Numeric Mutation: Improved Search in Genetic Programming

Matthew Evett and **Thomas Fernandez**
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, Florida 33431
{matt, tfernand}@cse.fau.edu

Abstract

Genetic programming is relatively poor at discovering useful numeric constants for the terminal nodes of its s-expression trees. In this paper we outline an adaptation to genetic programming, called *numeric mutation*. We provide empirical evidence and analysis that demonstrate that numeric mutation makes a statistically significant increase in genetic programming's performance for symbolic regression problems.

Introduction

One of the weaknesses of genetic programming (GP, henceforth) is the difficulty it suffers in discovering useful numeric constants for the terminal nodes of the s-expression trees. GP's difficulty with numeric constant generation is relatively well known. In a speech at a recent conference John Koza said:

> The finding of numeric constants is a skeleton in the GP closet... [and an] area of research that requires more investigation.(Koza 1997)

GP's difficulty in evolving a numeric constant derives from its representation as a tree node, while the reproduction operations (including mating and mutation) affect only the structure of the trees, not the composition of the nodes themselves. Consequently, individual numeric constants are not affected by mutation and mating and thus cannot benefit from them.

The traditional way of generating new numeric constants is indirect, by combining existing numeric constants within novel arithmetic s-expressions. The leaves of the trees corresponding to such s-expressions are all numeric constants. Each such s-expression can thus be viewed as a single numeric constant terminal node, with a value equal to that of the s-expression (usually distinct from that of any of the leaves). We call this process of numeric constant creation *arithmetic combination.*

It is also possible to generate numeric constants even when none are provided in the original terminal set. For example, a terminal representing a variable could appear in an s-expression consisting of the variable being divided by itself, effectively yielding the constant 1.0. Once the constant 1.0 exists, 2.0 can evolve via an s-expression that adds 1.0 to itself, etc. In this way the GP process can generate an arbitrary number of constants, even when no numeric constants are included in the original terminal sets. We call this process of numeric constant creation *arithmetic genesis.*

Although the spontaneous emergence of constants is possible via arithmetic genesis and arithmetic combination, the techniques are tedious and inefficient. We are examining several techniques for facilitating the creation of useful, novel numeric constants during a GP run. In this paper we report on one such technique, *numeric mutation.* We demonstrate that numeric mutation provides a significant improvement in the ability of GP to solve symbolic regression problems.

History

Some of the early enhancements to the GP process facilitated the creation of constants. These enhancements (Koza 1992) consisted of including a small number of numeric constants and/or the *ephemeral random constant*, $\Re$, in the original terminal set. Both of these techniques seed the genospecies with numeric constants, providing a larger initial pool of numeric constants in the genospecies, making arithmetic combination more likely. Even so, GP still has difficulty generating sufficient numeric constants. In his first GP book (Koza 1992), John Koza uses GP on a problem consisting of discovering just a single numeric constant. Despite the use of the ephemeral random constant, the GP system still required 14 generations to create a solution, an s-expression comprising almost half a page. This is but one simple example, yet it illustrates that the creation of numeric constants remains a weak point of GP.

Numeric Mutation

Numeric mutation is a technique for facilitating the creation of useful, novel numeric constants during a GP run. Numeric mutation is a reproduction operation that, like mutation or cross-over, is applied to a portion of each population each generation. Numeric mutation replaces all of the numeric constants with new ones in the individuals to which it is applied. The new numeric constants are chosen at random from a uniform distribution within a specific se-

lection range. The selection range for each numeric constant consists of the old value of that constant plus or minus a *temperature factor*. The terminology derives from the similar concept of temperature in simulated annealing ((Kirkpatrick, Gelatt, & Vecchi 1983; Rumelhart & McClelland 1987) *et al*) in that when the temperature factor is larger, numeric mutation creates greater changes in the affected numeric constants.

The temperature factor is determined by multiplying the raw score of the best individual of the current generation by a user specified *temperature variance constant*, in this case[1], 0.02. The fitness score (raw or standardized, depending on the problem domain) of the best-of-generation individual approaches zero as it approaches a perfect solution to the problem domain. Consequently, the effect of this method for selecting the temperature factor is that when the best individual of a population is a relatively poor solution, the selection range is larger, and therefore there is an overall greater potential for change in the numeric constants of the individuals undergoing numeric mutation.

