2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

A Self-Selecting Crossover Operator

Robin Harper and Alan Blair, *Member, IEEE*

Abstract - This paper compares the efficacy of different crossover operators for Grammatical Evolution across a typical numeric regression problem and a typical data classification problem. Grammatical Evolution is an extension of Genetic Programming, in that it is an algorithm for evolving complete programs in an arbitrary language. Each of the two main crossover operators struggles (for different reasons) to achieve 100% correct solutions. A mechanism is proposed, allowing the evolutionary algorithm to self-select the type of crossover utilised and this is shown to improve the rate of generating 100% successful solutions.

I. Introduction

Both genetic algorithms (GA) and genetic programming (GP) stress the role of the crossover operator in searching the problem space. There are a large number of variations on the crossover that can be performed in GA and variation in the choice and frequency of the subtree crossover points in GP. There has been debate over which crossover operator is *better* or if, say, one type is better for small populations and another better with more diverged populations [2]. We believe the question that arises is: whether certain crossover types are more effective in solving different types of problems and whether such operators might be more effectively utilised at different stages of the problem solving attempt. If this is the case then, given that we must assume we have no *a priori* knowledge of the specific task domain, is there a way we can use the evolutionary algorithm to select the best crossover for the problem at hand?

This paper discusses the previous work on self-adaptive crossover operators in GA and GP. It then recaps on how Grammatical Evolution (GE) extends GP to allow the use of radically different crossover operators rather than variations on location and number of sub-tree crossover points. It then describes the numeric regression domain and the classification problem used to illustrate the different results obtained from the two crossover operators analysed in this paper and demonstrates how allowing both crossover operators to be used provides better results than using only one of these two crossover operators.

II. Previous Work

Angeline [3] separates adaptive evolutionary techniques (which he distinguishes from absolute update rules) into three classifications: (i) *population-level* techniques which dynamically adjust parameters that are global to the entire population; (ii) *individual-level* adaptive methods which modify the way an individual itself is treated by the system (e.g. its mutation rate or method of crossover); and (iii) *component-level* adaptive methods which alter one or more of the components of an individual.

A. Population-Level Adaptation

Population-level techniques utilise a higher-level analysis of the overall population in an attempt to alter and/or guide the evolutionary process. In other words an analysis of the population as a whole is used to adapt the crossover technique. For instance [4] describes a system where the statistical distribution of alleles in the population is used to adjust the swapping probability of each gene locus. Similarly [5] describes a population-level genetic programming method that utilises statistics gathered over all subtrees in the population to determine crossover points that are likely to be more advantageous.

However, given that the population can be thought of as a parallel search of the solution space, it could be argued that there is no need, in fact it may be incorrect, to assume that any adaptive technique should be global to the population. For this reason, as well as for the reason that it is closer to the biological method that ultimately inspired the evolutionary algorithm, this paper focuses on individual-level adaptive methods.

B. Individual-Level Adaptation

Several papers have analysed individual level adaptation in both GA and GP.

1) Individual-level adaptation in GA

There have been a number of papers relating to individual level adaptation, perhaps the most relevant to this paper being [6]. In [6] Spears appended to each individual a one-bit tag that allowed the individual to choose between a two-point and a uniform crossover in a GA system. As discussed by Spears the types of crossover represented by two-point crossover and uniform crossover are radically different in their effect on the GA representation. With two point crossover building blocks may well be maintained, whereas with uniform crossover building blocks will almost certainly be disrupted unless the populations are highly converged. Spears noted a difference between the two-point and the uniform crossover operators not so much between the two problems examined, but rather on the size of the populations and the level of redundant bits utilised in different runs. This is primarily down to the fact that for half of the problems 90% of the bits were redundant, only the first 10% having any impact on the solution. With traditional two-point crossover in such a system most crossovers (90% of all

Robin Harper and Alan Blair are with the School of Computer Science and Engineering, University of New South Wales, Sydney, 2052, Australia and the ARC Centre of Excellence for Autonomous Systems. Email: {robinh,blair}@cse.unsw.edu.au

0-7803-9487-9/06/$20.00/©2006 IEEE

crossovers) would have no impact. For this reason we believe that the impact of the self-adaptation he implemented was somewhat muted. In addition when he removed the self-adaptation relating to the different crossover points and instead applied them randomly, largely the same benefits were observed. Spears paper is geared towards GA. Although the questions explored by Spears are similar to those discussed in this paper, the representation of the problems chosen by him were such that, arguably, the two point operator's effectiveness was highly constrained by the unlikelihood of a crossover occurring in the active part of the genome.

