Crossover in Grammatical Evolution: A Smooth Operator?

Michael O'Neill & Conor Ryan

Dept. Of Computer Science And Information Systems
University of Limerick
Ireland
{Michael.ONeill|Conor.Ryan}@ul.ie

Abstract. Grammatical Evolution is an evolutionary algorithm which can produce code in any language, requiring as inputs a BNF grammar definition describing the output language, and the fitness function. The usefulness of crossover in GP systems has been hotly debated for some time, and this debate has also arisen with respect to Grammatical Evolution. This paper serves to analyse the crossover operator in our algorithm by comparing the performance of a variety of crossover operators. Results show that the standard one point crossover employed by Grammatical Evolution is not as destructive as it might originally appear, and is useful in performing a global search over the course of entire runs. This is attributed to the fact that prior to the crossover event the parent chromosomes undergo alignment which facilitates the swapping of blocks which are more likely to be in context.

1 Introduction

While crossover is generally accepted as an explorative operator in string based G.A.s [4] the benefit or otherwise of employing crossover in tree based Genetic Programming is hotly debated. Work such as [2] went as far as to dismiss GP as a biological search method due to its use of trees, while [1] presented results which suggested that crossover in GP provides little benefit over randomly generating subtrees. Langdon and Francone have also addressed this issue, the former on tree based GP and the latter on linear structures, and have both introduced new crossover operators in an attempt to improve exploration [3] [6].

Both of these works exploit the idea of a homologous crossover, which draws inspiration from the molecular biological crossover process. The principal exploited being the fact that in nature the entities swapping genetic material only swap fragments which belong to the same position and are of similar size, but which do not necessarily have the same functionality. This, it is proposed, will result in more productive crossover events. Indeed, results from both Langdon and Francone et. al. provide evidence to back up this claim. A consequence arising from these conservative crossover operators is the reduction of the bloat phenomenon, occurring due to the fact that these new operators are less destructive and therefore the production of increasingly longer genomes being deemed unnecessary. Suggestions have been made that destructive crossover events could be responsible for bloat, bloat arising as a mechanism to prevent destructive crossover events occurring by acting as buffering regions in which crossover can occur without harming functionality. As with many non-binary representations, it is often not clear how much useful genetic material is being exchanged during crossover, and thus not clear how much exploration is actually taking place.

Grammatical Evolution (GE) an evolutionary algorithm which can evolve code in any language, utilises linear genomes [12] [11] [9] [8] [7]. As with GP systems, GE has come under fire for its seemingly destructive crossover operator, a simple one-point crossover inspired from GA's. In this paper we address crossover in GE, and seek answers to the questions of how destructive our one-point crossover operator is, and could the system benefit from a homologous-like crossover operator as proposed for tree-based GP, and the linear AIM GP. Results suggest that GE's one-point crossover operator is in fact a form of an homologous crossover, which acts as a global search operator throughout entire runs. As regards the destructive nature of GE's crossover, at the beginning of runs results show that the number of crossover events which produce individuals which are better than those in the current population is very high. On average this ratio remains relatively consistent during the duration of runs, which tells us that crossover is in fact a useful operator in recombining individuals effectively, rather than causing mass destruction.

2 Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm which can evolve computer programs in any language. Rather than representing the programs as parse trees, as in GP [5], we use a linear genome representation. Each individual, a variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. BNF is a notation which represents a language in the form of production rules. It is comprised of a set of non-terminals which can be mapped to elements of the set of terminals, according to the production rules. An example excerpt from a BNF grammar is given below. These productions state that S can be replaced with either one of the non-terminals *expr*, $if-stmt$, or *loop*.

```
S ::= expr       (0)
    | if-stmt    (1)
    | loop       (2)
```

In order to select a rule in GE, the next codon value on the genome is generated and placed in the following formula:

$$Rule = Codon\ Integer\ Value\ MOD\ Number\ of\ Rules\ for\ this\ non{-}terminal$$

If the next codon integer value was 4, given that we have 3 rules to select from as in the above example, we get $4\ MOD\ 3\ =\ 1$. S will therefore be replaced with the non-terminal $if - stmt$.

Beginning from the the left hand side of the genome codon integer values are generated and used to select rules from the BNF grammar, until one of the following situations arise: (a) A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar. (b) The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during this individuals mapping process. (c) In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value.

