Self-Adapting Semantic Sensitivities for Semantic Similarity Based Crossover

Nguyen Quang Uy NCRA, UCD, Ireland,
Email: quanguyhn@gmail.com Bob McKay School of CSE, SNU, Korea,
Email: rimsnucse@gmail.com Michael O'Neill NCRA, UCD, Ireland,
Email: m.oneill@ucd.ie Nguyen Xuan Hoai School of IT, VMTA, Vietnam
Email:nxhoai@gmail.com

***Abstract*—This paper presents two methods for self-adapting the semantic sensitivities in a recently proposed semantics-based crossover: Semantic Similarity based Crossover (SSC) [1]. The first self-adaptation method is inspired by a self-adaptive method for controlling mutation step size in Evolutionary Strategies (1/5 rule). The design of the second takes into account more of our previous experimental observations, that SSC works well only when a certain portion of events successfully exchange semantically similar subtrees. These two proposed methods are then tested on a number of real-valued symbolic regression problems, their performance being compared with SSC using predetermined sensitivities and with standard crossover. The results confirm the benefits of the second self-adaption method.**

***Index Terms*—Genetic Programming, Semantics, Crossover, Self-Adaptation, Symbolic Regression.**

I. Introduction

Genetic Programming (GP) is an evolutionary algorithm, inspired by biological evolution, for finding problem solutions in the form of computer programs [2], [3], [4]. The program is usually presented through syntactic formalisms such as s-expression trees [2], a linear sequence of instructions, grammar derivation trees, or graphs [5]. The genetic operators in such GP systems are generally designed to ensure the syntactic closure property – i.e. to always produce syntactically valid children from syntactically valid parent(s). GP evolutionary search is performed on the syntactic space of programs using such purely syntactic operators, with the only semantic guidance coming from the fitness of individuals.

Although GP has shown its effectiveness in solving diverse problems, the limitations to (finite) behavior-based semantic guidance, and purely syntactic genetic operators, are somewhat alien to the perspective of programmers. In programming, the search for appropriate computer programs is generally constrained not merely by syntax, but also by semantics. In normal practice, a change to a program is only made after careful attention to the change in semantics. To incorporate more of this flavour into GP, a number of researchers have proposed a variety of semantically based methods for controlling the genetic operators [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16].

In recent work [1], Uy et al presented *Semantic Similarity based Crossover* (SSC), which pays attention to the scale of semantic difference between two subtrees. The results reported in [1] show that SSC helps to improve the performance of GP in solving a family of real-valued symbolic regression problems. However the work in [1] has some undoubted drawbacks:

1) The definition of semantic distance leaves its value dependent on the number of sample points used in approximating the semantics.
2) SSC was tested on only one family of problems with a single domain (range of input values).
3) Most important, the performance of SSC depends strongly on some manually tuned parameters (semantic sensitivity). This leaves to users the problem of determining appropriate semantic sensitivities.

This paper aims to reduce the above problems; the main contributions are:

1) Modification of the semantic distance to use the mean, rather than the sum, of absolute distances, so removing the dependence on the number of sample points.
2) Testing on a wider range of regression problems.
3) Proposing and testing of two self-adaptive schemas for tuning semantic sensitivities

The remainder of the paper is organised as follows. In the next section, we give a review of related work on semantic based crossovers in GP and a brief review of self-adaptation in Evolutionary Computation (EC). Section III describes SSC and the two proposed self-adaptive mechanisms. The experimental settings are detailed in Section IV. The results of the experiments are presented and discussed in section V. Section VI concludes the paper and highlights some potential future work.