Over successive generations, the best-of-generation individual tends to improve and so the temperature factor becomes proportionally smaller. As the temperature factor decreases, numeric mutation causes successively smaller changes to the numeric constants. This should allow the GP process to "zero in on" (i.e., retain across generations with little change) those numeric constants that are useful in solving the given problem.

Experimental Evaluation

Our research with numeric mutation is at an early stage. Eventually, we plan to investigate the efficacy of numeric mutation in general. In this paper, however, we investigate the use of numeric mutation only in the problem domain of symbolic regression. Our experimental hypothesis was that numeric mutation increases the effectiveness of the GP process in solving symbolic regression problems. Our initial experiment involved the study of just one problem, defined by 11 pairs of numbers representing the x and y coordinates of 11 points (*target points*) on a plane. These *target* points correspond to the value of an objective function:

$$y = x^3 - 0.3x^2 - 0.4x - 0.6 \qquad (1)$$

(shown in Figure 1, along with the target points) at the 11 integer values of x from 0.0 to 10.0. This function objective function is considered the target or goal of the symbolic regression only indirectly. An infinite number of curves pass through these 11 points, and the goal is to discover *any* function that passes within a distance of plus or minus 0.1 along the y–axis for the x value of each of the eleven target points.

At the end of each generation, the numeric mutation technique is applied to 40 randomly selected individuals of the 200 with the best fitness from a population of

[1]We are still experimenting with methods for determining the value of the temperature variance constant.

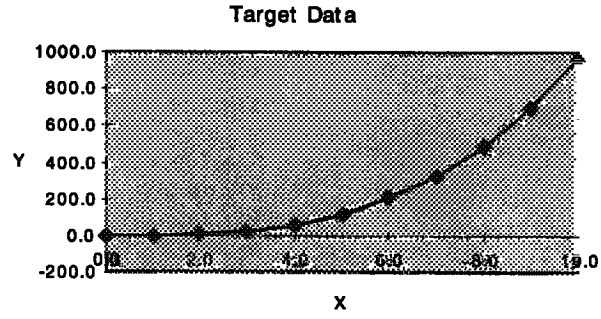

Figure 1: The target points for the symbolic regression, and their generating function.

1081. Each selected individual is replaced with a copy wherein each numeric constant has been mutated, as described in above. The fitness function is reevaluated for each of the new individuals, so that the fitness-ranking of the population corresponds to the altered population.

The choice of the number of elements to be numerically mutated, the size of the group that they are selected from, and the use of 0.02 as the temperature variance constant, were based on experiments involving other regression problems that suggested that these values tended to maximize the benefit of the numeric mutation (Fernandez 1997).

To test our hypothesis, we conducted 1000 runs of the GP system with numeric mutation and 1000 times without. Each generation of a numeric mutation run included the evaluation of the fitness function on $M = 1081$ individuals plus 40 additional individuals created by the numeric mutation process.

To compensate for the extra work done by the numeric mutation runs, the populations of runs not using numeric mutation contained 40 more individuals than those that did. To make comparisons between the results of the two techniques as equitable as possible, the populations of runs not using numeric mutation consisted of $M = 1121$ individuals. (Otherwise any performance advantage observed in the numeric mutation runs might be ascribed to the additional individuals evaluated therein.)

Each run was allowed to continue until a function was found that met the criterion described above, or until 50 generations were completed. Runs that discovered a function matching the target points within the 50 generation limit were considered successful. We ran our experiments on an AMD 166Mhz K6 running *Microsoft Windows 95*. We used the AGPS GP system (Fernandez & Evett 1997; Fernandez 1997), using the control parameters specified in the tableau shown in Table and an *elitist graduated overselection strategy* (Evett & Fernandez 1997) to select individuals from the population for reproduction and crossover.

Population size	1121 or 1081(NM)
% ramped complete growth	100
% ramped partial growth	0
Crossover Percentage	90
Mutation Pertcentage	0
Max Number of Runs	1000
Max Number of Generations	50
Max Nodes per Tree	200
Selection Strategy	Graduated Elitist
Initial Tree Minimum Depth	3
Initial Tree Maximum Depth	7
RandomSeed	0

Table 1: The GP tableau.