GE uses a steady state replacement mechanism [11], and the standard genetic operators of mutation (point), and crossover (one point). It also employs a duplication operator, which duplicates a random number of codons and inserts these into the penultimate codon position on the genome. A full description of GE can be found in [12] [9] [8] [7].

3 Crossover Operators in GE

By default, GE employs a standard one point crossover operator as follows: (i) Two crossover points are selected at random, one on each individual (ii) The segments on the right hand side of each individual are then swapped. As noted in Section 1, it has been suggested that the cost of exploration via crossover is the possible destruction of building blocks. Indeed, there has been work which shows that a reason for the increase in bloat in many GP runs is to protect individuals from destructive crossover. It is argued here that bloat occurs as a mechanism to prevent the disruption of functional parts of the individual arising from crossover events. To counteract the potentially destructive property of the crossover operator, and to indirectly reduce the occurrence of bloat, novel crossover operators have been developed [6][3]. Dubbed *homologous* crossover, these operators draw inspiration from the molecular biological process of crossover in which the chromosomes to crossover align and common crossover points are selected on each individual (i.e. at the same locus), and typically a two point crossover occurs.

This paper serves to investigate crossover in GE, and proposes a new form of operator inspired by the novel homologous crossover designed for GP. We then compare the standard GE one point crossover to two different versions of this homologous crossover, as well as two alternative forms of a two point crossover operator.

The standard GE homologous crossover proceeds as follows: (i) During the mapping process a history of the rules selected in the grammar is stored for each individual. (ii) The histories of the two individuals to crossover are aligned. (iii) Each history is read sequentially from the left while the rules selected are identical for both individuals, this region of similarity is noted. (iv) The first two crossover points are selected to be at the boundary of the region of similarity, these points are the same on both individuals. (v) The two second crossover points are then selected randomly from the regions of dissimilarity. (vi) A two point crossover is then performed. The reasoning behind this operator is to facilitate the recombination of blocks which are in context with respect to the current state of the mapping process. In this case these blocks are of differing lengths. The second form of the homologous crossover operator differs only in that is swaps blocks of identical size. In step (v) the two second crossover points are at the same locus on the individual. The two point crossover employed was the standard two point operator, and one in which the size of the fragments being swapped are the same.

Table 1. Grammatical Evolution Tableau for Symbolic Regression

Objective :	Find a function of one independent variable and one dependent variable, in symbolic form that fits a given sample of 20 (x_i, y_i) data points, where the target functions is the quartic polynomial $X^4 + X^3 + X^2 + X$
Terminal Operands:	X (the independent variable)
Terminal Operators	The binary operators $+, *, /$, and $-$ The unary operators Sin, Cos, Exp and Log
Fitness cases	The given sample of 20 data points in the interval $[-1, +1]$
Raw Fitness	The sum, taken over the 20 fitness cases, of the error
Standardised Fitness	Same as raw fitness
Hits	The number of fitness cases for which the error is less than 0.01
Wrapper	Standard productions to generate C functions
Parameters	$Population$ = 500, $Generations$ = 20 $pmut$ = 0.01, pcross = 0.9

Table 2. Grammatical Evolution Tableau for the Santa Fe Trail

Objective :	Find a computer program to control an artificial ant so that it can find all 89 pieces of food located on the Santa Fe Trail.
Terminal Operators:	left(), right(), move(), food_ahead()
Fitness cases	One fitness case
Raw Fitness	Number of pieces of food before the ant times out with 615 operations.
Standardised Fitness	Total number of pieces of food less the raw fitness.
Hits	Same as raw fitness.
Wrapper	Standard productions to generate C functions
Parameters	$Population = 500$, $Generations = 20$ $pmut = 0.01$, pcross = 0.9

4 Experimental Approach

For each type of crossover 100 runs were carried out on the Santa Fe trail and a Symbolic regression problem. Performance of each operator was measured for the following (1) Cumulative frequency of success, (2) Average size of fragments being swapped at each generation, (3) Ratio of the average fragment size being swapped to the average genome length and, (4) Ratio of crossover events resulting in successful propagation of the individual to the next generation and the total number of crossover events. Measures (2) and (3) have been used previously in [10]. Tableau's describing the parameters and terminals are given in Tables 1 and 2.