II. Related Work

This section presents the review of related work on semantic based crossovers in GP and a brief review of self-adaptation in Evolutionary Computation (EC)

A. Semantics in Genetic Programming

Recently, GP researchers have paid increasing attention to the use of semantics to improve the ability of GP to solve problems. Generally, researchers have taken this to mean

making use of additional information to guide GP search. The work falls into three main strands:

1) using formal methods [7], [8], [11], [12], [13]
2) using grammars [6], [9], [10]
3) using structures such as GP trees [14], [15], [16], [1]

The first approach was advocated by Johnson in a series of papers [7], [8], [11]. In these methods, semantic information extracted through formal methods (e.g., Abstract Interpretation and Model Checking) is used to quantify fitness in problems where it is difficult to measure by sample point fitness. Katz and Peled subsequently used model checking to solve the Mutual Exclusion problem [12], [13]. Again, individuals' fitness is measured through model checking. These formal methods have a strict mathematical foundation, that potentially may aid GP. Perhaps because of high complexity, however, these methods have seen only limited research despite the advocacy of Johnson [17]. Their main application to date has lain in evolving control strategies.

In the second category, Attribute Grammars are the most popular formalism. Attributes added to a grammar can generate some useful semantic information about individuals, which can be used to eliminate bad individuals [10], or to avoid generating semantically invalid ones [6], [9]. However the attributes used to represent semantics are problem dependent, and it is not always easy to design such attributes for a new problem.

In the last category, semantics has mainly been used to control the GP operators. In [14], the authors investigated the effect of semantic diversity on Boolean domains, checking the semantic equivalence between offspring and parents by transformation to a canonical form, Reduced Ordered Binary Decision Diagrams (ROBDDs) [18]. Their information is used to determine whether the offspring are copied to the next generation. The method improved GP performance, presumably because it increased semantic diversity. The method has also been applied to mutation [19] and to initialisation [20].

While, most of previous research on semantics in GP were focused on combinatorial and boolean problems [9], [14], [15], [12], research on real-valued domains [16], [1], [21] is much more recent. Krawiec and Lichocki [21] based the semantics of individuals on fitness cases, using it to guide crossover (*Approximating Geometric Crossover* - AGC). AGC turned out no better than standard crossover (SC) on real-valued problems, and only slightly better on Boolean.

Uy et al. [16] proposed Semantics Aware Crossover (SAC), another crossover operator promoting semantic diversity, based on checking semantic equivalence of subtrees. It showed limited improvement on some real-value problems; it was subsequently extended to Semantic Similarity based Crossover (SSC) [1], which turned out to perform better than both SC and SAC [1]. However, the performance of SSC depended on some predetermined parameters, the semantic sensitivities. Our aim here is to test whether, on a broad class of problems, self-adaptation of semantic sensitivity could be used.

B. Self-Adaptation in Evolutionary Computation

There have been a number of studies of automated control of different aspects of evolutionary algorithms. Angeline [22] distinguished three categories:

1) *population-level* – techniques that statistically analyse the global information from the whole population and use this information to dynamically adjust parameters. For example, in [23], the authors recorded the statistics of all subtrees to figure out the points that are likely to be more advantageous for crossover. A self-adaptive scheme that measures the ratio of successful mutations to all mutations in [24] is also a kind of *population-level* techniques.
2) *individual-level* – adaptive methods that alter the way an individual itself is treated by the system. Harper and Blair [25] used self-adaptation to choose between crossover operators in Grammatical Evolution (GE [26]) for a particular pair of parents
3) *component-level* – adaptive techniques that automatically adjust one or more elements of an individual. Angeline's scheme for determining multi-point crossovers [27] can be seen as an example.

Most early research on self-adaptation focused on controlling operators (crossover and mutation). More recently, self-adaptive schemas have also been developed for other aspects. One parameter that has been attracted much attention is the population size [28], [29]. Silva and Dignum [30] presented a self-adaptive method for setting the maximum length of GP individuals.

The self-adaptive methods proposed in this paper can be seen as population-level methods, in that they gather information from the whole population and use this information in the next generation. They differ from previous self-adaptive schemas in self-adapting two new parameters, the *semantic sensitivities*, in SSC.

III. Methods

This section briefly presents SSC and then details two adaptive methods for it.

