Random Number Generation using Genetic Programming Specification

Philip Leonard

Abstract

This is the specification document 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 the project description, the statement of deliverables, the conduct of the project & plan, and the bibliography.

Keywords: Genetic Algorithm (GA), Genetic Programming/Program (GP), Pseudo Random Number Generator (PRNG), Random Number Generator (RNG), Entropy, Single Node Genetic Programming/Program (SNGP), Crossover, Mutation.

1 Project Description

Inspired by Evolutionary Algorithms, Genetic Programming is a method used to evolve computer programs using the Darwinist theory of natural selection (survival of the fittest);

1) To begin with, a random initial population of computer programs (traditionally in a tree structure) is generated from a set of functions/operators and terminals.

2) A fitness function is used to assess the fitness of each program, i.e. how well the program does it's job, or how accurate the computed output is.

3) Genetic operations such as crossover and mutation are used in order to generate new offspring programs.

4) The fitness of the new population is then assessed again, and the programs for the next generation are then selected.

The process from steps three and four are then repeated, creating new generations of "fitter" programs. Generally, Genetic Program runs terminate when a member of the population reaches a certain pre-defined fitness, or when the run has reached a certain number of generations. Fitness functions, probabilities and genetic operations all vary from problem to problem.

A Pseudo Random Number Generator (PRNG) is a program/algorithm which generates a number that possesses the characteristics of a truly random number. These programs typically employ seed values (an example being the number of nanoseconds since the Unix epoch[1]), or they use a sequence of consecutive numbers as an input. Mathematical operations are applied to this input in order to fabricate a "random" number or bit sequence. This differs from true random number generators which typically use a piece of hardware to measure an unpredictable environmental occurrence such as radiation, measured by a Geiger Counter.

Using the paper by Koza [1], the aim of this project is to replicate/implement his research of 'Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm'; taking a sequence of consecutive numbers as the input to produce a random bit sequence. I shall then implement the same idea of evolving a PRNG but by using a newly developed technique known as Single Node Genetic Programming (SNGP). Unlike traditional Genetic Programming, with SNGP every member of the population contains a single program node[2, p.1]. I can then compare the results of the two Genetic Programming methodologies, to find which method performs the best in this specific problem domain.

2 Statement of Deliverables

This section describes what components will formulate the project in its entirety.
Documentation will primarily be technical, due to the projects problem solving nature. User guides will be

[1] 00:00:00 (UTC), Thursday, 1 January 1970

minimal as there will be limited interaction and inputs to the software, as the aim of the program is for it to autonomously solve a problem. The technical documentation will cover the structure of the program and the program code, which in turn will partially act as the end user documentation, as users of the software will be technically educated in the area.

The program will be developed in the C programming language, and the software itself will be small but complex in order to solve such a specific problem. As covered above, user interaction with the program will be limited. It will possibly include basic program initialisations such as; maximum runs, mutation & crossover probabilities, population size and terminating fitness value etc via a command line interface. This will be there to aid the testing of the program under different conditions. For each run of the program, it will output the fittest random number generator in a tree form, and it's fitness value. The fitness of a program will be measured by implementing a fitness function which measures the entropy of the output of each program. Members of the population will be represented as S expressions in prefix notation. By implementing a prefix calculator, the output of each PRNG can be calculated. From this, the entropy of the bit sequence (which in turn becomes the fitness of the program) will be calculated. The idea and explanation of mathematical entropy shall be introduced in the technical documentation. The tree structures of the fittest PRNG from each run, and the smallest PRNG produced[2] from all of the runs shall be written to files so that the user can then easily obtain the results. The same idea will then be implemented for SNGP.

Experimenting will allow me to gather data on running the Genetic Program under different variable inputs for both SNGP and classical GP. For both SNGP and GP, I shall run test cases with varying; population sizes, number of generations, mutation & crossover probabilities and fitness level termination values etc... As covered before this could possibly be done via the command line interface making it easier to execute test cases. From this I can gather entropy data for both approaches, under different conditions.

After test data has been gathered a variety of possible methods can be ultilised to evaluate the programs. By taking the program and it's entropy produced by a run of the GP, I could compare it against widely used Random Number Generators (RNGs) such as; the *rand()* function in the stdlib.h C library, and other common commercial RNGs. The same can be done for the SNGP approach. But most importantly, I will be comparing both SNGP and tree based GP approaches according to entropy and the conditions of each run. This will give me a three way comparison between different PRNGs (commercial and those evolved by GP), thus giving me an abundance of methods to deduce which is the best solution for generating random numbers.

3 Conduct of the Project and Plan

Researching has been and will be a dominating factor in making preparations for the project. As the topic of GP was relatively new to me, I began studying the basics of Genetic Algorithms (GAs) by reading [3, p.1 - 24]. After introducing myself to the idea of GAs, I refined my studying to the Genetic Programming paradigm where I read [4, p.1 - 35] and [5, p.73 - 191]. From this I felt that I had gained a solid introduction, so I moved onto reading the paper on which this project is based by Koza [1], which describes the method of evolving a PRNG using GP. I also studied Genetic Programming code for an implementation of the Multiplexer Problem by Koza [6], which gave me a good idea of how Genetic Programming is implemented in C code. Now I have a strong conceptual image of the workings of my project, which will allow me to write detailed design documents. Future research will include studying SNGP using sources like [2], which will then allow me to implement the same idea of evolving a RNG but using the SNGP methodology. I shall use [7] as a reference to assist me in the implementation stage.

Genetic Programs typically either randomly generate their own initial population or they obtain a seed population from somewhere. In this project I shall be randomly generating the initial population, so no data will need to be gathered from external sources.

