# Semantics Based Crossover for Boolean Problems

Nguyen Quang Uy
Natural Computing Research & Application Group
University College Dublin
Belfield, Dublin 4, Ireland
quanguyhn@gmail.com

Nguyen Xuan Hoai
School of Information Technology
Military Technical Academy
Hanoi, Vietnam
nxhoai@gmail.com

Michael O'Neill
Natural Computing Research & Application Group
University College Dublin
Belfield, Dublin 4, Ireland
m.oneil@ucd.ie

Bob McKay
School of Computer Science and Engineering
Seoul National University
Seoul, Korea
rimsnucse@gmail.com

## ABSTRACT

This paper investigates the role of semantic diversity and locality of crossover operators in Genetic Programming (GP) for Boolean problems. We propose methods for measuring and storing semantics of subtrees in Boolean domains using Trace Semantics, and design several new crossovers on this basis. They can be categorised into two classes depending on their purposes: promoting semantic diversity or improving semantic locality. We test the operators on several well-known Boolean problems, comparing them with Standard GP Crossovers and with the Semantic Driven Crossover of Beadle and Johnson. The experimental results show the positive effects both of promoting semantic diversity, and of improving semantic locality, in crossover operators. They also show that the latter has a greater positive effect on GP performance than the former.

## Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

## General Terms

Algorithms, Representation, Experimentation

## Keywords

Genetic Programming, Trace Semantics, Crossover Operators, Boolean Problems

## 1. INTRODUCTION

Genetic Programming (GP) is an evolutionary paradigm, inspired by biological evolution, for finding solutions (in the form of a program) to a user-defined task [19, 21, 26]. The program is usually presented through a syntactic formalism such as s-expression trees [19], a linear sequence of instructions, grammars, or graphs [26]. The genetic operators in such GP systems are usually designed to ensure the syntactic closure property, i.e., to produce syntactically valid children from any syntactically valid parent(s). Using purely syntax-based genetic operators, GP evolutionary search is conducted on the syntactical space of programs, with the only semantic guidance coming from the fitness of individuals.

Although GP has demonstrated its effectiveness in solving a range of problems, the limitation to (finite) behavior-based semantic guidance and pure syntactic genetic operators, is somewhat alien to the practice of human programmers. Computer programs are constrained not only by syntax, but also by semantics. In programming practice, any change to a program should pay careful attention to the change in program semantics. Thus a number of researchers have proposed a wide variety of semantically based methods for controlling the genetic operators [5, 6, 32, 14, 15, 16, 18, 17, 1, 23, 30, 31].

One useful piece of semantic information is the scale of change. Beadle and Johnson [1] showed, for Boolean problems, that eliminating crossover between semantically equivalent subtrees could substantially improve the computational efficiency of GP. Subsequently, Uy et al [31] showed that information about the scale of semantic difference between crossover candidates could be even more useful: limiting semantic change to an intermediate range further improved performance. Potentially, though, discrete domains could show a different behaviour: the benefits of an intermediate semantic change might disappear with a quantised fitness function.

A second potentially useful piece of information is the resulting change in semantics at parent nodes. Even when the exchanged children are semantically different, this might not necessarily result in a change of semantics of the parent nodes after crossover. In this work, we propose two new operators which control the extent of this change directly, rather than relying on the difference in semantics of the children. We study the performance of these new operators on


*GECCO'10,* July 7–11, 2010, Portland, Oregon, USA.

Boolean domains, comparing them with operators more directly analogous to those reported in [1, 31].

The remainder of the paper is structured as follows. In section 2, we give a review of related work using semantics in GP. Section 3 details the Boolean semantic measure and attribute-based representation, and introduces the new crossover operators. Experimental settings are described in section 4. The results of the experiments are presented and discussed in section 5. Section 6 concludes the paper and highlights potential future work.

## 2. RELATED WORK

The use of semantics in GP has been the subject of increasing attention. While the precise meaning attached to "semantics" may vary from field to field, in GP it has generally meant the use of semantic information to provide additional guidance to the GP search. There are at least three ways in which semantics can be represented, extracted and used to guide GP:

1. using grammars incorporating semantic information about the domain [32, 5, 6]
2. through semantics-based formal methods [14, 15, 16, 18, 17]
3. based on GP tree-like structures [1, 23, 30, 31].

