CATEGORY: Genetic Programming

Fault-Tolerant Computing with N-Version Genetic Programming

KOSUKE IMAMURA   JAMES A. FOSTER

Initiative for Bioinformatics and Evolutionary STudies (IBEST)

Computer Science Department

University of Idaho

Moscow, ID 83844-1010

{kosuke, foster}@cs.uidaho.edu

(208) 885-7062

Submitted to Genetic and Evolutionary Computing Conference (GECCO 2001)

Fault-Tolerant Computing with N-version Genetic Programming

KOSUKE IMAMURA JAMES A. FOSTER

Initiative for Bioinformatics and Evolutionary STudies (IBEST)
Computer Science Department
University of Idaho
Moscow, ID 83844-1010
{kosuke, foster}@cs.uidaho.edu
(208) 885-7062

Abstract

Software reliability is an increasingly important issue today. Yet, reliability of genetic programming has not been studied fully. A genetic program to be deployed is often the one which performs the best on sample tests. One of the techniques to improve reliability is N-version programming. Our question is whether N-version genetic programming (NVGP) improves reliability over a single version. We applied NVGP to a path prediction problem, and compared the performance with a single version. Statistics from the experiment suggests that NVGP is a viable method to increase reliability.

1. INTRODUCTION

Hecht cites Dijkstra, “program testing can be used to show the presence of bugs but never to show their absence” [Hecht and Hecht]. Exhaustive testing may prove the correctness of programs built with sample training based methods such as genetic programming and artificial neural networks, but this is a nearly impossible task in real applications because of the vast input space. Moreover, without exhaustive testing, inferred programs are likely to be incorrect [Imamura et al.].

Therefore, it is necessary to provide a means to detect and recover from faults in order to sustain continuing operation of a system. A fault is an undesirable behavior in a system, such as incorrect output from a program module or a dropped bit in a communication line. A failure is when a system behaves incorrectly, for example by returning an incorrect value to the user or by forwarding an erroneous packet. Fault tolerant systems detect and correct faults before they become failures. Fault tolerant software by N-version programming is one of the techniques to cope with software faults. N-version programming (NVP) is defined as the independent generation of $N \geq 2$ functionally equivalent programs (versions) from the same initial specifications [Avizienis and Kelly]. Its goal is to adapt the technique used in hardware redundancy with majority voting to build programs that tolerate software design faults [Hilford et al.].

A fundamental assumption of the NVP is that independent programming effort will reduce the probability that similar errors will occur in two or more versions [Avizienis and Kelly]. Independent programming is believed to promote diversity in algorithms, data structures, programming languages, platforms, and error recoveries. Different approaches to attaining program design diversity are discussed in [Hillford et al.], including NVP.

In genetic programming (and evolutionary techniques in general) there are no design specifications or path testing. Consequently, the conventional software testing or design diversity techniques cannot be applied to genetic programming. With lack of testing and verification methods, ensuring overall reliability of genetic programming is a difficult task. To cope with software faults under such circumstances, we applied NVP to genetic programming. Statistics from our experiment suggests that N-version genetic programming may be a viable option for increased reliability of genetic programming.

2. BACK GROUND AND PREVIOUS WORKS

A fundamental NVP assumption is that faults are independent among the N versions. Independent faults can be masked by votes. For simplicity, if we assume the probability of fault of each version is uniformly *p* and that faults are independent, then the probability of the system failure, *f*, is:

$f = \sum_{k=m}^{n} \binom{n}{k} (1-p)^{n-k} (p)^{k}$ where *n* is the total number of versions, *m* is the minimum number of faulty votes to cause system wide failure.

Typical N-version software is constructed by isolating N individuals or N groups of programmers. Difficulties of attaining diversity among the versions, which is prerequisite for an independent fault, by human effort may be attributed to the fact that we have similar education and training backgrounds, and therefore we tend to use the similar programming techniques and algorithms to solve common problems.

