Structural and Nodal Mutation in Grammatical Evolution

Jonathan Byrne, Michael O'Neill, James McDermott, Anthony Brabazon
Natural Computing Research & Applications Group
University College Dublin, Ireland
jonathanbyrn@gmail.com, m.oneill@ucd.ie, anthony.brabazon@ucd.ie

ABSTRACT

This study focuses on mutation in Grammatical Evolution and divides mutation events into those that are structural in nature and those that are nodal. A structural event being one that alters the length of the phenotype. A nodal event simply alters the value at any node of a derivation tree. We analyse and compare the effect of integer, nodal and structural mutations on fitness for randomly generated individuals before continuing this analysis to their relative problem-solving performance over full runs. The study highlights the importance of understanding how the search operators of an evolutionary algorithm behave. The result in this case being a form of mutation for Grammatical Evolution, node mutation, with a better property of locality than standard integer-based mutation, which does not discriminate between structural and nodal contexts.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]; I.2.2 [**Automatic Programming**]; F.4.2 [**Grammars and Other Rewriting Systems (D.3.1)**]

General Terms

Algorithms, Experimentation

Keywords

grammatical genetic programming, grammatical evolution, mutation, locality

1. INTRODUCTION

Much attention has been directed towards the behaviour of crossover in Grammatical Evolution (GE) 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 with the notable exception of Rothlauf and Oetzel's locality study on binary mutation [5] and a study comparing performance of binary and integer forms of mutation [6].

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 in turn helps us to design more efficient search operators. This study closes 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 it's impact on evolutionary performance.

The remainder of the paper is structured as follows. First some background on the locality of binary mutation is provided in Section 2 before an analysis of the behaviour of mutation in GE is undertaken in Section 3. Section 4 and Section 5 describes the experimental setup, experiments undertaken and results obtained. In light of the results further analysis is described in Section 7, before finishing the paper in Section 8 with Conclusions and Future Work.

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 can result in small changes to the genotype resulting in not so small 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 greater than one unit of change at the phenotypic tree level occured approximately ten percent of the time. 14% of these had a distance of greater than 5 unit 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 codon values to correspond to the same production rule choice. The genotype change therefore being neutral upon phenotype structure and fitness in these cases. It is not clear how invalid phenotypes (i.e., phenotypes that are incompletely mapped containing at least one non-terminal symbol) were handled in Rothlauf and Oetzel's study []. It is possible that any remaining non-terminal symbols were treated as different node content values in the tree distance calculations, but this is not exposed in their study.

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 is not perfectly correlated 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.

3. AN ANALYSIS OF MUTATION IN GE

To expose the impact of mutation on derivation tree structure we design a simple grammar, which adopts binary rule choices. This allows us to condense codons to single bits, which simplifies our analysis without loss of generality to more complex grammars with greater than two productions for each non-terminal.

Below is a simple binary grammar which might be used in the case of application to 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 example 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 an arrangement outlined in Fig. 1. In this particular example we see that a mutation event at the first codon corresponds in 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) results 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 genomes 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 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 categorise mutation events as being either structural or nodal, and by logical extension we could define two new types of mutation operator, *structural mutation* and *nodal mutation. 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 determinental 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. *Perhaps then crossover combined with a nodal form of mutation might produce more efficient search?* The following analysis and experiments seek to determine the relative importance of these forms of mutation and begin to answer these kinds of questions.

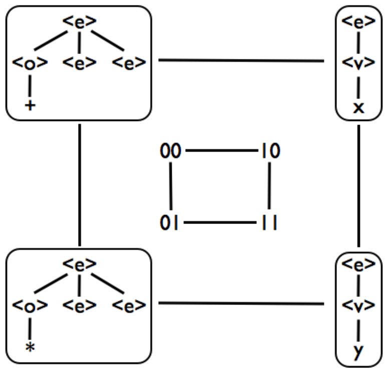

Figure 1: The 2D neighbourhood for the example grammar (i.e., using the first two codons).

