Positional Effect of Crossover and Mutation in Grammatical Evolution

Tom Castle and Colin G. Johnson

School of Computing, University of Kent,
Canterbury, CT2 7NF, UK
{tc33,C.G.Johnson}@kent.ac.uk
http://www.kent.ac.uk

Abstract. An often-mentioned issue with Grammatical Evolution is that a small change in the genotype, through mutation or crossover, may completely change the meaning of all of the following genes. This paper analyses the crossover and mutation operations in GE, in particular examining the constructive or destructive nature of these operations when occurring at points throughout a genotype. The results we present show some strong support for the idea that events occurring at the first positions of a genotype are indeed more destructive, but also indicate that they may be the most constructive crossover and mutation points too. We also demonstrate the sensitivity of this work to the precise definition of what is constructive/destructive.

Key words: Grammatical Evolution, crossover, mutation, position, bias

1 Introduction

A desirable trait of the genetic operations of crossover and mutation in evolutionary algorithms, is that they should have high locality; small genotypic changes should result in similarly small changes in the phenotype [1]. This allows the algorithm to more smoothly navigate the search space. Grammatical Evolution (GE) [2] represents programs as a linear genome (genotype) which is then converted to executable code (phenotype), through a mapping operation.

It has been proposed that as a result of the mapping operation, the genetic operators in GE do not maintain high locality [3]. One potential reason for this is that an alteration in the chromosome will change the meaning and context of all following codons, even if those codons remain the same. The expectation therefore, is that mutation and crossover events that occur at points towards the beginning of the genome will on average be far more destructive than those at the end. This paper investigates the extent to which this is true.

The following section gives an introduction to the GE technique and the crossover and mutation operators being considered, followed by a brief review of some existing, similar work. A detailed description of the experiments performed is given, before the results of the experiments are presented along with some discussion as to what they show. Finally, the paper will be concluded with some suggestions for future work.

2 Background

2.1 Grammatical Evolution

Grammatical Evolution [2] is an evolutionary approach to generating computer programs which is similar to Genetic Programming (GP) [4]. GE allows the generation of programs in any language definable in Backus Naur Form (BNF). The GE algorithm uses a variable length binary string called a *chromosome* to represent individuals. The chromosome is made up of a sequence of *codons*, which are simply integer values (we use a full range of positive 32-bit integers in this study). During fitness evaluation the chromosome is translated into a phenotype source string, which is valid according to the BNF grammar. This conversion is achieved using the mapping operation.

Mapping is performed by traversing the grammar. At each rule with multiple productions, the production to use is decided by the program's next codon. The integer codon value taken is then divided by the number of available productions with the remainder used as the index of the rule to map to. For example, given the example grammar rule below, the expr may be replaced by any of the 3 productions. The codon C_x is taken from the program to be mapped, and then the rule to use will be decided by: C_x MOD 3, where 3 is the number of productions in this case.

```
expr ::= conditional       {0}
       | loop              {1}
       | assignment        {2}
```

Mapping is complete when all non-terminal grammar rules have been replaced by terminals, or, if the program is deemed to be invalid, which may occur if there are insufficient codons to complete the mapping or some maximum depth/length limit is exceeded. In the case of insufficient codons it is typical to use a wrapping operator whereby mapping continues from the first codon, but this is usually accompanied by a maximum number of wraps.

GE uses the standard genetic operators of crossover and mutation. Mutation is a single point mutation where an individual selected to undergo mutation has a codon selected at random, and this codon is replaced with a new randomly generated codon. Similarly, crossover uses single point crossover, where two parents selected for crossover will have points chosen randomly within each of them, and the two sets of codons following these points exchanged.

A characteristic of the mapping process is the presence of a number of unused codons at the end of an individual. Crossover and mutation events still occur within this unused portion, but the change has no impact on the phenotype or fitness. Throughout the rest of this paper we will only be considering those change events that occur within the *active* region, defined to be those codons that are translated in program code.

2.2 Related Work

Numerous studies have looked into the effect of crossover in various forms of genetic programming. Nordin et al [5] and Johnson [6] independently looked at the fitness change caused by crossover for tree-based GP. Similar studies have also been conducted for linear [7] and graph [8] representations. The consistent conclusion is that most crossover events result in either a reduction in fitness or no change. Our work seeks to confirm whether this is the case in GE too, but in particular is more concerned with the effect crossover position has on this fitness change.

