An Analysis of the Behaviour of Mutation in Grammatical Evolution

Jonathan Byrne, Michael O'Neill, James McDermott, and Anthony Brabazon

Natural Computing Research & Applications Group
University College Dublin, Ireland
{m.oneill,anthony.brabazon}@ucd.ie

Abstract. This study attempts to decompose the behaviour of mutation in Grammatical Evolution (GE). Standard GE mutation can be divided into two types of events, those that are structural in nature and those that are nodal. A structural event can alter the length of the phenotype whereas a nodal event simply alters the value at any terminal (leaf or internal node) of a derivation tree. We analyse the behaviour of standard mutation and compare it to the behaviour of its nodal and structural components. These results are then compared with standard GP operators to see how they differ. This study increases our understanding of how the search operators of an evolutionary algorithm behave.

1 Introduction

Much attention has been directed towards the behaviour of crossover in Grammatical Evolution due to the traditional importance placed on this search operator in Genetic Programming [1] in general (e.g., [2,3,4]). However, aside from simple studies which examined mutation rates, there has been little analysis of the behaviour of mutation on search in GE. The notable exceptions to this include Rothlauf and Oetzel's locality study on binary mutation [5], a study comparing performance of binary and integer forms of mutation [8] and our first study on standard GE mutation [11]. This study extends this research by examining how each behavioural component of mutation moves the solution through the problem space. Search operators are a key component of any genetic and evolutionary computation representation, and as such it is critical that we understand their behaviour. This study addresses this important research gap by conducting an analysis of the behaviour of GE's mutation operator, focusing on the types of changes that occur when it is applied, and their impact on evolutionary performance.

The remainder of the paper is structured as follows. Firstly, the related research in this area is discussed in Section 2. A brief explanation on the locality of binary mutation is provided in Section 3 before an analysis of the behaviour of mutation in GE is undertaken in Section 4. Three separate experiments are carried out and their results are discussed in Sections 5, 6 and 7. In light of the results further analysis is described in Section 8, before finishing the paper in Section 9 with Conclusions and Future Work.

A.I. Esparcia-Alcazar et al. (Eds.): EuroGP 2010, LNCS 6021, pp. 14–25, 2010.

2 Related Research

In a recent study examining the locality of the mutation operator in Grammatical Evolution it was found that in some cases (less than ten percent of the time) mutation events resulting in small changes to the genotype can result in large changes to the structures generated [5]. More specifically, given a single unit of change at the genotype level (i.e., a bit flip), changes of one unit or greater at the derivation tree level occurred approximately ten percent of the time. 14% of these had a distance of greater than 5 units at the tree level. A unit of change at the phenotypic tree level corresponded to tree edit distance calculations which included deletion (delete a node from the tree), insertion (insert a node into the tree) and replacement (change a node label) change types. It is worth stating that the other 90% of the time mutation has no effect due to the many-to-one mapping adopted in GE which allows multiple values to correspond to the same production rule choice. The genotype change therefore is neutral upon phenotype structure and fitness in these cases.

In this paper we turn our attention to what is occuring that critical 10% of the time when a unit of change arising from mutation at the genotype level does not perfectly correspond with a unit of change at the phenotype level. We wish to establish if it is possible to design a mutation-based search operator that exhibits better properties of locality than the one currently adopted in GE.

3 A Component-Based View of Mutation in GE

In order to expose the impact of mutation on derivation tree structure we design a simple grammar, which uses binary rule choices. This allows us to condense codons (elements in the string representing the individual) to single bits. This simplifies our analysis without loss of generality to more complex grammars with more than two productions for each non-terminal.