The most popular form of the first uses attribute grammars. GP individuals expressed in the form of attribute grammar trees can incorporate semantic information, which can be used to eliminate bad individuals from the population [6] or to prevent the generation of semantically invalid individuals as in [32, 5]. However, these attribute grammars are generally designed for a specific problem, and so not readily extended to other problems.

Recently, Johnson has advocated the use of formal methods as a means to incoporate semantic information in GP [14, 15, 16]. In this work, semantic information is extracted by formal methods (e.g., abstract interpretation or model checking) and used to measure individual fitness in problems where more traditional sample point fitness is difficult to use. Katz et al. used model checking to solve the Mutual Exclusion problem [18, 17]; semantics were also used to calculate individual fitnesses.

In expression tree GP, semantic information has been incorporated mainly through modification of the crossover operator. At first, this focused on the syntax and structure of individuals – in [13], the authors modified the standard subtree crossover operator to take into account the depth of trees. Others modified crossover based on awareness of the shape of individuals [24]. More recently, context has been used as additional information in determining GP crossover points [11, 29, 22]. However these methods generally pay a high computational cost in evaluating the context of every subtree of every individual in the population.

Beadle and Johnson [1] investigated the direct use of semantic information to guide GP crossover in Boolean domains. Their crossover, Semantically Driven Crossover (SDC), checks the semantic equivalence between offspring and parents by transforming Boolean expression trees to Reduced Ordered Binary Decision Diagrams (ROBDDs) [7]. Two trees are semantically equivalent exactly when they reduce to the same ROBDD. This controls the survival of children in the crossover operation: if they are semantically equivalent to their parents, they are discarded and the crossover is restarted; this is repeated until semantically new children are found. This increases semantic diversity and consequently improves performance. Subsequent papers [3, 2] extended the approach to GP mutation and initialisation.

McPhee et al. [23] take a different approach to extracting semantic information from a Boolean expression tree: enumerating all possible inputs. They evaluate (and measure) two different aspects: the semantics of subtrees and the semantics of context. They particularly emphasised fixed-semantic contexts, where replacing one subtree by another does not change the semantics. They showed that as tree size increases during evolution, many such fixed-semantic contexts may arise; thus it becomes increasingly difficult to change the semantics using standard crossover and mutation operators.

Krawiec [20] proposed a way to measure the semantics of an individual based on fitness cases, using it to guide crossover (*Approximating Geometric Crossover* - AGC).

Uy et al. [31] proposed two new crossover operators, Semantics Aware Crossover (SAC) and Semantic Similarity based Crossover (SSC); the former eliminates crossovers resulting in small semantic change, while the latter additionally eliminates those resulting in large change, confining semantic change to an intermediate range.

Variants of most of these semantically-based crossover operators have been tested on continuous domains, with SSC giving the best overall performance, while only a few have been tested on discrete domains, with the record so far going to SDC. The aim of this paper is to compare the performance of our new crossover operators, both with more standard syntactically-based operators, and with some of the newer semantically-based operators, on Boolean domains. In doing so, we also provide a more detailed and direct comparison of these operators on Boolean domains than has previously been available..

## 3. METHODS

### 3.1 Measuring Semantics

The exact meaning of "semantics" varies from field to field, though there is always a common element contrasting the semantics (or meaning) of a syntactic expression with its surface syntax. In programming, the definition is more tightly defined: semantics is the relationship between the sequence of inputs to a program and the corresponding sequence of outputs. In GP, we generally restrict still further, requiring a program to have only a single (albeit possibly complex) input and a single corresponding output, so that the semantics becomes a simple mathematical relation – and in the commonest case, of deterministic programs, a mathematical function. This is the meaning we will attach to "semantics" here.

There are two primary ways we can define such a function, and correspondingly two kinds of semantics. An intensional definition (thus intensional semantics) defines a function by relating it to simpler previously-defined functions. This is the form of semantics underlying formal methods in programming, as for example in the work of Beadle and Johnson [1]. More common in GP, though, is extensional definition (and extensional semantics), in which a function

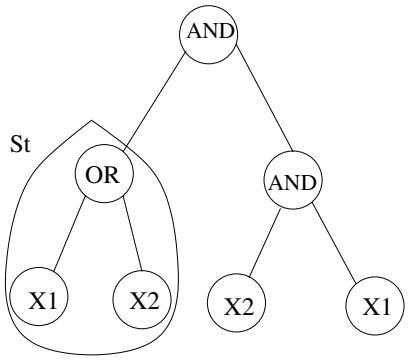


**Figure 1: Example for Trace Semantics**

is defined through listing its set of input-output pairs (or in many cases, a finite approximation of this set).