Knight and Leveson applied a probabilistic metric to measure the assumed independence of NVP [Knight and Leveson]. Interestingly, the theoretical system fault rate (*f* in the above case) was not observed in their experiment and they rejected the hypothesis of the assumed independence of fault by independently developed programs. However, the rejection of the independent fault hypothesis does not mean we should not try N-version. Hatton followed up the Knight and Leveson's study, and his finding was that N-version is more reliable and cost effective than one good version [Hatton], e en with non-independent faults. He also observed non-independent faults, and his 3-version system (with fault probability is 0.004 each) increased the reliability 45 times, while the complete independent faults would have increased it 833.6 times. This result suggests that N-version can substantially increase the reliability even with non-independent faults.

Another important aspect of NVP is ability to detect faults. Even with mildly non-independent faults, if a large majority of modules agree in their output, the minority outputs are likely to be incorrect. This probability increases with the number of voters. Naturally, if the voting is highly dependent, then there is chance that a majority of voters will agree to produce erroneous output.

N-version genetic programming (NVGP) is identical to a conventional NVP in concept, but without design/implementation issues of man-effort. NVGP automates the creation of redundant modules by using speciation with genetic programming.

A similar study to our NVGP is found in Hasem's Linearly Optimal Combination of ANN. An artificial neural network is another example of a black box approach. Hashem reports superior performance of a linearly combined neural network (LOCANN) [Hashem 1995 and Hashem 1997]. In his approach, redundant modules (trained ANNs) are assigned optimal weights for computing a weighted average of the module outputs. NVGP is dissimilar to LOCANN in that: NVGP allows retraining of faulty GPs, while LOCANN NN modules are fixed after initial training; and NGVP's weights are all 1 because we cannot assume that some GP is better than another, while LOCANN estimates the best weight vector from the given sample set.

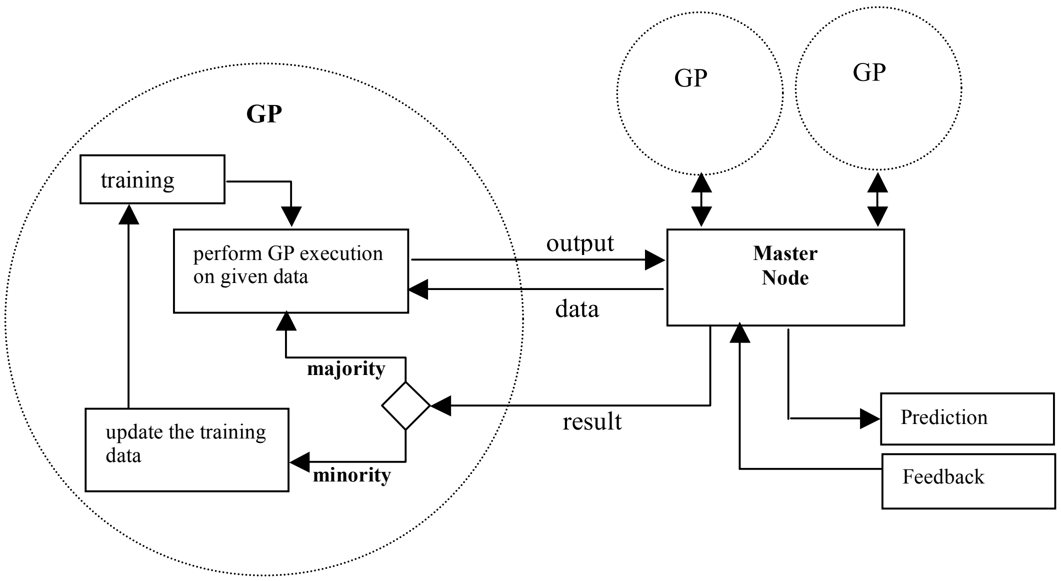
Since isolating groups of programmers is very much like an isolated island model, and since it is the only method known to us for GP speciation, we developed an N-version genetic programming using the isolated island model. In the next section, we describe our system.

3. DESCRIPTION OF NVGP

A server-client model implements the system. The system consists of one master and multiple GPs. GPs receive training data from the master. Our test application is a time series prediction problem, described below. After training is completed, GPs from slave processes submit their predictions based on given data. The master averages all the GP outputs to determine the final prediction value. The master receives feedback on the prediction value, and then based on the feedback, it sends the "majority" message to slave processes whose vote values were close to the desired value and the "minority" message to the rest. If majority, then the GP uses the same program to predict the next. If minority, then the GP starts retraining with the most recent data. Retraining is done with a new population. Preliminary experiment showed difficulty in convergence otherwise. Voting is synchronized by a barrier synchronization method; no GP can cast the next prediction value until all the GPs receive the majority/minority results. The system is implemented on a Unix network, and we are currently moving it to a Beowulf cluster. GPs are built using lilgp1.1 library package [Punch and Zongker].