A number of researchers have considered crossover in the grammatical evolution algorithm. Harper and Blair proposed alternatives to the standard one-point crossover, with self-selecting crossover [9] and structure preserving crossover [10], and Keijzer et al. introduced a form of ripple crossover to GE [11]. O'Neill and Ryan [12] sought to identify a homologous crossover for GE. They determined that rather than causing the "mass destruction" expected, the standard one-point crossover operator itself acts as a form of homologous crossover, recombining individuals such that the context of building blocks is preserved.

There has been much less work focusing on the mutation operator in grammatical evolution. Rothlauf and Oetzel [3] examined mutation in relation to locality, determining that the representation used by GE does lead to lower locality. Another study by Hugosson et al. [13] compared a number of mutation operators, leading to the conclusion that the standard bit-flipping mutation is the best choice for locality. They also question whether higher locality in GE is actually beneficial.

3 Method

For each of even-five parity, Santa Fe trail, 6-bit multiplexer and a symbolic regression problem, 1000 runs were carried out to provide a large quantity of crossover and mutation data. The fitness of each individual selected for crossover and mutation was logged before and after the operation, along with the point at which the operation was carried out. The destructive or constructive nature of the operation could then be analysed in relation to the position at which it occurred. If the standardised fitness decreased then this was considered a positive change, if it increased it was negative and no change signalled a neutral effect.

Since crossover operations involve the exchange of material on two candidate solutions, it is more difficult to define what is positive/neutral/negative than for mutation. There are many ways in which to compare the resultant children's fitness to the fitness of the parents:

- average of both children compared to average of both parents
- each child compared to the average fitness of the population
- each child compared to one parent
- each child compared to the average fitness of the parents
- ...

Table 1. Even-five parity parameter tableau for GE

Raw fitness:	Number of inputs producing incorrect outputs, on all 2^5 possible cases.
Standardised fitness:	Same as raw fitness.
Population size:	500
Number of generations:	100
Mutation probability:	0.1
Crossover probability:	0.9

Table 2. Santa Fe trail parameter tableau for GE

Raw fitness:	Number of pieces of food before the ant times out with 600 operations.
Standardised fitness:	Total number of pieces of food, minus the raw fitness.
Population size:	500
Number of generations:	100
Mutation probability:	0.1
Crossover probability:	0.9

None of these definitions are perfect, so two approaches are presented here. In one approach the fitness of each child is compared to one parent program, in the other to the average of both of its parents. These approaches will allow us to consider each crossover operation as two events—one for each crossover point.

Variable length chromosomes make it necessary that some form of normalisation take place in order to compare separate operator events. This was performed by grouping into deciles, so that all crossovers occuring in the first 10% of the chromosome are grouped together, and so forth. Events at points outside the active portion of the chromosome were ignored due to their neutrality and as such, deciles were resolved based upon the position of the change within the active portion of the chromosome.

The parameters used for each run are outlined in tables 1, 2, 3 and 4. Chromosomes were allowed to wrap, but it should be noted that the effects of wrapping were negligible as a result of few individuals requiring it on the given problems.

Table 3. Multiplexer 6-bit parameter tableau for GE

Raw fitness:	Number of inputs producing incorrect outputs, on all 2^6 possible cases.
Standardised fitness:	Same as raw fitness.
Population size:	500
Number of generations:	100
Mutation probability:	0.1
Crossover probability:	0.9

Table 4. Symbolic regression parameter tableau for GE

Raw fitness:	Sum of the error, for a sample of 20 data points in the interval -1.0 to 1.0.
Standardised fitness:	Same as raw fitness.
Population size:	500
Number of generations:	100
Mutation probability:	0.1
Crossover probability:	0.9

4 Results and Discussion

The results of our experiments are presented in this section. We consider firstly mutation, and then crossover (measuring fitness change in two different ways). For each operator, three groups of bar charts are presented. These represent the proportion of positive, neutral and negative changes. Each bar represents one decile, as explained above.

4.1 Mutation

The results of our mutation experiments are presented in figures 1, 2 and 3. The results are not strongly problem-dependent: the same trends can be observed for each problem.