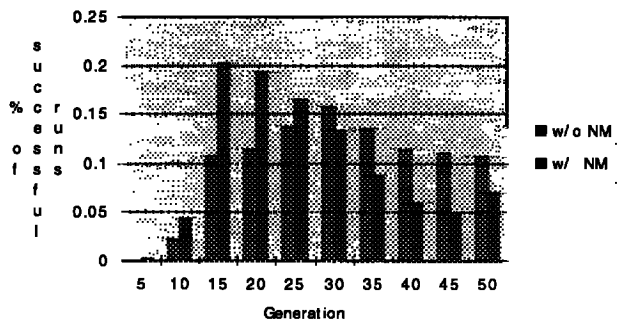

Figure 2: Percentage of successful runs that terminated at each generation for GP using and not using numeric mutation.

Results

Of the 1000 runs without numeric mutation, 328 were successful, while 541 of the runs with numeric mutation were successful. Thus, runs using numeric mutation were about 65% more likely to terminate successfully than the plain runs. The success ratio of the GP system was clearly higher when using numeric mutation. To determine whether this outcome was statistically significant, we performed a Large-Sample Statistical Test for Comparing Two Binomial Proportions (as described in (Mendenhall & Lyman 1972), page 203).

The null hypothesis for the significance test was that the populations have the same success ratios, and the alternate hypothesis was that they were not the same. This choice of the alternate hypothesis necessitated the use of a two-tail test. While the hypothesis could be phrased in a way to make a one-tailed test applicable, we have used the described hypothesis and the corresponding two-tailed test because it is more stringent(Fogel 1997).

The results of the test were that we rejected the null hypothesis with 95% confidence. Thus we conclude that numeric mutation's improvement to GP is statistically significant for this problem. A further indication of this is that not only does numeric mutation yield successful runs more frequently, but also the successful runs require, on average, fewer generations than the successful runs on the GP system without numeric mutation. The average number of generations in a successful run with numeric mutation was 24.44, while the average without numeric mutation was 29.67.

Figure 2 is a histogram of the percentage of successful runs that terminated each generation, with and without numeric mutation. For example, the figure shows that 20% of the numeric mutation runs finished successfully between generations 10 and 15. The shape of the curves formed by the two data sets in the figure clearly indicates that numeric mutation runs terminated successfully earlier than the non-numeric mutation runs.

This increase in efficiency was also reflected in runtime performance. The 1000 runs using numeric mutation required 8.28 hours to complete, while the 1000 runs without numeric mutation required 11.63 hours. The average time to complete a successful run with numeric mutation was 16 seconds while the average time without numeric mutation was 24 seconds.

Interpreting the Results

We have demonstrated that numeric mutation provides an improvement to the GP algorithm as it is applied to this symbolic regression problem. The next step is to understand from whence this benefit derives.

It is apparent that the numeric mutation technique provides a much greater diversity of numeric constants to the GP. Plain GP starts with a fixed number of numeric constant leaf nodes in the genospecies. Whenever the selection process causes all copies of a numeric constant to be removed from the population, that numeric constant is effectively lost for the remainder of the run. Thus, with each generation the number of unique numeric constant leaf nodes can never increase and, indeed, typically decreases monotonically. In contrast, with numeric mutation, the GP process gains many new numeric constants each generation.

We conducted experiments to determine if the steady influx of new numeric constants, alone, accounted for the benefit of the numeric mutation technique. We completed 1000 GP runs in which 40 elements were selected after each generation in the same way as in numeric mutation, and all of their numeric constant leaf nodes replaced with new numeric constants. These new constants were selected randomly from the interval $(-1000.0, 1000.0)$ using a uniform distribution. We call this process *numeric replacement*. Numeric replacement is similar to the technique referred to as *small-mutation* in (Harris & Smith 1997) except that numeric replacement concerns only numeric constant leaf nodes, while small-mutation can affect any type of node.