A fault in this system is a large deviation by a slave GP from the system output—which is the average of all slave process outputs. A failure is a large deviation of the system output from the correct target value. "Large" is a system parameter. **Figure 1** illustrates the system.

Figure 1. SRGP System Diagram



4. EXPERIMENT

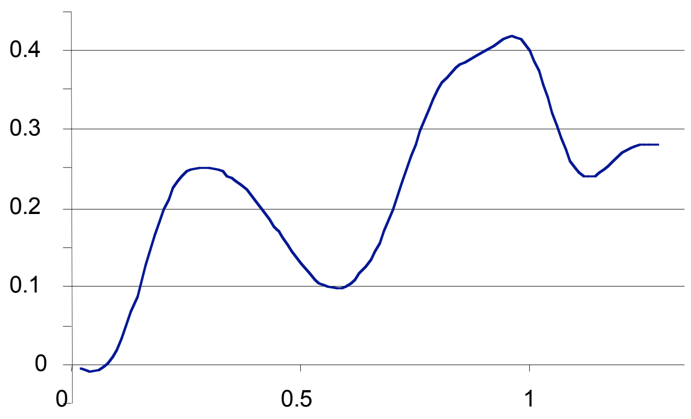
The problem is to predict the next location of a moving object on a constantly changing path. The path was simulated by a piecewise cubic spline, which is differentiable everywhere. The data points for the spline were arbitrarily placed by hand using a spline package [Rauch]. The path is shown in the **Figure 2**. This makes it difficult for a GP to retain the same approximation program over a long range, since a cubic spline is a series of connected different cubic polynomial segments. The prediction is made based on the most recent 5 points on a two dimensional space. Our first hypothesis is that averaging voted values from multiple GPs tracks the path with a smaller standard deviation of prediction errors. Our second hypothesis is that the means of prediction errors are the same between NVGP and single version GP. If these hypotheses hold true, then NVGP has some degree of independent faults.

There are 5 GPs participating in the voting. After receiving the initial 5 points, (P_0, ..., P_4) from the master, each GP predicts P_5. If the prediction is within the retraining-threshold, it retains the same equation to predict P_6. Otherwise, it uses (P_1, ..., P_5) to retrain itself. This process continues, using the most recent 5 data points, until P_{60} has been predicted.

Each vote participating (slave) GP must be retrained when it casts a minority vote, which is when its prediction and feedback difference exceeds a retraining threshold of 0.02. Essentially, a module whose prediction exceeds the retraining threshold (0.02) is considered faulty, and retraining is a fault recovery process. The retraining-threshold value was set in the preliminary experiment as a compromise between performance and retraining time. The GP completes its retraining when it generates an equation that contains all the 5 recent points within a 0.015 tolerance. Without the recovery process, the overall system may fail. System prediction fails when the difference between prediction and feedback values exceeds the performance guarantee error-threshold. We considered three levels of the performance guarantee error-threshold, 0.02, 0.03, 0.04, and 0.05.

The experiment was repeated 28 times. Also, we ran 28 times a single version GP for performance comparison. The single version is NVGP with only one voter. We performed the F-test on the standard deviations of errors, each standard deviation of the 28 runs of NVGP is compared with each of the 28 runs of a single GP, totaling 784 (28×28) comparisons. Four hundred eighty two comparisons out of the 784 comparisons (61.5%) indicated that the NVGP's standard deviations of errors are significantly smaller than that of the single GP at a significant level 0.10. It should be noted that every standard deviation of the NVGP are smaller than all the standard deviations of a single GP. We also performed two-sample t-test (Satterthwaite's Method) at a significant level 0.10; one sample is the 28 runs of NVGP and the other sample is the 28 runs of the single GP. The results are shown in the **Table 1**. The mean of NVGP's standard deviation of errors is significantly smaller than the single GP's. This is consistent with the F-test. On the other hand, the means of errors of NVGP and the single GP are not significantly different. This statistic is also consistent with individual 784 comparisons of the means. Only 1.3% of the comparisons indicated that the means of errors are significantly different at a significant level 0.10.