In this paper, we use the simplest possible extensional semantic measure, namely the input-output relation defined by the fitness cases as inputs. For concreteness, we call this semantics the *Trace Semantics* (TS). This definition of semantics goes back at least to Poli and Page [27], who however only used it for full trees; more recently, McPhee et al. [23] used it to also define the semantics of subtrees. Formally, given a particular GP problem with a pre-defined set of fitness cases, the *Trace Semantics* of a (sub)tree is defined as follows:

Let $P = \{p_1, p_2, ..., p_N\}$ be the fitness cases of the problem in domain $D$, and let $F$ be the function expressed by a (sub)tree $T$ on $D$. Then the *Trace Semantics* of $T$ relative to $P$ on domain $D$ is the set $S = \{(p_1, s_1), (p_2, s_2), ...(p_N, , s_N\}$ where $s_i = F(p_i), i = 1, 2, ..., N$.

Consider subtree $St$ in figure 1. Assume that the problem has only two input variables $(X_1, X_2)$, so there are four fitness cases $P = \{p_1, p_2, p_3, p_4\}$ with $p_1 = \{0, 0\}$, $p_2 = \{0, 1\}$, $p_3 = \{1, 0\}$, $p_4 = \{1, 1\}$. Thus the trace semantics of $St$, since it computes OR, is $S = \{(p_1, s_1), (p_2, s_2), (p_3, s_3), (p_4, s_4)\}$ $=\{((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 1)\}$.

For the remainder of this paper, we will assume a canonical ordering of the possible inputs (for example, for Boolean domains we might use lexicographical ordering, as for $P$ above), so that we can abbreviate the TS to the sequence (0,1,1,1).

Trace semantics is also closely related to the sampling semantics of Uy et al. [31] for real-valued problems. Sampling semantics differs in using an independent set of test points sampled from the domain, rather than the test cases.

We follow [31] in defining the semantic distance between two subtrees as the sum of the absolute distances between their outputs over all fitness cases, and in defining relations of semantic equivalence and similarity based on this distance. We use the Hamming distance as the underlying distance metric. Let $U = \{(p_1, u_1), (p_2, u_2), ..., (p_N, u_N)\}$ and $V = \{(p_1, v_1), (p_2, v_2), ..., (p_N, v_N)\}$ be the TS of (sub)trees $St_1$ and $St_2$. Then the *Trace Semantics Distance* (TSD) between $St_1$ and $St_2$ is:

$$TSD(St_1, St_2) = |u_1 - v_1| + |u_2 - v_2| + .... + |u_N - v_N|$$

Two subtrees are said to be *Semantically Equivalent* (SE) if they have the same TS (or in other words, their TSD is zero).

They are said to be semantically similar (SS) if their TSD

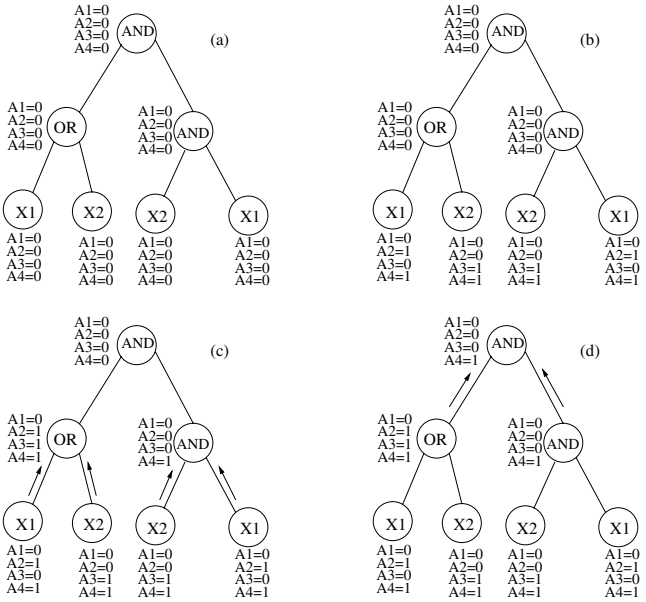


**Figure 2: An individual in AGP and the process of evaluating the value of its attributes.**

lies within a predefined range. That is:

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

where $\alpha$ and $\beta$ are predefined constants, known as the *lower* and *upper* bounds for semantic sensitivity. Uy et al. used a small nonzero value for the lower bound to exclude close numerical approximation. In a discrete domain, this seems unnecessary, so we set the lower bound to zero, investigating the effects of a range of values for the upper bound.

