Numeric Mutation as an Improvement to Symbolic Regression in Genetic Programming

Thomas Fernandez and Matthew Evett

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

Abstract. A weakness of genetic programming (GP) is the difficulty it suffers in discovering useful numeric constants for the terminal nodes of the s-expression trees. We examine a solution to this problem, called *numeric mutation*, based, roughly, on simulated annealing. We provide empirical evidence to demonstrate that this method provides a statistically significant improvement in GP system performance for symbolic regression problems. GP runs are more likely to find a solution, and successful runs use fewer generations.

1 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. This problem is interesting because genetic algorithms, from which GP is derived, are highly suited to the task of optimizing numeric parameters. 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.[8]

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 these s-expression are all numeric constants, and so the s-expression necessarily evaluates to a numeric value. The entire s-expression can thus be viewed as a single numeric constant terminal node, with a value equal to that of the s-expression. 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. Having the constants 1.0 and 2.0, the constant of 0.5 can evolve via an s-expression that divides 1.0 by 2.0, 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 a symbolic regression problem.

2 History

Some of the early enhancements to the GP process facilitated the creation of constants. These enhancements [7] consisted of including a small number of numeric constants and/or the *ephemeral random constant*, $\Re$, in the original terminal set. (Each time the ephemeral random constant is selected as a terminal in the creation of the population of generation 0, it is replaced with a randomly generated number within some specified range.)

Both of these techniques seed the genospecies with numeric constants. The ephemeral random constant is particularly helpful because it provides many different numeric constants in generation 0. Even so, most problems require a solution that uses numeric constants other than those provided in the original terminal set or generated by the ephemeral random constant. Such constants must be evolved tediously by arithmetic combination or arithmetic genesis (though the larger initial pool of numeric constants in the genospecies does make arithmetic combination more likely.)

Even with the use of the ephemeral random constant and/or the presence of predefined constants in the terminal set GP still has difficulty generating sufficient numeric constants. In [7], GP is used 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.

3 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 which, 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 selection range. The selection range for each numeric constant is specified as the old value of that constant plus or minus a *temperature factor*. The terminology derives from

the similar concept of temperature in simulated annealing ([6, 10] and others) 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.

Note that the form of mutation expressed here differs from that of most other evolutionary algorithms in that the mutation is not localized in the genotype, i.e., the mutation does not affect just a single GP subtree, or a single bit in a GA genome. Those techniques yield results that are, in a sense, "close" to the original. For example, GA mutation produces strings that are at a Hamming distance of 1 (sometimes a bit more) from the original.

In a sense, numeric mutation also yields mutants that are close to the original, but this closeness is measured in terms of the n-dimensional space of all individuals having the same GP-tree structure containing n numeric constants. Numeric mutation yields an individual that is only a short distance (limited by the temperature factor) from the original, in this space.

4 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. Note that all of these points lie along the curve defined by the generating function:

$$y = x^3 - 0.3x^2 - 0.4x - 0.6 \tag{1}$$

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

This function (shown in Fig. 1, along with the target points) 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 11 target points.

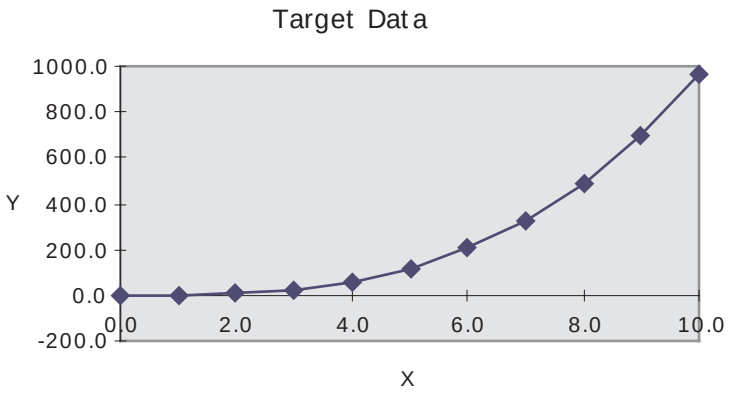

Fig. 1. The 11 target points for the symbolic regression, and generating function.