Figure 2. Path simulated by a cubic spline



Summarizing these statistical data, we accept our hypothesis that the NVGP predicts the path with the narrower error margin than a single GP system on overall bases. We also accept our second hypothesis that the NVGP and the single GP produce statistically the same mean of errors. This suggests that there is a certain degree of independent faults in the NVGP. The mean time between failures (MTBF) of NVGP is significantly smaller than that of the single GP at all 4 levels of the performance guarantee error-threshold. For visual confirmation, **the Figure 3** plots all the prediction errors. Notice the NVGP plot band of is narrower than the single GP band.

Figure 3. Prediction Error Plot. Y axis is difference between predicted value and actual value in target series in Figure 2. X axis is same as for target series. Each line is a different run (28 total).

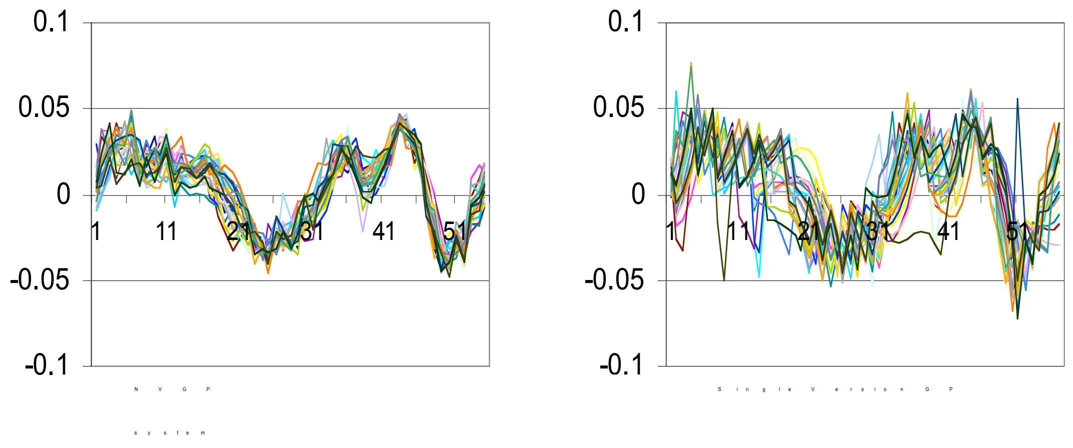


Table 1. Statistics Summary

$f = s_1^2 / s_2^2$

$\lambda = (r_1 - r_2) / \sqrt{s_1^2/n_1 + s_2^2/n_2}$

$df \approx (s_1^2/n_1 + s_2^2/n_2)^2 / ((s_1^2/n_1)^2(n_1-1) + (s_2^2/n_2)^2(n_2-1))$

where s^2: variance, n: number of samples, r: mean, subscript: sample 1 and 2.

Average of average error=$\sum x_i/28$

Std of average error=stdev(x_i)

Average of std of average error=$\sum$stddev(x_i)/28

Std of std of average error =stddev(stddev(x_i))

where x_i is the average error of *i*th run of 28 runs.

MTBF=1/frequency of failures

	NVGP	Single GP	
Average of average error	0.00411	0.00409	f=1.933 λ=0.0360 df≈100
Std of average error	0.00195	0.00271	$\lambda < t_{60} = 1.296 < t_{56}$ Average error is the same.
Average of std of average error	0.02199	0.02671	f=1.435 λ=21.672 df≈107
Std of std of average error	0.00104	0.00125	$\lambda > t_{40} = 1.303 > t_{56}$ Std of error is different.
Error threshold=0.02			
Average failure frequency	25.07	30.80	f=1.879 λ=10.529 df≈101
Std of failure frequency	2.40	3.29	$\lambda > t_{40} = 1.303 > t_{56}$ 22.9% increased reliability.
MTBF	0.0399	0.0325	
Error threshold=0.03			
Average failure frequency	10.64	17.36	f=0.577 λ=15.764 df≈102
Std of failure frequency	2.54	1.93	$\lambda > t_{40} = 1.303 > t_{56}$ 163.2% increased reliability.
MTBF	0.0940	0.0576	
Error threshold=0.04			
Average failure frequency	2.28	7.57	f=3.104 λ=18.645 df≈87
Std of failure frequency	1.04	1.85	$\lambda > t_{40} = 1.303 > t_{56}$ 332.0% increased reliability.
MTBF	0.4386	0.1321	
Error threshold=0.05			
Average failure frequency	0	1.75	no failures with NVGP
Std of failure frequency	0	1.24	
MTBF	0	0.5714	