Below is a simple binary grammar which might be used in a symbolic regression type problem with two variables (`x` and `y`).

```
<e> ::= <o><e><e>  (0)
      | <v>        (1)

<o> ::= +          (0)
      | *          (1)

<v> ::= x          (0)
      | y          (1)
```

We can then construct genomes with binary valued codons to construct sentences in the language described by the above grammar. Consider all genomes of length two codons (2^2 of them) and draw an edge between genomes that are a Hamming distance of one apart. If we then present the corresponding partial derivation trees resulting from those genomes we see the arrangement outlined in

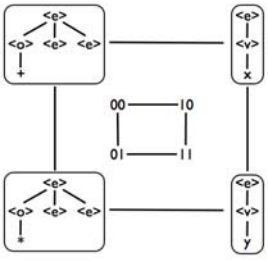

Fig. 1. The 2D neighbourhood for the example grammar (i.e., using the first two codons)

Fig. 1. In this particular example we see that a mutation event at the first codon corresponds to a new derivation tree structure. Here we define a new derivation tree structure as being one that has changed in length, that is, it contains more non-terminal symbols than its neighbour. Mutations from `00` to `10` (and vice versa) and from `01` to `11` (and vice versa) result in these structural changes. Whereas the remaining mutation events result in node relabelling.

Extending the genomes by an additional codon we can visualise the Hamming neighbourhood between the 2^3 genomes both in terms of codon values and partial phenotype structures. These are illustrated in Fig. 2. Again, we see a clear distinction between mutation events that result in structural and non-structural modifications.

Mapping these codons back to the grammar we see that structural mutations occur in the context of a single non-terminal symbol, `<e>`. We can see from this grammar that this non-terminal alone is responsible for structural changes, as it

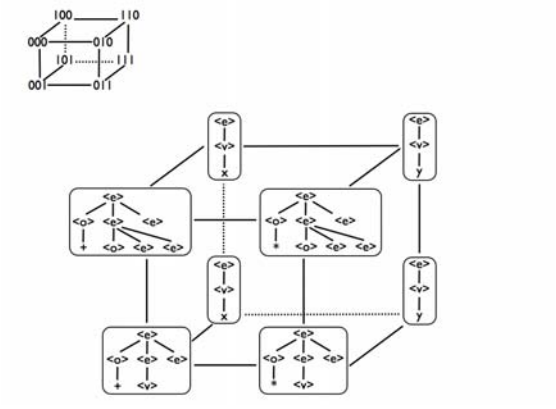

Fig. 2. The 3D neighbourhood for the example grammar (i.e., using the first three codons)

alone can increase the size of the developing structure. The rules for the <o> and <v> non-terminals are non-structural as they simply replace an existing symbol without changing structural length.

Effectively we can now decompose the behaviour of mutation into two types of events. The first are events that are structural in their effect and the second are those which are nodal in their effect. By logical extension we could consider both types of events as operators in their own right, and therefore define a *structural mutation* and a *nodal mutation*. It should be noted, however, that this is a specialisation of standard GE mutation, as it is possible for both types of events to occur during standard application of GE mutation to an individuals genome. *Perhaps the locality of mutation could be improved by simply reducing the number of occurances of the structural form of mutation, or even removing this form of mutation completely?*

If mutation was the sole search operator employed in a GE search, its elimination would have the consequence of removing structural change and structural search. This of course should have detrimental consequences for search as in Genetic Programming we must explore both structures and their contents. Part of the strength of GP approaches as problem solvers is their ability to search variable-length structures, so the removal of this ability would be undesirable. The following analysis and experiments seek to determine the relative importance of these behavioural components of mutation and begin to answer these kinds of questions.

4 An Analysis of Mutation in GE

To show the impact of decomposing mutation into its constituent parts we will look at how well each component of the operator performs on the Max problem. The aim of the Max problem is to generate a tree that returns the largest real value within a set depth limit. The optimal solution to this problem is to have addition operators at the root of the tree so that it creates a large enough variable for multiplication to have an effect. This problem is considered difficult for GP as populations converge quickly on suboptimal solutions that are difficult to escape from, except through a randomised search [7]. As such, this problem should be amenable to the right form of mutation operator. The grammar for this problem is given below:

```
<expr> ::= <op> <expr> <expr> | <var>
<op>   ::= + | *
<var>  ::= 0.5
```

This problem is an interesting application for the new component behaviours of the mutation operator as it highlights the different methods each component uses for exploring the search space. The grammar is suitable for illustrating the mutational differences as it consists of one structural rule(<expr>) and one nodal rule(<op>). Its simplicity also removes any extraneous factors that could complicate the result. The Max problem also requires that every element of the

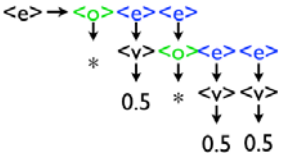

Fig. 3. An Example Max problem parse tree. Total =0.125.