### 3.1.1 Attribute-based Representation

Repeatedly computing semantics of subtrees may, in general, be costly. To avoid this, we store the trace semantics of each subtree directly in the root node of that subtree, as a sequence of attribute values (corresponding to the sequence of inputs). In Figure 2, four attributes $(A_1, A_2, A_3, A4)$ are used to represent the TS of a problem with only two input variables. These attributes are evaluated bottom-up, as shown in figure 2, in which the values are initialised to zero (subfigure a), then propagated upwards (subfigures b, c, d).

In doing so, we are trading off time for space. How much? Let $P$ be the population size, each individual having an average of $S$ nodes, with $k$ Boolean variables. If all Boolean combinations are used as fitness cases, the memory cost will be $O(P * S * 2^k)$. In general, for small Boolean problems, this is affordable. For larger problems, computational cost will preclude using all Boolean combinations as fitness cases. If $N$ cases are used, the memory cost will be $O(P * S * N)$, which is again generally affordable.

## 3.2 Diversity Promoting Semantic Crossovers

It is widely recognised that maintaining high diversity is very important in GP [10]. Higher diversity helps GP to

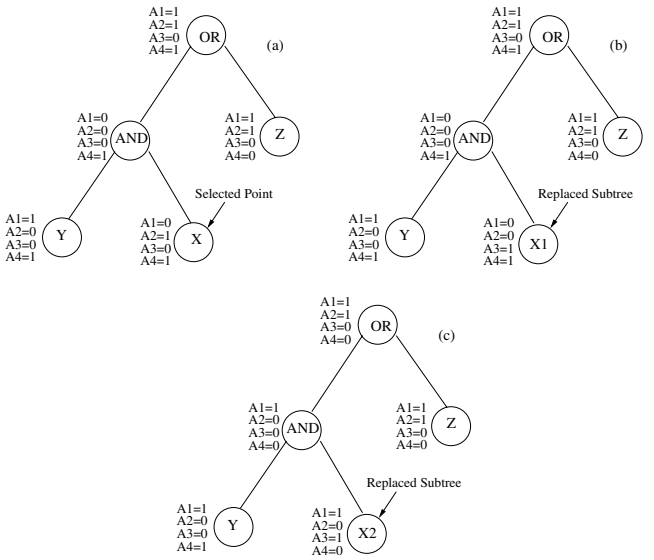


**Figure 3: An individual in AGP and the process of evaluating the value of its attributes.**

find new solutions through exploration, and thus improve its performance [4], while a rapid loss of diversity can cause premature convergence to local optima [8, 25]. Thus for example, the Semantics Aware Crossover (SAC) of [30] aborts crossovers that exchange semantically equivalent subtrees, with the aim of promoting diversity. Whenever two subtrees are chosen for crossover, they are tested for trace semantic equivalence; if the test succeeds, the crossover is aborted and new candidates are selected.

However SAC cannot guarantee to produce children semantically different from their parents. For example, consider figure 3(a), showing part of a GP tree with attached semantic attributes ($X$, $Y$ and $Z$ represent whole subtrees, not merely single nodes). Assume that $X$ is the point selected for crossover. Suppose that it is replace by subtree $X1$, with the TS shown in 3(b). Although the TS of $X$ and $X1$ differ, the TS at the parent 'AND' node does not change, nor does that at the root 'OR' node. However when $X$ is replaced with $X2$, the TS of the ancestors does change.

Thus the form of fixed semantics investigated in [23] is not the only cause of crossover failing to change semantics. The replacement might simply not change the right components of the semantics for change to propagate to the root. To remedy this, we propose a crossover operator known as *Guaranteed Change Semantic Crossover* (GCSC). The objective of GCSC is to guarantee a change in the semantics of the children in the new population. The detail of GCSC is presented in algorithm 1.

How can we efficiently compute whether two crossover points are change-inducing? Before we try to exchange $X$ and $X1$ in figure 3 (a, b), we compute which attributes differ (in this case, forming the set $\{A2, A3\}$. We then propagate this upward, discovering that at the parent 'AND' node, since Y guarantees that A2 and A3 will be 0, the root difference set $R$ becomes the empty set, $\phi$. Therefore, this crossover is not change-inducing, so it is aborted. If we try to exchange $X$ and $X2$ in figure 3 (a, c), the difference set is $\{A1, A2, A3, A4\}$, which propagates to the 'AND' as $\{A1, A4\}$, and then to the 'OR' as $R = \{A4\}$. Since the root change-set is nonempty, this crossover is change-inducing.

