The Effect of Extensive Use of the Mutation Operator on Generalization in Genetic Programming using Sparse Data Sets

Wolfgang Banzhaf[1], Frank D. Francone[2] and Peter Nordin[1]

[1] Department of Computer Science, Dortmund University,
Joseph-vonFraunhofer-Str. 20, 44227 Dortmund, GERMANY
banzhaf,nordin@LS11.informatik.uni-dortmund.de
[2] Law Office of Frank D. Francone,
4806 Fountain Ave. #77, Los Angeles, California 90027, USA
ytns65a@prodigy.com

Abstract. Ordinarily, Genetic Programming uses little or no mutation. Crossover is the predominant operator. This study tests the effect of a very aggressive use of the mutation operator on the generalization performance of our Compiling Genetic Programming System ('CPGS'). We ran our tests on two benchmark classification problems on very sparse training sets. In all, we performed 240 complete runs of population 3000 for each of the problems, varying mutation rate between 5% and 80%. We found that increasing the mutation rate can significantly improve the generalization capabilities of GP. The mechanism by which mutation affects the generalization capability of GP is not entirely clear. What is clear is that changing the balance between mutation and crossover effects the course of GP training substantially — for example, increasing mutation greatly extends the number of generations for which the GP system can train before the population converges.

1 Introduction

Evolutionary Algorithms may be classified into two different groups based on their relative use of Crossover and Mutation. Genetic Algorithms ('GA') and Genetic Programming ('GP') tend to use little or no mutation. In these paradigms, crossover is by far the dominant operator [4, 3]. On the other hand, Evolutionary Strategies ('ES') and Evolutionary Programming ('EP') have their traditional focus on the mutation operator [11, 12, 2]. Recently, the crossover operator has been challenged by researchers as being relatively ineffective in both Genetic Algorithms [2] and in Genetic Programming [5]. This paper therefore undertakes the first systematic study of the effect of changing the relative balance of mutation and crossover in genetic programming.

We undertook this study with our Compiling Genetic Programming System ('CGPS'). Compiling Genetic Programming is the direct evolution of binary machine code. A CGPS program is a sequence of 32 bit binary machine instructions

in memory. Genetic operators act directly on the 32 bit instructions. When executed, the instructions in a CGPS program cause the CPU to perform simple operations on the CPU's hardware registers.[1] By way of example, a CGPS program might, when executed, cause the CPU to add the value in register 1 to the value in register 2 and then place the result in register 3. Each such CGPS instruction corresponds to three nodes in a hierarchical (tree-based) GP-system. Functions of great complexity can be evolved with simple arithmetic functions in a register machine [9].

2 Specification of the Experiments

Table 1 summarizes our experimental setup [4]. Some items merit separate discussion and for those items, the experimental setup is described in the text.

2.1 The CGPS Mutation Operator

The CGPS mutation operator flips bits inside the 32-bit instructions that comprise a CGPS program. The mutation operator ensures that only the instructions in the function set are generated and that the register and constant values are within the predefined ranges allowed in the experimental setup. To keep the number of individuals evolved even as between mutation and crossover, we either crossover or mutate two individuals each time a genetic operator is executed.

2.2 The Data Sets Used In This Study

The Gaussian 3D Data Set. Gaussian 3D is an artificially generated Classification problem. Both output classes are comprised of points with zero mean and normal distribution. The standard deviation of Class 0 is 1. The standard deviation of Class 1 is 2 [13] (p. 14). Using 5000 training samples, the ELENA KNN benchmark error rate is 22.2%. Therefore, a CGPS hit rate of 77.8% would match the benchmark. The Gaussian problem is by far the most difficult problem reported in this paper.

Phoneme Recognition Data Set. The Phoneme recognition database requires an ML system to classify two sets of spoken vowels into oral or nasal vowels. Using 5404 training samples, the KNN benchmark error rate is 14.2% [13] (p. 30-1).Therefore, a CGPS hit rate of 85.8% would be match the benchmark. This problem is the next most difficult problem reported in this paper.

[1] For a more detailed description of CGPS, please see [7, 8, 9, 10].