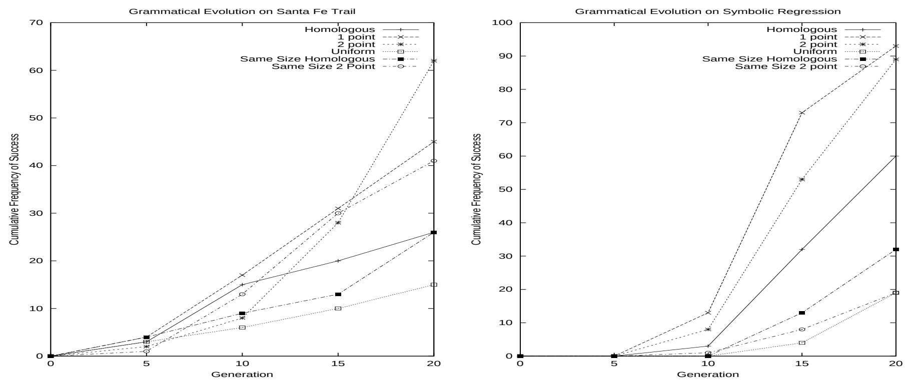

Fig. 1. Comparison of the cumulative frequencies of success for each crossover operator

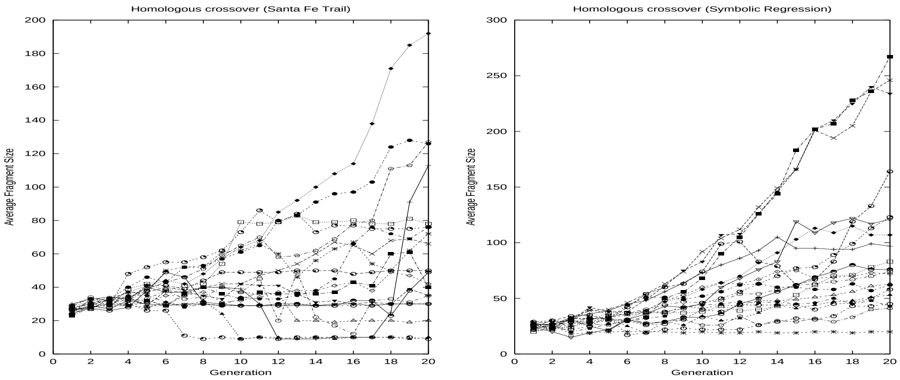

Fig. 2. Average fragment size being swapped each generation for Homologous crossover

5 Results

As can be seen in Figure 1, the cumulative frequencies of success clearly show that standard one and two point crossover are superior to the other operators on both problem domains. We will now describe the results for each operator under the other measures stated in Section 4.

Figure 2 shows the average fragment size being swapped at each generation with homologous crossover. Data can be seen for 20 separate runs, note that we are interested in general trends over the set of runs as opposed to the precise details for individual runs in each of these figures. Overall the fragment size increases as each generation passes, however the length of the chromosomes are also increasing, so a more useful measure of the ratio of average fragment size to the average chromosome length can be seen in Figure 3. The results for same size homologous, two point, same size two point, and one point can be seen in Figures 4, 5, 6, and 7 respectively. Figure 8 shows the ratio of individuals undergoing crossover which have been successfully propagated to the next generation and the total number of crossover events which have occurred. The results of this measure for all other operators can be seen in Figures 9,10,11,and 12.

6 Discussion

Examining figures 13 and 14, we can see the average over 20 runs of the ratio of individuals undergoing crossover which are propagated to the next generation to the total number of crossover events for each generation, and the ratio of the average crossover fragment size to the average chromosome length, respectively. In terms of the ratio of individuals being propagated to the next generation having undergone crossover, we can see for one point crossover in the case of the Symbolic Regression problem the rate of transfer remains relatively constant throughout the run around the value 0.35. A similar trend can be seen for one point crossover in the case of the Santa Fe trail problem although there is a slight