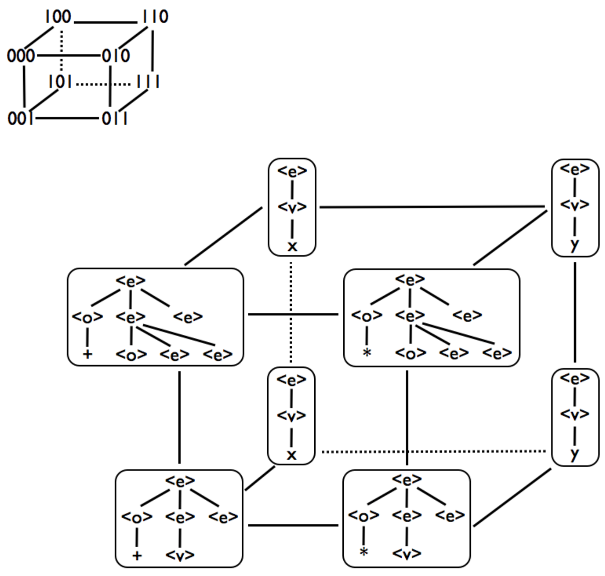

Figure 2: The 3D neighbourhood for the example grammar (i.e., using the first three codons).

4. EXPERIMENTAL PROCEDURE

This experiment was implemented using GEVA[7, 8], this is an open source framework for Grammatical Evolution in Java designed by the NCRA group in UCD. A number of properties were kept constant over the 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. Single point crossover was used for all crossover operations. Wrapping was turned off as it could lead to conditions where a codon was both structural and nodal. Generational replacement with an elite size of 1 was used as our replacement operator.

The fitness for Symbolic Regression was calculated using 50 randomly selected points between the range -1 and 1. The target function for Symbolic Regression is the polynomial $x + x^2 + x^3 + x^4 + x^5$. The fitness for the Santa-Fe ant trail is the amount of food collected by the time the energy is exhausted, and the fitness for the Word Match is the amount of characters correctly placed in the target word. The word used was "experimental". In the Even Five Parity problem the fitness was the hamming distance between the result and the correct value of the boolean function. In our experiment the GE version of ramped half-and-half initialisation was used, A Ramped full grow initialiser with the tree depth set to 10.

5. EXPERIMENT DESCRIPTION

5.1 Analysis of Mutations Effect on Chromosomal Fitness

This experiment conducted an analysis of the operator's impact on individual codon changes. The experiment was run against the four problems described above with the same settings. The fitness was recorded by generating a random individual and then traversing each codon with a 10% probability of mutation, when a mutation occurred the change in fitness was recorded and then the codon was returned to its original value. This meant that each mutation could be considered independent of those preceding it. 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 10,000 mutations was gathered for each operator. This experiment does not accurately reflect 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 and, conversely, a good mutation is always of benefit to a population regardless of its size. Instead these metrics attempt to show some of the mechanics at play during mutation.

5.2 Analysis of Mutations Effect on Search

This experiment attempted to investigate whether these different mutation operators could have a beneficial impact on traversing the search space. Our experiment was carried out on four problems described above. Every problem was tested without crossover and with crossover set to a probability of 0.9. For each test problem a combination of mutation operators were used, Nodal mutation, Structural mutation, Intflip mutation and a varying composite of Nodal and Structural Mutation. Each of these operators were tested using two different probabilities of mutation 0.01 and 0.1. In the case of variable mutation the mutation rate was moved proportionately from Structural to Nodal during the course of the run. At any point in the experiment the sum of the Structural and Nodal mutation probabilities equalled the overall probability. It must also be highlighted that the number of mutation events for Intflip ended up being higher as the mutation probability was applied to *every* codon, whereas the other operators were only applied to a subset of these codons.