IRIS Data Set. The IRIS database requires an ML system to classify measurements from photos of Irises into three different types of Irises. Using 150 training samples, the KNN benchmark error rate is between 0% and 7.3% (95% confidence interval)[13] (p. 35-6). Therefore, a CGPS hit rate of 92.7% would match the benchmark. The Iris data set is by far the easiest of the problems attacked here. It turned out to be almost trivial for CGPS to solve.

Sparse Data Sets. We made each of the above problems more difficult for CGPS by withholding between 50% and 98% of the data upon which the benchmarks were calculated. On Phoneme and Gaussian, we gave our system only 100 data points for training and another 100 for testing rather than the over 5000 data points used for the KNN Benchmarks[13]. For the Iris problem, we used 75 training examples as opposed to the 150 training samples for the benchmark.[13]

Table 1. Experimental Specification. EDI's were enabled on half of the runs.

Description	*Value*
Objective :	Classification of data sets from ELENA Database
Terminal set :	Integer constants initialized from 0 - 255
Function set :	Addition, Multiplication, Subtraction, Division
Number of Hardware Registers :	One more than the number of classes in the data
Number of Fitness/Testing Cases :	100/100
Wrapper :	None
Population Size :	3000
Selection:	Tournament. 4/2
Maximum Individual Size:	256 instructions.
Total Number of Runs :	480 (Gauss), 240 (Phoneme), 240 (Iris)

2.3 Other Important Features of the System

Variable Parameters/Number of Total Runs. We varied three of the parameters from run to run. These parameters were: Parsimony Factor (0, 0.1, 1), Explicitly Defined Introns (enabled/not enabled), and Crossover/Mutation Mix (5%, 20%, 50%, 80% mutation). There are twenty-four different combinations of the above parameters (see Table 1). We ran each such combination on ten different random seeds. As a result, we conducted a total of 240 runs per problem set.

Measuring Fitness. For our fitness function, we selected one hardware register as the output register. We also arbitrarily assigned a target output value to each class (the 'Class Target Value'). If a particular fitness case should have been classified as being a member of Class n, then the fitness of the individual for that fitness case is the absolute value of the difference between the value in the output register and the Class Target Value for class n. The sum of these absolute deviations over all fitness cases was the fitness of the individual.

Measuring Hits. We counted a prediction as a 'Hit' if the absolute value of the difference between the output register and the Class Target Value was less than 50.

Measuring Generalization. We assessed generalization by separating our data into training and testing sets. Each generation, the best individual on the training set is evaluated on the testing set and the results of that test are preserved. [2]

Terminating a Run. We have previously reported that where destructive crossover falls to below 10% of all crossover events, all effective training is over [10]. Accordingly, we monitored the rate of destructive crossover during training. When destructive crossover fell to 10% of the total crossover events for any given generation, we terminated the run. This approach saves a substantial amount of CPU time. Otherwise, we terminated a run at 200 generations.

3 Results.

We use two measures of how much mutation affected the performance of our GP system — the mean of all runs and the probability that a particular mutation rate would result in a particularly good run.

3.1 Results on the Iris Data Set.

We report the results on the Iris Data Set here because they are so straightforward. The Iris problem was very easy for CGPS. Almost all runs matched or exceeded the benchmark — most of the runs did so by the end of the first generation. On this data set, changes in the mutation operator had no measurable effect on generalization. The remainder of this Results section applies only to the more difficult Phoneme and Gaussian Data sets.

[2] While our method does provide much useful information about the generalization capabilities of our system[6] (pp. 335-376), the use of a third data set would improve it. The ability of the best generalizer from a run to generalize on this third data set would measure the performance of that run more accurately. We plan to implement this capability in the future.

3.2 Effect of Varying Mutation Rates on the Mean of The Data Set.

Table 2 reports the effect of varying the mutation rate on the generalization capabilities of CGPS. The measure of generalization capabilities used is the performance in % hits on the testing set as a percentage of the ELENA KNN Benchmark.

Table 2. Effect of Varying the Mutation Rate on CGPS Generalization, including a Statistical Analysis: C.C.: Correlation Coefficient, S.S.: Statistical Significance.