2) Individual-level adaptation in GP

With respect to previous individual-level adaptation in GP, Angeline [7] introduces a modification to the standard crossover in GP that allows two types of self-adaptation. The first, which he terms *Selective Self-Adaptive Crossover (SSAC)* is a method that utilises individual-level adaptation. SSAC involves keeping associated with each parse tree a parse tree of probabilities; these probabilities impact the choice of the crossover point. The second, *Self-Adaptive Multi-Crossover (SAMC)*, keeps a similar mirror parse tree of probabilities, but each probability equates to the chance of a crossover occurring at that point (i.e. rather than influencing the choice of where the one crossover point occurs as in SSAC, with SAMC multiple crossover points can occur, each point occurring with a probability determined by the corresponding point in the mirror probability parse tree). Unlike SSAC, SAMC is a component level adaptation (as explained in section II.C).

Whilst the results obtained by Angeline are promising they differ from the study in this paper as they are a means of adapting the way the crossover works rather than choosing between different crossover operators. For instance, in all the cases explored by Angeline, the different crossovers act to swap one or more complete subtrees – there is no equivalent of the uniform crossover or the "ripple crossover" of GE [14]. Whilst the self-adaptation is designed to make the appropriate type of crossover more efficient, and in the test domains analysed – with certain caveats – this appears to be the case, our interest is more in using the evolutionary algorithm to choose between different crossover types (which appear to traverse the problem domains in different ways) in an attempt to make the algorithm more robust for problem domains where one type of crossover operator on its own may have problems.

C. Component Level Adaptation.

Arguably, the SAMC operator introduced by Angeline (see II.B.2) is a component level operator in GP, since it impacts on the probabilities of any one component (or node of the expression tree) being swapped during crossover. Whilst this operator certainly can have a radical impact on the crossover (allowing multiple nodes to be swapped in any one operation) it still utilises the same principle of crossover, namely the swapping over of subtrees.

Otherwise component level adaptation would appear to have been most extensively studied in EP and ES and is of limited relevance to the aims of the paper. The reader is referred to [3] for a fuller discussion of this type of adaptation.

III. Grammatical Evolution

A. Introduction

Grammatical Evolution (GE) is a form of genetic programming which utilises the evolutionary algorithm to evolve code written in any language, provided the grammar for the language can be expressed in a Backus Naur Form (BNF) style of notation [8]. Traditional genetic programming, as exemplified by Koza [9] has the requirement of "Closure". Closure, as defined by Koza, is used to indicate that a particular function set should be well defined for any combination of arguments. Previous work by Montana [10] suggests that superior results can be achieved if the limitation of "closure" required in traditional genetic programming can be eliminated, for example through typing. Whigham [11] demonstrates the use of context free grammars to define the structure of the programming language and thereby overcome the requirement of closure (the typing being a natural consequence of evolving programs in accordance with the grammar). GE utilises the advantages of grammatical programming, but, unlike the method proposed by Whigham, separates the grammar from the underlying representation (or as this is commonly referred to; the genotype from the phenotype).

It has been argued that the separation of genotype from phenotype allows unconstrained genotypes (and unconstrained operations on genotypes) to map to syntactically correct phenotypes. Keller [12] presents empirical results demonstrating the benefit of this type of mapping. One of the interesting aspects of having such a simple underlying genotype as GE (a bit string) is that it is possible to design a number of operators that act on this simple bit string. For instance crossovers can be designed which mirror uniform crossover, single bit crossover and crossovers which swap complete expansions in the underlying grammar, each of which represent (after the genotype to phenotype mapping has been performed) radically different methods of searching the phenotype space. It is likely that different crossovers have advantages in different fitness landscapes. The question that this paper explores is whether there is a way to allow the choice of which crossover to apply to come under the control of the genetic algorithm itself.

The way that GE maps the genotype to the phenotype is discussed in the next subsection. The different types of crossovers that are used in this paper are discussed in subsection III.C.

B. Grammatical Evolution – the mapping

Rather than representing programs as parse trees GE utilises a linear genome representation to drive the derivation of the encoded program from the rules of an arbitrary BNF

grammar. Typically the genome (being a variable length bit string) is split up into 8 bit codons and these codons are used sequentially to drive the choices of which branch of the grammar to follow. The maximum value of the codon is typically many times larger than the number of possible branches for any particular non-terminal in the grammar and a mod operator is utilised to constrain it to the required number.

