Implementing an Intuitive Mutation Operator for Interactive Evolutionary 3D Design

Jonathan Byrne, James McDermott, Edgar Galván-López, and Michael O'Neill

Abstract— **Locality - how well neighbouring genotypes correspond to neighbouring phenotypes - has been described as a key element in Evolutionary Computation. Grammatical Evolution (GE) is a generative system as it uses grammar rules to derive a program from an integer encoded genome. The genome, upon which the evolutionary process is carried out, goes through several transformations before it produces an output. The aim of this paper is to investigate the impact of locality during the generative process using both qualitative and quantitative techniques. To explore this, we examine the effects of standard GE mutation using distance metrics and conduct a survey of the output designs. There are two different kinds of event that occur during standard GE Mutation. We investigate how each event type affects the locality on different phenotypic stages when applied to the problem of interactive design generation.**

I. Introduction

How an algorithm explores and exploits a landscape is a key element during evolutionary search. Rothlauf [19], [20] has described and analysed the importance of locality in performing an effective evolutionary search of landscapes. Locality refers to how well neighboring genotypes correspond to neighboring phenotypes. Rothlauf's research distinguished two degrees of locality: low and high locality. A representation has high locality if all neighbouring genotypes correspond to neighbouring phenotypes. On the other hand, a representation has low locality if many neighboring genotypes do not correspond to neighboring phenotypes. Low locality is undesirable in an operator as it is akin to random search. Rothlauf demonstrates that a representation that has high locality is necessary for efficient evolutionary search.

One area where locality is beneficial to search is interactive design. If the user wants to indirectly influence a design through the use of an operator it must perform in an intuitive manner. This type of interactivity in the evolutionary process is categorised as active intervention [23]. If the user applies a small change to the genotype then they expect a small change in the phenotype. Low locality can also increase user fatigue as the user perceives it as a random search and will stop using the operator. In this study we examine the locality of the mutation operator when applied to designs in Grammatical Evolution (GE).

GE is a form of Genetic Programming where there is a mapping process from genotype to phenotype, similar to how our DNA encoding is transformed into our physical characteristics. This mapping process makes it different from standard GP as it goes though several transformations before it is finally output. This means that there are several different stages during the process i.e, where neighbours are mapped to non-neighbours. We will investigate the locality using distance metrics on different stages of the generative process.

Natural Computing Research & Applications Group, Complex and Adaptive Systems Lab, University College Dublin, jonathanbyrn@gmail.com

This paper is organised as follows. Section II is a summary of the related research in this area. A brief overview of Grammatical Evolution is given in Section III. The generative process involved in GE is discussed in Section IV. In Section V we describe the different components of standard GE mutation. In Section VI, we describe the distance metrics that will be used to analyse the different phenotypic stages in GE. We present and discuss our findings in Section VIII. Finally, in Section IX we discuss our conclusions and future work.

II. Related Research

Understanding of how well neighbouring genotypes correspond to neighbouring phenotypes is a key element in evolutionary search [17], [20]. In the abstract sense, a mapping has *locality* if neighbourhood is preserved under that mapping. In EC this generally refers to the mapping from genotype to phenotype [17]. Rothlauf [17] is perhaps the authoritative work on the topic of locality in EC. It and other work [20], [18] has shed new light on several problems. The definition of locality assumes that a distance measure exists on both genotype and phenotype spaces, and that for each there is a minimum distance, and that neighbourhood can defined in terms of minimum distance.

There has already been some investigations into locality in GE. Rothlauf and Oetzel's locality study on binary mutation [20] investigated how binary changes in the genome impacted the derivation tree. This study was extended to compare the performance of binary and integer forms of mutation [7]. There is also our previous work examining the locality of components of standard GE mutation [2].

