

# Mutation as a Diversity Enhancing Mechanism in Genetic Programming

David Jackson  
Dept. of Computer Science  
University of Liverpool  
Liverpool L69 3BX, United Kingdom  
Tel. +44 151 795 4248  
djackson@liverpool.ac.uk

## ABSTRACT

In various evolutionary computing algorithms, mutation operators are employed as a means of preserving diversity of populations. In genetic programming (GP), by contrast, mutation tends to be viewed as offering little benefit, to the extent that it is often not implemented in GP systems. We investigate the role of mutation in GP, and attempt to answer questions regarding its effectiveness as a means for enhancing diversity, and the consequent effects of any such diversity promotion on the solution finding performance of the algorithm. We find that mutation can be beneficial for GP, but subject to the proviso that it be tailored to enhance particular forms of diversity.

## Categories and Subject Descriptors

D.1.2 [Programming Techniques]: Automatic Programming; I.2.2 [Artificial Intelligence] Automatic Programming – *program synthesis*; I.2.6 [Artificial Intelligence] Learning – *induction*.

## General Terms

Algorithms, Design, Performance, Experimentation.

## Keywords

Genetic programming, implementation, evolution strategies, empirical study, performance measures.

## 1. INTRODUCTION

There are essentially two reasons for using mutation as an operator in evolutionary computing algorithms. The first is as an alternative to crossover in exploring the search space. This approach is found in, for example, evolutionary programming [6], various hill climbing techniques (e.g. Stochastic Iterated Hill Climbing – SIHC [13]), and Cartesian Genetic Programming [11]. However, the most common reason for employing a mutation operator, particularly when it appears alongside crossover, is to ensure that a certain level of diversity is maintained in the population. Without it, the argument goes, a population that is converging on a minimum that is not a solution may be unable to escape because of the homogeneous

nature of its members. Mutation acts as a brake on this by throwing in random fragments of chromosome which either have never existed in the population or which have been deleted because of their presence in only low-fitness individuals. These new fragments may cause sufficient modification of the relatively high fitness members selected for mutation to allow wider and more fruitful exploration of the search space.

This argument is a cogent one for evolutionary techniques such as genetic algorithms [12]. It is certainly the case with GAs that standard crossover of two identical parents will spawn a child that is also a clone of the parents, and that crossover of two very similar parents will produce a child that is also very similar. If a certain allele of a gene is not present in the parents, then it will not appear in the child. Moreover, if it is not present in the population at all, then no amount of crossover or reproduction will create it. The importance of mutation in the form of random gene flipping is that it is able to reintroduce such alleles, which may be crucial to the success of the evolutionary process. However, if used with abandon, the mechanism can hinder convergence, and so the usual guidance is that mutation should be invoked in only a small percentage of the total number of evolutionary operations, ideally with only a small amount of genetic material being altered on each occasion.

In the particular case of genetic programming, on the other hand, it would appear that the argument in favour of mutation carries less weight. The conventional method of performing mutation in GP, as described by Koza [9], is to replace an arbitrary sub-tree of the selected individual with a randomly generated code tree. This bears so little relation to the concept of allele replacement as used in GAs that its worth is questionable. It has been argued that, in standard tree-based GP, mutation and crossover are really two sides of the same coin, and this has provoked several attempts to settle the debate [10,16]. Whatever the evidence, it is clear that many GP practitioners (Koza included) are not convinced of the value of mutation, as its absence is frequently notable in GP experiments described in the literature.

The aim of this paper is not to add fuel to the crossover versus mutation controversy, but instead to examine more closely what the effects of mutation are, if and when it is used in GP. Specifically, there are a number of questions we attempt to address here. The first and most fundamental of these questions is: does the GP mutation operator actually achieve its stated aim of increasing population diversity? Following on from this: if it does have an effect on diversity, does this in turn have an impact on the performance of the GP process? And finally: what can we do to make mutation more powerful in terms of its ability to enhance diversity, and what are the consequences of doing so?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GECCO'11, July 12–16, 2011, Dublin, Ireland.

Copyright 2011 ACM 978-1-4503-0557-0/11/07 ...\$10.00.

In order to answer these questions, we must first define our terms of reference. In particular, we must establish what we mean by diversity, and how we propose to measure it.