Using, by way of example, the numerical regression grammar contained in listing 1; the following illustrates the translation process. If an individual had the following initial DNA and codon pattern

```
DNA: 00100001 00010100 00100000 00000001 00010000…
Codons:   33       20       32       1        16…
```

it would be translated as follows: No codon would be used for the first expansion, since there is only one choice: <code> (being the start of the parse) is expanded to <i_value>. The initial codon of the genome would be used to determine which of the five possible expansions to apply to <i_value>. The value of 33 would be MOD'd with five (since there are five choices) to give a value of 3. The fourth choice (<i_value><op><i_value>) would then be chosen. The expression now is "<i_value><op><i_value>". The first non-terminal <i_value> is expanded by using the next codon (20). <i_value> has five choices, so 20 would be MOD'd with 5, to give zero and XVAL would be chosen. The expression is now "XVAL <op> <i_value>". The next non-terminal (<op>) is expanded using the codon 32. 32 Mod 4 = 0, so "+" is chosen, leaving us with "XVAL + <i_value>". <i_value> is then expanded using the next codon (1), to give us <number>. The expression is now "XVAL + <number>". The next three codons are used to decode the three digits in number, giving us, say, "XVAL + 0.621" There being no further non-terminals in the expression the expansion is complete. Any remaining codons are not used.

If the expression can be fully expanded by the available codons (i.e. the expansion reaches a stage where there are no non-terminals) the individual is valid; if the codons run out before the expression is fully expanded the individual is invalid and assigned a fitness of zero, effectively removing it from the population.

```
code:     i_value
i_value:  XVAL
          | number
          | i_value op i_value
          | ( i_value op i_value )
          | - i_value
number:   0. digit digit digit
digit:    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
op:       + | - | * | %
```

Listing 1 – Extract of Numerical Regression Grammar

C. Crossover Operators

In previous work [13] we have explored a number of different crossovers which can be utilised in GE. Here we briefly describe the two crossover types that are utilised in this paper.

- Single Point Bit Operator: The standard GE crossover [8] is a simple one-point operator. Here random points are chosen in the genotype (the bit string) and the two parents have their "tails" swapped. Our implementation of the operator constrains the choice of crossover point to occur during the coding part of the genome (i.e. in the part of the genome which is used to "decode" the grammar). As illustrated in Diagram 1 the values of the codons in the tails being swapped may change (unless the crossover points happen to fall on codon boundaries). In addition the codons in the new tails may be used to interpret a different part of the grammar than the part they interpreted in their parent. Given this, the single point bit crossover has been criticised on the grounds that it is seemingly destructive of the information contained in the second contributing parent. However, despite the apparent destructive effect of the crossover, its efficacy as a search operator has been confirmed in [14].

- LHS Operator: The LHS crossover acts as a two-point crossover whereby the expansion of a rule of the grammar (or the Left Hand Side of a grammatical rule) is replaced by an expansion of the same rule found in the other parent. The mechanism by which this is achieved is fully described in [13]. The LHS crossover is similar to the traditional GP sub-tree crossover, save that it operates on the parse tree rather than the complete phenotype.

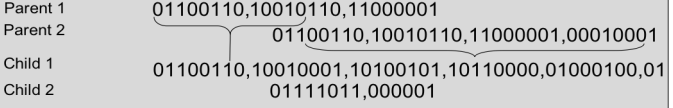

Diagram 1 - Bit Crossover Operator

Of the two crossover operators used in this paper the Single Point Bit Operator is arguably the most destructive, as it preserves neither the codon values of the tail nor their context. The LHS Operator is the least destructive of the operators since codon sequences are swapped whilst preserving their values and the part of the grammar they are being used to interpret.

An important point to note for the purposes of this paper is that these two crossover operators represent quite distinct methods of searching the phenotype space.

D. The GE environment

The following constant GE strategy was utilised:

- Selection of individuals was based on a probability selector, that is the chance of any particular individual being chosen to breed was directly proportional to its fitness.
- A strategy of retaining the fittest 5% of individuals was adopted.

- A constant mutation rate of 1 in 2,000 was applied to each child generated.
- Two children were generated for each crossover operator.
- Invalid individuals were given zero fitness and were not eligible for selection or breeding.
- Individuals started with a random bit string; in particular no attempt was made to ensure that the first random individuals were a minimum size, although if an individual was invalid it was regenerated.
- At each generation a population of 250 (Balance Scale Problem) or 500 (Regression Problem) individuals was generated.
- Each run consisted of 1000 generations.

E. Bloat control

Like all GP, GE has the potential to suffer from what has been termed "bloat". However, bloat appears to be easier to control in GE than in traditional GP, by putting a mild fitness pressure on DNA size [13]. The rule used was: if DNA size exceeds 3000 bits, then if two entities had the same fitness then the entity with the smaller DNA is ranked ahead. This only impacts on the decision as to the top 5% of individuals to copy across to the next generation (it does not impact on the probabilities of selection for crossover). Finally DNA sizes were capped at 7000 bits; if they exceeded this then the entity is deemed invalid.

IV. The Need for the Variable Crossover

The need for a crossover which could combine the various search strategies encapsulated by the different crossovers available for GE became apparent when the test domains described in this paper were analysed.