The study of locality also has relevance to interactive evolutionary computation. Hart allowed the user to specify which type of mutation (and mutation rate) to apply because of the assumption that users sometimes have an intuition about what type of change is required to an individual [6]. The purpose of defining nodal and structural operators stems from our study of interactive design evolution [13]. During this study it was found that the user applied the mutation operator to explore areas of the design space that had produced interesting aesthetic designs. For mutation to be useful in this circumstance, it requires locality to be maintained from the genotype to the output.

```
<expr> ::= (<op><expr><expr>)          (0)
         | <var>                       (1)
<op>   ::= +                           (0)
         | -                           (1)
<var>  ::= x                           (0)
         | y                           (1)
```

Fig. 1. A simple BNF grammar

III. Grammatical Evolution

Grammatical Evolution is an evolutionary algorithm that is a grammar based form of GP [15]. It differs from standard GP by replacing the parse-tree based structure of GP with a linear genome. It generates programs by using a list of integers (also called a chromosome) to select rules from the grammar which are then applied to generate a program. The chromosome is made up of codons. Each codon in the string is used to select a production rule from a Backus Naur Form (BNF) grammar. The BNF represents a language in the form of production rules. Each rule is comprised of non-terminals that map to either terminals, non-terminals or both, depending on the production rules. A simple example BNF grammar that could be used for symbolic regression is given in Figure 1. `<expr>` is the start symbol from which all programs are generated. The grammar states that `<expr>` can be replaced with either one of `<expr><op><expr>` or `<var>`. An `<op>` can become either +, or -, and a `<var>` can become either x, or y.

The integer codons decide which rule is chosen by simply calculating the modulus of the codon value with the number of rules. This can be represented with the following formula:

$$Rule(idx) = Codon\ Value\ \%\ Num.\ Rules \tag{1}$$

By iterating through the codons the BNF rules are applied and a derivation tree is built. This in turn generates a string from the grammar which represents the program.

IV. Generative processes in EC

The genotype to phenotype mapping is a terminology that has been adapted from the field of biology. A genotype is an encoding upon which the evolutionary operators of mutation and recombination act. An example of this would be human DNA. Changes in the genotype are translated into changes in the phenotype. A phenotype is an observable characteristic of an individual. An example of this in humans would be eye-color or height. Selection is performed on the phenotype. This biological concept was introduced to the field of EC as a method for separating the search and solution space [1] and as a metaphor for describing the representations and mapping processes.

It could be argued that in traditional GP there is no generative process. The trees that define the programs are directly manipulated by the operators, therefore there is no generative stage. Although, this is not the complete picture. The tree itself could be viewed as a genotypic encoding and the output of the program as the phenotype. Furthermore, in Genetic Algorithms there is a mapping of the evolved integer string translating it into a meaningful application. This highlights that a genotype to phenotype to fitness mapping exists in all forms of evolutionary computation except the most basic.

In GE there are several stages in the mapping process, each with its own observable characteristics. As each stage also contains a version of the final instantiation, it can be classified as a phenotype. The three main phenotypic stages are shown in Figure 10. The integer list is translated into a derivation tree using a BNF grammar defined in Figure 1 and the mod rule. The terminal rules that make up the finished string are shaded. The string is then evaluated to produce the final output phenotype, in this case a 3 dimensional plot of a plane.

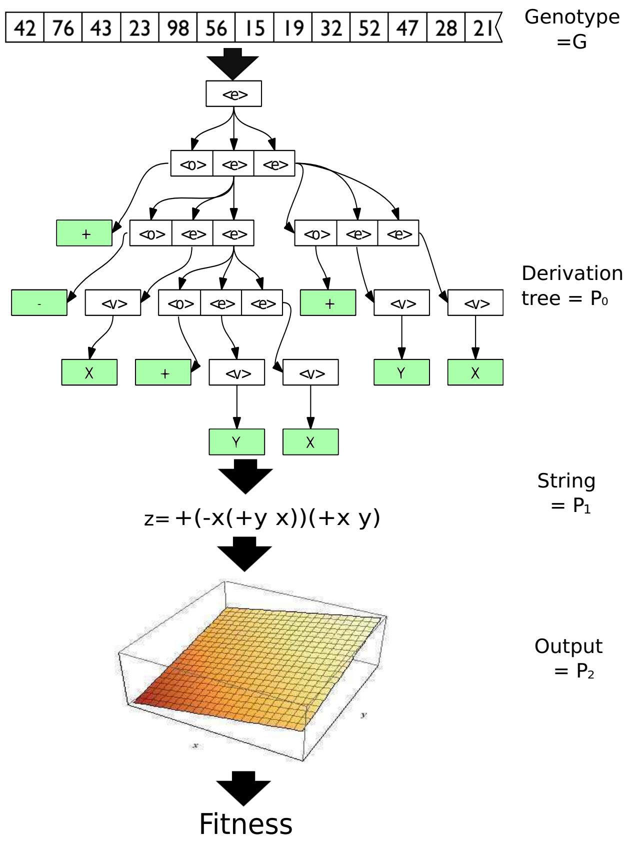

Fig. 2. stages of derivation in GE

To study locality, it is necessary to define a metric on the search space. In a genotype-phenotype mapping representation. In his work, Rothlauf claimed that for two different search spaces (e.g., genotypic and phenotypic search space) it is necessary to define two different metrics. As we are examining three different phenotypic stages we need to use different metrics for each one.