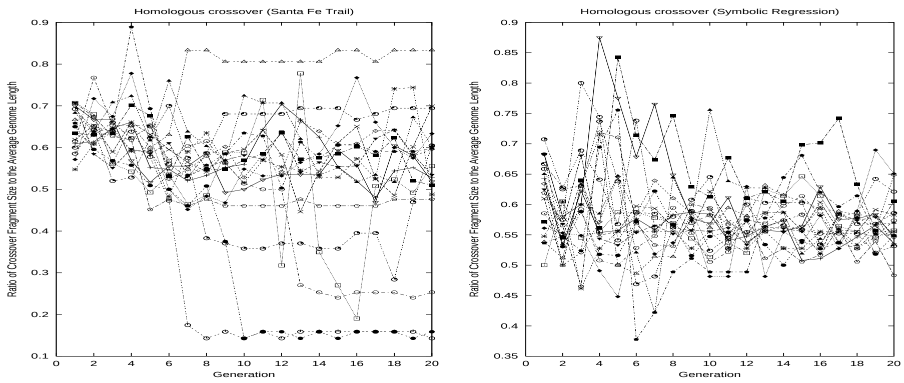

Fig. 3. Ratio of the average fragment size being swapped and the average chromosome length at each generation for Homologous crossover

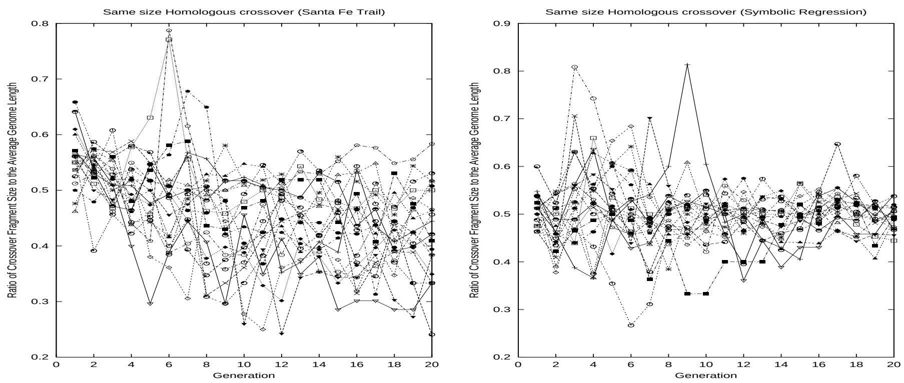

Fig. 4. Ratio of the average fragment size being swapped and the average chromosome length at each generation for Same size Homologous crossover

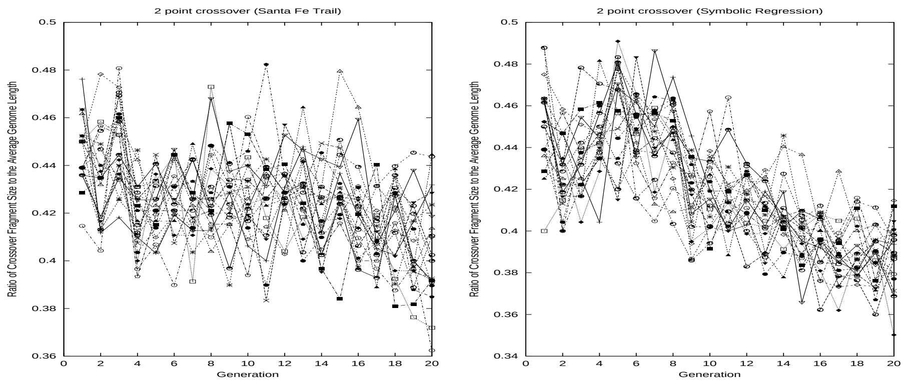

Fig. 5. Ratio of the average fragment size being swapped and the average chromosome length at each generation for two point crossover

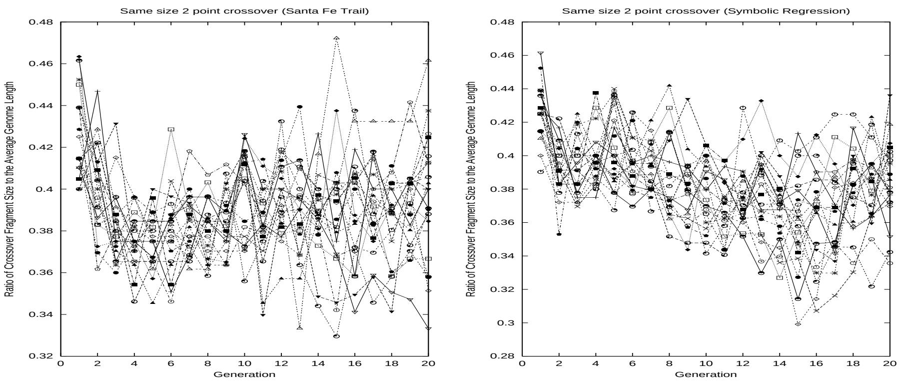

Fig. 6. Ratio of the average fragment size being swapped and the average chromosome length at each generation for same size two point crossover