Mutation Rate	*Gaussian 3D*	*Phoneme*	*Both Problems*
5%	72.3%	90.8%	81.5%
20%	75.3%	90.6%	82.9%
50%	75.7%	91.5%	83.6%
80%	73.4%	91.4%	82.4%
C.C.	0.199	0.111	0.157
S.S.	99%	92%	99%

Thus, the effect of changing the mutation rate from 5% to 50% on the mean of 480 runs varies from a 1% to a 3.5% improvement on the testing set, depending on the problem. This effect is both important and statistically significant. Important, because in higher ranges small changes in generalization are disproportionally difficult to achieve and often make the difference between a successful prediction model and a failure. The result is also statistically significant. Table 2 shows the correlation coefficients and statistical significance level over all 480 runs for the correlation between the negative of the square of the mutation rate and the mean generalization capability over all 480 runs on both the Gaussian and Phoneme problem sets.

3.3 Effect of Varying the Mutation Rate on the Probability of Evolving Outstanding Runs.

In real world applications, the point is not to evolve a lot of runs with a somewhat higher mean. The point is to evolve one or a few runs where the best individual's generalization capability is outstanding. Our results strongly suggest that increasing the mutation rate dramatically increases the probability of generating such outstanding runs. Table 3 shows the proportion all runs that are in the top 5% of runs by mutation rate.

Table 3. Percentage of Top 5% of Runs Comprised of Various Mutation Rates. 480 Runs. Phoneme and Gaussian Data Sets.

Mutation Rate	*Gaussian 3D*	*Phoneme*	*Total*
5%	0%	0%	0%
20%	33%	8%	21%
50%	58%	58%	58%
80%	8%	33%	21%

This effect is statistically significant. Table 4 presents the correlation coefficients and the statistical significance levels of the coefficients for the top 5% , 10% and 20% of runs.

Table 4. Correlation Coefficients and Statistical Significance Levels for Correlation Between Mutation Rate and the Probability that A Run Will Be Among the Best Runs. 480 Runs. Gaussian and Phoneme Data Sets.

	C.C.	*S.S.*
Top 5% of Runs	0.887	99%
Top 10% of Runs	0.760	95%
Top 20% of Runs	0.828	99%

Table 3 somewhat overstates the effect of higher mutation rates on the actual probability of a run being in the best 5% of runs. Simply put, it took longer for higher mutation runs to find a best generalizer than it took lower mutation runs. Table 5 shows the increases in the probability of a run being in the top 5% and the top 10% of runs adjusted for this difference in CPU Time.

Table 5. Effect in increasing Mutation Rate On Probability of A Run Being One of the Best Runs. Adjusted for CPU Time. Factors of Increase in Probability of Run being in Top 5 and 10 %, respectively. 480 Runs. Gaussian and Phoneme Data Sets

Change in Mutation Rate	*Factor 5%*	*Factor 10%*
From 5% to 20%	1.8	1.7
From 5% to 50%	3.4	3.0
From 5% to 80%	1.6	2.3

A factor of 3.4 in Table 5 means that, adjusted for CPU time, a run with a 50% mutation rate is 3.4 times more likely to be in the top 5% of runs than a run with a 5% mutation rate. Put another way, one effect of higher mutation rates is to increase the size of the high generalization tail of the distribution of multiple runs.

3.4 Effect of Varying the Mutation Rate on Indicia of Training.

Introduction. Increasing the mutation rate profoundly changes the way a CGPS run trains. The purpose of this section is to describe some of those changes.

Table 6 sets forth the effect of various mutation rates on various training indicators. We discuss each of these items below.

Table 6. Various Training Indicators as a Function of Mutation Rate. Mean of 480 Runs: Introns As Percent Of Total Instructions; Effective Size of Individual; Number of Generations to Best Generalizer / Number of Generations to Termination of Run.