We are interested in determining how locality is affected on the different phenotypic stages of the generative process. In this study we apply a single change on the genotypic stage, the equivalent of a hamming distance of one, and observe the corresponding locality on each stage of derivation.

V. A COMPONENT-BASED VIEW OF MUTATION IN GE

Standard GE mutation can be divided into two types of events, those that are structural in nature and those that are nodal. A nodal mutation changes a single leaf of the derivation tree. A structural mutation changes one or more internal nodes of the derivation tree (and zero or more leaves). This can result in a change to the length of the phenotype. This work was originally shown in [3]. In order to expose the impact of mutation on derivation tree structure we will use the binary grammar as shown in Figure 1. This allows us to condense codons (elements in the string representing the individual) to single bits.

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. 3. In this particular example we see that a mutation event at the first codon impacts a non-terminal rule that 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.

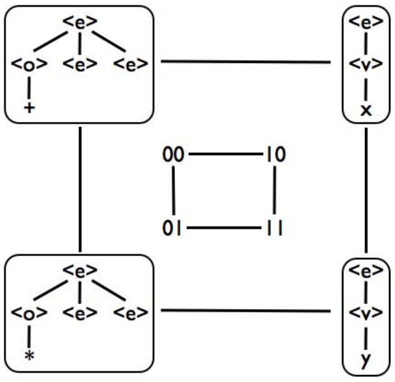

Fig. 3. The 2D neighbourhood for the example grammar (i.e., using the first two codons).

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. 4. 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 change the size of the developing structure. The rules

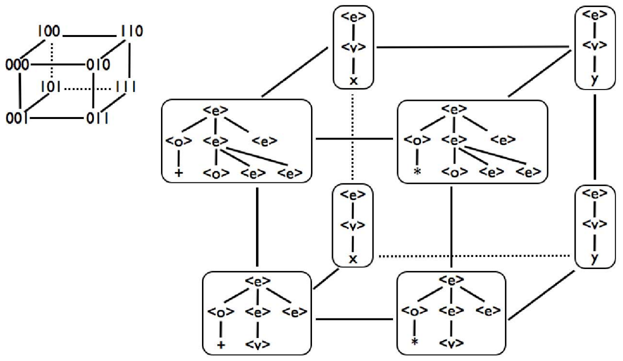

Fig. 4. The 3D neighbourhood for the example grammar (i.e., using the first three codons).

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 these events are only 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 individual's genome. In order to understand the impact of change arising from mutation events in GE we need metrics of distance between the different stages of the mapping as outlined earlier in Section IV.

VI. DISTANCE MEASURES FOR GRAMMATICAL EVOLUTION

In this section we review and provide motivation for three distance measures, namely, (a) tree edit distance, (b) levenshtein distance, and (c) normalised compression distance. Traditionally different metrics are used to analyse different types of operators but as we are studying different effects caused by the standard mutation operator and we intend to compare those effects, the same metric was used for both. We will later use these in our analysis reported in Section VIII.

A. Tree Edit Distance

O'Reilly [16] proposed an approach, called edit distance, with the main goal of having a metric that specifies the degree of dissimilarity between two individuals in the form of tree-like structures. The idea of edit distance is to calculate the minimum cost (number of moves) that is required to transform a given tree to a target tree step by step. For this purpose, the author defined the use of three types of edits: (a) Substitution: changing a node into another, (b) Insertion: adding a node within the tree and (c) Deletion: removing a node from the tree. O'Reilly's approach was inspired by the work reported in [21], [22]. This distance is notable because it is closely aligned with a mutation operator defined in the same paper. In GE this metric is applied to calculate a distance at the derivation tree stage of the mapping.

B. Levenshtein Distance

Levenshtein distance is a metric for comparing two strings, or in our case, generated computer programs. This has been used as a metric to compare representations in GP [9] [8]. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character. In our study the output string is tokenised and the levenshtein metric is applied to phenotype symbols.

C. Normalised Compression Distance

The NCD has been used as a distance measure for linear structures within EC [5]. The so-called "universal similarity metric" is a theoretical measure of similarity between any two data structures (for example strings), defined in terms of *Kolmogorov complexity* [10]. This is defined as the length of the shortest program which creates the given string. Informally, two strings are very similar if the Kolmogorov complexity of their concatenation is close to the complexity of just one of them. This idea was made practical by [4]: they approximated the (uncomputable) Kolmogorov complexity of a string by the length of its compressed version, as calculated by off-the-shelf compression software. The "normalised compression distance" or NCD is defined as follows:

$$d(x,y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))}$$

where x and y are two strings, xy is their concatenation, and the function C gives the length of the compressed version of its argument. This metric is applied to calculate a distance between phenotype strings.

VII. Experimental Procedure

In this section we examine the locality of phenotypic changes using the metrics described in Section VI and describe the setup that was used for the design survey. The experiment was implemented using GEVA [11], [12], this is an open source framework for Grammatical Evolution in Java designed by the NCRA group in UCD. We used a grammar which generates designs through the use of higher order functions. The designs are then rendered using Blender, an open source 3D modelling software. This allowed us to produce designs which could then be subjectively evaluated by users. The grammar used to generate the objects is shown in Figure 14 and may be downloaded from the GEVA website [11]. We may then use this to compare similarity at the output stage of the evolved program. Instead of executing runs in the traditional sense we carried out single mutation events (ie; a hamming distance of one on the genotype) on randomly generated individuals and recorded the change in output. As we required a large sample size of the mutation events where there was a phenotypic change, the experiment was allowed to run until 5000 non-neutral mutation events occurred. During the run nodal mutation produced 953 neutral mutation and structural mutation produced 2803, a ratio of nearly 3 to 1.

The distance metrics were then applied to record changes on the derivation and string stages of the phenotypes. We also randomly sampled 100 of the mutation events where both a structural and nodal change occurred. These designs were then rendered in Blender and a survey was taken as to how closely the mutated designs resembled the original. Survey participants are shown the original image and then presented with the same design after both a nodal mutation event and a structural mutation event (Figure 5). The participant evaluated 100 individuals and were given as much time as needed to complete the task The order that these images were shown was randomised so as to avoid bias. There was also the option to pass on a selection if no clear preference was presented. Although a users interpretation of similarity is a somewhat subjective metric, the survey allowed us analyse the semantic stage of the generated program

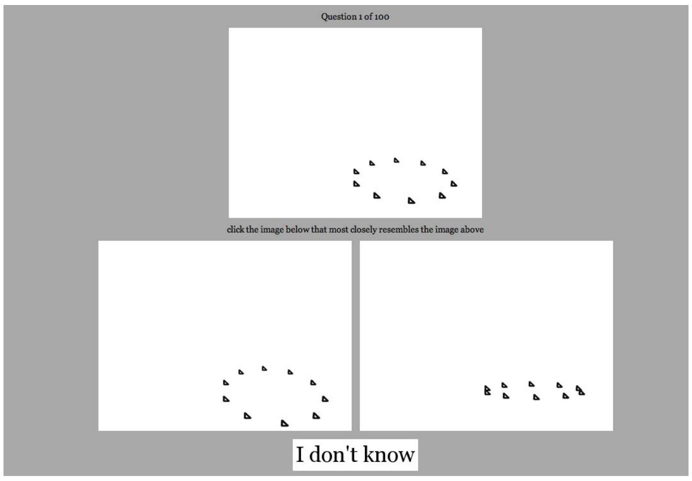
Question 1 of 100
click the image below that most closely resembles the image above
I don't know

Fig. 5. User screen for design survey

VIII. Results and Discussion

The experiment set out to investigate whether a small change at the genotypic level would translate into a small or large change at the output stage. A randomly selected sample of the output is shown in Figures 6, 7, 8 and 9. The images suggest that there was a significant difference between a nodal and structural mutation event. The results from the survey also showed a high degree of agreement that the nodal mutations produced an output more similar to the original than a structural mutation.The results for the survey are shown in Table I. Despite the subjective nature of the survey, the result implies that nodal mutation is a high locality operator whereas structural mutation is a low locality operator.

The results from the distance metrics also support the hypothesis that nodal mutation has high locality whereas structural mutation has low locality. The distance results were only recorded when both a structural and nodal mutation

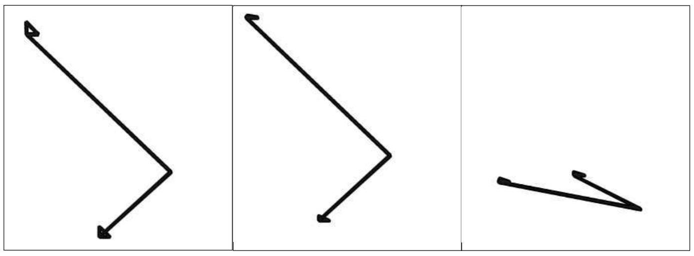

Fig. 6. The original design (left) and the same design after nodal (middle) and structural (right) mutations

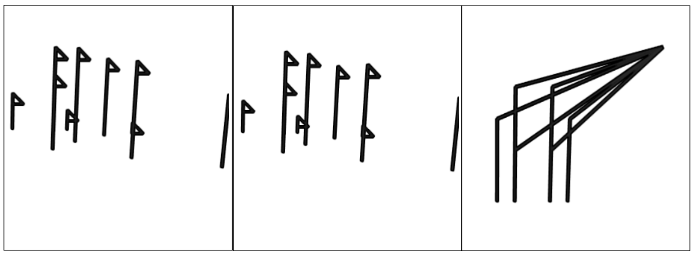

Fig. 7. The original design (left) and the same design after nodal (middle) and structural (right) mutations

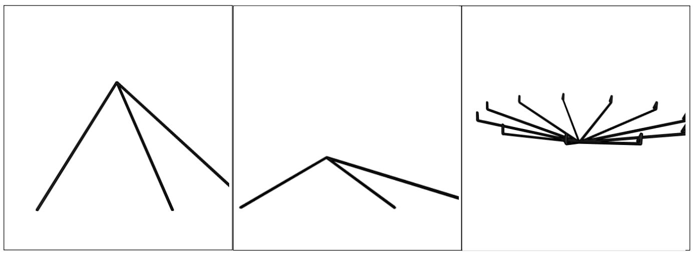

Fig. 8. The original design (left) and the same design after nodal (middle) and structural (right) mutations

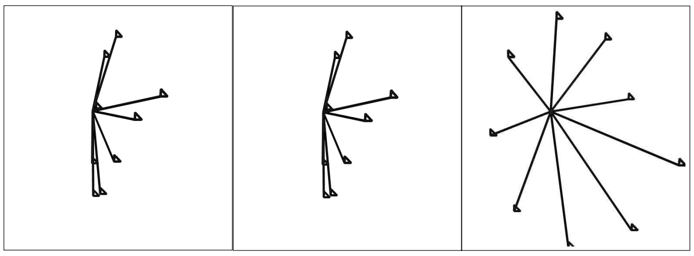

Fig. 9. The original design (left) and the same design after nodal (middle) and structural (right) mutations

occurred. The results for the three distance metrics are shown in the in the violin plots in Figs. 10, 11 and 12. The violin plot is a combination of a standard box plot (in black) and a kernel density estimation (shaded area). This shows the distribution of the results and can be thought of as a continuous histogram that has been smoothed by a weighted kernel and reflected through the Y axis. As can be seen from these results the nodal mutation produced more localised changes than structural mutation on both the derivation tree stage and the string stage. By definition, nodal mutation will only change one terminal on the derivation tree so it will always have a tree edit distance of one. The normalised compression distance also showed a non-normal distribution for the structural mutation events. A structural event either caused a small change or a large change but few intermediate changes as shown in Figure 13. We hypothesise that small changes were caused by structural mutation substituting one non-terminal for another that leaves the subtree size the same. The large changes, on the other hand, could have been caused by substituting a non-terminal that changes the size of that derivation subtree. This means that it would either add or remove codons to the following subtrees and cause a "ripple" event. This would have a consequence of changing the following codons and so redefine the meaning of the remaining chromosome. A ripple event has previously been discussed in [14].

TABLE I
RESULTS FOR SIMILARITY SURVEY

Preference	Total	Percentage
Nodal	1313	82%
Structural	247	15.5%
Did not know	40	2.5%

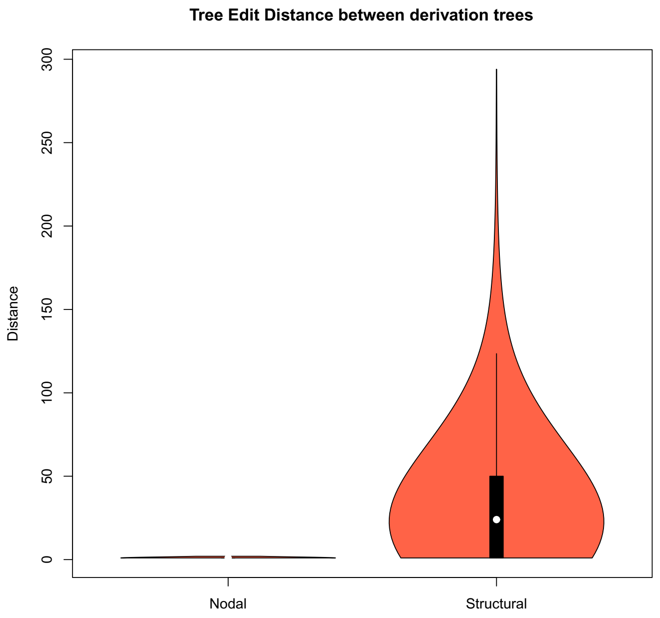

Fig. 10. Tree Edit Distance for HOF grammar.

The results highlight that nodal and structural mutation events have distinctive locality on the different stages in the derivation process. The results from the survey also highlight that this locality translates into similarity in designs at the subjective user level. Our interactive design software contains buttons on the GUI to allow the user to apply the mutation operator to an individual when they think it is appropriate.

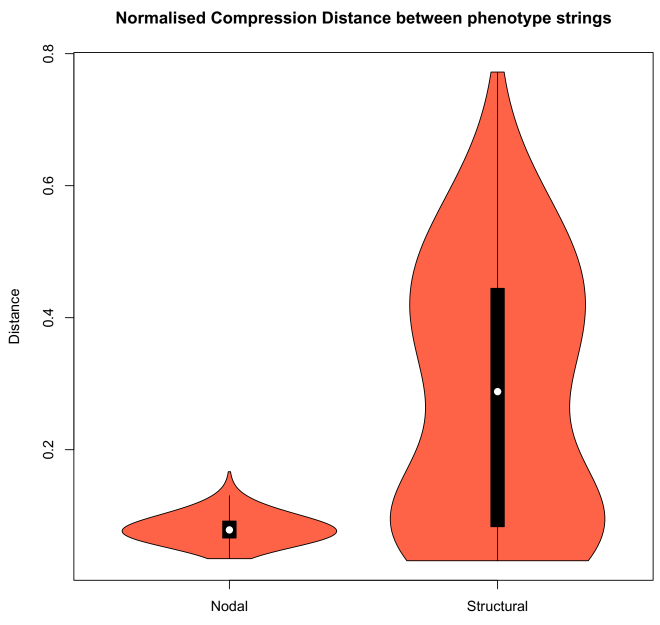

Fig. 11. Normalised Compression Distance for HOF grammar.

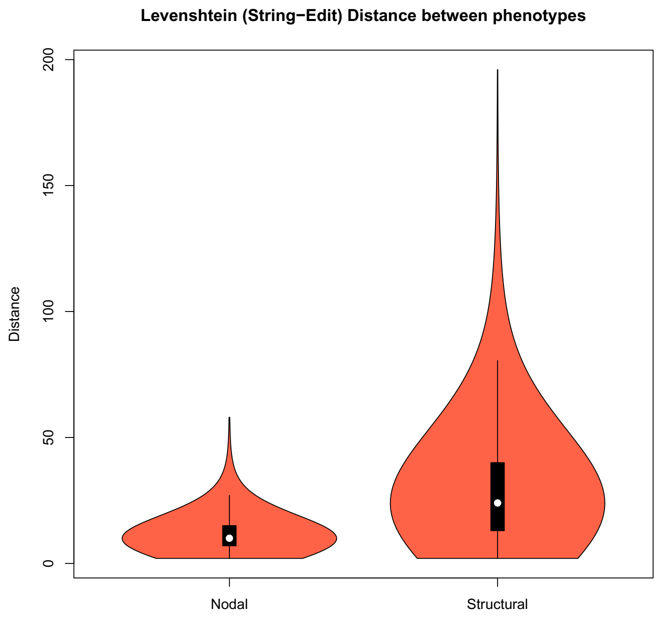

Fig. 12. String Edit Distance for HOF grammar.

This is an implementation of the active intervention described in [23] and [6]. The feedback that we have received from architectural students that have used our interactive design software states that this was a desirable quality for exploring an aesthetically pleasing area of the design space. By creating operators that the user can intuitively use, it enables the users themselves to act as the search operators, driving the

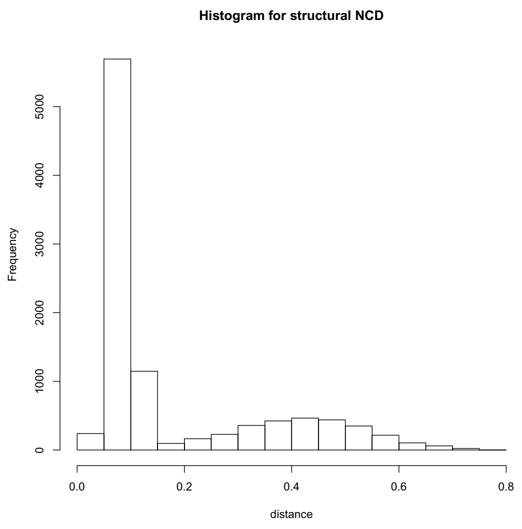

Fig. 13. Histogram of NCD for structural mutation.

evolutionary process into more fruitful regions of the design space.

IX. Conclusions

This study set out to analyse locality on the successive phenotypic output generated by GE. We defined the different phenotypic stages and described the components within standard GE mutation that were being investigated. Experiments were carried out to show how the different mutation events compared in their locality. Our results showed that it was possible to generate both high and low locality events during standard mutation that maintained their locality throughout the mapping process. The results will enable us to give users control over how much change they expect on a phenotypic level and increase the efficiency of their search. It also allows the user to search the design space exclusively with high locality operators and optimise designs that appeal to the user. This paper distinguished two different components of standard mutation in GE that each have different scales of change. By making this distinction it opens up the possibility of a dynamically changing operator. The operator would change the granularity of it's search as it was carried out, something we hope to address in future work.

X. Acknowledgments

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

References

[1] W. Banzhaf. Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 322–332, Jerusalem, 9-14 Oct. 1994. Springer-Verlag.

[2] J. Byrne, J. McDermott, M. O'Neill, and A. Brabazon. An analysis of the behaviour of mutation in grammatical evolution. In *Genetic Programming, Proceedings of EuroGP'2010*. Springer-Verlag, 2010.

[3] J. Byrne, M. O'Neill, and A. Brabazon. Structural and nodal mutation in grammatical evolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1881–1882. ACM, 2009.

[4] R. Cilibrasi and P. M. B. Vitanyi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[5] F. J. Gomez. Sustaining diversity using behavioral information distance. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 113–120, Montréal, Canada, 2009. ACM.

[6] D. A. Hart. Toward greater artistic control for interactive evolution of images and animation. In M. Giacobini, editor, *Applications of Evolutionary Computing*, volume 4448 of *LNCS*, pages 527–536. Springer, 2007.

[7] J. Hugosson, E. Hemberg, A. Brabazon, and M. ONeill. Genotype representations in grammatical evolution., 2007.

[8] C. Igel. Causality of hierarchical variable length representations. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 324–329, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

[9] R. E. Keller and W. Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 116–122, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[10] M. Li and P. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer Verlag, 1997.

[11] M. O'Neill, E. Hemberg, E. Bartley, A. Brabazon, and C. Gilligan. Geva - grammatical evolution in java. *ncra.ucd.ie/GEVA*, 2008.

[12] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. GEVA: grammatical evolution in Java. *ACM SIGEVOlution*, 3(2):17–22, 2008.

[13] M. O'Neill, J. McDermott, J. M. Swafford, and J. Byrne. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. In *International Journal of Design Engineering.*, volume In Press. 2010.

[14] M. O'Neill and C. Ryan. Crossover in grammatical evolution: A smooth operator? In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 149–162, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

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

[16] U.-M. O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In *IEEE International Conference on Systems, Man, and Cybernetics: Computational Cybernetics and Simulation*, volume 5, 1997.

[17] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, 2nd edition, 2006.

[18] F. Rothlauf. On the bias and performance of the edge-set encoding. *IEEE transactions on evolutionary computation*, 13(3):486–499, June 2009.

[19] F. Rothlauf and D. Goldberg. Redundant Representations in Evolutionary Algorithms. *Evolutionary Computation*, 11(4):381–415, 2003.

[20] F. Rothlauf and M. Oetzel. On the locality of grammatical evolution. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 320–330, Budapest, Hungary, 10 - 12 Apr. 2006. Springer.

[21] D. Shasha and K. Zhang. Fast Parallel Algorithms for the Unit Cost Editing Distance Between Trees. In *SPAA '89: Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 117–126, New York, NY, USA, 1989. ACM.

[22] M. Tacker, P. F. Stadler, E. G. Bornberg-Bauer, I. L. Hofacker, and P. Schuster. Algorithm Indepedent Properties of RNA Secondary Structure Predictions. *European Biophysics Journal*, 25(2):115–130, 1996.

[23] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, Sept. 2001. Invited Paper.

```
<scene> ::= <shapes>
<shapes> ::= <list_of_shapes>

# Given a list of functions and a list of points, return a list of shapes
<list_of_shapes> ::= map(<list_of_point_to_shape_funcs>, <list_of_points>)

# Only one method of creating a list of points.
<list_of_points> ::= map(<scalar_point_func>, make_scalar_list(<n>))

# Functions which return a point, given a scalar.
<scalar_point_func> ::= <unitcircle> | <ellipse> | <diagonal> | <bezier> |
                        <spiral> | <add_scalar_point_funcs> | <sinusoid>

# Given a scalar t, return a point on the spiral around a bezier curve.
# The radius, initial phase, and number of revolutions can be specified.
<spiral> ::= spiral(t, <radius>, <phase>, <revs>, <scalar_point_func>)

# Given a scalar t, return a point on a diagonal between two points.
<diagonal> ::= interpolate(t, (<pt>, <pt>))

# Given a scalar t, return a point on a circle with given radius and centre
# in the plane indicated by <dimension>.
<unitcircle> ::=  circlePath(t, <radius>, <pt>, <dimension>)
<ellipse> ::=  ellipsePath(t,<radius>,<radius>,<pt>,<dimension>)
<add_scalar_point_funcs> ::= pt_plus_pt((<scalar_point_func>)(t),
                             (<scalar_point_func>)(t))

<sinusoid> ::= (0.0, 0.0, <xcos>) | (0.0,<xcos>, 0.0)
             | (<xcos>, 0.0, 0.0)

# use 1.0 + cos() to keep it positive, avoid negative z values
# use 4pi * t so that we get 2 full revolutions, for t in [0, 1]
<xcos> ::= <x> * (1.0 + cos(<ndoublerevs> * 4 * pi * t))

# Given a scalar t, return a point on a cubic bezier curve.
<bezier> ::= bezier_form(t, (<pt>, <pt>, <pt>, <pt>))

# Functions which return a shape, given a point.
<point_shape_func> ::= [connect(<pt>, x)]
                       | [dropPerpendicular(x, 2)]
                       | connect3(x, <dimension>)
<list_of_point_to_shape_funcs> ::= [<point_shape_funcs>]
<point_shape_funcs> ::= <point_shape_func>
                       | <point_shape_funcs>,<point_shape_func>
<ndoublerevs> ::= 1 | 2 | 3 | 4
# points are represented as tuples
<pt> ::= (<x>, <x>, <x>)
# <x> is used for point coordinates
<x> ::= [0, 15]
# <dimension> indicates x, y or z
<dimension> ::= 0 | 1 | 2
<radius> ::= <x>
<phase> ::= [0.0, 1.0]
<revs> ::= [1, 7]
```

Fig. 14. Shape Grammar for GEVA Blender