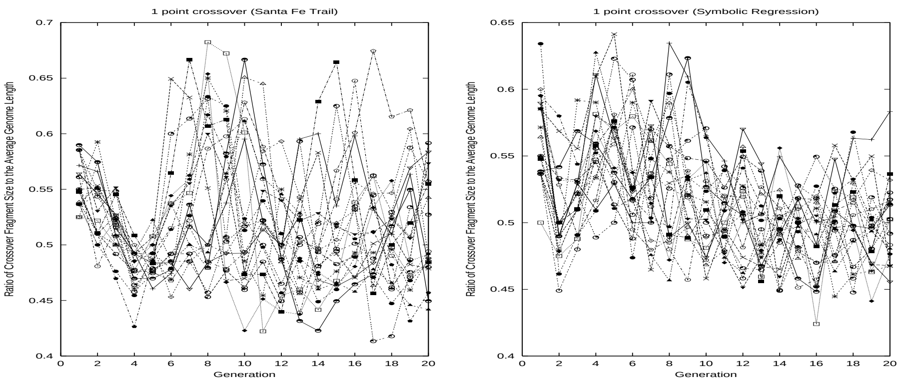

Fig. 7. Ratio of the average fragment size being swapped and the average chromosome length at each generation for one point crossover

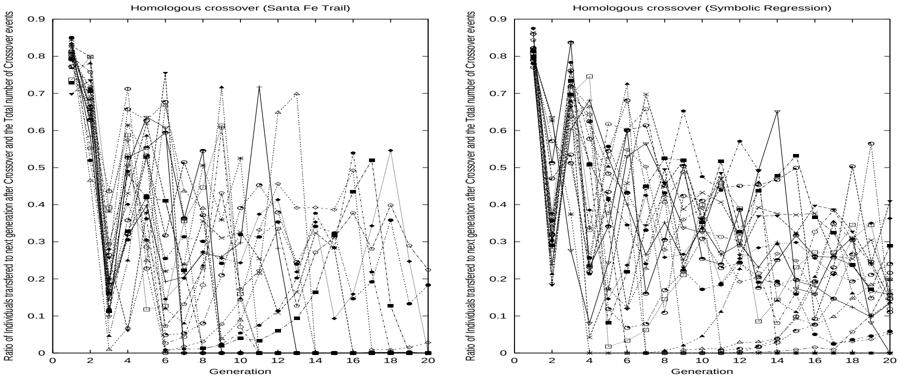

Fig. 8. Ratio of the number of individuals undergoing Homologous crossover which have been propagated to the next generation and the total number of crossover events occurring in that generation

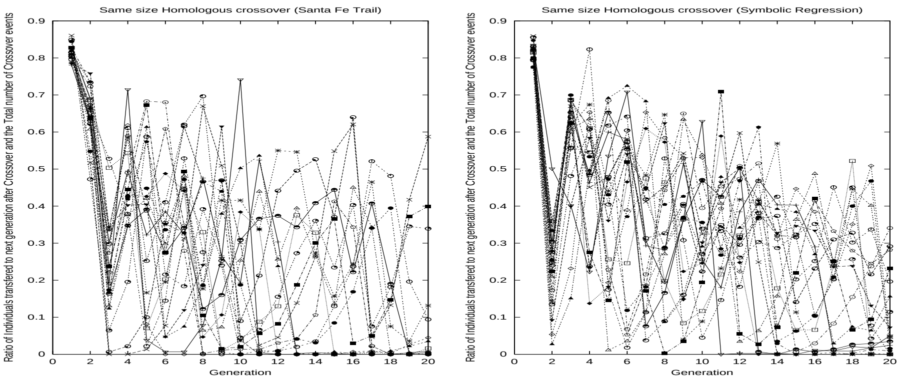

Fig. 9. Ratio of the number of individuals undergoing Same size Homologous crossover which have been propagated to the next generation and the total number of crossover events occurring in that generation