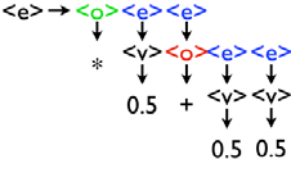

Fig. 4. Nodal mutation performed on parse tree. Total = 0.5.

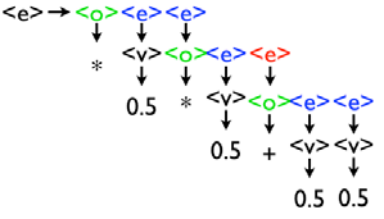

Fig. 5. Structural mutation performed on parse tree. Total = 0.25.

tree contributes to the final solution. i.e. there are no introns [7], which further removes any confounding factors.

4.1 Experimental Procedure

The experiments described in Sections s 5, 6 and 7 were implemented using GEVA[12,13], this is an open source framework for Grammatical Evolution in Java designed by the NCRA group in UCD. The following properties were kept constant for each experiment execution, these are as follows: Population size = 500, replication rate = 0.1, maximum generations = 50, the Mersenne Twister as the random number generator and fitness proportionate selection using the tournament selection operator with the tournament size set to 3. Generational replacement with an elite size of 1 was used as our replacement operator.

Wrapping was turned off as it could lead to conditions where a codon was both structural and nodal. The experiment was run using mutation exclusively as crossover would have had a confounding effect in combination with the mutation operators. A ramped half and half initialiser was used to create the derivation trees with an initialisation depth of 10. This equates to a phenotype tree of depth 8, the maximum depth allowed for this problem. This was necessary because nodal mutation by itself cannot alter the length of a phenotype and it required trees initialised to the maximum depth for a fair comparison.

5 Analysis of Mutations Effect on Search

This experiment investigates whether there is a statistically significant difference in performance between standard GE mutation and its structural and nodal

components. The two component behaviours have been implemented as operators in their own right so that we may measure the relative impact on the overall behaviour of integer mutation. We then examine whether these different components could have a beneficial impact on traversing the search space. Our experiment was carried out on the problem described above. As Nodal and Structural mutation act on subsets of the chromosome, the standard method of mutation had to be altered. Instead of applying the operator to each codon with a certain probability of mutation, only one mutation event was allowed per individual. This meant that each operator produced the same number of mutation events. 500 trial runs were carried out for each operator.

5.1 Experiment Results

The results from this experiment show that selectively altering subsets of codons from the chromosome can have a dramatic effect on how GE navigates the search space. The results are shown in the graphs below, see Figures 6 and 7. The fitness was based on a minimising function (1 over the result), so smaller results are better. As the results were on a logarithmic scale, a Wilcoxon rank-sum test (two-tailed, unpaired) was performed on the nodal and structural distributions that showed they were significantly different. The nodal component behaviour performs the best as it will explore the configurations of the particular tree structure to optimise it, something this problem requires to reach an optimal solution. Conversely structural changes does not explore the contents of the tree but instead explores configurations of derivation tree structures. This leads to poor performance on this problem. This is even more discernible when looking at the average fitness of the population (Figure 7). The fitness of standard mutation matches more closely with structural mutation suggesting it has a damaging effect on the nodal search component, changing the tree structure before it can be fully explored. This result clearly indicates that there are two separate behavioural components operating in standard GE mutation and it also indicates that they might not be complementary behaviours.