## 2. DIVERSITY IN GP

As with other forms of evolutionary computation, diversity in GP provides an assessment of how different the individuals in a GP population are from one another. There are, however, a number of ways in which such differences may be measured [3,4].

Genotypic diversity is defined in terms of the structure of the program trees making up the population. That is, it takes into account the size and shape of trees and the tokens used at each node position. The importance of genotypic diversity was recognized by Koza, who proposed a ‘ramped half-and-half’ algorithm for creating an initial population containing a wide variety of program sizes and shapes. He also advised the GP practitioner to strive for structural uniqueness in this initial population. At its simplest, structural diversity metrics may reduce to nothing more than a straightforward node-by-node comparison of the linearised form of such trees, although more sophisticated techniques have been proposed [5,3].

By contrast, phenotypic diversity focuses not on static structure but on what occurs when program trees are evaluated. Approaches to measuring phenotypic diversity have tended to focus on the spread of fitness values present in the population [14,15], although alternatives have been suggested which focus more on semantic analysis of individuals [1,2]. Recently, another form of phenotypic diversity metric has been proposed which considers the execution behaviour of individuals [7,8]. In this approach, a program is executed for each test case, and the complete set of outputs or behaviours is recorded. This record is then compared with that of other individuals in order to establish the level of behavioural diversity. In the experiments which follow, we will make use of this form of behavioural diversity, together with appropriate structural and fitness metrics.

## 3. EXPERIMENTAL SET-UP

To investigate the role of diversity in GP, we consider four typical GP benchmark problems. Two of these are taken from the Boolean domain (6-multiplexer and even-4 parity), one is a path navigation problem (the Santa Fe ant trail), and the fourth is a numeric problem (symbolic regression of a polynomial).

In the 6-mux problem, the aim is to evolve a program that interprets the binary value on two address inputs in order to select which of the four data inputs to pass onto the output. In the even-4 parity problem we search for a program which, given 4 binary inputs, determines whether the number of inputs set to logic 1 is even or odd. For both problems, fitness evaluation is exhaustive over all combinations of input values, with fitness being given as a count of mismatches with expected outputs.

The Santa Fe ant trail problem consists of evolving an algorithm to guide an artificial ant through a 32x32 matrix in such a way that it discovers as many food pellets as possible. To prevent exhaustive or interminable searching, the ant is allowed a maximum of 600 steps in which to complete the task.

Our final problem is symbolic regression of a polynomial. In our version of this, the polynomial we attempt to match through evolution is  $4x^4 - 3x^3 + 2x^2 - x$ . The only terminal is  $x$ , and the function set is  $\{+, -, *, /\}$ , with the division operator being protected

to ensure that divide-by-zero does not occur. The fitness cases consist of 32  $x$ -values in the range  $[0,1)$ , starting at 0.0 and increasing in steps of  $1/32$ , plus the corresponding  $y$ -values. Fitness is calculated as the sum of absolute errors in the  $y$ -values computed by an individual, whilst success is measured in terms of the number of ‘hits’ – a hit being a  $y$ -value that differs from the expected output by no more than 0.01.

All of the above are commonly-used benchmark problems that are described more fully elsewhere (e.g. by Koza [9]). Other parameters as they apply to the experiments described in the remainder of this paper are shown in Table 1.

**Table 1.** GP system parameters common to all experiments.

Population size	500
Initialisation method	Ramped half-and-half
Evolutionary process	Steady state
Selection	5-candidate tournament
No. generations	51 generational equivalents (initial+50)
No. runs	100
Prob. reproduction	0.1
Prob. crossover	Variable, 0.4 to 0.9
Mutation	Variable, 0.0 to 0.5
Prob. internal node used as crossover point	0.9

In measuring structural diversity, we can simply perform node-by-node comparisons of the program trees for each individual, as described above. Similarly, fitness diversity is measured by considering the number of different fitness values present in the population. Our approach to measuring behavioural diversity, however, requires some further detail.