The positive events show the weakest overall pattern in the results: towards the end of the active region, there is typically a declining trend in the proportion of positive events; towards the beginning there is not a strong pattern, though three of the problems show a noticeably lower proportion in the first decile. The neutral events all have an upward trend, and the negatives all demonstrate a rapid downward trend.

Considering the neutral changes, one notable feature is the very high proportion of neutral changes in the later positions in the active region. This suggests that a mutation operator that was biased towards the earlier part of the active region might have a better effect. The results suggest that such an operator would create more movement in the search space; whilst by far the largest proportion of this would be negative, it would also increase the proportion of positive events. Using this alongside a local search operator (e.g. a mutation variant on brood crossover [14]) might help to counteract the number of negatives generated using such a mutation method.

4.2 Crossover

The crossover results are presented using the two different definitions of the positive-neutral-negative classification discussed above.

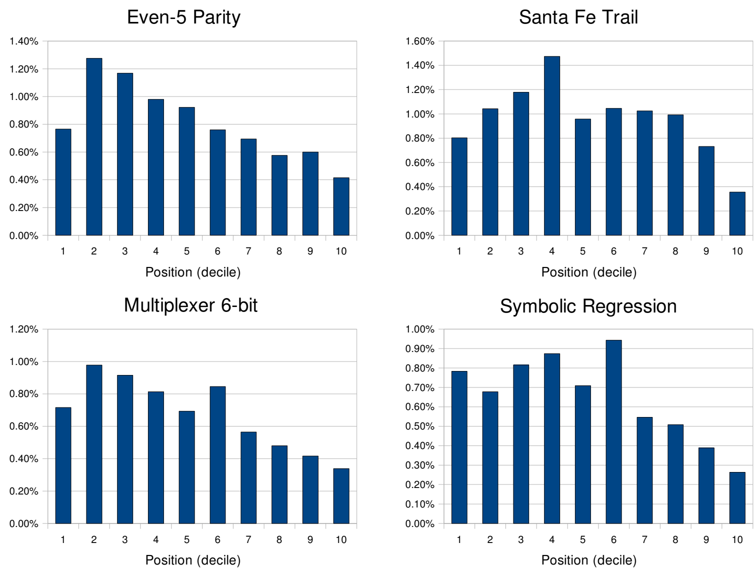

Fig. 1. Proportion of mutations in each decile that had a positive effect on fitness.

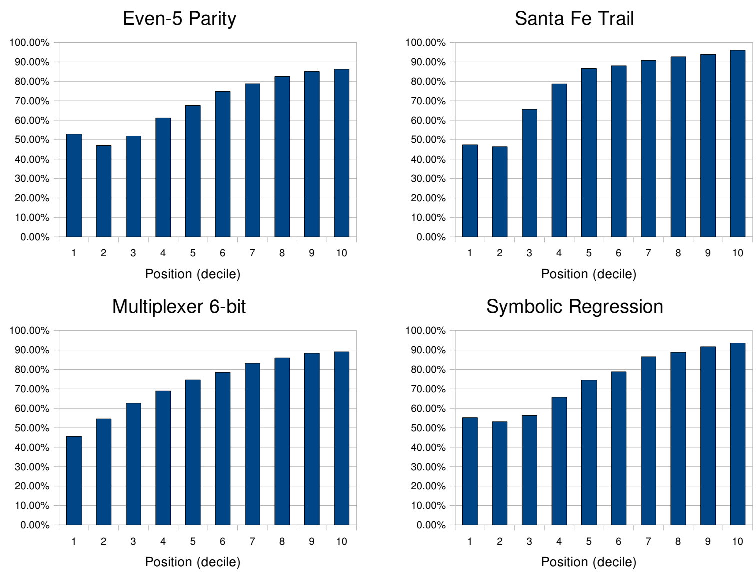

Fig. 2. Proportion of mutations in each decile that had no effect on fitness.