6 Comparison with GP Mutation

By splitting GE mutation into its components we now have two operators that are analogous to standard tree-based GP operators, point mutation and subtree mutation. Nodal mutation is identical in behaviour to point mutation so no comparison is needed but while subtree and structural mutation both explore the structure of trees there is a significant difference. There is a dependency on codon placement in GE as it uses a linear genome representation. As the genome is read from left to right, it means a change to a structural codon could change the meaning of the codons that follow it. In the most extreme case a single mutation early on in the genome could generate an entirely different tree. This effect is called ‘ripple’ mutation, and is similar in effect to ripple crossover [14]. Subtree mutation, on the other hand, replaces a subtree with another randomly

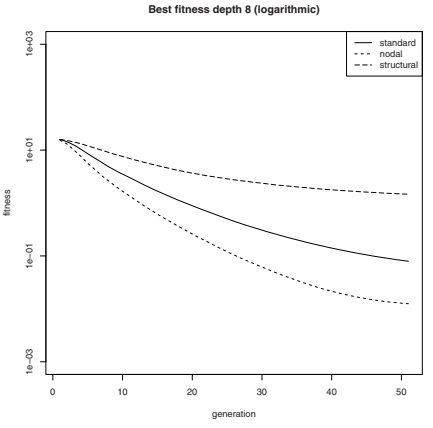

Fig. 6. Log of best fitness for GE operators on Depth 8 Max Problem

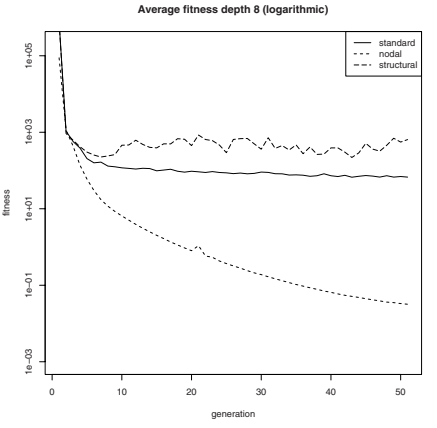

Fig. 7. Log of average fitness for GE operators on Depth 8 Max Problem

generated subtree, while leaving the rest of the tree intact [9]. In the following experiment we will apply standard tree-based mutation operators, point and subtree mutation, to the GE derivation tree representation. This allows us to compare a search of the structural space both with and without the ripple effect. We can then determine whether this ripple effect is advantageous to the search process. 500 trial runs were carried out for each operator.

6.1 Experiment Results

This experiment investigated how structural mutation performed against GP subtree mutation and whether the ripple effect was beneficial to the search process. A pairwise Wilcoxon rank-sum test was performed on the final results of both best and average fitness. It showed that all the results were significant except for nodal compared with standard mutation. Figure 8 shows that subtree mutation outperformed structural mutation. When the average individual fitness was examined it showed that despite subtree mutations better performance, most mutations created exponentially worse individuals during the course of the run (Figure 9). This shows that the ripple effect of structural mutation did not have a significantly detrimental effect on the search.

A further experiment was run where the maximum depth was set to 100 while leaving the maximum initialisation depth at 8. This in effect gave nodal mutation a significant disadvantage over the other operators as it cannot increase the derivation tree depth. The results for this experiment are shown in figures 10 and 11. Nodal mutation still performed better than structural and standard mutation but subtree mutation greatly outperformed it. Average fitness in subtree mutation also improved at the greater depth. Upon closer investigation the cause for this was found to be an explosive growth in the derivation tree depth and