6. EXPERIMENTAL RESULTS

6.1 Chromosomal Fitness Results

This experiment attempted to highlight the effect of selectively altering codons that correlated with either a terminal or non-terminal phenotype. The results from this experiment are shown in the barcharts below, see Figures 3, 4 and 5. It shows that on average Nodal mutation produced more good results of higher fitness. In the cases where IntFlip performed better there was also a disproportionate amount of bad mutations. The only exception to this was the fitness results for Symbolic Regression. The most likely cause of this is the difference in invalids being produced by the operators, a possibility also supported by the fact that overall it produced a greater number of good mutations. Nodal mutation, by its definition, cannot produce an invalid, only substitute one terminal for another. whereas the other operators produced more invalids on this problem than any other. The greater quantity of valid but bad mutations also worked against Nodal mutation because a sum of squared errors formula was used to calculate the fitness, this magnified any bad result.

These results shows that Nodal mutation was able to exploit an existing solution to maximise the fitness from it. Structural mutation performed worst overall with both the poorest fitness gain and the greatest number of invalids. While the creation of invalids is generally regarded as detrimental to the search process. It does at least show that Structural mutation is attempting to fully explore the search space, regardless of whether the result is beneficial to the population or not.

6.2 Search 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 6-23. Structural mutation by itself performs worst. As shown in the graphs it initially makes some improvement to the overall fitness but then it soon plateaus as shown in figs 14 and 22. This operator could also be affected by the initialisation depth which might have generated many of the solutions that it would have explored. Nodal mutation matched the performance of IntFlip on the Word Match and Even Five Parity problem and surpasses it at higher mutation rate. Nodal mutation has an advantage on these types of problems as much of the changes for improved fitness would be on the letters the nodes represent.

The Variable mutation operator matches the the best results of both the structural and nodal mutation operators for each experiment and for higher mutation rates it surpasses the Intflip operator on several of the experiments. In some cases it even surpasses nodal results which would

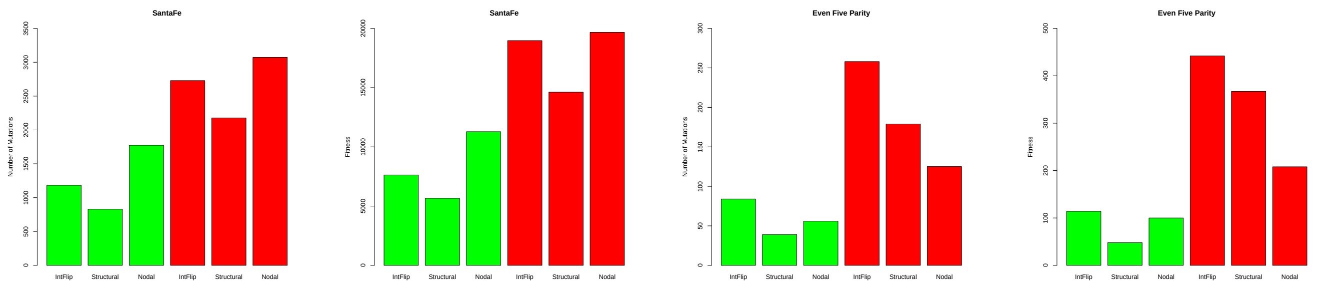

Figure 3: Comparison of the number of good versus bad mutations (far left) and fitness gain (middle left) for Santa Fe Ant trail, and the number of good versus bad mutations (middle right) and fitness gains (far right) for Even Five Parity. Good mutations have a positive fitness impact and are represented in green with bad mutations having negative impact and are shown in red. The order of the bars in each chart are as follows: integer (good), structural (good), nodal (good), integer (bad), strucutral (bad), nodal (bad).