The result of the numeric replacement experiment was that only 278 of the runs were successful by the 50th generation as compared to 328 successful runs with

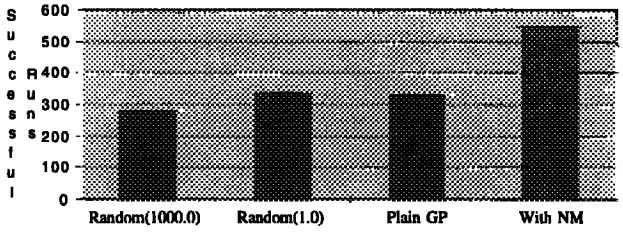

Figure 3: Number of successful runs (of 1000), handling numeric constants four different ways.

plain GP. To determine if this difference was statistically significant we again used the Large-Sample Statistical Test for Comparing Two Binomial Proportions described above. We determined, with 95% confidence, that numeric replacement produces a statistically significant *degradation* of performance when compared to plain GP. Therefore it is highly probable that the benefit derived from numeric mutation does not derive solely from the influx of new numeric constants, but also from the values of those constants.

To ensure that the range from which the the new constants were selected was not biasing our results, we conducted similar experiments, but used a smaller range, $(-1.0, 1.0)$. This range corresponded better with the coeffecients of the generating function (see Equation 1. All had an absolute value no greater than 1.0).

Of the resulting 1000 runs, 336 were successful by the 50th generation. This, at least, was more than the 328 successful runs that occurred with the plain GP, but the Large-Sample Statistical Test for Comparing Two Binomial Proportions determined that this improvement is not statistically significant at the 95% confidence level. We again conclude that the benefit of numeric mutation does not derive solely from the influx of a large number of new numeric constants. A summary of all these results is shown in Figure 3. The entries in the figure labelled "Random" correspond to numeric replacement using the two different ranges.

Having eliminated other possible explanations, we speculate that the benefit of numeric mutation derives not simply from the introduction of new numeric constants into the genospecies, but also from these new constants being introduced only into s-expressions at locations in genomes where arithmetically similar numeric constants have already demonstrated some measure of success, insofar as they appear in individuals in the top 18.5% of the population (the top 200 out of 1081 as scored by the fitness function). We further speculate that the choice of new numeric constants is further enhanced by making them increasingly more similar to the existing "successful" constants as the population comes closer to finding an acceptable solution. (As reflected by the raw score of the best individual in each generation.)

Conclusion and Future Work

We conclude that the use of numeric mutation should be considered for any GP problem in which numeric constants are used as terminal nodes. Numeric mutation is easy to implement and does not add significant additional overhead to the GP algorithm.

Several additional experiments are suggested by this work. We are already in the process of collecting data for other symbolic regression problems to determine if numeric mutation is generally useful for that problem domain. We also plan to measure the value of numeric mutation in other problem domains so as to be able to characterize those domains where numeric mutation is especially beneficial. We plan to see if additional benefit can be derived by applying numeric mutation only to a portion of the numeric constants in selected individuals, and to experiment with alternative methods for determining the temperature factor, such as using the raw score of the individual to be mutated rather than the raw score of the best element in the generation.

References

Evett, M., and Fernandez, T. 1997. A distributed system for genetic programming that dynamically allocates processors. Technical Report TR-CSE-97-39, Dept. Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL.

Fernandez, T., and Evett, M. 1997. The impact of training period size on the evolution of financial trading systems. Technical Report TR-CSE-97-41, Florida Atlantic University, Boca Raton, FL.

Fernandez, T. 1997. The evolution of numeric constants in genetic programming. Master's thesis, Florida Atlantic University, Boca Raton, FL. In preparation.

Fogel, D. 1997. The burden of proof. Invited lecture at Genetic Programming 1997, Palo Alto, CA.

Harris, K., and Smith, P. 1997. Exploring alternative operators and search strategies in genetic programming. In Koza, J.; Deb, K.; Dorigo, M.; Fogel, D.; Garzon, M.; Iba, H.; and Riolo, R., eds., *GP-97, Proceedings of the Second Annual Conference*, 147–155. San Francisco, CA, USA: Morgan Kaufmann.

Kirkpatrick, S.; Gelatt, C.; and Vecchi, M. 1983. Optimization by simulated annealing. *Science* 220:671–680.

Koza, J. 1992. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press.

Koza, J. 1997. Tutorial on advanced genetic programming, at genetic programming 1997.

Mendenhall, W., and Lyman, O. 1972. *Understanding Statistics.* Belmont, CA: Duxbury Press.

Rumelhart, D., and McClelland, J. 1987. *Parallel Distributed Processing*, volume 1. Cambridge, MA: MIT Press.