**Algorithm 1**: Guaranteed Change Semantic 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;
    if Subtree1 and Subtree2 are change-inducing
    then
        execute crossover;
        add the children to the new population;
        return true;
    else
        Count=Count+1;
if Count=Max_Trial then
    choose a random crossover point Subtree1 in P1;
    choose a random crossover point Subtree2 in P2;
    execute crossover;
    return true;
```

The objective of GCSC is similar to that of SDC [1]: to guarantee change. The differences lie in the use of extensional rather than intensional semantics, and in the checking of only a single child.

## 3.3 Locality Promoting Semantic Crossovers

The two crossover operators in the previous subsection focus on promoting semantic diversity. In Evolutionary Computation (and GP in particular), locality (a small change in genotype leading to a small change in phenotype) also plays a crucial role in algorithm efficiency [9, 12, 28]. However, most current GP representations and operators have low locality: small (syntactic) variations in individuals can cause large changes in semantics.

Uy et al [31] introduced SSC as an extension of SAC with controlled semantic step size. In SSC, the crossover is aborted not only when the two children are semantically too similar, but also when they are too dissimilar. Two parents are randomly selected and a subtree in each parent is stochastically chosen. The two subtrees are checked for semantic similarity. If they are similar, the crossover is executed by exchanging these subtrees and the children are added to the next generation. If they are not, SSC uses multiple trials to find a semantically similar pairs, only reverting to random selection after passing a bound on the number of trials.

Here we propose a variant, *Locality Controlled Semantic Crossover* (LCSC). LCSC is an extension of GCSC, bearing the same relation to it as SSC does to SAC. LCSC is implemented in the same way as GCSC, propagating the difference set of changed attributes upward in the tree. However when the root is reached, the crossover is aborted not only if the difference set is empty, but also if it is too large. In other words, LCSC is only executed if $0 < |\mathrm{R}| \leq \epsilon$, where $R$ is the difference set at the root, and $\epsilon$ is a predefined constant called the semantic sensitivity. In this paper, the value of $\epsilon$ is one of the experimental parameters.

**Table 1: Run and Evolutionary Parameter Values.**

| Parameter | Value |
|---|---|
| Population size | 200 |
| Generations | 50 |
| Selection | Tournament |
| Tournament size | 3 |
| Crossover probability | 0.9 |
| Mutation probability | 0.1 |
| Initial Max depth | 6 |
| Max depth | 15 |
| Max depth of mutation tree | 5 |
| Function sets | AND, OR, NAND, XOR |
| Terminals | $X_1, X_2, ..., X_N$ |
| # Fitness cases NFIT | |
| 6MUX | 64 |
| 5PAR | 32 |
| 6PAR | 64 |
| 5MAJ | 32 |
| Semantic Sensitivity | |
| SSC2 | NFIT/2 |
| SSC3 | NFIT/3 |
| SSC4 | NFIT/4 |
| LCSC4 | 4 |
| LCSC8 | 8 |
| LCSC12 | 12 |
| Fitness | total number of error bits overall fitness cases |
| Success | Zero errors |
| Trials per treatment | 100 independent runs for each value |

**Table 2: Percentage of Successful Runs**

| Xovers | 5PAR | 5MAJ | 6MUX | 6PAR |
|---|---|---|---|---|
| SC | 48 | 35 | 13 | 14 |
| SDC | 58 | 64 | 13 | 20 |
| SAC | 63 | 43 | 13 | 23 |
| GCSC | 65 | 68 | 14 | 26 |
| SSC2 | 63 | 48 | 16 | 26 |
| SSC3 | 68 | 49 | 21 | 23 |
| SSC4 | 58 | 42 | 10 | 22 |
| LCSC4 | 75 | 62 | 30 | 25 |
| LCSC8 | 76 | 69 | 32 | 33 |
| LCSC12 | 73 | 74 | 25 | 29 |

**Table 3: Mean and Standard Derivation of Best Fitness**

| Xovers | 5PAR | 5MAJ | 6MUX | 6PAR |
|---|---|---|---|---|
| SC | 1.02±1.22 | 1.01±0.93 | 3.96±2.69 | 5.76±3.53 |
| SDC | *0.61±0.81* | *0.45±0.62* | 3.33±2.27 | *4.76±3.67* |
| SAC | *0.60±0.88* | 0.82±0.85 | 3.84±2.37 | 5.03±3.68 |
| GCSC | *0.49±0.75* | *0.36±0.58* | 3.50±2.25 | *4.47±3.93* |
| SSC2 | 0.70±1.07 | *0.63±0.69* | 3.31±2.47 | *4.66±3.85* |
| SSC3 | *0.44±0.70* | *0.59±0.70* | *2.96±2.35* | *4.92±3.73* |
| SSC4 | 0.69±0.95 | 0.76±0.75 | 3.89±2.49 | *4.78±3.65* |
| LCSC4 | *0.34±0.65* | *0.44±0.60* | *2.83±2.79* | *4.67±3.81* |
| LCSC8 | *0.39±0.79* | *0.35±0.55* | *2.52±2.51* | *3.87±3.33* |
| LCSC12 | *0.43±0.78* | *0.27±0.46* | *2.79±2.23* | *4.03±3.34* |

# 4. EXPERIMENTAL SETTINGS

We investigated the effects of the new crossover operators GCSC and LCSC, comparing them with SC , SDC, SAC and SSC. We trialed all operators on four test-bed Boolean problems: 5 bit parity, 5 bit majority, 6 bit multiplexer, and 6 bit parity.