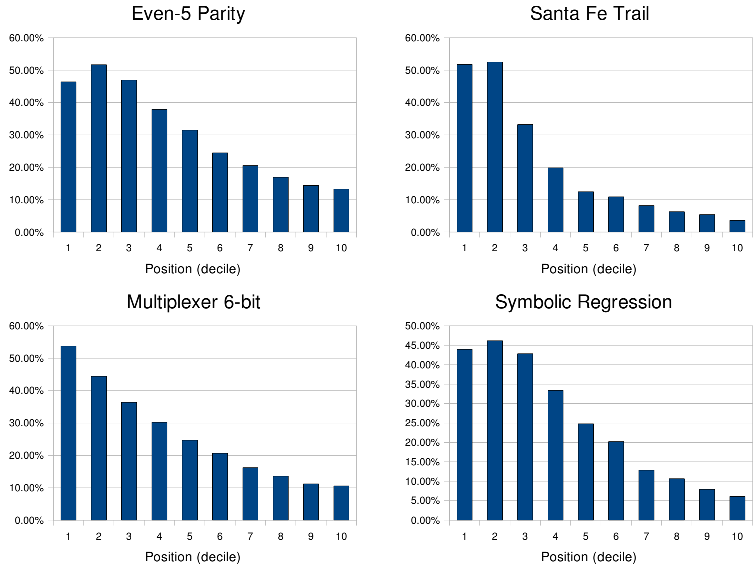

Fig. 3. Proportion of mutations in each decile that had a negative effect on fitness.

The results using the first defintion, the *averaged parent fitness* where the fitness of the child is compared with the average fitness of its two parents, are presented in figures 4, 5 and 6. As with the mutation results, these results do not show a strong problem-dependence, having similar patterns for all problems. The positive and neutral events both show a strong upwards trend, whereas the negative results show a steep downward trend.

The behaviour of both the negative and neutral events is similar to the behaviour observed in the mutation events. A large proportion of the crossovers towards the end of the active region result in a neutral change to the fitness. This perhaps suggests that these crossovers are placing the crossed over code from the second parent into a region on the first parent that, whilst encoded for, is redundant or inaccessible in the phenotype. One way to address this would be to use a data-flow analysis [15] to ensure that the crossover point is in an accessed part of the code (this idea of choosing a good crossover point has been explored in the technique called *context-aware crossover* [16]). Furthermore, it is possible that phenotypically identical material is being crossed over. This has a positional bias due to shorter regions needing to be equivalent towards the end of the active region.

When compared with the mutation events, the positive crossover events demonstrate a very different pattern of behaviour. Crossover events towards the end of the active region are much more likely to produce a positive effect than those towards the beginning. This suggests that crossover is not simply acting as another form of mutation operator; the code crossed in from the second parent

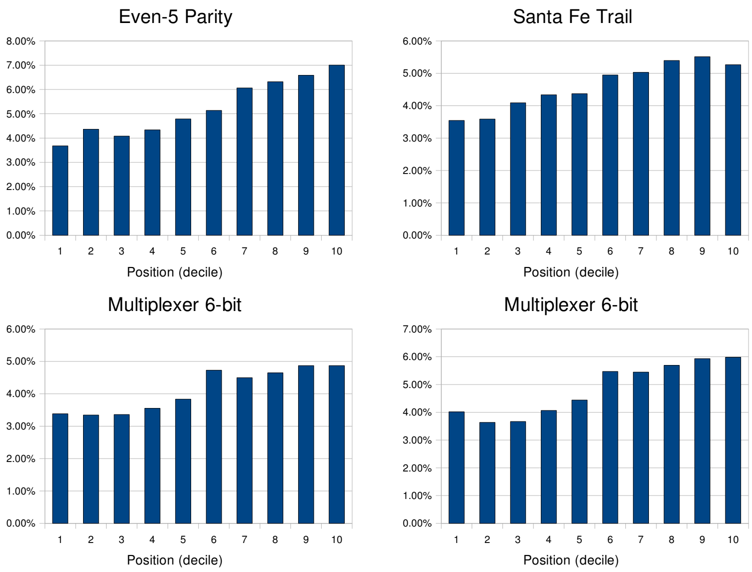

Fig. 4. Proportion of crossover events in each decile that had a positive effect on fitness. Using the *averaged parent fitness* definition of positive crossover.