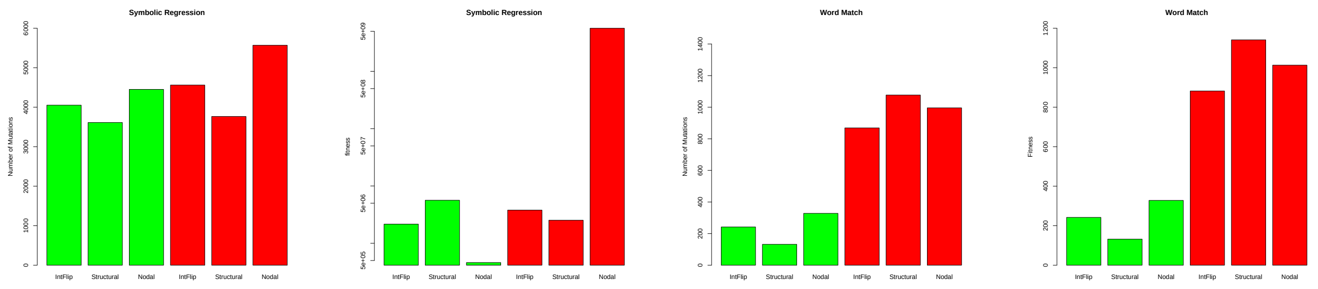

Figure 4: Comparison of the number of good versus bad mutations (far left) and fitness gain (middle left) for Symbolic Regression, and the number of good versus bad mutations (middle right) and fitness gains (far right) for Word Match.

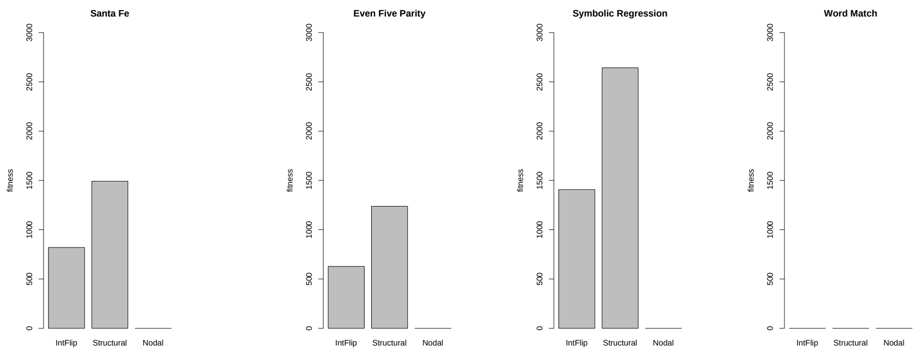

Figure 5: Number of invalids produced for each experiment

suggest that the combination of the two operators is of benefit. A pairwise T-Test was performed on the results and it showed that the Variable mutation operator offered a significant improvement over Intflip on both the Even Five Parity and Word Match problems. It should also be noted that increasing the mutation rate significantly improved the performance of the Nodal and Variable mutation operators. This result would either indicate that the smaller codon sets that these operators work on is distorting the results from IntFlip or that these operators might be less destructive than IntFlip.

7. DISCUSSION

It is thought that the main use of mutation is to fine-tune an existing solution. This may be the case in GAs as mutation may not greatly affect the solutions, but in GP small changes caused by mutation at root nodes can have a dramatic impact on the solution. This influence can be felt just as dramatically in GE, where the outcome of a particular rule choice is dependent on all the rules that preceded it. Nodal mutation was more akin to the standard GP mutation as phenotypic changes consisted of replacing one terminal for another. The analysis of chromosomal mutation showed that nodal mutation had a far greater number of fitness changes and would therefore be beneficial in fine tuning a solution.

On the other hand, Structural mutation did not act as a operation for exploiting a solution but instead as a technique for a more global exploration of the search space. While the benefits of structural mutation were not seen in the chromosomal experiments, it would be wrong to think that just because the changes it made didn't have an immediate effect that it had no effect at all. Preliminary results showed that a combination of Structural and Nodal mutations, used at different points of the search produced solutions at least as good as either of the operators individually and in some cases produced the best solutions of any operator.

This presents the intriguing idea that the lack of locality in the standard GE mutation could in fact help it to explore the search space more thoroughly and then as the general outline of a good solution has been selected, the direction of GE's mutation could be focused on the details of that solution to hone it down further

8. CONCLUSION & FUTURE WORK

This study analysed the behavior of mutation in GE. Three different types of mutation operator were investigated, Nodal, Structural and Integer mutation. We initially investigated the effects of these mutation operators at the chromosomal level to judge the impact of individual codon changes. An analysis was then conducted to find out how well each operator performed in practice on a set of problems. A new composite mutation operator consisting of Nodal and Structural mutation was also tested against these problems. The results indicated that the mutation operator had a distinct impact on evolutionary performance and in many cases improved performance over standard Integer mutation.