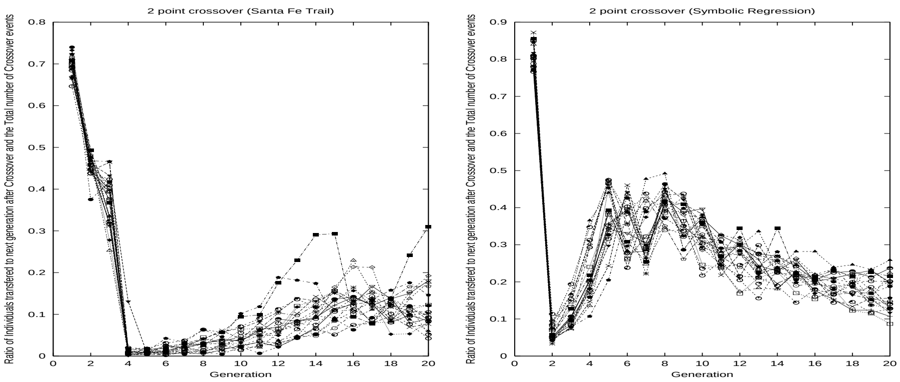

Fig. 10. Ratio of the number of individuals undergoing two point crossover which have been propagated to the next generation and the total number of crossover events occurring in that generation

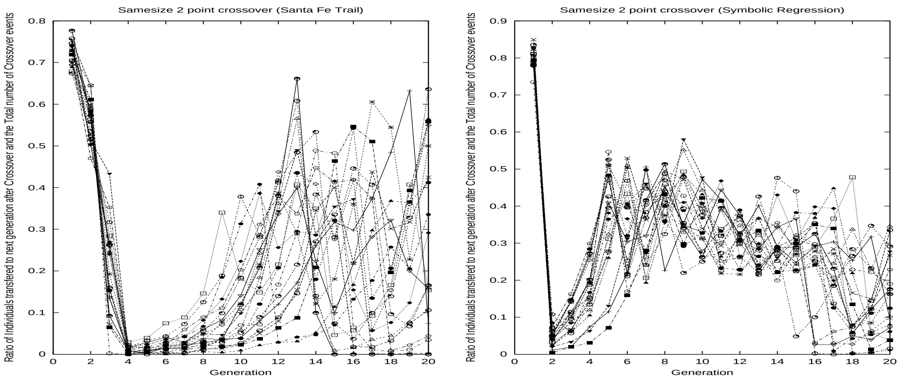

Fig. 11. Ratio of the number of individuals undergoing same size two point crossover which have been propagated to the next generation and the total number of crossover events occurring in that generation

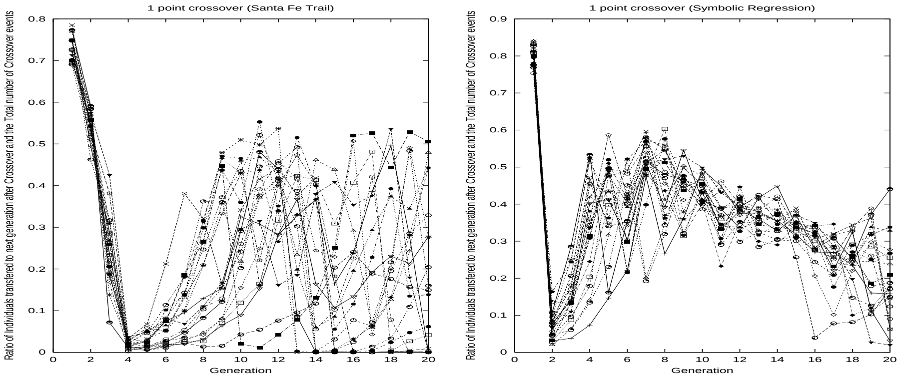

Fig. 12. Ratio of the number of individuals undergoing one point crossover which have been propagated to the next generation and the total number of crossover events occurring in that generation