A. Semantic Similarity based Crossover

The SSC is almost identical to that of Uy et al. [1], except for a slightly modified distance measure. We start with a clear defintion of (sub)tree semantics. Formally, the *Sampling Semantics* (SS) of a (sub)tree is defined as follows:

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

The optimal choice of N and P depend on the problems; we follow the approach of [1] in setting the number of points for evaluating the semantics equal to the number of fitness cases (20 for single variable functions and 100 for bivariate functions – Section IV) and in choosing the sequence of points P uniformly randomly from the problem domain.

Based on SS, we define a *Sampling Semantics Distance* (SSD) between two subtrees. It differs from that in [1] in using the mean absolute difference in SS values, rather than (as before) the sum of absolute differences. Let $U = (u_1, u_2, ..., u_N)$

and $V = (v_1, v_2, ..., v_N)$ represent the SSs of two subtrees, S_1 and S_2; then the SSD between S_1 and S_2 is defined in equation 1:

$$SSD(S_1, S_2) = \frac{\sum_{i=1}^{N} |u_i - v_i|}{N} \quad (1)$$

We follow [1] in defining a semantic relationship, *Semantic Similarity* (SSi), on the basis that the exchange of subtrees is most likely to be beneficial if they are not semantically identical, but also not too different. Two subtrees are semantically similar if their SSD lies within a positive interval. The formal definition of SSi between subtrees S_1 and S_2 is as the following equation:

$$SSi(S_1, S_2) = \begin{array}{l} \textbf{if } \alpha < \text{SSD}(S_1, S_2) < \beta \\ \quad \textbf{then } \text{true} \\ \quad \textbf{else } \text{false} \end{array}$$

α and β are two predefined constants, the *lower* and *upper* bounds for semantics sensitivity. In general, the best values for *lower* and *upper bound semantic sensitivity*(referred as LBSS and UBSS respectively) are problem dependent. In [1], the best settings found by manual tuning for the particular problems were $\alpha = 10^{-3}$ and $\beta = 0.4$.