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 1081. Each selected individual is replaced with a copy wherein each numeric constant has been mutated, as described in Sect. 3. 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 [2].

To test our hypothesis, we ran the GP system 1000 times with and without numeric mutation. Each generation of a numeric mutation run included the evaluation of the fitness function on 40 additional individuals (those 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. This makes comparisons between the results of the two techniques more fair, as both algorithms are then doing roughly the same amount of work. (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 our own hand-coded GP system (described in [3, 2]), using the control parameters specified in the tableau shown in Table 4 and an *elitist graduated overselection strategy* to select individuals from the population for reproduction and crossover, as described in [1].

Population size	1121 (plain), 1081 (NM)
% of individuals created by ramped complete growth	100
% of individuals created by 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 tableau used to test effectiveness of numeric mutation.

4.1 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 (LSTBP) (as described in [9], 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. It might be argued that numeric mutation is a modification to the GP technique involving additional work, and consequently we should be interested only if it provides an improvement to GP. An alternate hypothesis that reflects that argument is that GP with numeric mutation has a higher success ratio than GP without. Such a choice would permit the use a one-tailed test. We have used

the first hypothesis and the corresponding two-tailed test because it is more stringent[4].

The results of the test were that we rejected the null hypothesis at the 0.05 level of significance. 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 number of generations required to complete the successful runs with and without numeric mutation. Each column of the graph corresponds to a sum across five generations. For example, the figure shows that of the 1000 runs using numeric mutation, 103 finished successfully between generations 21 and 25, inclusive.

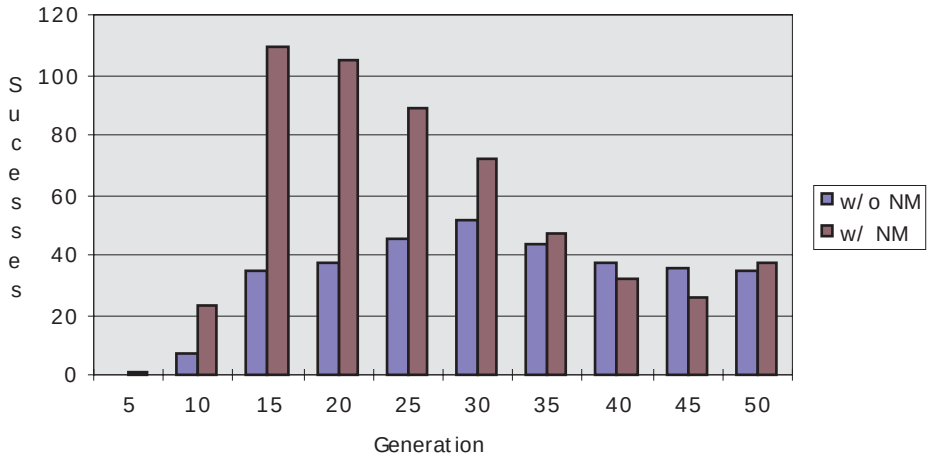

Fig. 2. Number of runs that terminated successfully at each generation for GP using and not using numeric mutation.

Figure 3 is a similar histogram, showing the same information but as 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 run-time performance. The 1000 runs using numeric mutation required 8.28 hours to complete, while the