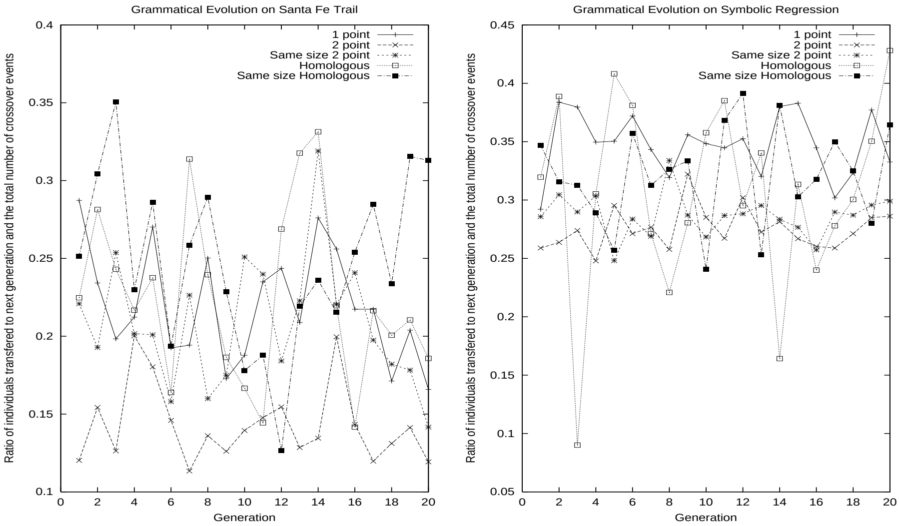

Fig. 13. Ratio of the number of individuals undergoing crossover which have been propagated to the next generation and the total number of crossover events occurring in that generation averaged over 20 runs

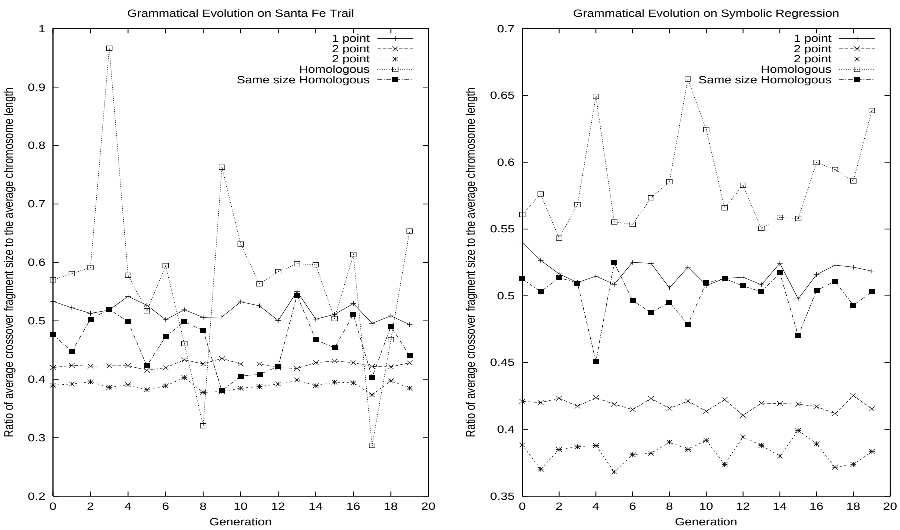

Fig. 14. Ratio of the average fragment size being swapped and the average chromosome length at each generation averaged over 20 runs

drop off as runs progress. Looking at the ratio of individuals transfered to the total number of crossover events for individual runs in figures 8, 9, 10, 11, and 12, we can see in all cases in the first few generations that this value is extremely high. Crossover then is acting in the initial stages of runs to aid in the removal of the poorer individuals randomly created in generation zero. These results for homologous crossover are not as consistent as those for one point, and directly compared appear erratic. The results show that the effort required to carry out our version of homologous crossover and the occasional peak it achieves in terms of individual transfer to each generation, is outweighed by the consistent results produced by the much simpler one point operator.

In general though, we can see the usefulness of this one point crossover operator by virtue of the fact it produces individuals which are capable of being propagated to successive generations given the steady state replacement strategy. This trend is observed over the course of entire runs.

Looking at the ratio of the average crossover fragment size to the average chromosome length, in the case of one point crossover we get a relatively smooth line around the 0.5 mark for both problems. Similar trends are observed for both types of two point crossover, however, these are localised to lower values. It can also be seen that the homologous operators are more erratic, although in the case of symbolic regression has a consistently higher value than all other operators. The experimental evidence shows us that crossover results in individuals which are being propagated to the next generation, and that the size ratio of these fragments to the actual genome length is consistent throughout the runs. This is in direct contrast to the results obtained in [10] which showed a drop off for all crossover operators except naturally for the uniform operator. These results showed how the operators examined in this case start off as global search operators but change rapidly to local search operators.