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;
    generate a number of random points (P) on the
    problem domain;
    calculate the SSD between Subtree1 and Subtree2
    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_Trail then
    choose a random crossover point Subtree1 in P1;
    choose a random crossover point Subtree2 in P2;
    execute crossover;
    return true;
```

The primary objective of SSC was to improve the locality of crossover. Algorithm 1 (adapted from [1]) shows the detailed operation of SSC. The value of Max_Trial was set at 12, a value which was determined through experimental results.

B. Self-Adaptation for Semantics Sensitivity

The self-adaptive methods proposed in this subsection focus primarily on automatically adjusting UBSS, as it is more difficult to tune than LBSS. LBSS is then adjusted, based deterministically on the UBSS.

Our first method is inspired by the 1/5 rule in an Evolutionary Strategy (ES). The 1/5 rule was first proposed by Rechenberg [24] for controlling mutation variance in ES. It was inspired by a theoretical analysis for optimising ES parameters. It may be stated as: *The optimum ratio of constructive mutations to all mutations is 1/5. If it exceeds 1/5, increase the variance; if less, decrease variance*. A mutation is said to be constructive if it improves the fitness of the parent. Let ${p_s}^t$ be the ratio of constructive mutations at generation t, and σ^t the mutation variance; then the mutation variance at generation $t+1$, σ^{t+1}, is adjusted using equation 2:

$$\sigma^{t+1} = \begin{cases} c_d.\sigma^t & \text{if } {p_c}^t < 1/5; \\ c_i.\sigma^t & \text{if } {p_c}^t > 1/5; \\ \sigma^t & \text{if } {p_c}^t = 1/5. \end{cases} \quad (2)$$

c_i is a predefined increment, generally set between 0.82 and 0.85; c_d is a predefined decrement, generally set to $1/c_d$ [31].

We propose a variant for semantic sensitivity, which we call Self-Adaptation based on Constructive Effects (SACE). UBSS is self-adapted using equation 3:

$$\beta^{t+1} = \begin{cases} c_d \cdot \beta^t & \text{if } {p_c}^t < \epsilon; \\ c_i \cdot \beta^t & \text{if } {p_c}^t > \epsilon; \\ \beta^t & \text{if } {p_c}^t = \epsilon. \end{cases} \quad (3)$$

β^t denotes the UBSS at generation t, while ${p_s}^t$ is the constructive ratio for SSC, and c_i, c_i are as above. A crossover is considered to be constructive if it produces two children that are fitter than their parents. The LBSS is adapted by the simple equation: α^{t+1}=10^{-3}*β^{t+1}. In this paper, we tested several values of c_d, c_i, and ϵ.

It might seem that we have gained little, replacing two parameters (α and β) by three; however if the algorithm is much less sensitive to the new values than the old, then there has been an important gain.

If UBSS is too small, SSC may mostly fail, it being very difficult to select subtree pairs satisfying the SSC condition. Conversely, if UBSS is too large, SSC will generally succeed, behaving the same as SC. Instead of controlling the constructive rate, we might instead aim to ensure that a certain portion of crossovers successfully exchange semantically similar subtrees (whatever the final effect on fitness). Self-Adaptation based on Successful Execution (SASE) aims to ensure that an intermediate proportion of SSC events successfully exchange similar subtrees. It is controlled using equation 4:

$$\beta^{t+1} = \begin{cases} c_d \cdot \beta^t & \text{if } {p_s}^t < \epsilon; \\ c_i \cdot \beta^t & \text{if } {p_s}^t > \epsilon; \\ \beta^t & \text{if } {p_s}^t = \epsilon. \end{cases} \quad (4)$$

TABLE I
SYMBOLIC REGRESSION FUNCTIONS.

Groups	Functions	Fitness Cases
Group$_1$	$F_1 = x^3 + x^2 + x$	20 points $\subseteq$ [-1,1]
	$F_2 = x^4 + x^3 + x^2 + x$	20 points $\subseteq$ [-1,1]
	$F_3 = x^5 + x^4 + x^3 + x^2 + x$	20 points $\subseteq$ [-1,1]
	$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	20 points $\subseteq$ [-1,1]
Group$_2$	$F_5 = sin(x^2)cos(x) - 1$	20 points $\subseteq$ [0,1]
	$F_6 = sin(x) + sin(x + x^2)$	20 points $\subseteq$ [0,π/2]