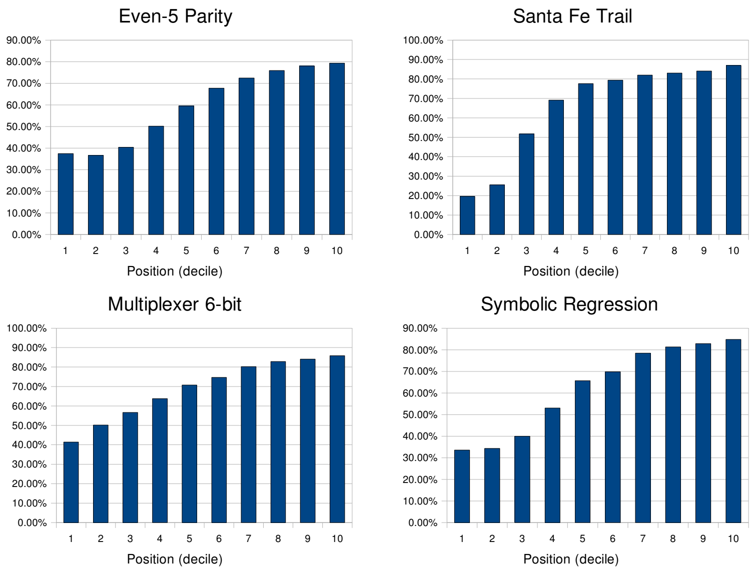

Fig. 5. Proportion of crossover events in each decile that had no effect on fitness. Using the *averaged parent fitness* definition of neutral crossover.

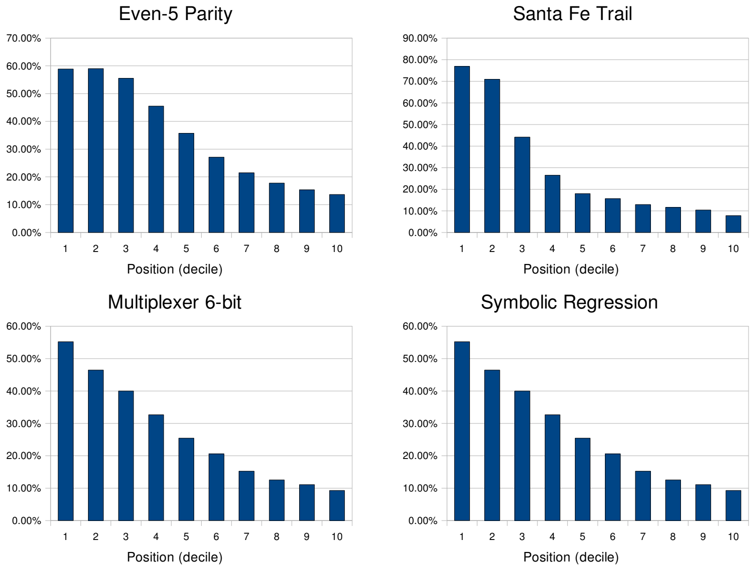

Fig. 6. Proportion of crossover events in each decile that had a negative effect on fitness. Using the *averaged parent fitness* definition of negative crossover.

is having a positive effect on the fitness, especially if it makes a smaller change by being towards the end of the active region.

One surprising feature of this analysis is how large a proportion of crossovers are neutral, given that this requires the child to have *exactly* the same fitness as the average of the two parents. This seems unlikely to occur as a result of a crossover event that takes two parents with different fitnesses; it would seem more likely that these are the result of identical or identical-fitness parents crossing over. An interesting piece of future work would be to do a further analysis in which these events were considered separately, as in [6].

By contrast with the mutation results, these suggest that a positional bias in crossover towards the end of the active region would be an obvious improvement, as this should increase the number of positive events and decrease the number of negative events. However, it is perhaps important not to eliminate too many negative events, as otherwise this will simply reduce the search to a hillclimber, which is unlikely to be effective in such a complex search space.

This analysis is made more complex, however, when we look at the alternative definition of positive-neutral-negative events using the *change to first parent* definition, i.e. the child is compared to the first of its parents. The results for this are given in figures 7, 8 and 9. In these results, the effects of crossover have very similar trends to those obtained using mutation, the most notable change being the larger proporion of positives in the first decile for three of the problems.