4. CONCLUSION AND FUTURE RESEARCH

Path prediction of a moving object is a difficult problem. Our 5-version GP implemented by an isolated island model alone increased the system reliability compared with a single version GP. Whether or not this increase is important depends on how critical an application is. For safety critical applications, such as aircraft control modules or nuclear power plant cooling system regulators, any improvement is significant. Reliability increases when faults are independent. Currently, we are exploring GP speciation, which will use a metric to measure the level of similarities between GPs. With this metric, we could discourage similar GPs from participating in voting, resulting in increased ability of fault masking. Another area of our research is to maintain an optimal voter pool in order to respond in real-time. A GP that needs to be retrained will be removed from the pool, and a GP that completed retraining will be brought into the pool.

We will also investigate how well statistical predictions of system reliability agree with our observations. One can compute the expected difference between the system output, which is the average of individual GP outputs, and the expected divergence of those individual outputs. This models the retraining rate of individuals. One can also compute the expected divergence of the system output, which is an approximation to the mean of expected individual outputs, from the actual mean. This is a crude approximation of mean time to failure, assuming that the target function and the mean are the same. Finally, we can compute the expected failure rate with the assumption that the system output is a random variable clustered around the actual target function—which expresses mean time to failure in terms of both the accuracy of the NVGP and its reliability.

Acknowledgement:

This research was funded in part by NSF EPS0080935 and NIH F33 GM20122-01. The authors are grateful to Dr. Axel Krings and Dr. Robert Heckendorn of University of Idaho for helpful discussions.

Reference:

Avizienis, A. and J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments", IEEE Computer, vol. 17 no. 8, August 1984, 67-80.

Sherif Hashem, "Optimal Linear Combinations of Neural Networks"
Neural_Networks, Vol. 10, No. 4, 1997, pp. 599-614.
(http://www.emsl.pnl.gov:2080/proj/neuron//papers/hashem.nn97.abs.html)

Sherif Hashem, "Improving Model Accuracy Using Optimal Linear Combinations of Trained Neural Networks, IEEE Transactions on Neural Networks, v.6(3), 1995, pp. 792-794
(http://www.emsl.pnl.gov:2080/proj/neuron//papers/hashem.tonn95.abs.html)

Leslie Hatton, "N-version vs. one good program", IEEE Software , Volume: 14 Issue: 6 , Nov.-Dec. 1997 Page(s): 71 -76

Herbert Hecht and Myron Hecht, "Fault-Tolerance in Software", in Dhiraj K. Pradhan, "Fault-Tolerant Computer System Design", Chapter 7, pp. 428-476, Prentice Hall PTR, 1996.

V. Hilford, M.R. Lyu, B. Cukic, A. Jamoussi, and F.B. Bastani, "Diversity in the Software Development Process", in Proceedings of WORDS'97, Newport Beach, California, February 1997.
(http://www.cse.cuhk.edu.hk/~lyu/papers.html#SFT_Techniques)

Kosuke Imamura, James A. Foster and Axel Krings. The test vector problem and limitations to evolving digital circuits. *Proc. NASA/DoD Workshop on Evolvable Hardware (EH),* IEEE Press, pp. 75—80, 2000

John C. Knight and Nancy B. Leveson, "An Experimental Evaluation of the Assumprion of Independence in Multiversion Programming", IEEE Transaction on Software Engineering, VOL SE-12, NO. 1, January 1986

Bill Punch and Douglas Zongker, http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html

Scott A. Rauch, XlXtrFun, http://www.netrax.net/~jdavita/XlXtrFun/XlXtrFun.htm