	$F_7 = log(x + 1) + log(x^2 + 1)$	20 points $\subseteq$ [1,3]
	$F_8 = \sqrt{x}$	20 points $\subseteq$ [0,4]
Group$_3$	$F_9 = sin(x) + sin(y^2)$	100 points $\subseteq$ [0,1]x[0,1]
	$F_{10} = 2sin(x)cos(y)$	100 points $\subseteq$ [0,1]x[0,1]
	$F_{11} = x^y$	100 points $\subseteq$ [0,1]x[0,1]
	$F_{12} = x^4 - x^3 + y^2/2 - y$	100 points $\subseteq$ [0,1]x[0,1]

TABLE II
RUN AND EVOLUTIONARY PARAMETER VALUES.

Parameter	Value
Generations	50
Population size	500
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	+, -, *, /, 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
Termination	max generations exceeded
Trials per treatment	100 independent runs for each value

as before β^t is the UBSS at generation t, and ${p_s}^t$ is the proportion of SSC that succeed. As before, the LBSS is set to α^{t+1}=10^{-3}*β^{t+1}. Several values of ϵ will be investigated in the following sections.

IV. EXPERIMENTAL SETTINGS

We investigated the effects of SACE and SASE, comparing them with both SSC (i.e. with predetermined semantic sensitivities) and SC. We tested these four operator schemas on twelve real-valued symbolic regression problems, classified into three groups. Group$_1$ includes four single variable functions in the form of binomial expressions. Group$_2$ includes four single-variable functions using trigonometric, log and square functions. Most are taken from [32] and [33]. The domains of these functions differ from those in $Group_1$. Group$_3$ contain four bivariate trigonometric, exponential, and binomial functions, again from [32]. All are presented in table I.

The experimental parameter settings are listed in Table II, and follow those of [1]. Our aim is to study the behaviour of crossover in the context of a normal GP run, so we have retained a low rate of mutation. Note that the raw fitness function is the sum of absolute error over all fitness cases, and a run is considered successful when some individual scores hits on all fitness cases (i.e. absolute error $<$0.01) . The LBSS for SSC is set at 10^{-3}, while UBSS is tested at good values from [1], namely 0.4, 0.5 and 0.6, giving SSCX with X being 04, 05 or 06.

For SACE, five configurations were tested. In the first three configurations, we set the rate of constructive crossover at 5% (this value being calibrated from experiments as one of the best for SACE), using c_i as $0.85, 0.9, 0.95$ (called SACEX, where X is 85, 90, or 95). For the last two configurations, we fixed c_i at 0.9 and tested different values of the constructive rate – 0.45 and 0.55 (called SACEX where X is 45 arond 55). For all configurations, the value of c_d was set to $1/c_i$.

For SASE, four configurations were used. As for SACE, c_i was fixed at 0.9, and c_d to $1/c_i$ (again, chosen by experimental testing). In the first three configurations of SASE, p_s^t is set to 65%, 75% or 85% (known as SASEX with X being 65, 75, or 85. In the final configuration of SASE, the rate of successful SSC is itself adapted. The rationale is that early in search, more global search is needed, so that the rate of successful SSC should be lower. In later generations, SSC should be encouraged to focus more on exploiting good solutions, so the rate of successful SSC should increase. To implement this, we used the schedule shown in equation 5 to control the rate of successful SSC:

$$p_s^t = \alpha + (\beta - \alpha) \cdot \frac{t}{t_{\max}} \tag{5}$$

where t is the current generation and $t_{\max}$ is the maximum evolution time. In these experiments, α was set to 65% and β to 85%. This setting of SASE is referred as SASES . The initial value of UBSS was set at 0.4 for all self-adaptive schemas.

V. RESULTS AND DISCUSSION

We recorded the performance of all crossover operators using two classic performance metrics, mean best fitness and the proportion of successful runs.