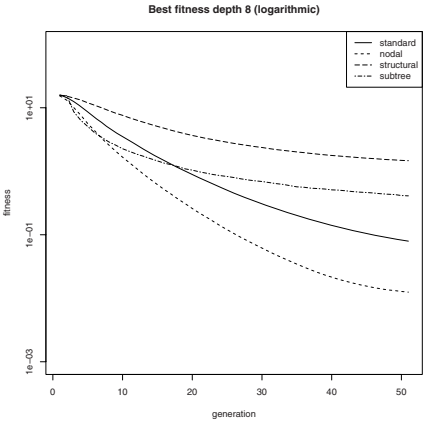

Fig. 8. Log of best fitness results for Depth 8 Max Problem

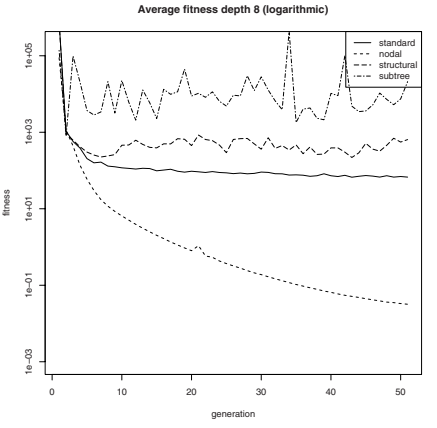

Fig. 9. Log of average fitness results for Depth 8 Max Problem

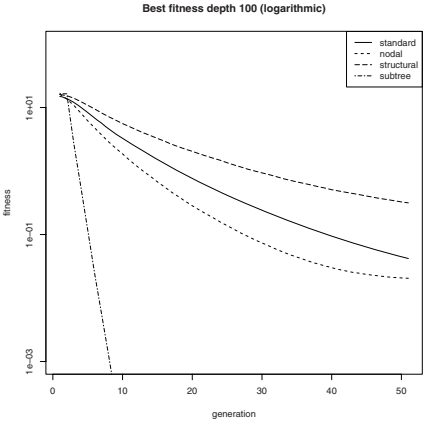

Fig. 10. Log of best fitness results for depth 100 Max Problem

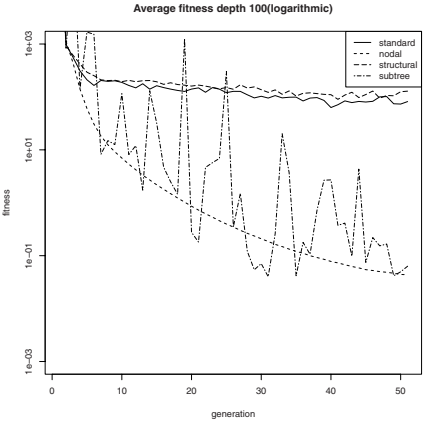

Fig. 11. Log of average fitness results for Depth 100 Max Problem

used codon length. This is shown in figures 12 and 13. Subtree mutation approached the problem by exploiting the fact that it makes larger subtrees. This meant that while it generated better mutations as regards fitness there was also a significant price to be paid in code bloat. Nodal mutation instead took the approach of optimising the structures present in the population to correctly target the position of the terminal set of operators(+,*). This led to the creation of highly efficient trees that could compete with subtree results despite their depth limitation.

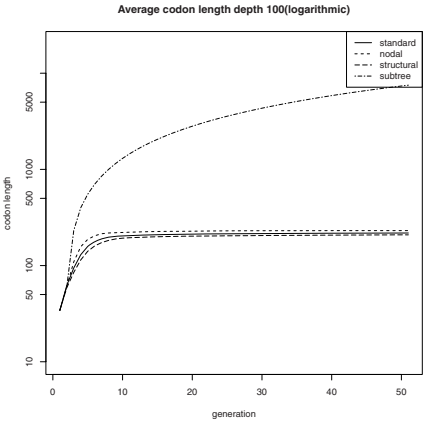

Fig. 12. Average codon length for depth 100 Max Problem (logarithmic)