In the case of the Boolean problems, the behaviour of an individual is measured in terms of the outputs it produces; this is recorded as a string of binary values for each of the test cases used during fitness evaluation. So, for the 6-mux problem, there is a 64-bit string associated with each member of the population, while for the even-4 parity problem only a 16-bit string need be recorded. To save memory, these can be packed into 64-bit and 16-bit integers, respectively. We say that two individuals exhibit phenotypic differences if they differ in any of the corresponding bits in their output strings, and that an individual is phenotypically unique in a population if there are no other members of that population with exactly the same binary output string.

For the symbolic regression problem, we again record the outputs produced for each test case, but this time it is a vector of 32 floating point results. In comparing phenotypes we choose not to look for exact matches, but instead check whether the differences between corresponding outputs lie within some pre-defined value epsilon. Hence, two individuals are said to be behaviourally identical if, for each  $x$ -value, the absolute difference between the corresponding  $y$ -values is less than epsilon. The value for epsilon was arbitrarily chosen to be the same as that used to check for ‘hits’ in fitness assessment, i.e. 0.01.

For the ant trail problem, the situation is complicated by the fact that the evolving programs do not produce outputs as such: their behaviour is measured in terms of the movements produced by function and terminal execution. Because of this, the record we make of an individual's behaviour is the sequence of moves it makes in the grid during execution. We are not concerned with recording any left or right turns that are executed while remaining on a square, nor with any decision making via execution of statements such as `IF_FOOD_AHEAD`.

To record the path histories, we associate with each individual in the population a vector of {north, east, south, west} moves. Each time the executing program moves to a different square, the heading is added to the end of the vector. Since a program times-out after a fixed number of steps (600), we know that the vector cannot be longer than that number of elements per individual, and so memory occupancy is not a huge issue. Determining behavioural differences between individuals becomes simply a matter of comparing these direction vectors.

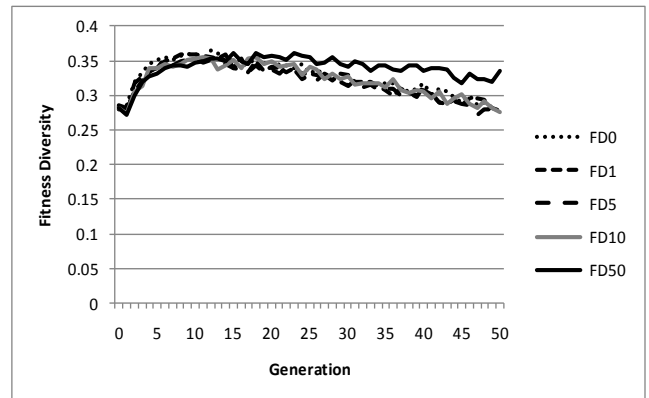
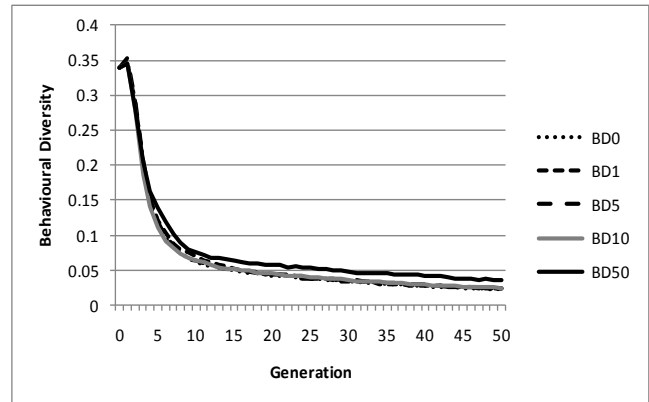
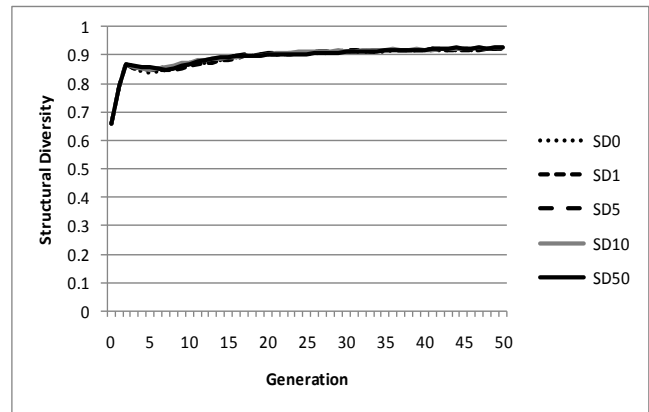
#### 4. EXPERIMENTAL INVESTIGATION

