Random Number Generation using Genetic Programming: Design

Philip Leonard

University of Liverpool

Primary Supervisor: Dr. David Jackson
Secondary Supervisor: Professor Paul E. Dunne

sgpleona@student.liverpool.ac.uk

Presentation Overview

Project Overview

The project can be broken down into 3 main stages, and therefore 3 design stages;

1. Evolving a Random Number Generator (RNG) by means of **Genetic Programming** (GP) in C using Koza's paper [1].

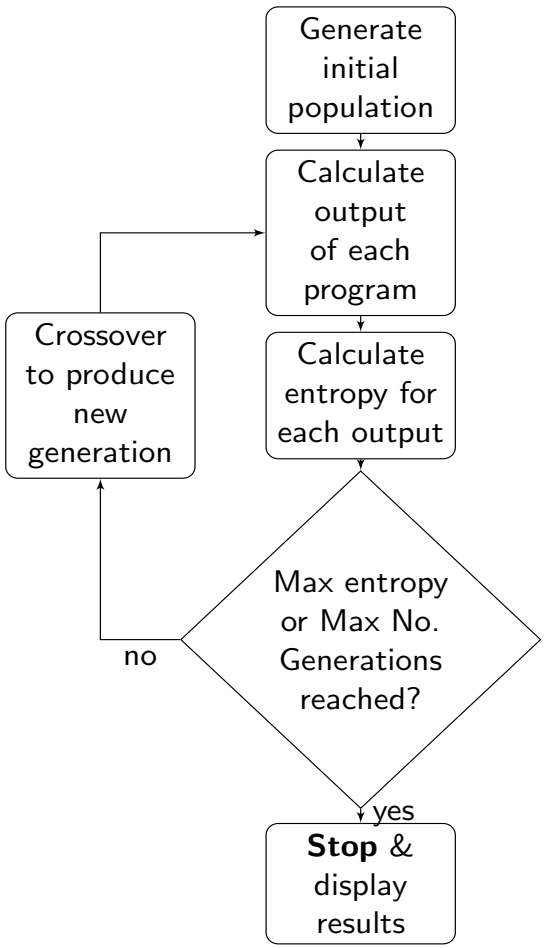
2. Evolving a Random Number Generator (RNG) by means of **Single Node Genetic Programming** (SNGP) in C using Jacksons paper [2].

3. Compare and evaluate GP and SNGP methods for this application and then compare their evolved RNGs against other Pseudo and True Random Number Generators

Genetic Programming Design

- The goal is to genetically breed a RNG to randomise a sequence of binary digits.
- The input to the RNG is the sequence J, which runs from $i = 1$ to 2^{14}.
- The output is a sequence of random binary digits of size J.
- The binary digit at location i is determined by the LSB (Least Significant Bit) of the output from the RNG on the i^{th} input.

Figure : GP flowchart



Inputs

The figure below displays the input parameters for a single Genetic Program run. Allowing these values to be changed by the user allows for easy testing and evaluation.

Figure : Input parameters

Data Type	Input description
Integer	Population size
Integer	Maximum number of generations
Integer	Maximum initial program depth and maximum crossover program depth
Double	Fitness proportionate reproduction to crossover operation balance
Double	Internal to external node crossover probabilities
Double	Target fitness value

Tree Structure

- The GP handles RNGs as binary tree structures coded as a prefix expressions, as shown here.

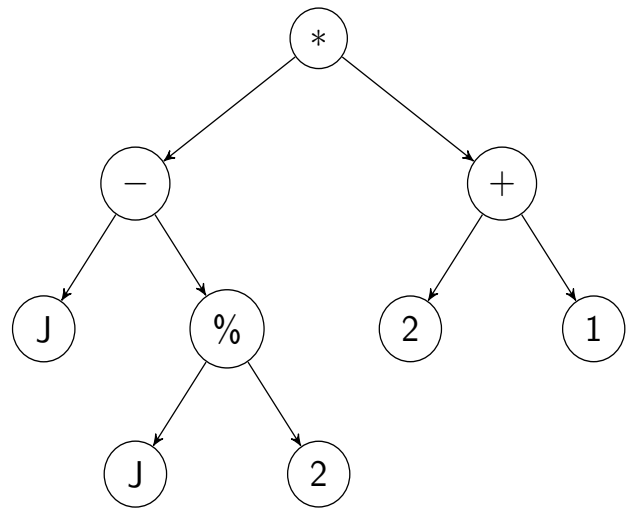
- These trees are initially randomly generated, selecting nodes from the function set F and the terminal set T;

$$F = \{+, -, *, /, \%\}$$
$$T = \{J, \Re\}$$

- The functions / and % (modulus) are protected, meaning that if division by 0 is attempted they return 1.