Mutation Rate	*Introns*	*Effective Size*	*Best Generalizer*	*Run Termination*
5%	74%	7	12	151
20%	68%	12	22	167
50%	63%	12	24	176
80%	58%	9	26	175

Introns As a Percentage of Total Instructions. First order Introns are evolved single instructions that have no effect on the fitness calculation [8]. We have frequently observed first order introns in our previous work [10]. A typical first order intron is:

```
register1 = register1 + 0 (1);
```

During training we measured the percentage of all instructions in the population that were first order introns at the time the Best Generalizer was found. We found that increasing the mutation rate greatly reduces the proportion of the entire population that is comprised of introns, see Table 6. Our previous work suggests that changes of this magnitude shown in Table 6 are related to important changes in the training of a GP run [10].

Mean Individual Effective Size. 'Effective Size' is the number of instructions in a GP individual that have an effect on the fitness of the individual [10]. We measured the effective size of all individuals in each GP run at the time the Best Generalizer was found and present the average for the population in Table 6. Here, raising the mutation rate significantly increases the average effective size of all the individuals in the population.

Mean Generations To Best Generalizer. Increasing the mutation rate effectively doubles the number of generations before a run locates the best individual generalizer. Table 6.

Mean Generations To Run Termination. Increasing the mutation rate increases the number of generations that it takes for a run to terminate. Table 6. In our system, this is mostly a measure of the time it takes for the number of introns to multiply so much that effective training is over [10]. This measure is, therefore, consistent with our observation above that there are more introns in low mutation rate runs. Table 6.

4 Discussion

Increasing mutation rates is a powerful way to improve the generalization capabilities of GP. Over a wide variety of parameters and over 480 runs, a 50/50 balance of mutation and crossover consistently performed better than the low mutation rates traditionally used in GP. However, higher mutation rates should only be expected to work on difficult data sets. As the data sets that we studied increased in difficulty, the effect of higher mutation rates also increased. The data sets we studied got more difficult in the following order: Iris, Phoneme and Gaussian. The effect of mutation on the mean of the generalization capabilities increases in the same order as follows: 0%, +1%, + 3.5%.

That said, the mechanism by which the mutation operator acts to improve generalization is not entirely clear. Several factors, however, point to increased diversity as a factor.

Diversity and Introns. Increasing the mutation rate reduces the number of introns. Table 6. Consider the typical first order intron described in (1). Now, imagine the effect of the possible mutations that could be applied to (1). Changing the operator from "plus" to "times" or "minus" will convert the intron into working code. Changing the constant or changing the register of the argument or the result are also likely to convert the intron into working code. In short, mutation tends with high probability to destroy typical first order introns by converting them into code that effects the fitness calculation.[3]

[3] It is possible to imagine first order introns that would be resistent to mutation under the correct circumstances. An example of such an intron would be:

Diversity And Effective Length. Table 6 also shows that higher mutation rates are associated with longer effective length in the entire population. Effective length is a measure of the number of instructions in an individual that effect fitness. Longer effective length could easily be a reflection of a constant infusion of new effective code into the population. Such an infusion could easily be the result of the mutation operator converting typical introns, such as (1), into effective code. Of course, such an infusion of fresh genetic material would tend to maintain the diversity of the population longer.

But for this mechanism to supply a continual flow of new genetic material into the population, the supply of introns in the population must somehow replenish itself. Otherwise, the mutation opreator in high mutation rate runs should rapidly exhaust the population's supply of introns. But Table 6 shows that introns are merely reduced, not eliminated. Our previous research strongly suggests that introns are formed by a GP population in response to the crossover operator [10]. The mechanism that suggests itself then is that crossover creates introns and mutation changes them into fresh genetic material. That the mutation operator works best at a 50/50 balance of crossover and mutation suggests the balance between crossover and mutation is a key to maintaining the flow of new genetic material into the population.

Diversity And Length of Effective Training. The effect of mutation on length of training is also consistent with our diversity explanation. Higher mutation runs continue to evolve better generalizing individuals for almost twice as many generations as do lower mutation runs, see Table 6. Of course, higher diversity in the population would be expected to cause such an effect. This observation hits at a central problem in GP — GP is plagued by premature convergence. That is, GP populations often lose their ability to evolve before the problem is solved. Increasing the mutation rate makes the runs continue to evolve for about twice as long (Table 6) - that is, a high mutation run maintans its ability to evolve for longer. This observation explains the better results in high mutation runs on difficult problems — with more time to explore the search space, such runs did better. This observation also explains why higher mutation rates did not effect the IRIS results. CGPS solved the IRIS problem almost immediately - most runs equalled or exceeded the benchmark by the end of the first generation. Simply put, CGPS did not need the extra evolvability lent by high mutation rates to solve the IRIS problem.