A. Numeric Regression

The numeric regression problem explored in this paper is simple regression of the sextic polynomial $x^6 - 2x^4 + x^2$. This problem domain has been explored by Koza [15], amongst others. For each run 50 random values of x in the range 0 to 1.0 are generated. To compute the fitness the absolute errors between the expected values of the function for each of these 50 input values and the value returned by the evolved function are summed. In our implementation, this cumulative error is then multiplied by 200 and subtracted from 1000. A fitness of 1000 therefore represents maximum fitness (no cumulative error). The minimum fitness of a valid individual is capped at 1. It is this measure of fitness that guides the evolution process.

In accordance with Koza's design there is another measure of success of the evolved function, being the number of "hits" achieved: for each of the x's chosen for that run a hit is defined as occurring if the value returned from the evolved function differs by less than 0.01 from the expected value. In this case achieving 50 hits represents a perfect solution to the numerical regression problem.

Listing 1 contains an extract of the relevant parts of the grammar used to evolve the solutions. XVAL represents the value of x in the evolved equation.

B. Balance Scale Problem

The balance scale problem is a data classification problem that is commonly used in Machine Learning. The data is available from the UCI Machine Learning Repository [16]. It involves a hypothetical balance scale where weights (ranging in value from 1-5) can be placed on the left hand and right hand side of the balance scale in one of five positions (1 being the closest to the fulcrum, 5 the furthest away). The data classification algorithm when given the weight and position of the left hand weight and the weight and position of the right hand weight needs to predict which position the balance scale will end up in (tilted to the left, tilted to the right or perfectly balanced). A formula which gives a perfect result is Left Weight * Left Position, compared to Right Weight * Right Position.

```
code:  lines , Class = class_value

lines: line, lines
       | line

line:  if (bool) then {lines } else {lines }
       | if (bool) then {lines }
       | Class = class_value

bool:  ( i_value bool_op  i_value )
       | Not ( bool )
       | bool AND bool
       | bool OR bool

bool_op: = | < | >

i_value:     attribute
             | i_value op i_value
             | ( i_value op i_value )
             | - i_value


attribute:   A1 | A2 | A3 | A4

class_value: Balanced | Left | Right

op: + | - | * | %
```

Listing 2 – Balance Scale Problem Grammar

The program evolved from this grammar is instantiated with the attributes A1-A4 (which take the value of the relevant query). A1 represents the weight on the left hand side, A2 the position of the left hand weight, A3 the weight on the right hand side and A4 the position of the right hand weight. The program is then evaluated until the first assignment to "Class" occurs, this being the value returned by the evolved classifier. As can be seen the expansion of <Code> ensures that at least one assignment to Class occurs. If the program has assigned the correct value to Class given the values of A1-A4 then it receives a positive score (or a hit), an erroneous classification would score zero.

The dataset contains 625 examples that comprehensively cover all possible combinations of A1-A4 for each of the five values they can take (representing the 5 weights or 5 positions). Of the 625 examples, 288 are correctly classified as tipping to the left, 288 as tipping to the right and 49 as balanced. For the purposes of evaluating a run the data were divided into two sets - a training set and a test set. The training set contained 80% of the samples (randomly drawn for each run from the full set). The test set was the remaining 20%. Only the training set was used to evolve the classification programs (i.e. the fitness function returned the number of hits based on the 80% training set) but for reporting purposes the whole set was used (so to score 625 out of 625 the evolved classifier had to correctly classify both the training set and the test set).

C. *Initial Runs – Regression Problem*

The initial runs of the regression problem, mapping the average fitness achieved by each of the two operators over 300 runs are reproduced as chart1.

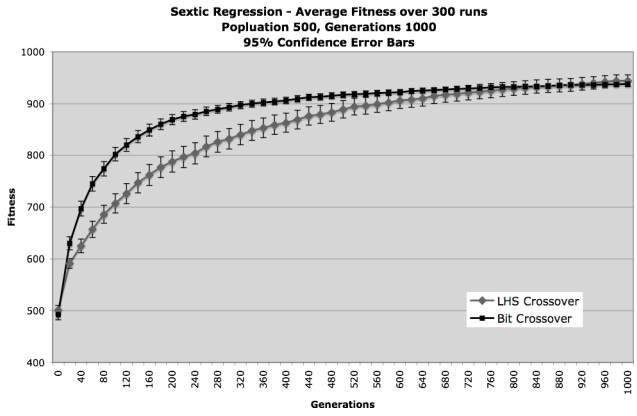

Chart 1 – Average fitness over 300 runs.

As can be seen from chart 1 (it should be noted the Y-axis does not commence at zero to allow the differences between the runs to be better observed) the single bit operator is significantly better for the first 450 generations (the error bars represent the 95% confidence level), with the LHS operator only catching up towards the end of the runs.