The results for 100 runs of Group$_1$ functions are presented in Table III. It is clear from the success rates that SSC generally out-performs SC (consistent with [1]). SACE, if anything, slightly lowers the performance of SSC, while SASE marginally improves it, consistently so in the case of SASE75 and SASES. Thus at least for these problems, SACE brings little benefit, while SASE out-performs SSC. Equally important, SASES is consistently better than SSC, while reducing the parameter tuning requirement. The best fitness table confirms these conclusions: SACE generally performed worse than SSC, while SASE performed as well, and often better. While SASEX with X=0.75 often gave the best performance, SASES gave comparable performance with a reduced parameter tuning requirement.

Table IV presents the results for Group$_2$ functions strengthen the previous results. SSC always out-performs SC by quite large margins. The comparison of SACE and SSC is mixed, but it is clear that SACE only performs better than SSC

TABLE III
PERFORMANCE METRICS FOR GROUP$_1$ FUNCTIONS

Crossovers	F_1	F_2	F_3	F_4
Percentage of Successful Runs				
SC	43	10	3	2
SSC04	66	34	16	8
SSC05	60	31	17	10
SSC06	62	25	12	4
SACE85	51	16	9	7
SACE90	55	21	5	3
SACE95	63	21	13	6
SACE45	56	26	6	11
SACE55	61	26	6	3
SASE65	61	45	17	11
SASE75	69	41	20	11
SASE85	59	38	17	9
SASES	67	37	20	17
Mean $\pm$ Standard Deviation of Best Fitness				
SC	0.18±0.24	0.30±0.23	0.40±0.37	0.49±0.27
SSC04	0.08±0.13	0.16±0.16	0.20±0.19	0.25±0.29
SSC05	0.09±0.16	0.15±0.15	0.21±0.20	0.28±0.31
SSC06	0.08±0.12	0.20±0.24	0.21±0.19	0.27±0.20
SACE85	0.12±0.19	0.21±0.18	0.29±0.24	0.38±0.32
SACE90	0.09±0.13	0.21±0.18	0.29±0.23	0.30±0.23
SACE95	0.10±0.15	0.18±0.17	0.23±0.19	0.31±0.30
SACE45	0.09±0.15	0.21±0.20	0.28±0.23	0.33±0.33
SACE55	0.10±0.16	0.22±0.20	0.27±0.22	0.33±0.21
SASE65	0.08±0.16	0.13±0.16	0.19±0.19	0.25±0.24
SASE75	0.07±0.13	0.13±0.16	0.19±0.18	0.26±0.24
SASE85	0.07±0.13	0.13±0.15	0.21±0.18	0.26±0.22
SASES	0.08±0.14	0.13±0.15	0.20±0.18	0.24±0.23

TABLE IV
PERFORMANCE METRICS FOR GROUP$_2$ FUNCTIONS

Crossovers	F_5	F_6	F_7	F_8
Percentage of Successful Runs				
SC	42	4	20	14
SSC04	71	24	49	38
SSC05	70	19	48	35
SSC06	72	21	53	34
SACE85	72	13	49	22
SACE90	74	17	48	32
SACE95	79	20	53	34
SACE45	76	15	48	30
SACE55	68	14	52	32
SASE65	87	29	57	43
SASE75	83	28	63	39
SASE85	84	34	63	42
SASES	84	35	57	50
Mean $\pm$ Standard Deviation of Best Fitness				
SC	0.11±0.12	0.27 ±0.14	0.15 ±0.09	0.25 ±0.20
SSC04	0.06 ±0.09	0.16 ±0.13	0.08 ±0.05	0.13 ±0.12
SSC05	0.06 ±0.08	0.17 ±0.15	0.10 ±0.07	0.12 ±0.11
SSC06	0.06 ±0.09	0.18 ±0.14	0.08 ±0.06	0.12 ±0.10
SACE85	0.07 ±0.09	0.22 ±0.16	0.10 ±0.08	0.17 ±0.15
SACE90	0.07 ±0.12	0.21 ±0.19	0.09 ±0.07	0.14 ±0.15
SACE95	0.05 ±0.08	0.17 ±0.12	0.08 ±0.05	0.13 ±0.11
SACE45	0.05 ±0.07	0.20 ±0.13	0.10 ±0.07	0.17 ±0.15
SACE55	0.07 ±0.12	0.21 ±0.15	0.08 ±0.06	0.15 ±0.15
SASE65	0.03 ±0.03	0.14 ±0.11	0.07 ±0.05	0.09 ±0.08
SASE75	0.04 ±0.07	0.16 ±0.12	0.07 ±0.04	0.11 ±0.10
SASE85	0.05 ±0.10	0.15 ±0.14	0.07 ±0.06	0.10 ±0.10
SASES	0.03 ±0.06	0.14 ±0.12	0.07 ±0.04	0.09 ±0.08

TABLE V
PERFORMANCE METRICS FOR GROUP$_3$ FUNCTIONS

Crossovers	F_9	F_{10}	F_{11}	F_{12}
Percentage of Successful Runs				
SC	35	14	0	0
SSC04	42	36	0	0
SSC05	48	25	0	0
SSC06	48	26	0	0
SACE85	79	52	0	0
SACE90	82	47	1	0
SACE95	71	60	1	0
SACE45	81	49	1	0
SACE55	80	47	1	0
SASE65	70	56	0	0
SASE75	68	49	1	0
SASE85	65	44	0	0
SASES	75	58	1	0
Mean $\pm$ Standard Deviation of Best Fitness				
SC	1.67 ±1.72	1.19 ±1.63	3.93 ±1.55	2.04 ±.053
SSC04	1.06 ±1.47	0.62 ±1.07	3.16 ±1.10	1.69 ±0.52
SSC05	1.13 ±1.58	0.71 ±0.59	3.27 ±1.25	1.74 ±0.53
SSC06	0.97 ±1.40	0.61 ±0.51	3.01 ±1.41	1.72 ±0.49
SACE85	0.42±1.09	0.68±1.27	3.46±1.56	1.72±0.57
SACE90	0.37±0.99	0.57±0.76	3.04±1.28	1.72±0.52
SACE95	0.55±1.18	0.39±0.60	2.98±1.16	1.62±0.55
SACE45	0.47±1.14	0.55±0.74	3.14±1.27	1.69±0.53
SACE55	0.44±1.10	0.52 ±0.67	3.13±1.23	1.67 ±0.50
SASE65	0.58 ±1.25	0.36 ±0.55	2.65 ±1.31	1.61 ±0.55
SASE75	0.62 ±1.18	0.47 ±0.62	2.53 ±1.15	1.57 ±0.52
SASE85	0.71 ±1.29	0.43 ±0.50	2.73 ±1.21	1.53 ±0.57
SASES	0.55 ±1.25	0.37±0.59	2.49 ±1.19	1.54 ±0.51