### *The 6-bit multiplexer problem (6MUX).*

(6MUX) interprets the two control bits $\{A_0, A_1\}$ as an address with which to choose the correct input bit from the binary input lines $\{D_0, D_1, D_2, D_3\}$ as its output.

### *The even 5- and 6-bit parity problems (5PAR and 6PAR).*

These problems take respectively 5 and 6 bits as input, returning true (1) if and only if an even number of the inputs are true (1).

### *The 5 majority problem (5MAJ).*

This problem takes 5 bits as input, and returns true(1) if and only if, the majority of the inputs are true(1).

In these (minimising) problems, the fitness of an individual is the number of error bits, not matching the target function. A run is considered successful if it finds an individual with no error (i.e. fitness zero).

The function set for all problems was {AND, OR, NAND, XOR} and the terminal set was $\{X_1, X_2, ..., X_N\}$, where $N$ is the number of input variables. The main GP parameters used for our experiments are given in Table 1. Although these experiments were purely concerned with crossover, the system also incorporated (a low rate of) mutation, because we wanted to study the behaviour of crossover in the context of a normal GP run.

The experimental settings for all GP systems are given in Table 1. The maximum number of trials permitted to select satisfied subtrees for SD, SAC, GCSC, SSC and LCSC was set at 20.[1]

The semantic sensitivities for SSC was set as varying proportions of the number of fitness cases NFIT. Because LCSC testing is more exhaustive than SSC, it is less likely to generate change for a given fitness case; thus semantic sensitivity levels need to be set correspondingly lower. We used NFIT/N, with $N = 2, 3, 4$ for SSC and $4, 8, 12$ bits for LCSC (the corresponding treatments being denoted SSCN and LCSCN).

# 5. RESULTS AND DISCUSSION

For all treatments, we recorded two classic performance metrics, the percentage of successful runs (table 2) and the mean best fitness (table 3). We tested the statistical significance of all differences from SC in the results in table 3, using the Wilcoxon signed-rank test with a confidence level

[1][1] in SDC repeated attempts at crossover until semantically different children were found. However the effect of continuing beyond 20 attempts is small, while the computational cost can be significant. For better comparability, we used the same bound as for the other algorithms.

of 95%. If a crossover operator is significantly better than SC, the result is printed in italic face.

## 5.1 Effects of Promoting Diversity

From [30], we would expect SAC to slightly out-perform SC, and indeed this is what we see in tables 2 and 3, but the differences are significant only on 5PAR. Similarly, from [1] we expect to see SDC more substantially out-perform SC, and again this is what we see, with all differences except 6MUX being significant at the 95% confidence level. GCSC being an extensional analogue of (intensional) SDC, we would expect to see similar performance from it – and again we do. It is possible that GCSC performance is slightly better than SDC (though not significantly); if so, it is likely to result from the small differences between the two comparison algorithms that we mentioned earlier.

In general, the results in this section provide strong evidence for the value of crossover operators promoting semantic diversity, with SDC and GCSC performing the best.

## 5.2 Effects of Improving Locality

From [31], we might anticipate that improving semantic locality of crossover operators through SSC might further enhance GP performance by comparison with SAC. The results in both tables confirm this: for every problem, at least one level of SSC performed better than SAC, the differences being quite substantial in most cases. However this improved locality was not always enough to compensate for the enhanced diversity promotion of SDC and GCSC, so that the comparison between them and SSC was mixed.

However when we combined diversity and locality promotion in LCSC, we saw the largest improvements: on every problem, the best performance came from some setting of LCSC (though the best setting varied with the problem). This is particularly clear in table 2, where the improvements in success rates are generally substantial.

It is particularly worth noting that, with the hardest problem (6MUX), the diversity-promoting mechanisms made only small improvements, especially in success rates, while LCSC was able to make substantial improvements in both mean best fitness and success rate performance measures.