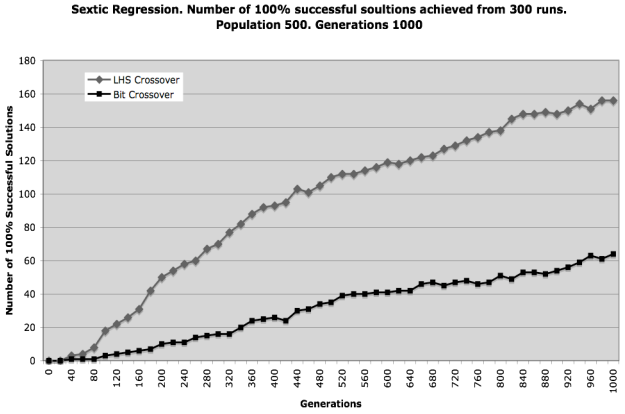

Chart 2 – Number of 100% successful solutions.

A different story emerges if one looks at the number of 100% successful runs achieved (i.e. the number of runs for which 50 "hits" were achieved).

As Chart 2 shows, the LHS Operator appears to be more successful in achieving 100% successful solutions than the Single-Point Codon operator despite the average fitness of the runs being lower.

Closer analysis of the data shows that the LHS operator appears more likely to get trapped in initial local maxima. Chart 3 shows the first 200 runs of the Single Bit operator and Chart 4 shows the first 200 runs of the LHS Operator. The distribution of the runs leads to the conclusion that a large number of LHS Operator runs are getting trapped early on (between 400 and 600 fitness), whereas the Single Bit Operator runs seem to get trapped at higher fitnesses.

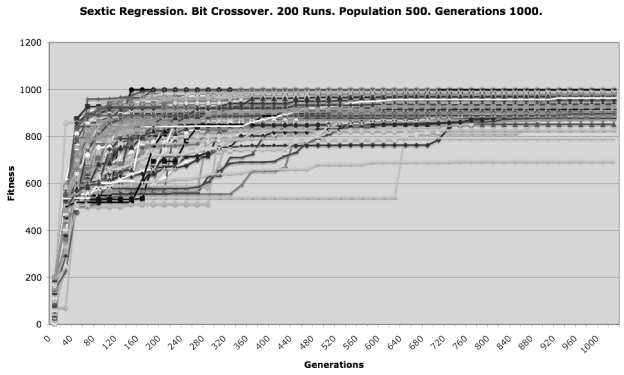

Chart 3 – Fitness for each of 200 Runs of Bit Crossover

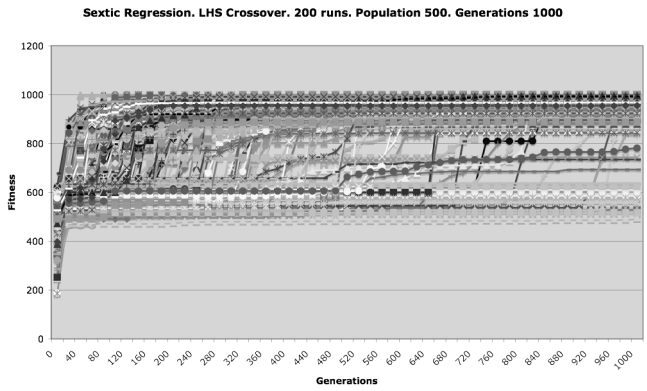

Chart 4 – Fitness for each of 200 Runs of LHS Operator

D. *Initial Runs – Balance Scale Problem*

The Balance Scale Problem exhibits similar characteristics. There is one very strong early local maximum corresponding to the "zero rule" of always classifying the scale as tipping to the left (or right) (e.g. Class = Left), which scores 288 out of 625 (there being 288 instances in the data where the scale tips to a particular side). This local maximum traps almost 50% of all the runs using the LHS Operator. None of the runs using the Bit Operator got trapped at this point.

There is another local maximum at 438, corresponding to a single test (e.g. if (A1>A3) then { Class = Left } else { Class = Right }). With the LHS operator a further 10% of the runs are trapped here, again none of the runs of the single bit operator were trapped at this point. So 60% of the runs using the LHS operator fail to make it past a score of 438 whereas every run with the bit operator gets past this point – and yet the LHS operator performs marginally better than the Bit operator when one analyses the number of 100% successful solutions generated.

E. Initial Runs - Analysis

The Bit and LHS operators by their different sectioning of the search space appear to get trapped at different stages of the search process.

One possible explanation is that the Bit Crossover performs a global search, whilst the LHS operator performs a local search. Chart 5 shows the average variance between the fitness of the top 50% of the population for the two operators (Regression Problem). As can be seen the Bit Crossover Operator has a larger disparity between the fitness of its population members - this is consistent with the concept of it being a global search operator and the LHS Crossover Operator being a local search Operator. [17] provides some analysis as to why the normal GP crossover operator (which is similar to the LHS operator) might be thought of as a local search operator.