if the parameters are carefully chosen (and not always even then). On the other hand, SASE performs substantially better than SSC, with the margins being greater than for Group$_1$ functions. If X is carefully chosen for the particular problem, SASEX can give the best performance for some functions, but SASES gives clearly the best all-round performance while reducing the requirement for setting parameters.

The results for Group$_3$ functions in Table V are interesting in a number of respects. First, they demonstrate very substantial improvements in performance, both of SSC over SC, and of self adaptation in SSC, whether measured by the success rate or the mean best fitness achieved. Again, the best overall performance comes from SASE, and particularly from SASES. However the results differ from the previous in that SACEX performed substantially better than SSC on F_9 and F_{10}, some settings of X for F_9 giving the overall best performance.

We tested the statistical significance of all differences from SC of mean best fitness in the results in Table III, Table IV, and Table V using the Wilcoxon signed-rank test with a confidence level of 99%. The results were that all methods (SSC with predetermined sensitivities, SACE, and SASE) are significant better than SC in terms of mean best fitness.

VI. CONCLUSION

In this paper, we proposed two simple self-adaptive methods for dynamically adjusting semantic sensitivities in semantic similarity based crossover (SSC). The first method, SACE, was inspired by a similar self-adaptive mechanism for mutation

variance in Evolutionary Strategies (ES). The second method, SASE, aims to ensure that a certain proportion of SSC can successfully exchange semantically similar subtrees. The two methods were tested on twelve problems with a range of domains and forms of target functions. The results were compared with SSC with predetermined sensitivities, and with standard crossover (SC).

The results yield two important conclusions. First, they extend the results of [1] to a wider range of functions, confirming the significant and substantial performance improvement of SSC over SC, independent of problem domain or form of target function. SSC is definitely worth the effort. Second, they show a particular form of self-adaption (SASE, particularly SASES), can yield further performance gains, while (again, especially in the case of SASES) reducing the number of tuning parameters. The performance gains are particularly marked for more difficult functions.

Overall, this further confirms the importance of semantic locality[1] in genetic operators in GP, as this is the main effect of SASE.

In the near future, we aim to further characterise the effects of the parameters introduced in SACE and SASE. Having now confirmed the importance of self-adaptation of semantic sensitivity, we also plan to investigate more sophisticated self-adaptation techniques from the literature in further work.

ACKNOWLEDGMENT

This paper was funded under a Postgraduate Scholarship from the Irish Research Council for Science Engineering and Technology (IRCSET). The Institute for Computer Technology at Seoul National University provided some research facilities for this study.

REFERENCES

[1] N. Q. Uy, M. O'Neill, N. X. Hoai, B. McKay, and E. G. Lopez, "Semantic similarity based crossover in GP: The case for real-valued function regression," in *Evolution Artificielle, 9th International Conference*, ser. Lecture Notes in Computer Science, P. Collet, Ed., October 2009, pp. 13–24.

[2] J. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*. MA: MIT Press, 1992.

[3] W. B. Langdon and R. Poli, *Foundations of Genetic Programming*. Berlin: Springer, 2002.

[4] 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). [Online]. Available: http://www.gp-field-guide.org.uk

[5] R. Poli and W. L. N. McPhee, *A Field Guide to Genetic Programming*. http://lulu.com, 2008.

[6] 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.

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

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

[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*. Springer Verlag, April 2005, pp. 34–45.