We now turn to the first of the research questions we wish to address, namely, does mutation in GP do anything to enhance diversity? To answer this, we define three separate metrics for structural diversity (SD), behavioural diversity (BD) and fitness diversity (FD). In each case, the diversity value at any point during evolution is calculated as the number of distinct sets of individuals present in the population, divided by the maximum possible number of such sets. For example, it is clearly possible in each of our benchmark problems for the number of distinct program trees in a population to be equal to the size of that population. Hence, if we were to find that there were 400 distinct tree structures in a population of 500, the value of SD at that point would be  $400/500 = 0.8$ . Similar reasoning applies to the calculation of BD. Note that a value of SD or BD equal to 1.0 would mean that every individual is structurally or behaviourally unique in the population.

The calculation of FD differs only slightly, in that the range of possible fitness values is often much smaller than the population size, and is problem-dependent. For example, if a given population in the even-4 parity problem contained a total of 5 different fitness values, then its FD would be  $5/17 = 0.294$ , since the problem allows for only 17 possible fitness values (0-16). Again, a value of  $FD=1.0$  would indicate full fitness diversity (which would also imply that the population held a solution!).

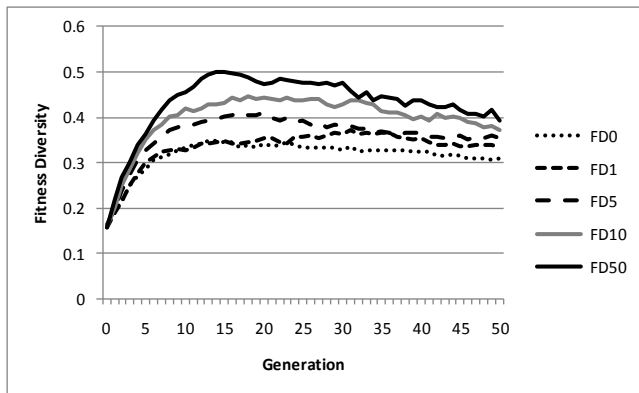
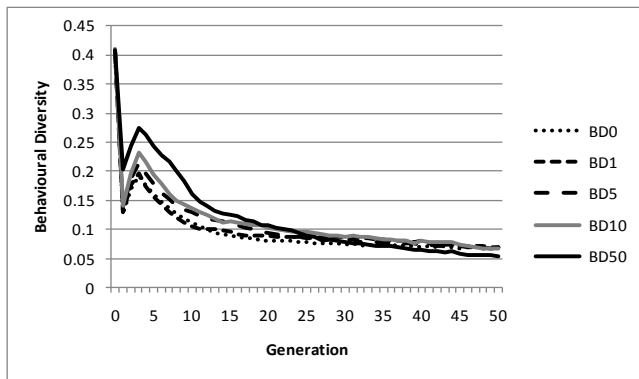
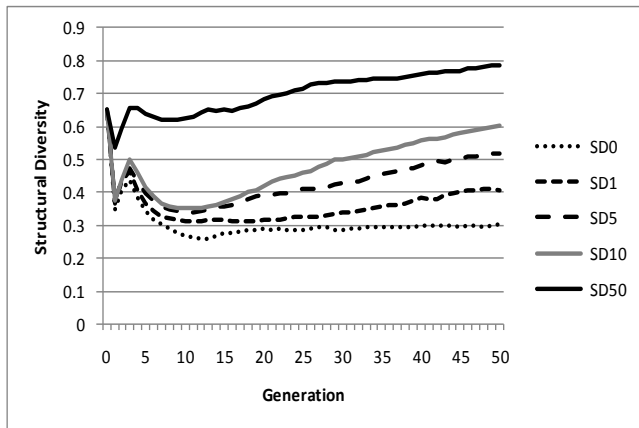
We can chart the changes in these diversity values as evolution proceeds, both with and without mutation. As mentioned earlier, mutation is conventionally applied infrequently. We therefore investigate the effect of mutation at a commonly used frequency of 1%. However, since this is so small a perturbation, we also attempt to magnify any diversity enhancing effects by using higher mutation frequencies of 5%, 10% and 50%. The results for the even-4 parity problem are plotted in Figure 1.

