such as crossover with bias on the depth of nodes [24], one point crossover and uniform crossover [39, 48]. Another potential research direction is to apply SSC to other problem domains, such as the Boolean problems that have been previously investigated in [43]. In this case, it may be even more difficult to generate children that differ semantically from their parents, so that the benefits may be greater. Last but not least, we plan to investigate suitable ranges for the *lower* and *upper bound semantic sensitivity* values for various classes of problems. In this paper, these values were manually and experimentally specified; however, it may be possible to allow these values to self-adapt during the evolutionary process [13].

Acknowledgements

This paper was funded under a Postgraduate Scholarship from the Irish Research Council for Science Engineering and Technology (IRCSET). The authors would like to thank the members of NCRA (Natural Computing Research & Applications Group) at University College Dublin.

References

1. C. Alan. *Meaning and Language: An introduction to Semantics and Pragmatics*. Oxford Textbooks in Linguistics, Cambridge, UK, 2004.
2. L. Altenberg. The evolution of evolvability in genetic programming. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press, 1994.
3. C. Baier and J. P. Katoen. *Principle of Model Checking*. MIT Press, 2008.
4. 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.
5. L. Beadle and C. G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307–337, Sep 2009.
6. L. Beadle and C. G. Johnson. Semantically driven mutation in genetic programming. In A. Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 1336–1342, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press.
7. 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.
8. R. Cleary and M. O'Neill. Solving knapsack problems with attribute grammars. In *Proceedings of the Grammatical Evolution Workshop*, 2004.

9. 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.
10. A. M. Collins and M. R. Quillian. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior,*, 8:240–247, 1969.
11. J. M. Daida, D. S. Ampy, M. Ratanasavetavadhana, H. Li, and O. Chaudhri. Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO'1999)*, pages 1851–1858. Morgan Kaufmann, 1999.
12. 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.
13. 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.
14. R. E.Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.
15. E. Galvan-Lopez and M. O'Neill. On the Effects of Locality in a Permutation Problem: The Sudoku Problem. In *CIG*. IEEE, 2009.
16. E. Galvan-Lopez and M. O'Neill. Towards Understanding the Effects of Locality in Genetic Programming. In *MICAI*, Lecture Notes in Computer Science. Springer, 2009.
17. 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.
18. S. Gustafson, E. K. Burke, and N. Krasnogor. On improving genetic programming for symbolic regression. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 912–919, Edinburgh, UK, 2005. IEEE Press.
19. 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.
20. 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, 2002.
21. N. X. Hoai, R. I. McKay, and D. Essam. Representation and structural difficulty in genetic programming. *IEEE Transection on Evolutionary Computation*, 10(2):157–166, 2006.
22. N. X. Hoai, R. I. B. McKay, D. Essam, and H. Abbass. Toward an alternative comparison between different genetic programming systems. In M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 67–77. Springer-Verlag, 2004.
23. T.-H. Hoang, D. Essam, R. I. B. McKay, and X. H. Nguyen. Building on success in genetic programming:adaptive variation & developmental evaluation. In *Proceedings of the 2007 International Symposium on Intelligent Computation and Applications (ISICA)*, Wuhan, China, Sep 2007. China University of Geosciences Press.

24. 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.
25. T. Ito, H. Iba, and S. Sato. A self-tuning mechanism for depth-dependent crossover. In *Advances in Genetic Programming*, page 377399. IEEE Press, June 1999.
26. 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.
27. C. Johnson. Genetic programming with guaranteed constraints. In *Recent Advances in Soft Computing*, pages 134–140. The Nottingham Trent University, 2002.
28. C. Johnson. What can automatic programming learn from theoretical computer science. In *Proceedings of the UK Workshop on Computational Intelligence*. University of Birmingham, 2002.
29. 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.
30. C. Johnson. Genetic programming crossover: Does it cross over? In *Proceedings of the 12th European Conference on Genetic Programming (EuroGP2009)*, pages 97–108. Springer, 2009.
31. 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.
32. 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.
33. M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *Proceedings of EuroGP'2003*, pages 70–82. Springer-Verlag, April 2003.
34. D. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2:95, 1968.
35. J. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, MA, 1992.
36. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.
37. K. Krawiec and P. Lichocki. Approximating geometric crossover in semantic space. In F. Rothlauf, editor, *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, pages 987–994. ACM, 2009.
38. K. Krawiec and B. Wieloch. Functional modularity for genetic programming. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 995–1002, Montreal, July 2009. ACM.
39. W. B. Langdon. Size fair and homologous tree genetic programming crossovers. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1092–1097. Morgan Kaufmann, July 1999.
40. W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, Berlin, 2002.
41. 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.

42. 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.
43. 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, 2008.
44. N. Mori, B. McKay, N. X. Hoai, D. Essam, and S. Takeuchi. A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 13(3):237–244, 2009.
45. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2005.
46. H. R. Nielson and F. Nielson. *Semantics with Applications: An Appetizer*. Springer, Springer-Verlag, London, UK, 2007.
47. U. M. O'Reilly and F. Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. *Lecture Notes in Computer Science*, 866(1):397406, 1994.
48. 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.
49. R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).
50. B. J. Ross. Logic-based genetic programming with definite clause translation grammars. *New Gen. Comput.*, 19(4):313–337, 2001.
51. F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, 2nd edition edition, 2006.
52. F. Rothlauf and D. Goldberg. Redundant Representations in Evolutionary Algorithms. *Evolutionary Computation*, 11(4):381–415, 2003.
53. 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.
54. R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.
55. W. A. Tackett. *Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, USA, 1994.
56. W. A. Tackett and A. Carmi. The unique implications of brood selection for genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.
57. 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*, pages 292–302. Springer, April 2009.
58. 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.
59. M. L. Wong and K. S. Leung. Learning programs in different paradigms using genetic programming. In *Proceedings of the Fourth Congress of the Italian Association for Artificial Intelligence*. Springer-Verlag, 1995.

60. P. Wong and M. Zhang. SCHEME: Caching subtrees in genetic programming. In J. Wang, editor, *2008 IEEE World Congress on Computational Intelligence*, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press.