$register1 = register2 >> register3(2)$; "Shift-right" (>>) is effectively a division by powers of 2. In this example, mutations that change argument registers or the content of argument registers are less likely to have effects on fitness than the similar changes in the typical intron as shown in (1), provided values in the argument registers are in certain broad ranges. In short, a type (2) intron may be relatively more resistent to mutation than a type (1) intron. We have, indeed, observed these type (2) introns to proliferate heavily in individuals from runs with very high mutation rates. We have never observed such introns in our previous work in low mutation runs. This new type of intron is an area in which further research is suggested.

Conclusion. In conclusion, many factors point to the improved diversity of the population as a primary candidate for further research to explain how mutation improves the generalization in GP runs. One other area for future research would be to incorporate some of the ES and EP type mutation strategies into GP mutation operators.

Acknowledgments

We would like to thank the ELENA partners for the invaluable work that they undertook in assembling the ELENA databases and in their analyses of the various data sets therein. We would also like to acknowledge partial support from the Ministerium für Wissenschaft und Forschung des Landes Nordrhein-Westfalen, under grant I-A-4-6037.I.

References

1. T. Bäck: Optimal Mutation Rates in Genetic Search. In: S. Forrest (Ed.): Proc. 5th Int. Conference on Genetic Algorithms, ICGA-93. San Mateo: Morgan Kaufmann 1993
2. D. Fogel, L. Slayton: On the Effectiveness of Crossover in Simulated Evolutionary Optimization. Biosystems **32** 171 - 182 (1994)
3. D. Goldberg: Genetic Algorithms in Search Optimization & Learning, Reading: Addison-Wesley 1989
4. J.R. Koza: Genetic Programming. Cambridge (USA): MIT Press 1992
5. K. Lang: Hill Climbing Beats Genetic Search on a Boolean Circuit Synthesis Problem of Koza's. In: A. Prieditis, S. Russell (Eds.): Proc. 12th Int. Conference on Machine Learning. San Mateo: Morgan Kaufmann 1995
6. T. Masters: Advanced Algorithms for Neural Networks. New York: Wiley 1995
7. J.P. Nordin, J.P.: A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In: K. Kinnear, Jr. (ed.). Advances in Genetic Programming. Cambridge MA: MIT Press 1994
8. J.P. Nordin, W. Banzhaf: Complexity Compression and Evolution. In: L. Eshelman (Ed.): Proc. 6th Int. Conference on Genetic Algorithms, ICGA-95. San Mateo: Morgan Kaufmann 1995
9. J.P. Nordin, W. Banzhaf: Evolving Turing Complete Programs for a Register Machine with Self Modifying Code. In: L. Eshelman (Ed.): Proc. 6th Int. Conference on Genetic Algorithms, ICGA-95. San Mateo: Morgan Kaufmann 1995
10. J.P. Nordin, F.D. Francone, W. Banzhaf: Explicitly Defined Introns and Destructive Crossover in Genetic Programming. In: K. Kinnear, Jr., P. Angeline (eds.): Advances in Genetic Programming 2. Cambridge MA: MIT Press 1996, in press
11. I. Rechenberg: Evolutionsstrategie 94. Stuttgart: Holzmann-Froboog 1994 (2nd. ed.)
12. H.P. Schwefel: Evolution and Optimum Seeking. New York: Wiley 1995 (2nd. ed.)

13. The ELENA Partners: C. Jutten, Project Coordinator: ESPRIT Basic Research Project Number 6891, Document Number R3-B1-P. Available via ftp at either ics.uci.edu or at satie.dice.ucl.ac.be, 1995

This article was processed using the LaTeX macro package with LLNCS style