As can be seen from these graphs, mutation appears to have very little effect on our diversity metrics. This is particularly true for structural diversity, which tends to remain high throughout the lifetime of the runs, even without mutation. It would seem that mutation can slightly increase the BD and FD values, but only when it is applied at the unusually high rate of 50%.



**Fig. 1.** Changes in diversity levels for even-4 parity problem, averaged over 100 runs. Note that SD10 indicates structural diversity with 10% mutation rate, etc.

Contrast this now with the graphs obtained for the symbolic regression problem, shown in Figure 2. Here, the effects of mutation on diversity is much more apparent, especially for SD and FD. There are also noticeable increases in BD in the earlier stages of evolution, but this tails off in later generations.



**Fig. 2.** Changes in diversity levels for symbolic regression problem, averaged over 100 runs.

Lack of space prevents us from including graphs for all the problems here, but we have attempted to summarize the various statistics in Table 2. In this and the later tables, the figures for diversity and fitness are averaged over 100 runs. The column headings can be explained as follows:

**Mut:** The mutation rate being applied.

**SD, BD, FD:** Average diversity value over all 50 generations.

**Solns:** Number of solutions found in 100 runs.

**Fitness:** Average of the best fitness values found at the end of the runs.

**Effort:** Amount of computation effort required, calculated as the total number of fitness evaluations divided by the number of solutions successfully evolved.

**Sig:** This states whether the best fitness values attained on all the runs using mutation are significantly different from those attained without mutation. Significance is determined using a statistical t-test with  $p=0.05$ .

It can be seen from this that the picture for the ant trail problem and the mux problem is similar to that for even-4 parity. Mutation has negligible impact on any of our three forms of diversity. Moreover, it does nothing to help in finding solutions. Fitness values are not altered to a statistically significant degree in any of the three problems (even though the solution rate for the ant problem is actually worse).

As the earlier graphs showed, somewhat different results are seen for the symbolic regression problem. Without mutation, the average SD value for this problem is much lower than it is for the other problems. Rising mutation frequencies are able to boost the SD considerably, from 0.3 up to 0.7 when 50% mutation is applied. This rise is accompanied by a significant increase in FD levels too, although BD levels are less affected. More importantly, the increase in the use of mutation also sees a concomitant increase in the solution finding performance of the GP algorithm. This is substantial, rising from a 6% solution rate when no mutation is used, right up to a 48% solution rate when 50% mutation is applied. The computational effort involved correspondingly reduces by a factor of 10.

Were it not for these statistics for the regression problem, one might indeed suspect that the introduction of a mutation operator into a GP system is not worth the implementation effort. However, the regression problem prompts us to wonder whether it is the increases in diversity which lead causally to the improvements in performance.

To investigate this further, we can make changes to the mutation operator with the objective in mind of attaining further increases in population diversity. It should be noted, however, that there is little we can do about increasing FD values directly. This is particularly true in the early stages of evolution, where the population contains a large number of low fitness individuals. Attempting to introduce fitter individuals would necessitate algorithms of the complexity of the evolutionary process itself, and would therefore defeat the object of the exercise. It may be possible to increase fitness diversity in later generations by reintroducing low fitness values that have already been winnowed out, but the random nature of mutation is such that this is likely to happen in any case.

What we can do quite easily is to force the mutation operator to search for structurally or behaviourally unique individuals, and thereby enhance population diversity. This may in turn impact upon fitness diversity. To implement this, we insert a loop into the mutation operator so that it contains to mutate the selected program tree until either an individual is generated that is unique in the population, or a predefined maximum number of attempts is reached (20 in our experiments).