Future work will involve analysing the impact each of the mutation operators has on fitness at different stages of a run. In the first set of experiments presented in this paper we effectively analysed the relative impact of each of the types of mutation on the randomly created individuals of generation zero. It is natural to expect that the importance of different types of operators will vary over the course of a run. For example, early on in the run a more global search driven by coarser-grained operators might be acceptable. However, as the population converges upon the more productive areas of the search space it becomes more appropriate to adopt finer-grained search operators (i.e., operators which make smaller step sizes allowing the population to perform a local search in the region of interest). 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. Indeed, expecting to see any difference in performance for randomly generated individuals might be too much to expect and would go some way to explain the observations made in the experiments reported here. 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.

9. REFERENCES

[1] Poli R., McPhee N.F., Langdon W.B. (2008). A Field Guide to Genetic Programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

[2] O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers.

[3] O'Neill M., Ryan C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. Genetic Programming and Evolvable Machines, 4(1):67-93.

[4] Harper R., Blair A. (2005). A Structure Preserving Crossover in Grammatical Evolution. In Proc. CEC 2005 IEEE Congress on Evolutionary Computation, Vol.3 , pp.2537-2544. IEEE Press.

[5] Rothlauf F., Oetzel M. (2006). On the Locality of Grammatical Evolution. In LNCS 3905 Proc. of EuroGP 2006 the 9th European Conference on Genetic Programming, pp.320-330. Springer.

[6] Hugosson J., Hemberg E., Brabazon A., O'Neill M. (2007). 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.

[7] O'Neill M., Hemberg E., Gilligan C., Bartley E., McDermott J., Brabazon A. (2008). GEVA - Grammatical Evolution in Java (v1.0). UCD School of Computer Science Technical Report UCD-CSI-2008-09. Available from `http://ncra.ucd.ie/geva/`.

[8] O'Neill M., Hemberg E., Gilligan C., Bartley E., McDermott J., Brabazon A. (2009). GEVA: Grammatical Evolution in Java. SIGEVOlution, 3(2):17-22.