If this explanation as to the operation of the two crossover Operators is correct then an algorithm that is able to utilise both crossover operators might be able to harness the power of both types of searches.

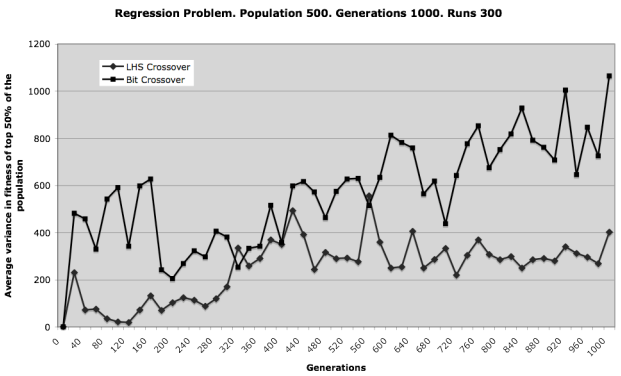

Chart 5 – Average variance in top 50% of the population.

V. VARIABLE CROSSOVER

A. Methodology

The methodology utilised was to associate with each individual the type of crossover that had been used to create it. Initially this was set to a random type (being one of the two crossovers described above). When a crossover was to be performed the following algorithm was employed.

```
If both parents were created using the same
crossover operator, use that operator:
    - The two children created inherit that
      crossover operator.
    - There is a chance (in our tests 10%) that
      the crossover mutates to a randomly set
      crossover (given these tests were limited
      to two crossovers this represents a 5%
      chance the crossover switches to the other
      type).
Else (if the crossovers which created the parents
differ), randomly choose to apply the crossover of
one of the parents:
    - The two children are created using the
      randomly chosen crossover;
    - The crossover preference for each child is
      inherited from one of its parents (randomly
      chosen).
```

It should be noted that the LHS Operator is more likely to create valid children than the Bit Operator. Consequently where an operator is to be applied, then it is continuously applied to the two parents until two valid children are produced. By applying the appropriate operator until valid children are produced we remove any bias induced by the operator's relevant success rate allowing the bias (if any) of producing more successful children to dominate.

The reasoning behind this algorithm is that if it is indeed true that one of the operators exhibits the characteristics of a global search then, when there are surrounding peaks with a higher fitness its children are likely to find them. Consequently, children created by the global search operator will include those with a higher fitness than the fitness of the children being found by the local operator (which will be slowly climbing the current peak). During this period the global operator will continue to have a substantial number of children in the population and will continue to be applied. However, once the global operator has shifted the population to the "best" peak it can find, the children of the local search operator will begin to dominate the population as it moves them to the top of that peak. The "mutation" included in the algorithm ensures that some of each crossover population is maintained to prevent a premature dominance of one type of crossover operator.

In order to complete the comparison tests (and in light of the results obtained by Spears) a random crossover (which selects the crossover type randomly) was also implemented. This algorithm simply ignores any preference expressed in a child and chooses the crossover operator to apply randomly. In this paper each of the two crossovers had a 50% chance of being chosen. Again once a crossover is chosen it is repeatedly applied until valid children are generated, removing any bias that results from a particular crossover being more likely to create valid children than another.

VI. RESULTS

Chart 6 shows the average fitness obtained over 300 runs for each of the four operators (the previously shown Single Point Bit operator and LHS operator, and the new Variable crossover operator and Random crossover operator). As can

be seen both the Random crossover operator and the Variable crossover operator surpass the previous crossover operators, with the Variable crossover operator achieving significantly better results after about 680 generations (the error bars represent the 95% confidence level). Again note that the Y-Axis does not start at zero to allow the differences between the runs to be more clearly observed.

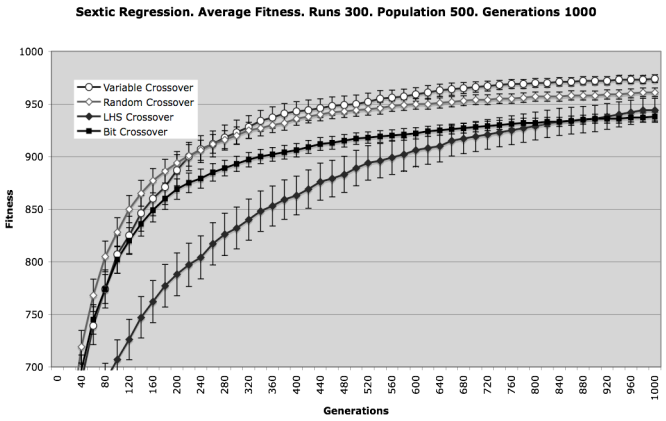

Chart 6 – Regression Problem. Average fitness over all runs.