**Table 2.** Standard mutation – diversity and performance.

<b>Problem</b>	<b>Mut.</b>	<b>SD</b>	<b>BD</b>	<b>FD</b>	<b>Solns</b>	<b>Fitness</b>	<b>Effort</b>	<b>Sig.</b>
Ant	0	0.81	0.11	0.24	14	18.9	146592	-
	1	0.83	0.11	0.23	12	19.6	172691	n
	5	0.84	0.12	0.25	10	21	210534	n
	10	0.86	0.11	0.24	18	19.1	112181	n
	50	0.89	0.12	0.26	5	21.2	440306	n
Even	0	0.89	0.06	0.32	12	1.7	177316	-
	1	0.89	0.06	0.32	8	1.8	273836	n
	5	0.89	0.06	0.32	15	1.8	139351	n
	10	0.89	0.06	0.32	16	1.7	132661	n
	50	0.89	0.08	0.34	10	1.9	220151	n
Mux	0	0.83	0.08	0.15	63	1.5	19056	-
	1	0.84	0.08	0.15	56	1.87	24556	n
	5	0.85	0.08	0.15	69	1.56	16482	n
	10	0.86	0.08	0.16	70	1.3	16628	n
	50	0.88	0.11	0.19	66	1.52	18520	n
Regress	0	0.30	0.1	0.32	6	0.92	369551	-
	1	0.36	0.1	0.33	12	0.68	181637	y
	5	0.43	0.1	0.36	20	0.48	103911	y
	10	0.47	0.11	0.4	23	0.36	88551	y
	50	0.7	0.12	0.43	48	0.21	34529	y

**Table 3.** SD-favourable mutation – diversity and performance.

<b>Problem</b>	<b>Mut.</b>	<b>SD</b>	<b>BD</b>	<b>FD</b>	<b>Solns</b>	<b>Fitness</b>	<b>Effort</b>	<b>SigT2</b>
Ant	0	0.81	0.11	0.24	14	18.9	146592	-
	1	0.81	0.16	0.24	9	20.2	239476	n
	5	0.84	0.17	0.24	9	21.6	237966	n
	10	0.85	0.12	0.24	11	21.2	189194	n
	50	0.9	0.12	0.25	15	20.8	136688	n
Even	0	0.89	0.06	0.32	12	1.7	177316	-
	1	0.89	0.07	0.33	14	1.7	152136	n
	5	0.89	0.06	0.32	21	1.6	97084	n
	10	0.89	0.07	0.33	11	2.0	199945	n
	50	0.9	0.08	0.34	6	1.9	375184	n
Mux	0	0.83	0.08	0.15	63	1.5	19056	-
	1	0.84	0.08	0.15	60	1.48	21080	n
	5	0.85	0.08	0.15	59	1.98	21653	n
	10	0.85	0.08	0.16	75	1.05	13434	n
	50	0.89	0.11	0.19	76	1.25	14470	n
Regress	0	0.30	0.1	0.32	6	0.92	369551	-
	1	0.32	0.1	0.34	11	0.49	197009	y
	5	0.42	0.11	0.36	26	0.43	75633	n
	10	0.49	0.12	0.40	32	0.32	59812	n
	50	0.75	0.11	0.43	39	0.25	47076	n

**Table 4.** BD-favourable mutation – diversity and performance.

Problem	Mut.	SD	BD	FD	Solns	Fitness	Effort	SigT2
Ant	0	0.81	0.11	0.24	14	18.9	146592	-
	1	0.82	0.12	0.24	22	16.6	89352	y
	5	0.82	0.13	0.27	19	17.5	123756	y
	10	0.82	0.11	0.24	20	18.1	135553	n
	50	0.89	0.14	0.29	17	15.6	245816	y
Even	0	0.89	0.06	0.32	12	1.7	177316	-
	1	0.89	0.07	0.33	14	1.8	162711	n
	5	0.89	0.07	0.35	26	1.5	104136	y
	10	0.89	0.09	0.36	13	1.7	297161	n
	50	0.84	0.19	0.43	21	1.4	568447	y
Mux	0	0.83	0.08	0.15	63	1.5	19056	-
	1	0.85	0.08	0.15	62	1.65	21331	n
	5	0.84	0.09	0.17	74	1.07	15489	n
	10	0.85	0.1	0.19	75	1.06	22267	n
	50	0.82	0.23	0.27	85	0.68	32063	y
Regress	0	0.30	0.1	0.32	6	0.92	369551	-
	1	0.34	0.1	0.36	12	0.54	179910	n
	5	0.34	0.12	0.4	25	0.35	85973	y
	10	0.37	0.14	0.42	26	0.27	86586	y
	50	0.47	0.27	0.56	47	0.17	61101	n

The effects of modifying mutation so that it promotes the chances of finding individuals that are unique in their structure is shown in Table 3. Although it is expected that this will most directly affect the SD levels, it is possible that this will have knock-on effects on the other diversity metrics, and so these are included for comparison.