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

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

[12] 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*, vol. 4963, pp. 141–156, 2008.

[13] ——, "Genetic programming and model checking: Synthesizing new mutual exclusion algorithms," *Automated Technology for Verification and Analysis, Lecture Notes in Computer Science*, vol. 5311, pp. 33–47, 2008.

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

[15] N. McPhee, B. Ohs, and T. Hutchison, "Semantic building blocks in genetic programming," in *Proceedings of 11th European Conference on Genetic Programming*. Springer, 2008, pp. 134–145.

[16] N. Q. Uy, N. X. Hoai, and M. O'Neill, "Semantic aware crossover for genetic programming: the case for real-valued function regression," in *Proceedings of EuroGP09*. Springer, April 2009, pp. 292–302.

[17] C. Johnson, "Genetic programming with guaranteed constraints," in *Recent Advances in Soft Computing*. The Nottingham Trent University, 2002, pp. 134–140.

[18] R. E.Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, 1986.

[19] L. Beadle and C. Johnson, "Semantically driven mutation in genetic programming," in *2009 IEEE Congress on Evolutionary Computation*, A. Tyrrell, Ed., IEEE Computational Intelligence Society. Trondheim, Norway: IEEE Press, 18-21 May 2009, pp. 1336–1342.

[20] ——, "Semantic analysis of program initialisation in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 10, no. 3, pp. 307–337, Sep 2009.

[21] K. Krawiec and P. Lichocki, "Approximating geometric crossover in semantic space," in *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, F. Rothlauf, Ed. ACM, 2009, pp. 987–994.

[22] P. J. Angeline, "Adaptive and self-adaptive evolutionary computations," in *Computational Intelligence: A Dynamic Systems Perspective*, M. Palaniswami and Y. Attikiouzel, Eds. IEEE Press, 1995, pp. 152–163.

[23] J. P. Rosca and D. H. Ballard, "Genetic programming with adaptive representations," University of Rochester, Computer Science Department, Rochester, NY, USA, Tech. Rep. TR 489, Feb 1994.

[24] I. Rechenberg, *Evolutionsstrategie*, ser. problemata. Stuttgart: Friedrich Frommann Verlag (Günther Holzboog KG), 1973, vol. 15.

[25] R. Harper and A. Blair, "A self-selecting crossover operator," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, G. G. Yen, L. Wang, P. Bonissone, and S. M. Lucas, Eds. Vancouver: IEEE Press, 6-21 Jul. 2006, pp. 5569–5576.

[26] M. O'Neill, "Automatic programming in an arbitrary language: Evolving programs with grammatical evolution," Ph.D. dissertation, University Of Limerick, Ireland, August 2001.

[27] P. J. Angeline, "Two self-adaptive crossover operators for genetic programming," in *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. Cambridge, MA, USA: MIT Press, 1996, ch. 5, pp. 89–110.

[28] A. E. Eiben, E. Marchiori, and V. A. Valkó, "Evolutionary algorithms with on-the-fly population size adjustment," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. LNCS, vol. 3242. Springer-Verlag, Sep 2004, pp. 41–50.

[29] T. Hu and W. Banzhaf, "The role of population size in rate of evolution in genetic programming," in *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, ser. LNCS, vol. 5481. Tuebingen: Springer, Apr. 15-17 2009, pp. 85–96.

[30] S. Silva and S. Dignum, "Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP," in *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, ser. LNCS, vol. 5481. Tuebingen: Springer, April 2009, pp. 159–170.

[31] H.-P. Schwefel, *Numerical optimization of Computer models*. Chichester: John Wiley & Sons, Ltd., 1981.

[1] Ensuring that operators make only a small bounded change in semantics

[32] M. Keijzer, “Improving symbolic regression with interval arithmetic and linear scaling,” in *Proceedings of EuroGP’2003*. Springer-Verlag, April 2003, pp. 70–82.

[33] C. Johnson, “Genetic programming crossover: Does it cross over?” in *Proceedings of the 12th European Conference on Genetic Programming (EuroGP2009)*. Springer, 2009, pp. 97–108.