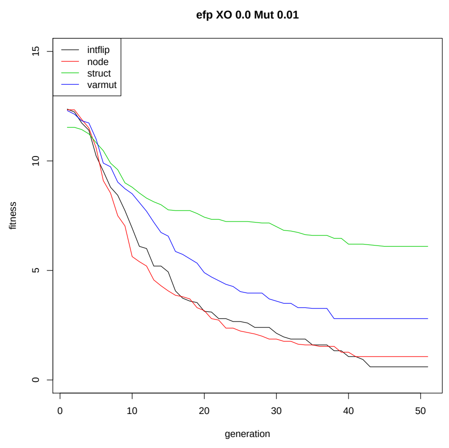

Figure 6: Results for Even Five Parity without crossover and mutation rate 0.01

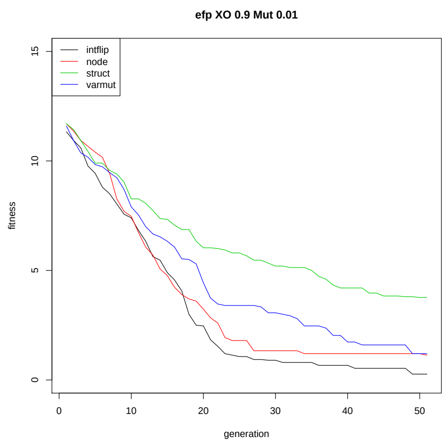

Figure 7: Results for Even Five Parity with crossover and mutation rate 0.01

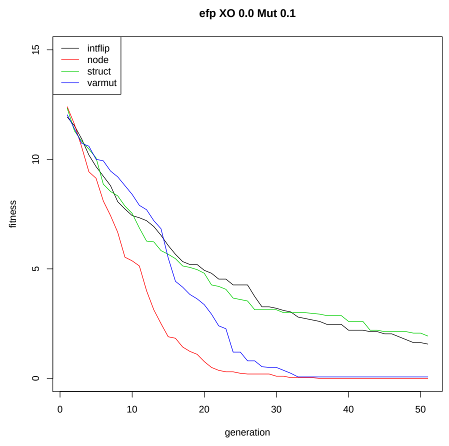

Figure 8: Results for Even Five Parity without crossover and mutation rate 0.1

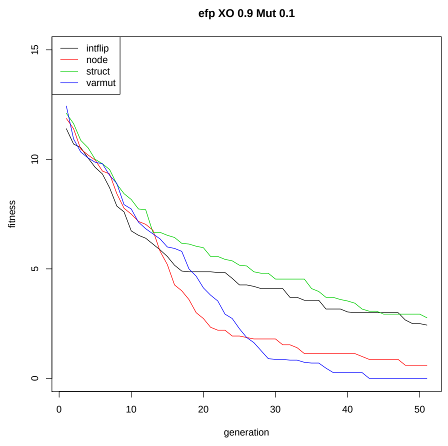

Figure 9: Results for Even Five Parity with crossover and mutation rate 0.1

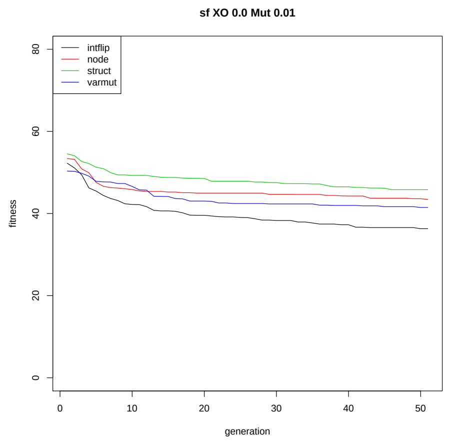

Figure 10: Results for Santa Fe without crossover and mutation rate 0.01

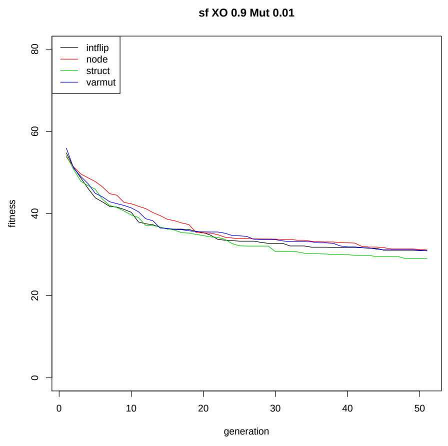