The architectural components which define the Genetic Programming paradigm are synonymous across most problems which use this technique. These include; initial population generation, a grow function, a fitness function and genetic operations (crossover and mutation). Therefore in the design stage, I will focus on the functional design of the software, representing the program in an algorithm like flow chart manner. I shall delve specifically into the job of each function and roughly define it's role in a pseudo code style.

[2]The smallest PRNG that still achieved greater or equal to the termination entropy value over all the runs. This allows the user to see the best PRNG and the most efficient PRNG that still meets the entropy criteria.

Implementing the program in terms of extra hardware and software will be relatively straight forward. The are no special hardware requirements for the program. The C implementation of [6] ran well on a standard desktop computer, which tells me that no extra computational power will be required to run my program, especially when it is coded efficiently. No extra software will be required for my project either, merely the standard C libraries.

There are certain risks associated with this project. The Genetic Programming Paradigm is relatively new to me, so there is a possibility of a hold up in the design and implementation stages, while I continue to refine my new skills. Sufficient studying of Genetic Programming resources such as [3, 4, 5], and the aid of the Biocomputation module[3] which I am currently taking should help resolve this issue. Making the Genetic Program efficient, and as efficient as the SNGP implementation could be a challenge in order to gain fair results by comparing the two methods.

Please see fig .1 for a detailed Gantt Chart, representing the full plan and the timescale for this project.

4 Bibliography for the Project

The sources referenced in this document from [1] to [7] have and will continue to be used in this project for referencing and for research. Other materials which will be used in the project are as follows;

- [8] For other SNGP material.
- [9] For another approach to Pseudo Random Number Generation using GP.
- [10] Entropy calculation for the fitness function.

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

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

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

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

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

[9] 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.

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

[3] COMP305, where Genetic Algorithm and Genetic Programming topics are covered

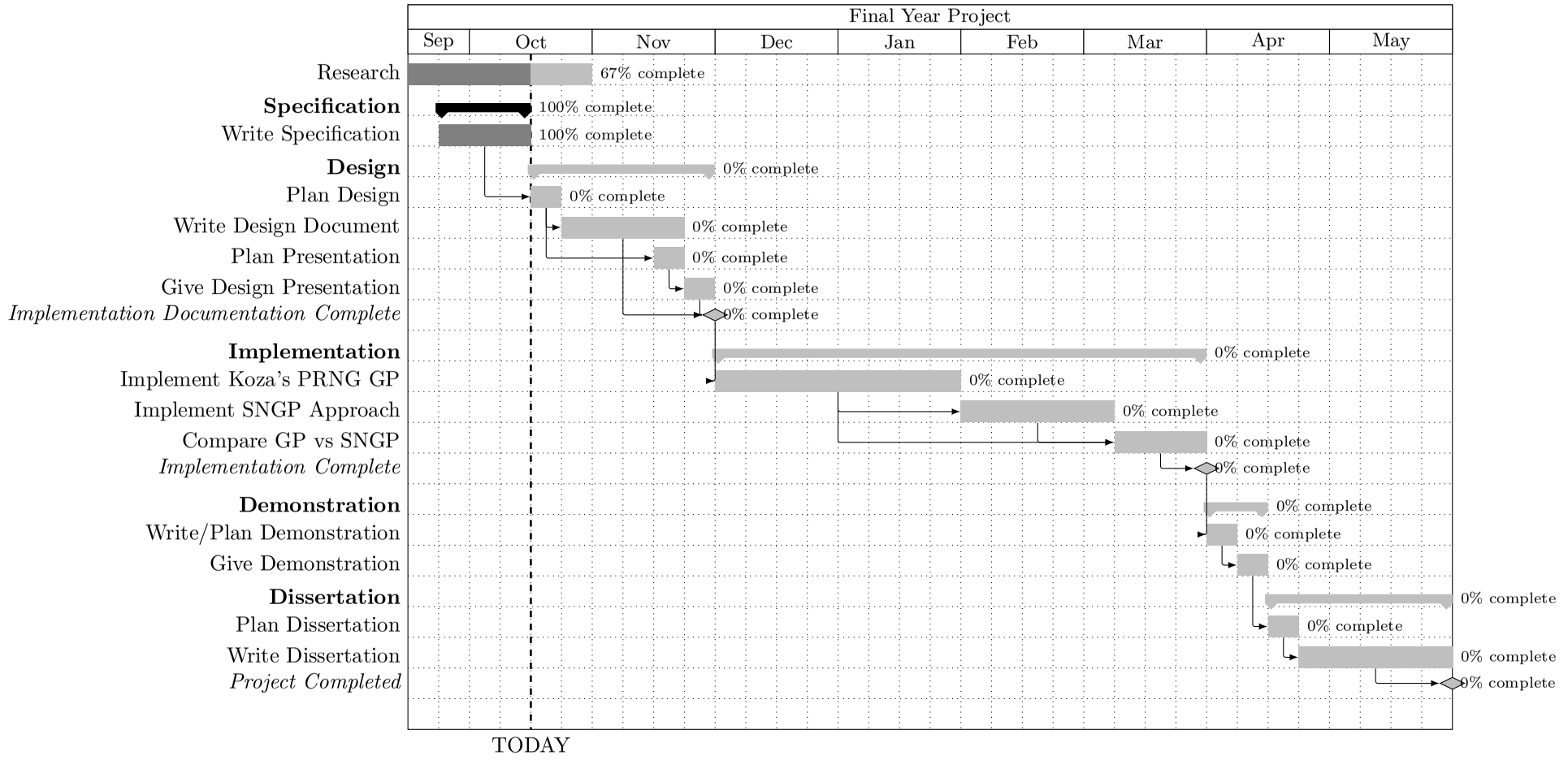

Figure 1: Gantt Chart - Work plan