In order to test for significance in the second test, a slightly different analysis was used. Rather than just counting the number of successful solutions in each of the 300 runs, the runs were split up into 10 groups of 30 runs, allowing the variances and confidence levels to be worked out.

As can be seen from chart 7 the Random Crossover appears to perform no better than the LHS Replacement crossover operator (the differences are not significant) whereas the Variable Crossover gives a significant increase in the number of correct solutions found.

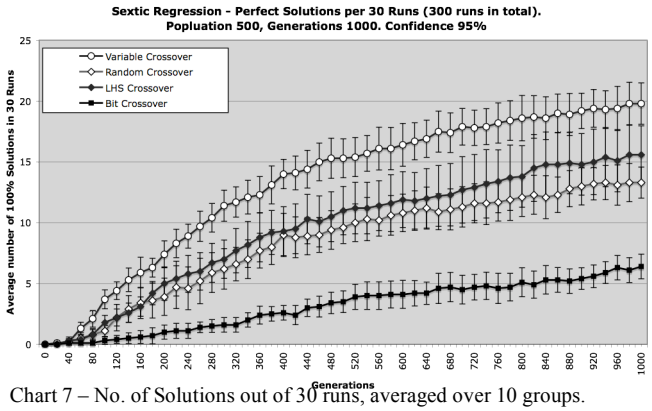

Chart 7 – No. of Solutions out of 30 runs, averaged over 10 groups.

Chart 8 shows the results for the Balance Scale Problem. The Variable Crossover and the Random Crossover far outperform either of the two crossovers on their own, although for this problem domain there does not appear to be a statistical difference between the Variable and the Random crossover.

Finally we analysed the percentage of the population that had a preference for the Single Point Crossover. Typically (in both problem domains), this would start off at 50% and

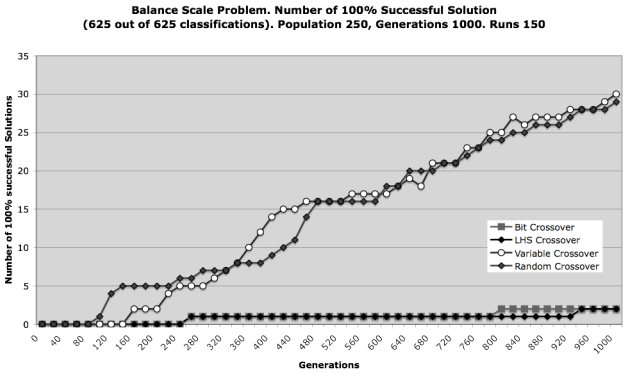

Chart 8 – No. of Solutions found in the Balance Scale Problem.

in the course of the first 100 or so generations settle to approximately 20%). Of interest was whether this decay was as a result of a switch from a global to a local search or one of natural decay. To analyse this we started several runs of the Balance Scale problem domain with all the individuals having a preference for the LHS Operator. Chart 9 shows the average percentage (averaged over 60 runs) of the population with a preference for the Single Point Operator. The chart shows the first 600 generations, with a granularity of 3 generations (i.e. the information was saved every 3 generations).

As can be seen the Single Bit Operator builds up (from zero) to account for over 30% of the population (indicating that it must be finding fit individuals) until it drops down to 20%. This is in line with our suggestion as to the operation of the two different crossover operators.

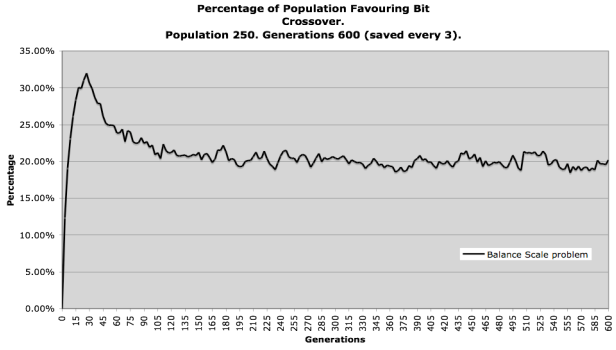

Chart 9 – Analysis of application of the different crossover

VII. Conclusions and Future Work

We have shown that for the Sextic Regression Problem and the Balance Scale problem each of the two crossover operators detailed in this paper encounter some difficulty in producing 100% correct solutions. Our analysis suggests that, for these problems, each of the two crossover operators is performing a different type of search. We posited that a combination of the two operators would be helpful and suggested a mechanism that would allow the population to utilise the different searches in a manner that did not use a population-level adaptation, but rather an individual level

adaptation. This Variable crossover operator was shown to perform substantially better than either of the two crossover operators on their own, and at least as well as, and in some cases significantly better than, the alternative strategy of randomly selecting a crossover operator to apply. Of note is that the strategy of using more than one type of crossover does appear, in both cases examined, to allow the population to escape from local maxima that would otherwise trap some runs.