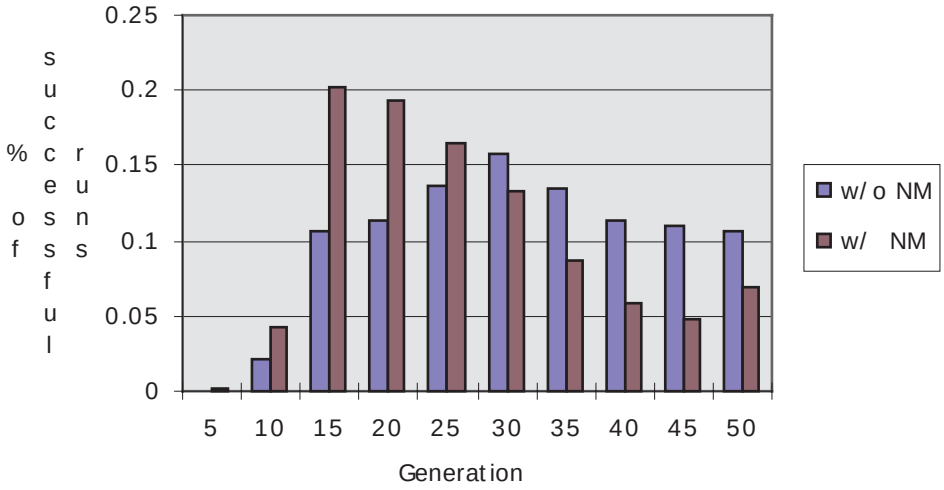

Fig. 3. Percentage of successful runs that terminated at each generation for GP using and not using numeric mutation.

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. (The exact value of the timings is not important, but their relative values are. The timing information under Windows 95 is somewhat imprecise, but serves to support the conclusions derived from the other experimental results.)

4.2 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 (without numeric mutation) starts (in generation 0) with a fixed number of numeric constant leaf nodes in the entire population (i.e., 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. Numeric mutation replaces all of the numeric leaf nodes with new numeric constants in all of the elements to which it is applied. Thus the GP process gains many new numeric constants each generation by using numeric mutation.

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 [5] 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 plain GP. To determine if this difference was statistically significant we again used the LSTBP statistical test described above. We determined, at the 0.05 level of significance, 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.

A potential criticism of this experiment is that the range (e.g. $(-1000, 1000)$) from which the new constants were chosen in numeric replacement did not correlate well with the problem domain. The function used to generate the regression's target points, (1), contains no numeric coefficients with an absolute value greater than 1.0. So the range $(-1000, 1000)$ may be introducing numeric constants into the genospecies that are unlikely to be useful in solving the specific symbolic regression problem under study.

To investigate this possibility we again conducted the numeric replacement experiment, but used the range $(-1.0, 1.0)$ from which to select the new constants. Of the 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 LSTBP statistical test 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 Fig. 4. The entries in the figure labelled "Random" correspond to numeric replacement.

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

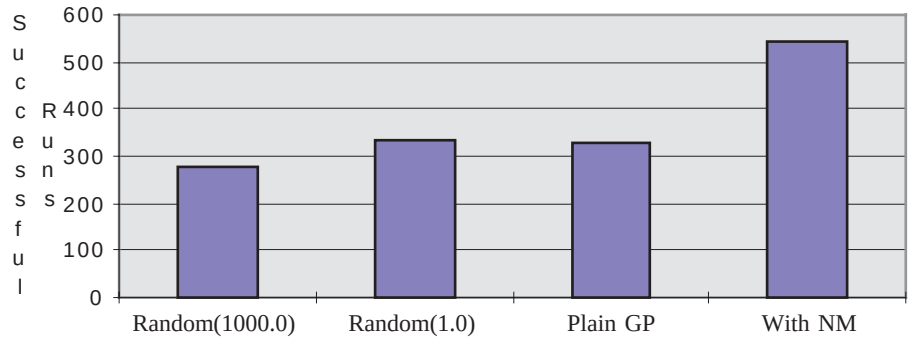

Fig. 4. Number of successful runs (of 1000), handling numeric constants four different ways.

the best individual in each generation.)

5 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.

Acknowledgements

We thank the reviewers for identifying several weaknesses in the original draft of this paper.

References

1. M. Evett and T. Fernandez. 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, 1997.
2. T. Fernandez. The evolution of numeric constants in genetic programming. Master's thesis, Florida Atlantic University, Boca Raton, FL, 1997. In preparation.
3. T. Fernandez and M. Evett. 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, 1997.
4. D. Fogel. The burden of proof. Invited lecture at Genetic Programming 1997, Palo Alto, CA, July 1997.
5. K. Harris and P. Smith. Exploring alternative operators and search stratagies in genetic programming. In J. Koza, editor, *GP-97, Proceedings of the Second Annual Conference*, pages 147–155. Morgan Kaufmann, 1997.
6. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
7. J. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.
8. J. Koza. Tutorial on advanced genetic programming, at genetic programming 1997, Palo Alto, CA, July, 1997.
9. W. Mendenhall and O. Lyman. *Understanding Statistics*. Duxbury Press, Belmont, CA, 1972.
10. D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA, 1987.

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