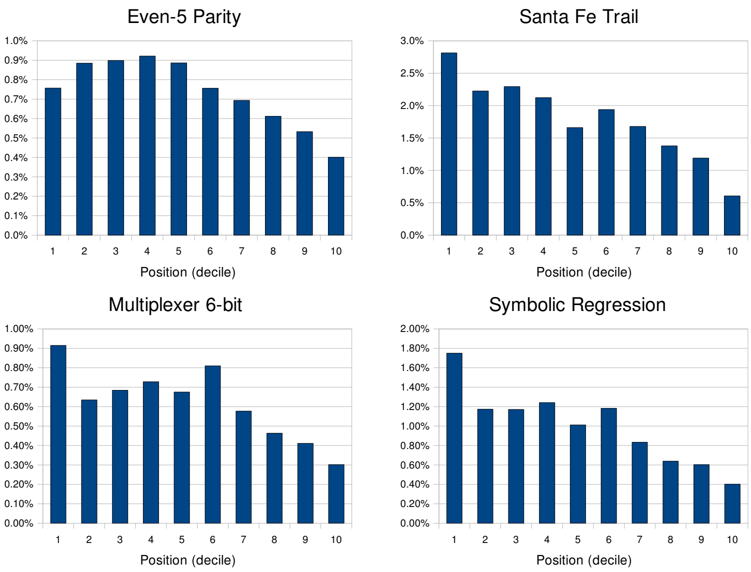

Fig. 7. Proportion of crossover events in each decile that had a positive effect on fitness. Using the *change to first parent* definition of positive crossover.

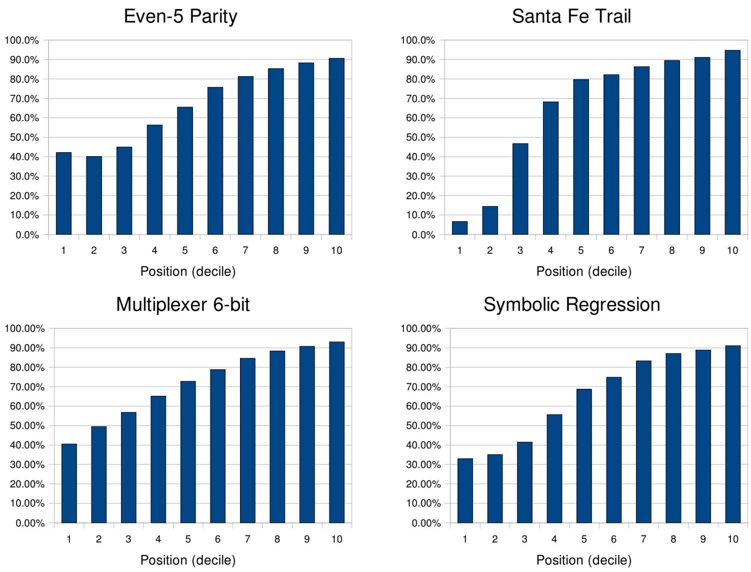

Fig. 8. Proportion of crossover events in each decile that had no effect on fitness. Using the *change to first parent* definition of neutral crossover.

By contrast with the results from the previous definition, these results suggest that crossover is acting as a kind of mutation operator.

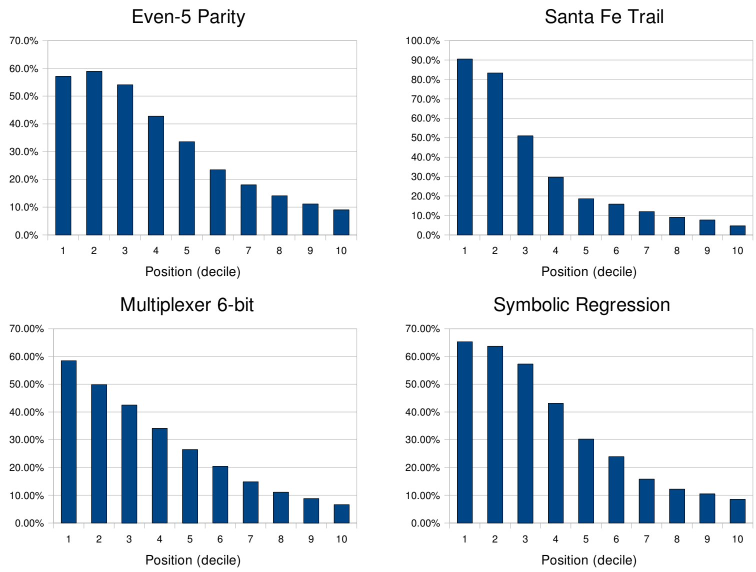

Fig. 9. Proportion of crossover events in each decile that had a negative effect on fitness. Using the *change to first parent* definition of negative crossover.