Initial results on other domains seem to indicate that the two operators discussed in this paper habitually traverse the fitness landscape in different ways suggesting that the utility of using more than one type of crossover operator may generalise. Part of the problem in analyzing this is the large number of runs needed to obtain statistically significant results when one is looking at the number of 100% successful solutions obtained rather than, say, the average fitness of the population.

Given that the Variable crossover operator did not perform significantly better then the Random crossover operator in one of the domains examined, we believe that further work needs to be done in relation to verifying that the Variable crossover operator generalises and in particular with respect to problem domains which favour either the Single-Bit crossover or LHS crossover.

We also believe that the algorithm presented can be further generalised to include more than one operator. [13], for instance, provides a sample of the large number of different crossover operators available in GE. Further work will have to be done to investigate whether increasing the number/types of crossovers under evolutionary control aids or hinders the process.

One of the motivations in writing this paper is an attempt to find a crossover operator that represents a "good choice" in the absence of any empirical evidence as to the best crossover operator to use. Even setting aside the difficulty of gathering empirical evidence (especially in novel and/or complex domains) this paper illustrates the difficulties and danger of using, for instance, average fitness as a guide to the crossover operator to use if one were interested in evolving complete solutions.

References

[1] Angeline, P. J. 1994. Genetic programming and emergent intelligence. In *Advances in Genetic Programming*, K. E. Kinnear, Ed. Mit Press In Series In Complex Adaptive Systems. MIT Press, Cambridge, MA, 75-97.

[2] Beasley, D., Bull, D.R., & Martin, R.R. (1993) "An Overview of Genetic Algortihms: Part 2, Research Topics", University Computing, 15(4) 170-181

[3] Angeline, P. J. (1995) Adaptive and Self-Adaptive Evolutionary Computations, In M. Palaniswami, et. al. (eds.), Computational Intelligence: A Dynamic Systems Perspective, Piscataway, NJ: IEEE Press, pp 152-163.

[4] Yang, S. 2003. Adaptive crossover in genetic algorithms using statistics mechanism. In Proceedings of the Eighth international Conference on Artificial Life (December 09 - 13, 2002). R. K. Standish, M. A. Bedau, and H. A. Abbass, Eds. MIT Press, Cambridge, MA, 182-185.

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

[6] Spears, William M. (1995). Adapting Crossover in Evolutionary Algorithms. Proceedings of the Evolutionary Programming Conference, 367-384.

[7] Angeline, P. J. 1996. Two self-adaptive crossover operators for genetic programming. In *Advances in Genetic Programming: Volume 2*, P. J. Angeline and K. E. Kinnear, Eds. MIT Press, Cambridge, MA, 89-109.

[8] Ryan C., Collins J.J., O'Neill M. Grammatical Evolution: Evolving Programs for an Arbitrary Language. Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming 1998.

[9] J.R. Koza Genetic Programming MIT Press/Bradford Books, Cambridge M.A., 1992Keijzer 1991] Keijzer M., Ryan C., Cattolico M., and Babovic V. Ripple Crossover in Genetic Programming. In proceedings of EuroGP 2001.

[10] Montana D, Strongly Typed Genetic Programming. Technical Report 7866, Bolt Beranek and Newman Inc.

[11] Whigham P.A, Grammatically-based Genetic Programming (1995) Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, pages 33-41. Morgan Kaufmann Pub.

[12] Robert E. Keller, Wolfgang Banzhaf 1996. Genetic Programming using Genotype-Phenotype Mapping from Linear Genomes into Linear Phenotypes (1996) Genetic Programming 1996: Proceedings of the First Annual Conference

[13] Robin Harper and Alan Blair, A Structure Preserving Crossover in Grammatical Evolution, 2005 IEEE Congress on Evolutionary Computation, 2537-2544.

[14] Keijzer, M., Ryan, C., O'Neill, M., Cattolico, M., and Babovic, V. 2001. Ripple Crossover in Genetic Programming. In Proceedings of the 4th European Conference on Genetic Programming (April 18 - 20, 2001). J. F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. Tettamanzi, and W. B. Langdon, Eds. Lecture Notes In Computer Science, vol. 2038. Springer-Verlag, London, 74-86

[15] J.R. Koza Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, Mass., 1994. Chapter 5.

[16] http://www.ics.uci.edu/~mlearn/MLRepository.html

[17] R. Poli and W.B. Langdon. On the search properties of different crossover operators in genetic programming. In J.R. Koza et al., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 293–301, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann