Genotype Representations in Grammatical Evolution

Jonatan Hugosson, Erik Hemberg, Anthony Brabazon & Michael O'Neill

Natural Computing Research & Applications
Complex & Adaptive Systems Lab
University College Dublin, Ireland
jhugosson@gmail.com, erik.hemberg@ucd.ie, anthony.brabazon@ucd.ie,
m.oneill@ucd.ie

Abstract

Grammatical evolution (GE) is a form of grammar-based genetic programming. A particular feature of GE is that it adopts a distinction between the genotype and phenotype similar to that which exists in nature by using a grammar to map between the genotype and phenotype. Two variants of genotype representation are found in the literature, namely, binary and integer forms. For the first time we analyse and compare these two representations to determine if one has a performance advantage over the other. As such this study seeks to extend our understanding of GE by examining the impact of different genotypic representations in order to determine whether certain representations, and associated diversity-generation operators, improve GE's efficiency and effectiveness. Four mutation operators using two different representations, binary and gray code representation respectively, are investigated. The differing combinations of representation and mutation operator are tested on three benchmark problems. The results provide support for the use of an integer-based genotypic representation as the alternative representations do not exhibit better performance, and the integer reprensentation provides a statistically significant advantage on one of the three benchmarks. In addition, a novel wrapping operator for the binary and gray code representations is examined, and it is found that across the three problems examined there is no general trend to recommend the adoption of an alternative wrapping operator. The results also back up earlier findings which support the adoption of wrapping.

Key words: genetic programming, grammatical evolution, representation

Published in Applied Soft Computing 10(2010):36-43

1 Introduction

Grammatical evolution (GE) [20,19,25] is a form of grammar-based genetic programming. A special feature of GE is that, unlike genetic programming, it has a clear distinction between the genotype and phenotype. The mapping of the genotype and phenotype is governed by a grammar and this grammar can contain domain knowledge to bias the form a phenotypic solution can take. By separating the search and solution spaces, GE allows the implementation of generic search algorithms without a requirement to tailor the diversity-generating operators to the nature of the phenotype. A substantial literature has emerged on GE and its applications [20,2,24,22,4]. Some of the more recent developments of GE are focused on the various components of the GE approach including the use of alternative search engines [15,14,18], the use of alternative grammar constructs [16,5,12,21], and the examination of different mapping processes [17]. One aspect of GE which has seen less research is the examination of the impact of the choice of genotypic representation, and associated diversity-generation operators, on GE's efficiency and effectiveness. A recent paper by Oetzel and Rothlauf [24] examined the locality properties of a binary representation in GE and found that a genotypic bit-mutation operator produced non-local changes in the phenotype. The authors of this study proposed that further research be undertaken in order to find other representations and associated mutation operators which would produce higher locality, suggesting that this would increase the performance and effectiveness of GE. This study addresses this research issue by investigating the impact of four mutation operators using two different representations, binary and Gray code representation respectively, on the performance of GE. In addition, for the first time, a direct comparison is made between the two forms of genotypic representation adopted in the GE literature, namely integer versus binary codons. The combinations are tested using three standard benchmark problems, symbolic regression, the Santa Fe ant trail and the even-5-parity problem. In addition, a novel wrapping operator is proposed for the binary and gray representations and its performance compared to the standard wrapping operator.

The remainder of the paper is structured as follows. Section 2 describes GE and provides background on earlier work on representations. Section 3 details the experimental approach adopted and results, and finally section 5 details conclusions and future work.

2 Background

This section provides an introduction to GE and to some prior work on the importance of representation in evolutionary algorithms. GE is a grammar

based form of genetic programming (GP) [9]. Rather than representing the programs as parse trees, as in GP, a linear genome representation is used. A genotype-phenotype mapping is employed such that each individual's variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar, see Fig. 1. Consequently, the genetic operators such as crossover and mutation are applied to the linear genotype in a typical genetic algorithm (GA) [7] manner, unlike in a tree-based GP approach where they are applied directly to the phenotypic parse trees. The grammar allows the generation of programs in an arbitrary language that are guaranteed to be syntactically correct. The user can tailor the grammar to produce solutions that are purely syntactically constrained, or they may incorporate domain knowledge by biasing the grammar. The mapping process creates a clear distinction between the search and solution space.

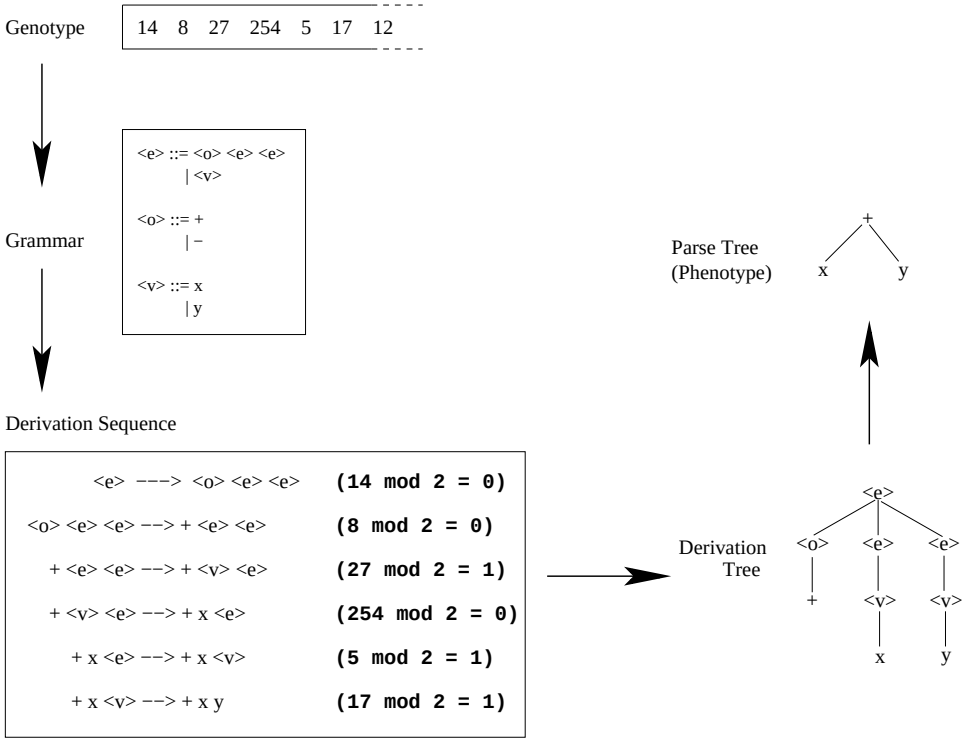

Fig. 1. An example GE genotype-phenotype mapping, where the genotype is used to select production rules from a grammar to produce a derivation sequence. The derivation sequence represents the development of a program from the embryonic non-terminal start symbol (<e>). The derivation sequence can be represented as a derivation tree, which can then be simplified to correspond to the parse tree adopted in standard tree-based GP.

An important element in a successful application of evolutionary methodologies is a careful co-selection of a representation and associated diversity-generating operators which are well-suited for a specific problem landscape. These choices can radically change the performance of an algorithm. Easy

problems in one representation can be hard in another. A substantial literature exists on the importance of representation choice and readers are referred to [23] for a detailed discussion of this issue. The study focuses on redundancy, scaling of building blocks and the modification of distance between individuals when mapping the genotype to the corresponding phenotype. It suggests that redundancy in a representation typically has a neutral or negative effect on algorithmic performance.

In contrast, GE always maps via the BNF grammar to the solution space, producing a much more complex genotype-phenotype mapping than typically exists in a GA. The locality of a genotype-phenotype mapping describes how well genotypic neighbors correspond to phenotypic neighbors. Rothlauf and Oetzel [24] investigate the locality of this mapping in GE and suggest that it has low locality as neighboring genotypes do not necessarily correspond to neighboring phenotypes. Based on this finding, they suggest that further work is needed in order to develop representations with higher locality in order to maximise the efficiency of GE.

In prior investigations on GAs, Hollstien [8] claimed that gray code works slightly better than the binary representation. Gray code has some advantages compared to binary code [23] as it is not affected by scaling and it has perfect locality concerning small changes. Therefore the difficulty of a problem remains unchanged when using mutation-based search. However gray coding may not produce the same effect in GE since GE uses a complex many to one mapping (arising from the BNF grammar) causing neutral mutations whose effects also must be considered. The neutral mutations are believed to cause higher diversity and thus higher fitness [3,28,6,24]. If neutral mutations are important mutation operators with high neutrality should be advantageous. Yu and Miller, [27], state a hypothesis about the importance of the ratio between adaptive/neutral mutations. The mapping in GE gives it a non uniform representation and thus may take longer to converge. GE is also subject to a *ripple effect* when a standard GA is applied. As the function of a gene depends on the genes that proceeds it, a small genotypic change can lead to a big phenotypic change. Evidence suggests that this effect can promote a useful exchange of derivation sub-sequences during crossover [12].

This study seeks to extend our understanding of GE by examining the impact of different genotypic representations in order to determine whether certain representations, and associated diversity-generation operators, improve GE's efficiency and effectiveness. Four mutation operators using two different representations, binary and gray code representation respectively, are investigated.

3 Experimental Setup & Results

This section outlines the experimental setup used in this study, details the results of these experiments, and provides a discussion of the key findings.

3.1 Hypothesis and Setup

Given the best fitness value after 50 generations for each mutation operator: μ_0, best fitness using normal integer mutation. μ_1, plus minus mutation. μ_2, binary mutation and μ_3, gray code mutation. The following hypothesis is stated:

H_0: None of the representations and mutations proposed gains significant performance to GE in any of the experiments, i.e. $\mu_0 = \mu_1, \mu_2, \mu_3$.
H_1: At least one of the mutations gains significant performance for at least one experiment, i.e. $\mu_0 > \mu_1$ or $\mu_0 > \mu_2$ or $\mu_0 > \mu_3$.
α: The significance level of the test is 0.05. [1]

Since the mutation operator is believed to be very problem dependent all three experiments are run with the four different mutation operators over 30 runs. Ramped Half-and-half initialization with a maximum tree depth of 8 is used. The mutation probability was 0.01, crossover probability 0.9 and the number of wraps was 30. To raise the diversity within the population, unique children regarding phenotypic difference from its parents are more likely to be produced. Two parents are picked using tournament selection with tournament size ten (population size 500). Steady state replacement is used. The solution quality is measured by cumulative frequency (rate of success) and best fitness value for the last generation (50th generation). To analyze GE's performance a t-test is performed on the best fitness after 50 generations. The data is assumed to come from normal distribution with unknown, but equal, variance. Three problems are examined, namely the Even-5-Parity, Symbolic Regression, and Santa Fe Ant Trail, and a brief description of each problem follows along with the corresponding grammars adopted in Fig. 2.

3.2 Santa Fe Ant Trail

The Santa Fe ant trail problem is a standard problem in GP and can be considered a deceptive planning problem with many local optima. The objective

[1] α is the probability of making a type 1 error, i.e. The probability of rejecting H_0 given that H_0 is in fact true.

```
EVEN-5-PARITY
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr> | ( <expr> <op> <expr> ) | <var>
<op> ::= "|" | & | ^
<var> ::= d0 | d1 | d2 | d3 | d4
SYMBOLIC REGRESSION
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr> | ( <expr> <op> <expr> ) | <pre-op> ( <expr> ) | <var>
<op> ::= + | * | -
<pre-op> ::= sin | cos | exp | log | inv
<var> ::= X | 1.0
SANTA FE ANT TRAIL
<prog> ::=<code>
<code> ::=<line>|<code><line>
<line> ::=<condition>|<op>
<condition> ::= if (food_ahead()==1) { <line>} else {<line> }
<op> ::=  left(): | right(): | move():
```

Fig. 2. BNF grammars used in the experiments.

is to find a computer program sequence to control an artificial ant so that it can find all the 89 food pieces on a irregular trail in a 32 by 32 toroidal grid. The ant can turn left, right and move one square forward. The ant can also look one square ahead in the direction it is facing.

The Santa Fe problem has the features often suggested in real program spaces, it is full of local optima and many plateaus. The fitness landscape is riddled with neutral networks linking programs with the same fitness in a dense and suffocating labyrinth [10].

A limited GP schema analysis shows that it is deceptive at all levels. Longer programs are on average fitter but contain a lower density of solutions. There are low and middle order schemata which are required to build solutions but which are below average fitness. This means practical sized samples give noisy estimates of their fitness. Furthermore there are no beneficial building blocks, leading GAs to choose between them randomly.

3.3 Even-5-Parity, Boolean Functions

Boolean functions are functions whose arguments only can take two values, low or high, true or false etc. In the even-5-parity problem the goal is to find a boolean expression that returns true if an even number of the boolean input is true. The fitness value will be any of the natural numbers $[0, 1, ..., 31, 32]$, when the input is five boolean variables, $d0...d4$.

There are 2^{2^n} boolean logic functions of n inputs. This gives the even-5-parity problem around $4e10$ boolean functions, however since there are only 32 different fitness values many of them will have the same fitness. The fitness space is dominated by a central spike indicating almost all programs score exactly half marks. A small fraction, 2^{1-n}, of large programs solve the problem. So

in general either the solution is found or else you score half, implying that the problem is similar to seek for a needle in a haystack. There are not many beneficial building blocks.

3.4 Symbolic Regression

The symbolic regression problem involve finding a mathematical expression in symbolic form. The fitness function is the absolute error for 20 points in the interval $[-1, 1]$. The target function used is:

$$f(x) = x^4 + x^3 + x^2 + x \tag{1}$$

3.5 Description of different mutations

Mutations that change the genotype but not the phenotype are called neutral and can arise due to functional redundancy, implicit neutrality, or mutation on inactive genes [28,20]. Adaptive mutations are mutations that change the phenotype. Neutral mutations cause diversity within equally fit individuals in the search space, while adaptive mutations explore the solution space.

In canonical GE setup adopted in this study, a 32 bit integer representation is used (see Fig. 3), where each codon is defined by an integer. In a *integer mutation*, the current integer value in a codon is replaced by a new randomly-generated integer. This means that if this codon is mapped modulo 2: around half of the mutations will be neutral, and if it is mapped modulo 3: one third are neutral. In other words the real probability for mutation in the phenotype is in fact $P(mut) * 1/n$, where n is the number of grammar instructions to chose among for this codon (modulo n). If the algorithm is stuck at a local optima all points in the search space can be reached with the mutation operator in order to escape the local optima. To summarize, random integer mutation is believed to have good search space exploration, low locality and neutral mutations.

A *plus minus mutation* adds or subtract one from the current integer value of the codon in question. This type of mutation extinguishes the neutral mutations, and narrows the search space given some local optimum. The positive feature could be that given a smooth grammar, i.e. a grammar that changes smoothly (gradually) between the properties of each production rules, this mutation could explore the search space more smoothly. For example if the arithmetic expression $sin(\cdot)$ is mutated, with a smooth grammar, it could change to $cos(\cdot)$, or X^2 to X^3. To summarize: plus minus mutation is believed

551	13	...	87349	12387

Fig. 3. Example of a genotype in an integer representation

1010...1010	0010...0111	...	0100...0001	1000...0101

Fig. 4. Example of a genotype in a binary representation

Integer	Binary	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Fig. 5. Binary representation for Gray code and binary code

to have very poor global search space exploration, quite high locality and no neutral mutations.

The *binary mutation* used here flips one bit in the codon. Using the Gray code representation (see Figs. 4and 5) of the bitstring a *gray code mutation* will change the codons in a different pattern compared to the binary representation, notice that the gray code changes one bit every time adding or subtracting one. The choice of grammar affects the result of the different operators, for example a smooth grammar (where the semantics of the alternative rules for a non-terminal are similar) might be advantageous for the gray code and plus minus mutation while a grammar with an equal number of production rules for each non-terminal would make the neutral mutations vanish.

3.6 Mutation With Crossover

The amount of statistical analysis for these experiments is quite large, the tables for statistical analysis are omitted, instead significant difference is mentioned in the caption of the figures. Results for mutation with crossover can be seen in Fig. 6.

The only experiments that has a significant difference in the GE performance

Symbolic regression $f(x) = x^4 + x^3 + x^2 + x$			
Mutation operator	Mean best fit.	Cum. frequency	Standard deviation
Integer mutation	1.1	9	1.4
Binary mutation	1.3	2	0.8
Gray code mutation	1.5	4	1.2
Plus Minus mutation	1.6	1	1.4
Even-five-parity			
Mutation operator	Mean best fit.	Cum. frequency	Standard deviation
Integer mutation	3.2	0	1.9
Binary mutation	4.0	0	2.1
Gray code mutation	2.8	0	2.8
Plus Minus mutation	3.7	0	1.7
Santa Fe ant trail			
Mutation operator	Mean best fit.	Cum. frequency	Standard deviation
Integer mutation	15.6	14	17.4
Binary mutation	13.2	15	15.7
Gray code mutation	12.5	15	15.1
Plus Minus mutation	8.9	19	13.8

Fig. 6. Results for experiments using mutation and crossover for 30 runs. The mean best fitness (minimizing), cummulative frequency of success and the standard deviation.

(best fitness) is the even-five-parity experiment, see Fig. 7. In this experiment integer mutation outperforms binary mutation while no other significant differences exist. Thus the hypothesis H_0, that none of the proposed operators produces any significant performance advantage, cannot be rejected.

Regarding the mean fitness value, see Fig. 7 and 8, we can draw the conclusion that in all the experiments except even-five-parity, the mutation operator causing the most neutral mutations also has the lowest mean fitness value. Neutral mutations cause phenotypic diversity to decrease and thus produce reduced exploration. The adaptive mutation operators in all experiments have the highest cumulative frequency. Adaptive mutations therefore seems to be advantageous for finding the optimal solution, however maybe neutral mutation with higher mutation rate would perform as well or better using both good exploration and neutral features.

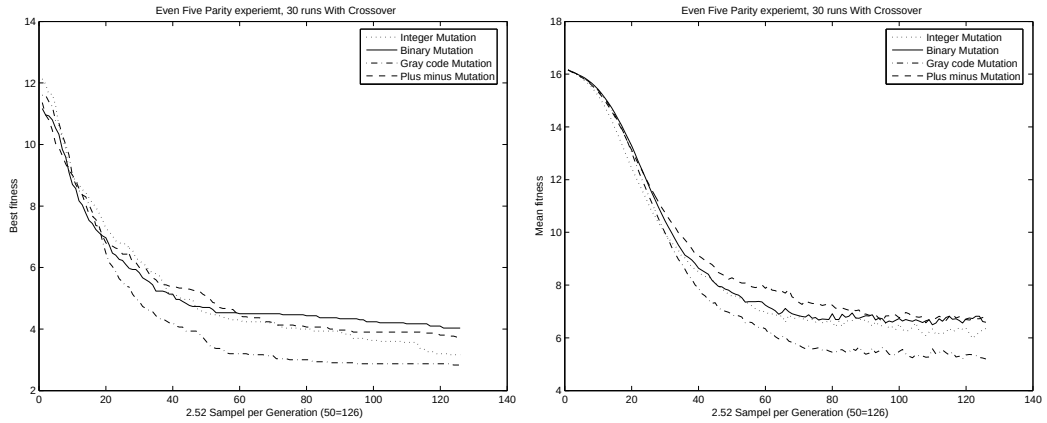

Fig. 7. *Left*: Even-five-parity GE performance, best fitness after 50 generations, for the four mutation operators. There is a significant difference between the binary mutation and the integer mutation. *Right*: Even-five-parity mean of the best fitness after 50 generations for the four mutation operators. There are significant differences between the Plus minus mutation and the three others.

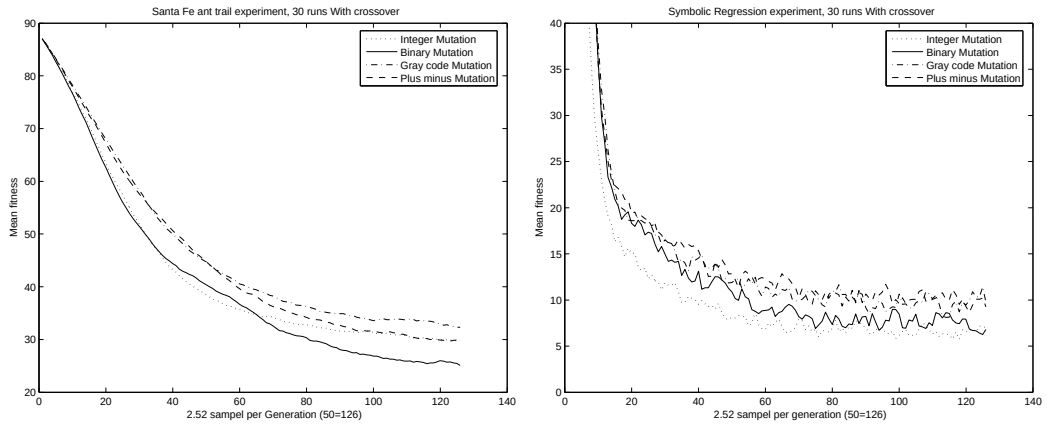

Fig. 8. *Left*: Santa Fe ant trail mean of the best fitness after 50 generations for the four mutation operators. There is a significant difference between the Gray code and binary mutation. *Right*: Symbolic regression mean of the best fitness after 50 generations for the four mutation operators. There are significant differences between the binary and Gray code and plus minus mutation respectively.

For each experiment no wrapping and no mutation is also tested. Since wrapping is turned off the length of the genotypes are increased to an average of 400 codons. As can be seen in Fig. 6 the canonical GE settings of using mutation and wrapping do produce higher performance than when these mechanisms are 'turned-off'.

Symbolic regression $f(x) = x^4 + x^3 + x^2 + x$			
Mutation operator	Mean best fit.	Cum. frequency	Standard deviation
Integer mutation	8.4	0	3.7
Binary mutation	7.9	0	3.8
Gray code mutation	7.4	0	3.8
Plus Minus mutation	7.2	0	3.5
Even-five-parity			
Mutation operator	Mean best fit.	Cum. frequency	Standard deviation
Integer mutation	5.0	0	2.5
Binary mutation	5.5	0	2.5
Gray code mutation	5.0	0	2.3
Plus Minus mutation	5.8	0	2.4
Santa Fe ant trail			
Mutation operator	Mean best fit.	Cum. frequency	Standard deviation
Integer mutation	33.2	2	15.7
Binary mutation	38.3	1	9.1
Gray code mutation	29.8	1	13.7
Plus Minus mutation	26.2	6	16.4

Fig. 9. Results for experiments using only mutation for 30 runs. The mean best fitness (minimizing), cumulative frequency of success and the standard deviation.

3.7 Mutation Without Crossover

In these experiments the crossover probability is set to zero, other settings are as in the previous experiments. The results are shown in Fig. 9.

No conclusions about GE performance and the mutation operators should be drawn since the only factor that seems to matter is the actual rate of exploration. This can be altered by increasing the mutation probability. In the Santa Fe ant trail experiment, it can clearly be seen that the binary and integer mutation are outperformed by plus minus and gray code mutation. Without crossover, neutral mutations produce less exploration and thus lower variance, diversity, and performance. Since the algorithm has a phenotypic diversity mechanism implemented this does not properly justify the way neutral mutations have been shown to work, see [27,28]. The best fitness for the plus

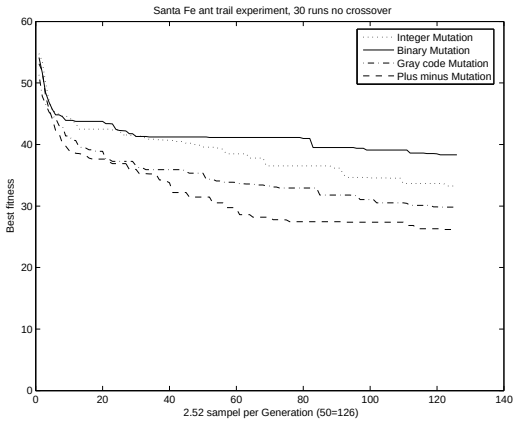

Fig. 10. Santa Fe ant trail GE performance for the four mutation operators without the crossover operator. The Binary mutation is significantly different from the plus minus and the Gray code mutation.

minus operator is not significantly better than that produced by any other operator. There is no bias induced by the grammar since the Santa Fe ant trail experiment always chooses among three or fewer production rules. The plus minus operator can reach all the points of the search space, but unlike the integer mutation no mutations are neutral and thus the plus minus mutation has a higher rate of exploration. The results do not indicate that any firm conclusions can be drawn regarding locality and mutation. None of the proposed operators appear to produce higher locality, the only factor that seems to be of importance is the exploration rate caused by fewer neutral mutations. It is likely that these results emerge because the GE grammar mapping is so complex that no matter what representation is used, the effect caused by a mutation causes a substantial change in the phenotypic structure.

3.8 Crossover with No mutation and No wrapping

For each experiment 30 runs are made using no wrapping and no mutations in the presence of crossover. Since wrapping is turned off the length of the genotype are increased to an average of 400 codons. Results are presented in Fig. 11.

As can be seen the standard GE settings of using mutation and wrapping do produce higher performance than when these mechanisms are 'turned-off'.

3.9 Wrapping

The standard wrapping operator adopted in GE is adopted on the integer representation. That is, when the end of the chromosome is reached, we *wrap*

Symbolic regression $f(x) = x^4 + x^3 + x^2 + x$			
	Mean	Cum. frequency	Standard deviation
Standard	1.1	9	1.4
No mutation, No wrapping	4.1	4	3.4
Even-five-parity			
	Mean	Cum. frequency	Standard deviation
Standard	3.2	0	1.9
No mutation, No wrapping	6.5	0	2.2
Santa Fe ant trail			
	Mean	Cum. frequency	Standard deviation
Standard	15.6	14	17.4
No mutation, No wrapping	33	2	12

Fig. 11. Results for experiments using only crossover for 30 runs. Both mutation and wrapping are not used in this case.

$$\overbrace{|0,1,2,...,29,30,X|}^{\text{Codon n}}\ \overbrace{|0',1',2',...,29',30',X'|}^{\text{Codon n+1}}$$

$$\underbrace{\qquad\qquad}_{\text{Reading frame 1}}$$

$$\text{Wrapping Operator} \qquad \Rightarrow$$

$$|0,1,...,11,12... \quad \underbrace{13,..,30,X|,|0',1',...,12'}_{\text{Reading frame 2}} \ ...13',14',...,29',30',X'|$$

Fig. 12. When the new wrapping operator acts on the genotype, the reading frame jumps 13 bits forward.

and begin reading integers from the start of the chromosome. In earlier studies with GE this is also the manner in which wrapping is conducted on binary chromosomes. In this study we also examined a novel wrapping operator for the binary representations (i.e., for both gray and binary) which allowed the reading frame to be changed upon wrapping (see Fig. 12). That is, the reading frame does not always start reading from the first bit on the chromosome. In this case the reading frame is offset by 13 bits after each wrap event. Fig. 13 summarises the features of the wrapping operators adopted for the integer, gray and binary representations.

Given the best fitness value after 50 generations for each mutation operator:

	Integer	Binary	Gray code
Conserving of Building Blocks	Yes	No	No
Generating of new sequences	Sometimes	Yes	Yes

Fig. 13. Table of the wrapping operators properties

GE performance 30 runs Santa Fe ant trail			
Wrapping operator	Mean	Cum. frequency	Standard deviation
Standard Integer Wrapping	10.5	17	13.3
Binary wrapping	16.2	12	16.4
Gray code wrapping	15.5	19	15.5

GE performance 30 runs Symbolic regression			
Wrapping operator	Mean	Cum. frequency	Standard deviation
Standard Integer Wrapping	0.9	5	0.6
Binary wrapping	1.6	1	1.3
Gray code wrapping	3.7	0	5.0

GE performance 30 runs Even five parity			
Wrapping operator	Mean	Cum. frequency	Standard deviation
Standard Integer Wrapping	4.0	0	1.8
Binary wrapping	3.2	0	1.4
Gray code wrapping	3.8	0	1.1

Fig. 14. A comparison of results for the three problems for wrapping on the three different representations.

(1) μ_0: Best fitness using normal integer wrapping, (2) μ_1: Binary wrapping and (3) μ_2: Gray code wrapping. The following hypothesis is stated for the performance of the wrapping operators:

H_0**:** None of the wrapping operators proposed gains significant performance to GE in any of the experiments, i.e. $\mu_0 = \mu_1, \mu_2$.
H_2**:** At least one of the wrapping operators gains significant performance for at least one experiment, i.e. $\mu_0 > \mu_1$ or $\mu_0 > \mu_2$.
α**:** The significance level of the test is 0.05 [2] .

[2] α is the probability of making a type 1 error, i.e. The probability of rejecting H_0

For all experiments it is assumed that the mean best fitness is normally distributed and that the mean best fitness variance is equal but unknown for all mutations.

Fig 14 details the results obtained for the three representations and their corresponding performance with the different wrapping operators. A t-test reveals that there is no significant difference between the wrapping operators on the Santa Fe Ant trail, the standard integer wrapping outperforms the others in the case of Symbolic Regression, while the binary wrapping outperforms integer wrapping on the even-5-parity problem.

4 Discussion

A number of experiments on different genotypic representations and operators have been presented, and are conducted on three diverse and well understood standard benchmark problems from the Genetic Programming literature. One of the primary motivations for this study was the fact that at least two variants of Grammatical Evolution exist within the literature, which adopt different genotypic representations (i.e., binary and integer). For the first time, in this study we compare the performance of these two underlying genotypic representations to determine if one presents an advantage over the other. Interestingly, in the standard algorithm setup which combines mutation, crossover and wrapping operators there is no observed statistically significant difference between these encodings on two of the three bencmarks. On the boolean even-five-parity problem the integer representation has a significant advantage over its binary counterpart. As such, we would recommend the adoption of an integer based representation.

On the experiments with the wrapping operator, earlier research [13] indicated that the presence of wrapping can be advantageous on some problems, while not having a negative contribution on performance on the other benchmarks examined. In this study alternative wrapping operators are explored, and again the main conclusion is that wrapping can present an advantage on certain problem instances. The main issue with adopting a wrapping operator as in Grammatical Evolution, is that the complexity of the genotype-phenotype mapping process can be compounded by the fact that different codons can be used in multiple contexts. Intuitively, this might make it more difficult to modify/mutate the values at these codons which have overlapping function. What is surprising is that despite this, wrapping can be a productive operator at enhancing performance of the algorithm on some problems. Some comfort can also be taken from the natural system which inspired the adoption of

given that H_0 is in fact true.

wrapping (plasmids), in that overlapping genes, such as those that might occur with wrapping, are quite common in the natural world [11].

By virtue of the fact we are adopting a genotype-phenotype mapping we end up with an algorithm that is more complex than without such a map, and the correlation between genotype and phenotype is not as clean cut as one might design intuitively. However, the claimed benefits of this separation are well documented (e.g., see [1,26]) and research in this domain is an active one. With respect to GE specifically, it has been claimed that benefits include, a generalised encoding that can represent a variety of structures (such a grammar-based approach inherently allows multiple-types to be included unlike traditional GP which is limited to a single type), efficiency gains for evolutionary search by taking advantage of neutral networks and neutral evolution, and facilitating genetic diversity and preservation of functionality due to the many-to-one mapping and degenerate encoding [13]. The numerous applications of GE in the literature are a clear statement of the success of this particular approach to representation in Genetic Programming. This paper does however, highlight the need for a better understanding of these kinds of genotype-phenotype representations and where the real performance benefits can be achieved. As such future research will be directed towards the genotype-phenotype map itself and also the impact of the grammar on this process.

5 Conclusions & Future Work

The object of this study was to examine the impact of different genotypic representations in GE in order to determine whether certain representations, and associated diversity-generation operators, improve GE's efficiency and effectiveness. Two main variants of genotype representation exist in the literature to date, namely, binary and integer encodings. This is the first time these two representations have been formally compared and analysed.

Four mutation operators using two different representations, binary and Gray code representation respectively, are investigated. The different combinations of representation and mutation operator were tested on three benchmark problems. The results provide support for the continued use of the integer variant of genotypic representation as none of the alternative gains a significant performance advantage, while the integer form produced statistically significant improvements on one of the benchmark problems.

Even though no clear improvements can be seen through new mutation operators by virtue of different underlying genetic representations, there is no doubt that the presence of mutation increases GE's performance (see Fig. 6).

Since GE has a complex mapping from genotype to fitness value, the results do not suggest that changing the representation of the genotype has a significant impact on GE performance. This does not imply that the representation is irrelevant, rather it suggests that the examination of the utility of a specific representation cannot be isolated from an examination of the mapping process embedded in the grammar. One way of investigating locality further would be to create a grammar that changes gradually between the properties of each production rules, this mutation could explore the search space more smoothly. Furthermore investigation of different mapping methods and how to create operators that perform a more local search, e.g. context sensitive mutations might improve the understanding of GE.

A comparison of a novel wrapping operator for the binary and gray code representations reveals that across the three problems there is no general trend that suggests one wrapping operator has a superior performance.

Overall the experiments reported here are conducted on three standard and well understood benchmark problems commonly found in the Genetic Programming literature, and as such we are limited in our ability to generalise our findings beyond these problems. The nature of the three problems are quite different however, ranging from finding a boolean expression, uncovering a symbolic expression from input-output pairs, to a simple robot control type problem. It will be useful to extend this study to harder benchmarks, and to real-world problem instances to determine if some of the differences (albeit small) may be amplified in more complex domains. One particular interest of our research is to determine how these representations will perform in dynamic environments and research to this end is underway.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under grant No. 08/IN.1/I1868.

References

[1] Banzhaf, W. (1994). *Genotype-Phenotype-Mapping and Neutral Variation - A case study in Genetic Programming.* In Proceedings of Parallel Problem Solving From Nature III, Vol. 866, pp. 322-332, Springer-Verlag.

[2] Brabazon, Anthony and O'Neill, Michael (2006), *Biologically Inspired Algorithms for financial Modelling*, Springer.

[3] Burke, E., Gustafson, S. and Kendall, G. (2005), *Evolutionary Optimization in uncertain Environments - A Survey*, IEEE Transactions on Evolutionary Computation 9(3) 2005, pp. 303-317.

[4] Dempsey, I. (2007). Grammatical Evolution in Dynamic Environments. PhD Thesis, University College Dublin, Ireland.

[5] Dempsey, I., O'Neill, M., and Brabazon, A. (2005). *Meta-grammar constant creation*, Proc. of GECCO 2005, pp. 1665-1672, ACM Press.

[6] Galvan-Lopez, E. and Rodriguez-Vasques, K. (2002), *The Importance of Neutral Mutations in GP*, PPSN IX, LNCS 4193, pp. 870-879, 2006.

[7] Goldberg, David E. (1989), *Genetic Algorithms*, Addison Wesley Longman.

[8] Hollstein, R. B. (1971), *Artificial Genetic Adaption in Computer Control Systems*, PhD thesis, University of Michigan.

[9] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.

[10] Langdon, William B. and Poli, Riccardo (1998), *Foundations of Genetic Programming*, Springer

[11] Lewin, B. (1999). *Genes VII*. Oxford University Press.

[12] Nicolau, M. and Dempsey, I. (2006), *Introducing Grammar Based Extensions for Grammatical Evolution*, CEC 2006, pp. 648-655.

[13] O'Neill, M. (2001). *Evolutionary Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution*, PhD Thesis, University of Limerick.

[14] O'Neill, M. and Brabazon, A. (2004). *Grammatical swarm*, Proc. of GECCO 2004 LNCS 3120, Part 1, pp. 163-174, Springer-Verlag.

[15] O'Neill, M. and Brabazon, A. (2006). *Grammatical Swarm: The Generation of Programs by Social Programming*. Natural Computing 5(4):443-462.

[16] O'Neill, M. and Brabazon, A. (2005). *mGGA: The meta-Grammar genetic algorithm*, EuroGP 2005 LNCS 3447, pp. 311-320, Springer.

[17] O'Neill, M., Brabazon, A., Nicolau, M., McGarraghy, S., Keenan, P. (2004). *πGrammatical Evolution*, Proc. of GECCO 2004, LNCS 3103, Part 2, pp. 617-629, Springer-Verlag.

[18] O'Neill, M. and Brabazon, A. (2006). *Grammatical Differential Evolution*, Proc. of ICAI '06, pp. 231-236, CSEA Press.

[19] O'Neill, M., Ryan, C. (2001). *Grammatical Evolution*, IEEE Trans. Evolutionary Computation 2001 5(4):349-358.

[20] O'Neill, M. and Ryan, C. (2003). *Grammatical Evolution*, Kluwer.

[21] O'Neill, M. and Ryan, C. (2004). *Grammatical evolution by grammatical evolution. The evolution of grammar and genetic code*, EuroGP 2004 LNCS 3003, pp. 138-149 Springer.

[22] O'Reilly, U. M. and Hemberg, M. (2007). *Integrating generative growth and evolutionary computation for form exploration*, Genetic Programming and Evolvable Machines, Vol. 8, pp. 163-186, Springer

[23] Rothlauf, F. (2002). *Representations for Genetic and Evolutionary Algorithms*, Physica-Verlag, Heidelberg.

[24] Rothlauf, F. and Oetzel, M. (2005). *On the Locality of Grammatical Evolution*, EuroGP 2005 LNCS 3905, pp. 320-330, Springer.

[25] Ryan, C., Collins, J.J., O'Neill, M. (1998). *Grammatical Evolution: Evolving Programs for an Arbitrary Language.*, EuroGP 1998 pp. 83-95, Springer-Verlag.

[26] Wagner G.P., Altenberg, L. (1996). *Complex Adaptations and the Evolution of Evolvability.* Evolution 50(3), pp.967-976.

[27] Yu, T. and Miller, J. (2002). *Finding Needles in Haystack is Not Hard with Neutrality*, EuroGP 2002, pp. 13-25.

[28] Yu, T. and Miller, J. (2002). *Neutrality and the Evolvability of Boolean Function Landscape*, EuroGP 2001, Vol. 2038, pp. 204-217.