Another point to make about Table 3 is that the final column has been altered slightly. Headed ‘SigT2’, this column now makes a direct comparison between the set of best fitness values obtained using this modified mutation operator and the set used to build Table 2, where standard mutation was used.

The first thing to note is that SD figures in this table are little different from those given in Table 2, meaning that the modified mutation operator does little to increase SD levels over and above those obtained by straightforward mutation. Indeed, execution monitoring of the GP system reveals that very few attempts are required to find structurally unique individuals.

With regard to performance, there is again barely any discernible improvement. In nearly all cases the changes in best fitness levels are not statistically significant. This is largely true even for the symbolic regression problem, despite there being a substantial increase in the solution rate when 10% mutation is applied, and a substantial drop in the rate when 50% mutation is used. The one exception in the whole table is for 1% mutation in the regression problem: although there is not much of a change in the solution rate, the improvement in fitness levels, from an average of 0.68 down to an average of 0.49, is substantial enough to be deemed significant.

If promoting structural diversity in this way has so little impact, what about promoting behavioural diversity? Table 4 shows what happens if we alter the mutation operator so that, instead of searching for unique structures, it searches for unique behaviours. Again, the column headed ‘SigT2’ tests for significance of fitness values when compared with those used in Table 2.

At first sight, it would appear that the average best fitness achieved at the end of the runs is better for all non-zero mutation rates in all problems. However, statistical analysis reveals that some of these differences are not significant. Despite this, the approach fares much better than the previous SD-promoting technique. In half of the cases (8 out of 16), the improvement in fitness levels is statistically significant. For some of these, the benefits are substantial. In the ant, even parity and mux problems, use of a 50% mutation rate leads to an increase in the solution rates by 240%, 110% and 29%, respectively.

Comparing this latest table with the figures for the SD-enhancing operator given in Table 3 also shows improvements. The technique leads to superior solution-finding performance in all cases except for the 5% and 10% mutation rates in the symbolic regression problem. Again, when 50% mutation is applied the benefits are substantial.

These improvements do, however, come at a cost. The effort columns in the tables reveal a significant increase in the number of fitness evaluations that must be performed for each solution obtained. Although generating new programs which are structurally distinct in the population seems relatively easy, finding unique behaviours is not. In general, larger numbers of offspring must be created before unique behaviours are found or the attempts aborted,

and comparing behaviours necessitates the fitness evaluation of each new candidate program.

## 5. CONCLUSIONS

In this paper we have attempted to investigate the importance or otherwise of the GP mutation operator with regard to its role of preserving diversity and thereby increasing the chances of finding solutions. We can now return to the questions we posed in our introductory remarks. The first of these was whether mutation does in fact do anything to enhance diversity, and our answer would have to be that it seems to depend on the nature of the problem. For three of our four benchmark problems it appears to have little or no effect on any of the diversity metrics we have defined. For symbolic regression, however, mutation is able to increase substantially the structural diversity levels, and to a lesser extent the fitness diversity.

Our next question was whether mutation can improve solution-finding performance. For the three problems in which there is no change in diversity levels, the answer again seems to be no. More interestingly, however, the symbolic regression problem *does* exhibit marked increases in performance as the mutation rate is increased.

Our final question concerned what would happen if we were to alter the mutation operator in such a way that it increases the chances of generating individuals that are unique in either structure or behaviour, thereby enhancing diversity. Here, the answer may have come as more of a surprise. If we alter mutation in favour of increased structural diversity, the results suggest that the effects are negligible. Even in the case of symbolic regression, where structural diversity tends to be much lower than it is in the other problems, the operator is unable to improve things beyond what can already be achieved via standard mutation.

If, on the other hand, we modify mutation so that it promotes the occurrence of individuals which are unique in behaviour rather than structure, we observe a related increase in fitness diversity, but more importantly, a statistically significant increase in solution-finding ability in half the test cases, with no evidence of deterioration in the other half. The price to be paid for this, however, is an increase in the computational effort required to discover those solutions.

