Random Number Generation using Genetic Programming Report

Philip Leonard

March 29, 2014

Abstract

This is the final report for the project; 'Random Number Generation using Genetic Programming' by Philip Leonard, supervised by Dr David Jackson (primary supervisor) and Professor Paul Dunne (secondary supervisor). This document covers all aspects of the project from the introduction and background, to the design implementation and analysis of the project in order to give the reader an understanding of how I implemented the project and what things I found when evaluating the project.

This report demonstrates that using methods described by Koza in [1], it is possible to genetically breed Random Number Generators that produce sequences of pseudo random bits with near maximal entropy. This report more importantly shows that it is possible to produce better results using the Single Node Genetic Programming methodology described by Jackson in [2]. As well as this, this report also demonstrates that the Random Number Generators Produced genetically by these methods also produce sequences of random bits with higher entropy than other widely used RNGs. It not only outperforms the C programming Linear Congruent Generator algorithm implemented in the *rand*() function, but can also perform better than a "True" Random Number Generator that creates random bit sequences from observed atmospheric noise.

Keywords: Genetic Algorithm (GA), Genetic Programming/Program (GP), Pseudo Random Number Generator (PRNG), Random Number Generator (RNG), True Random Number Generator (TRNG), Entropy, Single Node Genetic Programming/Program (SNGP), Crossover, Mutation, Fitness proportionate reproduction (FPR), Linear Congruent Generator (LCG).

Contents

1 Introduction

Random Number Generators have many computational, scientific and societal applications. They are essential in; gambling, lottery draws, cryptography, statistical sampling, Monte Carlo simulations and other applicaitons where and unpredicatble result is desired. Randomness can be interpreted in more than one way and has no strict mathematical definition. There are a series of tests that can be run on a set of data in order to evaluate some conception of randomness. These include the gap test, frequency test, runs test and information entropy test, and whilst there isn't a way to prove randomness, these tests are a good way of analysing randomness in the eyes of the user and for their practical application. In this project, randomness is tested using the shannon entropy equation [1, p.2], which is a measure of the unpredictability/uncertainty in a random variable.

The Genetic Programming paradigm is an evolutionary method of solving problems, where the implementer can define a task where solutions can be randomly generated and tested by some fitness function. The population can then be evolved based on their fitness in order to search through the problem space to find solutions which meet the fitness requirements of the implementer.

Evolving Random Number Generators is therefore an applicable problem instance for Genetic Programming. The aim of the project is to assess two Genetic Programming methodologies in order to determine which is best suited for genetically breeding RNGs. The method described by Koza in [**?**] was followed in order to implement the idea of evolving a random number generator in the widely known traditional Genetic Programming methodology. The next step was to implement the same idea using the Single Node Genetic Programming Methodology described by Jackson in [2]. The effecitvness of both where then compared by gathering data from both implementations and conducting some cross examinaton to determine the victor. In order to determine the effectiveness of the Random Number Generators as a real world solution, entropy data was gathered from pre existing and widely known and used RNGs. These were namely the C rand function, which is a Pseudo Random Number Generator which is makes use of a Linear Congruent Generator algorithm in order to produce random numbers, and a "True" Random Number Generator was also evaluated in order to compare the effectivness of the genetically produced RNGs against a non-deterministic source of random data.

What are the challenges of the project?

In terms of software three programs have been produced. The GP implementation and the SNGP implementation form two seperate single threaded command line programs. The command line interface displays information and run progress to the user, and all data for each generation and each run, including entropy values and is written to log files. The third program developed is to test both the C random number generator and the TRNG and to write the entropy data to log files as well.

From these implementations I was then able to start to collect data. In order to gather suffiecient data for analysis, I ran both the SNGP and GP implementations 50 times and also ran 50 tests of the C rand funtion and the TRNG.

The results of project are very promising. To begin with, SNGP proves a much faster way of evolving RNGs, finding solutions on average over 6 times quicker than the standard GP methodology. Not only is it quicker, but it also has a solution rate twice that of the Genetic Programming Methodology. On average the sizes and fitness of solutions were similar for both approaches. The solutions produced by the two genetic methodologies also outperformed the C rand funciton and even the "True" Random Number Generator.

2 Background

- Introduce Koza's paper to explain the GP implementation.
- Introduce Jacksons paper.
- Talk comparitively about the GP implementation to Koza's.
- Koza's and Jackson's papers used mainly for research into the implementations. Talk about using the GP and GA books for prior reading. Talk about adding the TRNG adn C Rand analysis as extra tests.
- Project Requirements from Spec sheet.