Figure 11: Results for Santa Fe with crossover and mutation rate 0.01

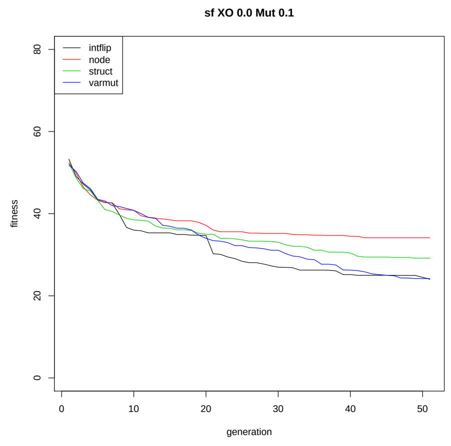

Figure 12: Results for Santa Fe without crossover and mutation rate 0.1

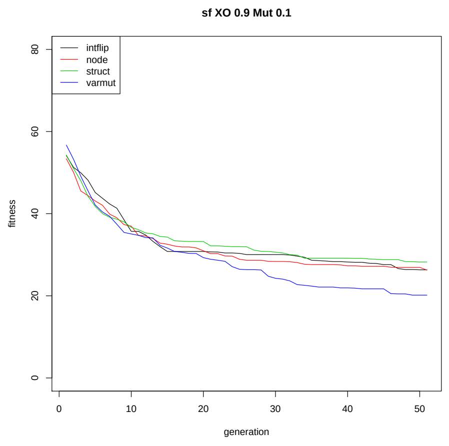

Figure 13: Results for Santa Fe with crossover and mutation rate 0.1

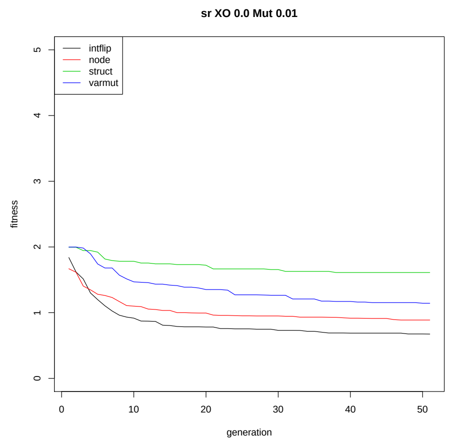

Figure 14: Results for Symbolic Regression without crossover and mutation rate 0.01

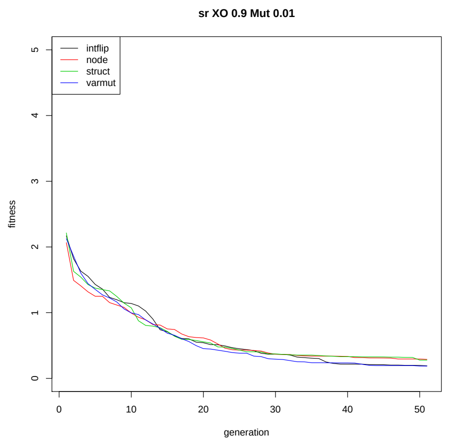

Figure 15: Results for Symbolic Regression with crossover and mutation rate 0.01

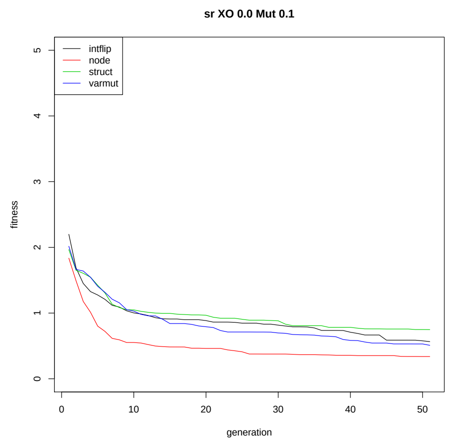

Figure 16: Results for Symbolic Regression without crossover and mutation rate 0.1