# 6. CONCLUSIONS AND FUTURE WORK

In this paper, we compared a number of different semantically-based crossover operators that have been previously proposed by various authors (SAC, SDC and SSC), and added two of our own (GCSC and LCSC). We can classify these operators in two ways. We can classify them according to whether they add only semantic diversity control to SC (SAC, SDC, GCLC) or also incorporate locality control (SSC, LCSC). We can also classify them accordingly as they measure the semantic effect only at the point of insertion (SAC, SSC) or at the root (SDC, GCLC,SCLC). Overall, it seems clear from the results that supporting semantic diversity is valuable, but that additionally supporting semantic locality is even more so. It also seems clear that it is worth the extra effort to evaluate these semantic changes at the root, not just at the point of insertion.

A number of directions for further research flow from this paper.

Firstly, for SSC and LCSC, semantic sensitivity is clearly a very important parameter, and not particularly robust (i.e. changes in sensitivity lead to large changes in performance). Thus we either need a good understanding of how to set the semantic sensitivity for a particular problem, or we need ways to set it automatically through parameter self-adaption techniques.

One issue, particularly with GCSC and LCSC, is computational cost. We can either pay a time cost to compute the necessary semantic values, or else (as here) a memory cost to cache them. Two potential alternatives present themselves:

- using only a subset of the fitness cases to compute trace semantics, thus removing the exponential cost of increases in the number of variables
- using compression methods to reduce the size of the stored TSs

Finally, we need to further analyse the effects of these operators on other aspects of GP runs. In particular, we plan to measure the effects on locality and on code bloat in future work.

# Acknowledgment

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

# 7. REFERENCES

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

[2] 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.

[3] 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.

[4] E. K. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Is increased diversity in genetic programming beneficial? an analysis of the effects on performance. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1398–1405, Canberra, December 2003. IEEE Press.

[5] 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.

[6] 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.

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

[8] C. Gathercole and P. Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 291–296, Stanford University, CA, USA, July 1996. MIT Press.

[9] 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*, pages 283 – 290. ACM, 2000.

[10] S. Gustafson. *An Analysis of Diversity in Genetic Programming*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, England, February 2004.

[11] 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.

[12] 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.

[13] 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.

[14] C. Johnson. Deriving genetic programming fitness properties by static analysis. In *Proceedings of the 4th European Conference on Genetic Programming (EuroGP 2002)*, pages 299–308. Springer, 2002.

[15] C. Johnson. What can automatic programming learn from theoretical computer science. In *Proceedings of the UK Workshop on Computational Intelligence*. University of Birmingham, 2002.

[16] C. Johnson. Genetic programming with fitness based on model checking. In *Proceedings of the 10th European Conference on Genetic Programming (EuroGP 2002)*, pages 114–124. Springer, 2007.

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] R. Poli and W. B. Langdon. On the search properties of different crossover operators in genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 293–301, University of Wisconsin, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann.

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

[27] R. Poli and J. Page. Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code gp and demes. *Genetic Programming and Evolvable Machines*, 1(1):37–56, 04 2000.

[28] F. Rothlauf and D. Goldberg. Redundant Representations in Evolutionary Algorithms. *Evolutionary Computation*, 11(4):381–415, 2003.

[29] W. A. Tackett. *Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, USA, 1994.

[30] 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 EuroGP 2009*, pages 292–302. Springer, April 2009.

[31] 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 P. Collet, editor, *Evolution Artificielle, 9th International Conference*, Lecture Notes in Computer Science, pages 13–24, October 2009.

[32] 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.