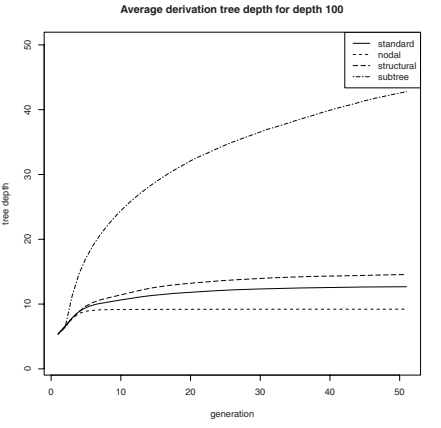

Fig. 13. Average derivation tree depth for Depth 100 Max Problem

7 Analysis of Mutation Events on Fitness

The previous experiments show how each operator performs during the course of a run but they do not have the granularity to see what was actually happening during each mutation event. Next we conducted an experiment in order to undertake an analysis of the operators' impact on individual independent mutation events. In previous studies [5] changes to the derivation tree were recorded but as we know what kind of phenotypic impact our operators have, we instead look at changes to the fitness of the individual. The experiment was run using the Max Problem with the same settings as described in Section 4.1. The mutation was carried out on a random codon in a randomly initialised individual. When a mutation occurred the change in fitness was recorded and then the codon was returned to its original value. Any mutation events that broke the depth limit were not counted. If a mutation created an invalid individual then this was recorded but no penalty was added to the fitness. This was continued until a sample size of 1,000,000 mutations was gathered for each operator. This experiment does not intend to show how these search operators work in practice. For example the quantity of bad fitness generated by an operator is generally irrelevant as it is bred out of the population(although it reduces the efficiency of the mutation operator). Conversely, a good mutation is always of benefit to a population regardless of its size. There is also a potential interplay of the operators as they are components of standard mutation and multiple codon sites can be mutation during standard application. Instead what these metrics do is focus on some of the mechanics at play during mutation.

7.1 Experiment Results

This experiment attempts to highlight the effects of individual mutation events on the overall fitness of an individual. The results are shown in table 1. It should be noted that over half the time the GE mutation operators generated a neutral mutation. This was because of the structure of the grammar. Each rule had two possible outcomes, mutation would either change the outcome or leave it untouched. Nodal and subtree mutation did not generate any invalids. In nodal mutations case it is because it is incapable of making them whereas it is in the definition of subtree mutation that it cannot generate invalids, so it would keep generating new subtrees until a valid one was produced. This factor also meant that subtree produced a large amount of neutral mutations as many of the alternatives would have made invalid individuals.

Nodal generated the greatest number of beneficial mutation events followed by subtree, standard and structural. As for the quality of these mutations, subtree outperformed the other operators by a factor of three. Nodal also produced the greatest number of bad mutations, followed by standard, and structural with subtree only producing bad mutations 2.9% of the time, unfortunately each bad subtree mutation was massively worse than the others. Overall nodal mutation was the least destructive operator, rarely generating large changes but cumulatively generating the greatest fitness change. Structural performed far worse than nodal but it was still not nearly as destructive as subtree mutation. Upon further investigation into the poor results for average fitness, as shown in Fig 9, it was found that the variance in average fitness increased exponentially at certain points during the run. This would indicate that subtree mutation occasionally produced very unfit subtrees that, depending on the root node of the tree, could make the overall fitness far worse.

Table 1. Results for mutation events on the Max Problem

Mutation Op	Good	Bad	Neutral	Invalid	Avr. Good	Avr. Bad	total Change
Standard	10.1%	26.2%	51.1%	12.6%	9.857	322.648	-138,580,712
Structural	5.2%	24.5%	51.6%	18.6%	11.385	1040.559	-137,691,951
Nodal	18.5%	31%	50.5%	0	10.462	63.106	-17,648,412
Subtree	11.4%	2.9%	85.6%	0	28.865	75461.321	-2,222,353,022

8 Discussion

This study highlighted the two contradictory components of standard GE mutation. The nodal behaviour was more akin to the standard GP point mutation as phenotypic changes consisted of replacing one terminal for another, leaving the derivation tree untouched. This could be beneficial in GE, where the outcome of a particular rule choice is dependent on all the choices that precede it. The behaviour of structural mutation was more explorative, creating variations of the tree structure itself. The results from the comparison with subtree mutation showed that it had a comparable effect on search despite the possibility of the

ripple effect changing every codons meaning. Even though it did not perform as well as subtree at the greater depth limit it showed that it was not as susceptible to code bloat as it explored the structure space of the trees.