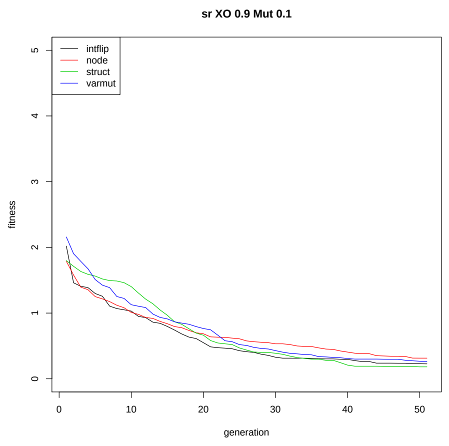

Figure 17: Results for Symbolic Regression with crossover and mutation rate 0.1

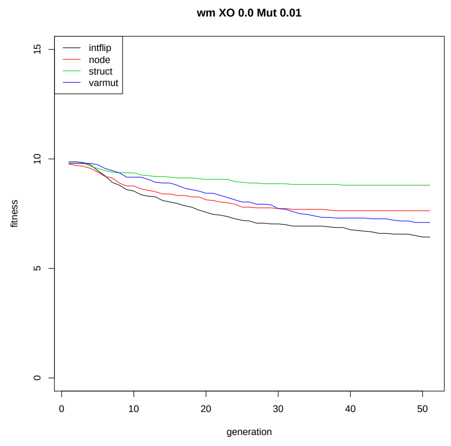

Figure 18: Results for Word Match without crossover and mutation rate 0.01

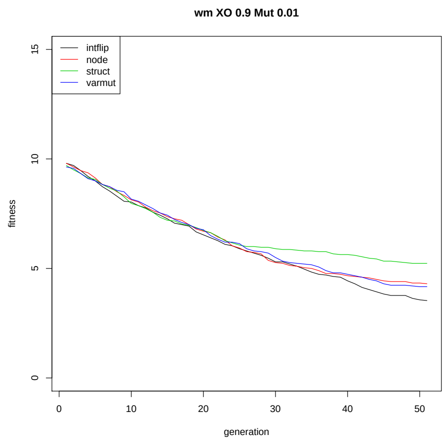

Figure 19: Results for Word Match with crossover and mutation rate 0.01

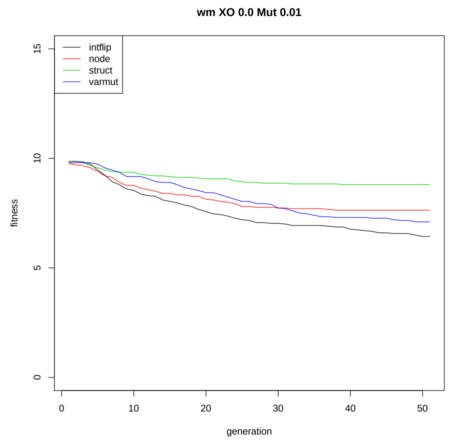

Figure 20: Results for Word Match without crossover and mutation rate 0.1

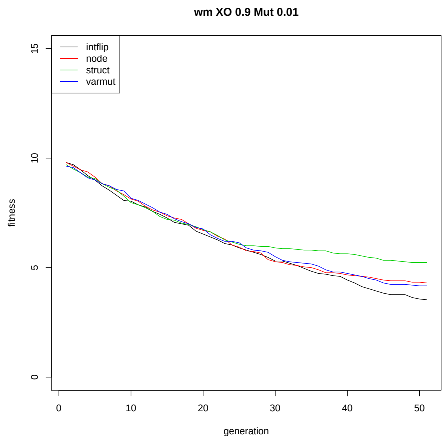

Figure 21: Results for Word Match with crossover and mutation rate 0.1

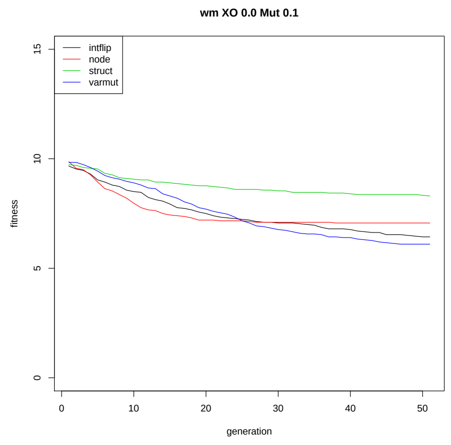

Figure 22: Results for Word Match without crossover and mutation rate 0.1

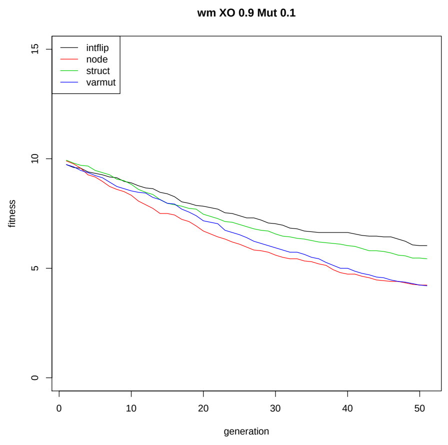

Figure 23: Results for Word Match with crossover and mutation rate 0.1