3 Design

Design Document

4 Implementation / Realisation

- GP implementation, go through algroithms to code.
- SNGP implementation, go through algorithms to code.
- C rand() and TRNG test implementation. - Describe changes; KMP matching algorithm, SNGP update list order realisation, added run data collection, integer to long for calculations, change of driving evolution in SNGP, terminating on smut to attempted smut. - Problems encountered, SNGP producing incorrect e ntropy values for some trees,

5 Evaluation

- Introduce how evaluation is based on collecting data from running all implemenations and then comparing the results. Any part where running time factors into the analysis then AMD Athlon X2 5600+ 2.8GHz Dual core processor used. Explain changes to the test criteria from design document.
- Introduce evaluation by comparing first GP resulst to that of Kozas results. Compare any criteria that Koza proposes in his paper. Then show results and findings.
- Then introduce comparisons between GP and SNGP implementations under standard conditions for the 50 runs. Explain what will determine which method is most effective, i.e. minimal solution size, run time, highest fitness/entropy, highest solution rate. Then show results and findings and conclude to determine the victor.
- Introduce experimentation with the run conditions of both GP and SNGP. Then show results and findings.
- Introduce comparisons of Genetic vs Converntional means of producing RNG's and random numbers.

5.1 Temporary Graphs

Figure 1: Reduced Pop size of SNGP

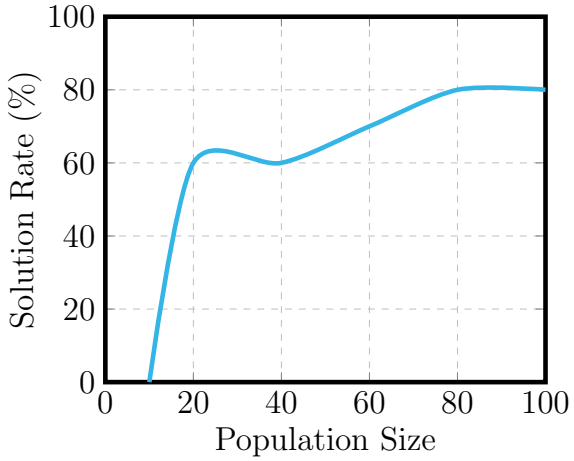
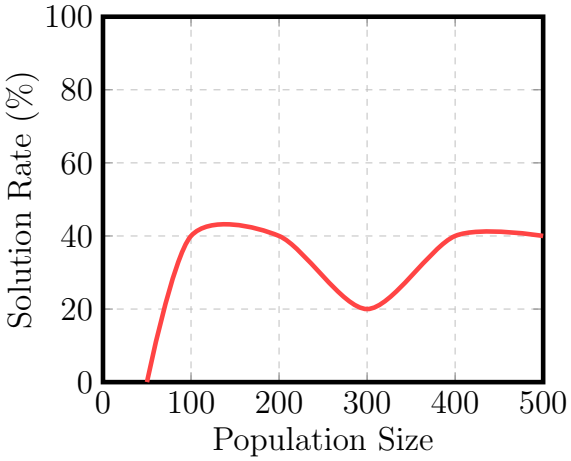


Figure 2: Reduced Pop size of GP



6 Learning Points

7 Professional Issues

References

[1] John R. Koza, *Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm*. Stanford University, 1991.

[2] David Jackson, *Single Node Genetic Programming on Problems with Side Effects*. University of Liverpool, 2012.

[3] Melanie Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1999.

[4] Phillip A. Laplante, *Biocomputing*. Nova Science Publishers Inc, 2003.

[5] R. Poli, W. B. Langdon, N. F. McPhee, J. R. Koza, *A Field Guide to Genetic Programming.* University of Essex, 2008.

[6] John R. Koza, *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[7] John R. Koza, *A Hierarchical Approach to Learning the Boolean Multiplexer Function* Stanford University, 1990.

[8] Brian W. Kernighan, Dennis M. Ritchie, *C Programming Language.* 2nd Edition, Prentice Hall Professional Technical Reference, 1988.

[9] David Jackson, *A New, Node-Focused Model for Genetic Programming.* Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012, Springer Verlag 2012.

[10] C. Lamenca-Martinez, J.C. Hernandez-Castro, J.M. Estevez-Tapiador, and A. Ribagorda *Lamar: A New Pseudorandom Number Generator Evolved by means of Genetic Programming.* University of Madrid, 2006.

[11] Robert M. Gray *Entropy and Information Theory.* First Edition (Corrected), Stanford Unviersity 2000.