To answer the wider question of whether mutation can be a useful operator in GP, the experimentation described here would prompt us to issue a carefully guarded yes, but only when the operator is coaxed into promoting diversity, and only when that diversity is measured in a particular way. Clearly, there is a lot more research that needs to be done in this area, as there is into the nature of GP operators in general.

## 6. REFERENCES

- [1] Beadle, L. and Johnson, C.G. Semantic Analysis of Program Initialisation in Genetic Programming. In *Genetic Programming and Evolvable Machines*, vol. 10, no. 3, 2009, 307-337.
- [2] Beadle, L. and Johnson, C.G. Semantically Driven Crossover in Genetic Programming. In *IEEE World Congress on Computational Intelligence*, IEEE Press, 2008, 111-116.
- [3] Burke, E., Gustafson, S., Kendall, G. and Krasnogor, N. Advanced Population Diversity Measures in Genetic Programming. In *Proc. 7th Int. Conf. Parallel Problem Solving from Nature (PPSN)*, *Lecture Notes in Computer Science*, vol. 2439, Guervos, J.J.M et al (eds), Springer-Verlag, Berlin Heidelberg, 2002, 341-350.
- [4] Burke, E., Gustafson, S., and Kendall, G. Diversity in Genetic Programming: An Analysis of Measures and Correlation with Fitness. In *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, Feb. 2004, 47-62.
- [5] de Jong, E.D., Watson, R.A. and Pollack, J.B. Reducing Bloat and Promoting Diversity using Multi-Objective Methods. In *Proc. Genetic Evolutionary Computation Conf.* Spector, L. et al (eds), San Francisco, CA, USA, 2001, 11-18.
- [6] Fogel, L.J. *Intelligence through Simulated Evolution: Forty Years of Evolutionary Programming*. Wiley, New York, 1992.
- [7] Jackson, D. Promoting Phenotypic Diversity in Genetic Programming. In: *Proc. 11th International Conference Parallel Problem Solving from Nature (PPSN XI)*, Krakow, Poland, *Lecture Notes in Computer Science* vol. 6239, Schaefer, R. et al (eds), Springer-Verlag, Berlin, Heidelberg, 2010, 472-481.
- [8] Jackson, D. Phenotypic Diversity in Initial Genetic Programming Populations. In *Proc. EuroGP 2010, Lecture Notes in Computer Science* vol. 6021, Esparcia-Alcazar, A.I. et al (eds), Springer-Verlag, Berlin, Heidelberg, 2010, 98-109
- [9] Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
- [10] Luke, S. And Spector, L. A Revised Comparison of Crossover and Mutation in Genetic Programming. In *Proc. 3rd Annual Conf. Genetic Programming*, Koza, J.R. et al (eds), Morgan Kaufmann, 1998, 208-213.
- [11] Miller, J.F. and Thomson, P. Cartesian Genetic Programming. In *Proceedings of the 3rd European Conference on Genetic Programming, Lecture Notes in Computer Science* vol. 1802, Springer-Verlag, Berlin Heidelberg, 2000, 121-132.
- [12] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998.
- [13] O'Reilly, U.-M. and Oppacher, F. Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing. In *Proc. Parallel Problem Solving from Nature (PPSN) III, Lecture Notes in Computer Science* vol. 866, Y. Davidor et al (eds), Springer-Verlag, Berlin Heidelberg, 1994, 39-46.
- [14] Rosca, J.P. Genetic Programming Exploratory Power and the Discovery of Functions. In *Proc. 4th Conf. Evolutionary Programming*, McDonnell, J.R. et al (eds), San Diego, CA, USA, 1995, 719-736.
- [15] Rosca, J.P. Entropy-Driven Adaptive Representation. In *Proc. Workshop on Genetic Programming: From Theory to Real-World Applications*, Rosca, J.P. (ed), Tahoe City, CA, USA, 1995, 23-32.
- [16] White, D.R. and Poulding, S. A Rigorous Evaluation of Crossover and Mutation in Genetic Programming. In *Proc. EuroGP 2009, Lecture Notes in Computer Science*, vol. 5481, Vanneschi, L. et al (eds), Springer-Verlag, Berlin Heidelberg, 2009, 220-231.