Therefore, it is hard to draw any definitive conclusions about crossover from these analyses—it appears that this depends highly on the specific definition of positive/neutral/negative chosen.

5 Future Work

Future work includes examining how these effects change through the generations of the GE run, looking at a wider range of definitions for constructive and destructive operators, and, finally, designing and testing new operators based on what we have learned from our analysis.

References

1. McKay, R.I., Nguyen, X.H., Whigham, P.A., Shan, Y.: Grammars in genetic programming: A brief review. In Kang, L., Cai, Z., Yan, Y., eds.: Progress in Intelligence Computation and Intelligence: Proceedings of the International Symposium on Intelligence, Computation and Applications, Wuhan, PRC, China University of Geosciences Press (2005) 3–18

2. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation **5** (2001) 349–358
3. Rothlauf, F., Oetzel, M.: On the locality of grammatical evolution. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A., eds.: Proceedings of the 9th European Conference on Genetic Programming. Volume 3905 of Lecture Notes in Computer Science., Budapest, Hungary, Springer (2006) 320–330
4. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
5. Nordin, P., Francone, F., Banzhaf, W.: Explicitly defined introns and destructive crossover in genetic programming. In Rosca, J.P., ed.: Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, Tahoe City, California, USA (1995) 6–22
6. Johnson, C.: Genetic programming crossover: Does it cross over? In Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M., eds.: Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009. Volume 5481 of LNCS., Tuebingen, Springer (2009) 97–108
7. Nordin, P., Banzhaf, W.: Complexity compression and evolution. In Eshelman, L., ed.: Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), Pittsburgh, PA, USA, Morgan Kaufmann (1995) 310–317
8. Teller, A., Veloso, M.: PADO: A new learning architecture for object recognition. In Ikeuchi, K., Veloso, M., eds.: Symbolic Visual Learning. Oxford University Press (1996) 81–116
9. Harper, R., Blair, A.: A self-selecting crossover operator. In Gary G. Yen et al., ed.: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, Vancouver, IEEE Press (2006) 5569–5576
10. Harper, R., Blair, A.: A structure preserving crossover in grammatical evolution. In David Corne et al., ed.: Proceedings of the 2005 IEEE Congress on Evolutionary Computation. Volume 3., Edinburgh, UK, IEEE Press (2005) 2537–2544
11. Keijzer, M., Ryan, C., O'Neill, M., Cattolico, M., Babovic, V.: Ripple crossover in genetic programming. In Miller, J.F., Tomassini, M., Lanzi, P.L., Ryan, C., Tettamanzi, A.G.B., Langdon, W.B., eds.: Genetic Programming, Proceedings of EuroGP'2001. Volume 2038 of LNCS., Lake Como, Italy, Springer-Verlag (2001) 74–86
12. O'Neill, M., Ryan, C.: Crossover in grammatical evolution: A smooth operator? In Poli, R., Banzhaf, W., Langdon, W.B., Miller, J.F., Nordin, P., Fogarty, T.C., eds.: Genetic Programming, Proceedings of EuroGP'2000. Volume 1802 of LNCS., Edinburgh, Springer-Verlag (2000) 149–162
13. Hugosson, J., Hemberg, E., Brabazon, A., O'Neill, M.: An investigation of the mutation operator using different representations in grammatical evolution. In: 2nd International Symposium "Advances in Artificial Intelligence and Applications". Volume 2., Wisla, Poland (2007) 409–419
14. Tackett, W.A.: Greedy recombination and genetic search on the space of computer programs. In Whitley, L.D., Vose, M.D., eds.: Foundations of Genetic Algorithms 3, Estes Park, Colorado, USA, Morgan Kaufmann (1994) 271–297 Published 1995.
15. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999)
16. Majeed, H., Ryan, C.: Using context-aware crossover to improve the performance of GP. In Maarten Keijzer et al., ed.: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation. Volume 1., Seattle, Washington, USA, ACM Press (2006) 847–854