The point of this paper was not to show which was the best operator, something which is problem dependent anyway. Instead it was to decompose GE's standard mutation operator into it's component behaviours. Despite our intentions, this information could now be used to create a mutation operator that applies these behaviours to a problem as and when they are needed. We have also applied this analysis on a range of other benchmark problems with similar results. The use of intelligent operators might help GE escape local optima as well as find the optimal solution more efficiently. This could be of particular benefit in dynamic environments where an evolutionary algorithm must be able to adjust quickly in response to a changing fitness function [10].

9 Conclusion and Future Work

This study analysed the behavior of mutation in GE. We initially described standard mutation in GE and then broke it into two components which we called Structural and Nodal mutation. We then investigated the effects of these mutation operators as applied to the Max Problem to ascertain if there was any discernible difference between these components. We then compared the operators against GP subtree mutation. Further investigation was carried out on the impact of individual mutation events.

Future work will involve using this information to design a better approach to mutation in GE. This could involve switching from one type of mutation to the other during the course of the run. It is natural to expect that the importance of different types of operators will vary over the course of a run as well as from problem to problem. In this respect we hypothesise that we would observe a greater difference in performance between the different forms of mutation during the end-phase of a run versus the mid and early-phases. In the later phase of a run we would hypothesise that nodal mutation might have a more positive impact than during the early phase, and we would hypothesise the opposite behaviour for structural mutation.

Acknowledgments

I would like to thank Erik Hemberg for his support and I would also like to thank SFI.This research is based upon works supported by the Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

1. Poli, R., McPhee, N.F., Langdon, W.B.: A Field Guide to Genetic Programming (2008), `http://lulu.com`, `http://www.gp-field-guide.org.uk`
2. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Dordrecht (2003)

3. O'Neill, M., Ryan, C., Keijzer, M., Cattolico, M.: Crossover in Grammatical Evolution. Genetic Programming and Evolvable Machines 4(1), 67–93 (2003)
4. Harper, R., Blair, A.: A Structure Preserving Crossover in Grammatical Evolution. In: Proc. CEC 2005 IEEE Congress on Evolutionary Computation, vol. 3, pp. 2537–2544. IEEE Press, Los Alamitos (2005)
5. Rothlauf, F., Oetzel, M.: On the Locality of Grammatical Evolution. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 320–330. Springer, Heidelberg (2006)
6. Langdon, W.B., Poli, R.: An Analysis of the MAX Problem in Genetic Programming. In: Proceedings of the second annual conference, Stanford University, July 13-16, pp. 222–230. Morgan Kaufmann Pub., San Francisco (1997)
7. Langdon, W.B., Poli, R.: An analysis of the MAX problem in genetic programming. Genetic Programming, 222–230 (1997) (Citeseer)
8. Hugosson, J., Hemberg, E., Brabazon, A., O'Neill, M.: An investigation of the mutation operator using different representations in Grammatical Evolution. In: Proc. 2nd International Symposium Advances in Artificial Intelligence and Applications, vol. 2, pp. 409–419 (2007)
9. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. The MIT press, Cambridge (1992)
10. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. Springer, Heidelberg (2009)
11. Byrne, J., O'Neill, M., Brabazon, A.: Structural and nodal mutation in grammatical evolution. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 1881–1882. ACM, New York (2009)
12. O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., Brabazon, A.: GEVA - Grammatical Evolution in Java (v1.0). UCD School of Computer Science Technical Report UCD-CSI-2008-09 (2008), `http://ncra.ucd.ie/geva/`
13. O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., Brabazon, A.: GEVA: Grammatical Evolution in Java. SIGEVOlution 3(2), 17–22 (2009)
14. Keijzer, M., Ryan, C., O'Neill, M., Cattolico, M., Babovic, V.: Ripple crossover in genetic programming. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tetamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, pp. 74–86. Springer, Heidelberg (2001)