In light of the data obtained, namely the transfer of individuals having undergone crossover to the next generation, one could suspect that within GE individuals there exist useful building blocks which are being recombined to produce better performing individuals. The default one point crossover operator then is acting in the fashion of an homologous crossover operator by swapping blocks which are in fact in context due to the method in which the chromosomes are aligned prior to the crossover event.

With respect to the homologous operator described in [6] for tree based GP, a lot of effort is required to carry out this operation. Whereas in GE we can get a homologous type crossover when using standard one and two point operators, by virtue of the linear nature of our individuals.

7 Conclusions & Future Work

In conclusion, we have examined various crossover operators and their performance in the Grammatical Evolution evolutionary algorithm. It has been found that the standard one point crossover operator already employed in the algorithm is the most consistent of those examined, in general producing more successful runs. This one point crossover can result in a type of homologous crossover by virtue of the linear chromosome representation, while being computationally much simpler than other homologous operators for tree based GP. It also achieves a smoother and, therefore, more consistent graph for the ratio of individuals transferred to the next generation having undergone this operation to the total number of crossover events. Further investigation is required to establish if indeed building blocks do in fact exist in GE, to determine what form these blocks may take, and to see if GE's current one point crossover is exploiting these blocks.

Acknowledgement

This paper benefitted from conversations with Bill Langdon.

References

1. Angeline, P.J. 1997. Subtree Crossover: Building block engine or macromutation? In *Proceedings of GP'97*, pages 9-17.
2. Collins, R. 1992. Studies in Artificial Life. PhD thesis. University of California, Los Angeles.
3. Francone F. D., Banzhaf W., Conrads M, Nordin P. 1999. Homologous Crossover in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 99*, pages 1021-1038.

4. Goldberg, David E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley.
5. Koza, J. 1992. *Genetic Programming.* MIT Press.
6. Langdon W.B. 1999. Size Fair and Homologous Tree Genetic Programming Crossovers. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 99*, pages 1092-1097.
7. O'Neill M., Ryan C. 1999. Genetic Code Degeneracy: Implications for Grammatical Evolution and Beyond. In *Proceedings of the Fifth European Conference on Artificial Life.*
8. O'Neill M., Ryan C. 1999. Under the Hood of Grammatical Evolution. In *Proceedings of the Genetic & Evolutionary Computation Conference 1999.*
9. O'Neill M., Ryan C. 1999. Evolving Multi-line Compilable C Programs. *Lecture Notes in Computer Science 1598, Proceedings of the Second European Workshop on Genetic Programming*, pages 83-92. Springer-Verlag.
10. Poli Riccardo, Langdon W.B. 1998. On the Search Properties of Different Crossover Operators in Genetic Programming. In *Proceedings of the Third annual Genetic Programming conference 1998*, pages 293-301.
11. Ryan C., O'Neill M. Grammatical Evolution: A Steady State Approach. In *Late Breaking Papers, Genetic Programming 1998*, pages 180-185.
12. Ryan C., Collins J.J., O'Neill M. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*, pages 83-95. Springer-Verlag.

Appendix

The grammars for the two problem domains addressed in this paper are as follows:

A Symbolic Regression Grammar

$$N = \{expr, op, pre\_op\}$$

$$T = \{Sin, Cos, Exp, Log, +, -, /, *, X, ()\}$$

$$S =< expr >$$

And P can be represented as:

```
(1) <expr> ::= <expr> <op> <expr>       (A)
             | ( <expr> <op> <expr> ) (B)
             | <pre-op> ( <expr> )    (C)
             | <var>                  (D)
```

```
(2) <op> ::= + (A)
           | - (B)
           | / (C)
           | * (D)

(3) <pre-op> ::= Sin (A)
               | Cos (B)
               | Exp (C)
               | Log (D)

(4) <var> ::= X
```

B Santa Fe Trail Grammar

$$N = \{code, line, expr, if - statement, op, if - true, if - false\}$$

$$T = \{left(), right(), move(), food\_ahead(), else, if, \{, \}, (,)\}$$

$$S =< code >$$

And P can be represented as:

```
(1) <code> :: =  <line>             (A)
                |<code><line>       (B)

(2) <line> :: =  <expr>

(3) <expr> :: =  <if-statement>     (A)
                |<op>               (B)

(4) <if-statement> :: = if(food_ahead())<if-true><if-false>

(5) <op> :: =    left()             (A)
               | right()            (B)
               | move()             (C)

(6) <if-true>:={<expr>}

(7) <if-false>:=else{<expr>}
```