- $\Re$ is a small random integer constant; 0, 1, 2 or 3

Figure : RNG binary tree



$$* \; - \; J \; \% \; J \; 2 \; + \; 2 \; 1$$

Fitness of a RNG

- The fitness of a RNG will be determined by the entropy of it's output.
- The entropy of a binary sequence is the evenness in the occurrence of a set of sub sequences in that sequence, and can be calculated using the formula below;

$$E_{total} = \sum_{h=1}^{N_{max}} \left[-\sum_{j} P_{(hj)} \ \log_2 \ P_{(hj)} \right]$$

- The fitness function in the GP will implement this equation to calculate he entropy and therefore fitness of each of the RNGs output. N_{max} will be 7 in order to calculate the entropy for subsequence lengths from 1 to 7.
- The maximum fitness a RNG can obtain is;

$$E_{max} = \sum_{i=1}^{7} i = 28$$

Crossover

Crossover will be the operator used to breed new RNG trees. It works by selecting[(1)] two parent trees with two random crossover points[(2)] which point to the root of a sub tree in each parent. These subtrees are swapped to create two new offspring trees.

(1) The two parents are selected using two different methods. Fitness Proportionate Reproduction gives members of the population with a higher fitness a greater probability of being selected. The second method is where one of the parents is selected relative to their fitness and the other uniformly.

(2) The probability of choosing an internal (function) to an external (terminal) crossover point is user defined as an input parameter.

After Crossover, the output and fitness of every individual in the population is recalculated.

Outputs & Termination

After every generation, the data for that population is written to a file;

Figure : Outputs

Data Type	Output description
Integer	Generation number
Double	Generation run time in seconds & hundredths of a second
Double	Total entropy/fitness of fittest candidate
Double	Entropy for subsequence size 1 of fittest candidate
⋮	⋮
Double	Entropy for subsequence size 7 of fittest candidate
Boolean	Entropy of fittest candidate is $\geq$ target fitness?
String	Prefix expression for fittest candidate

This data can be used for evaluation of the program.

If the current generation count is equal to the user defined maximum, or the fitness of the fittest candidate is greater than or equal to the user defined target, the GP terminates. Otherwise process continues, until one of these criteria is met.

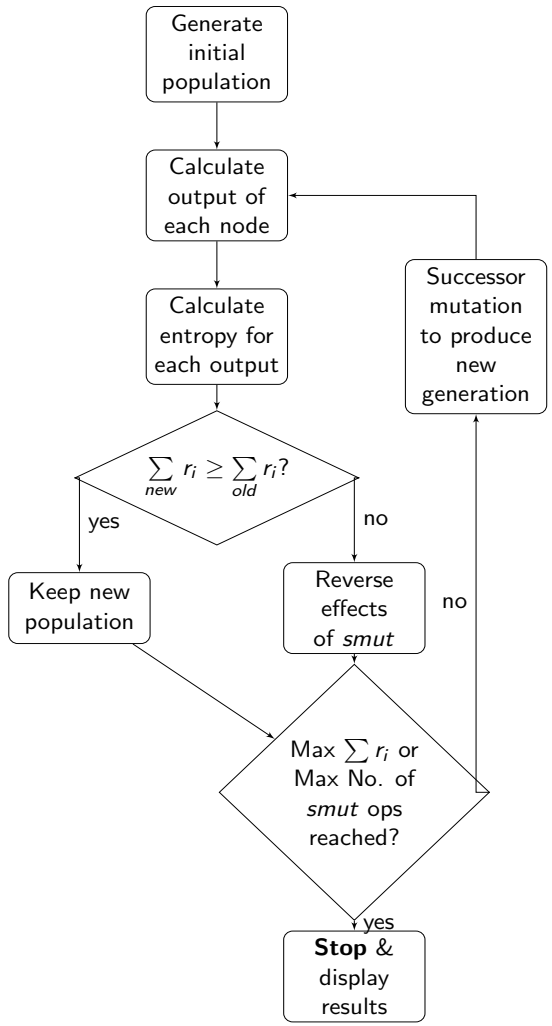
Single Node Genetic Programming Approach

- The aim is again to evolve a RNG, but in this case using the Single Node Genetic Programming (SNGP) methodology.
- There are fewer input parameters to the a SNGP as shown in the figure below;

Figure : SNGP Input parameters

Data Type	Input description
Integer	Length of a run (number of *smut* operations)
Integer	Population size (number of nodes)
Double	Total (i.e. sum of) target fitness value

Figure : SNGP Flowchart



SNGP Structure

- A population is a set M of N tuples;

$$M = \{m_0,\ m_1,\ ...,\ m_{N-1}\} \text{ where;}$$
$$m_i = < u_i,\ r_i,\ S_i,\ P_i,\ O_i > \text{ where;}$$

$u_i \in \{T \cup F\}$ - node in the terminal or function set
r_i - fitness of the node
S_i - set of successors of the node
P_i - set of predecessors of the node
O_i - output vector for this node

- In C the population will be implemented as an array of N *structs*, where the *structs* will represent the tuple above.
- The SNGP population represents a single graph structure.

Graph Structure

- SNGP makes use of Dynamic Programming. When evaluating a tree the outputs of the successor nodes are used rather than calculating them from scratch. Giving it efficiency gains over conventional GP.

- Every node in the graph is evaluated as the root node for a tree. Therefore there are as many RNG trees as there are nodes in the population, and these trees can be evaluated using the same method as the GP method.

- The same fitness function is used to calculate the entropy of a RNG output. (Pseudocode for all main methods are available in the design document);

$$E_{total} = \sum_{h=1}^{7} \left[- \sum_{j} P_{(hj)} \ \log_2 \ P_{(hj)} \right]$$

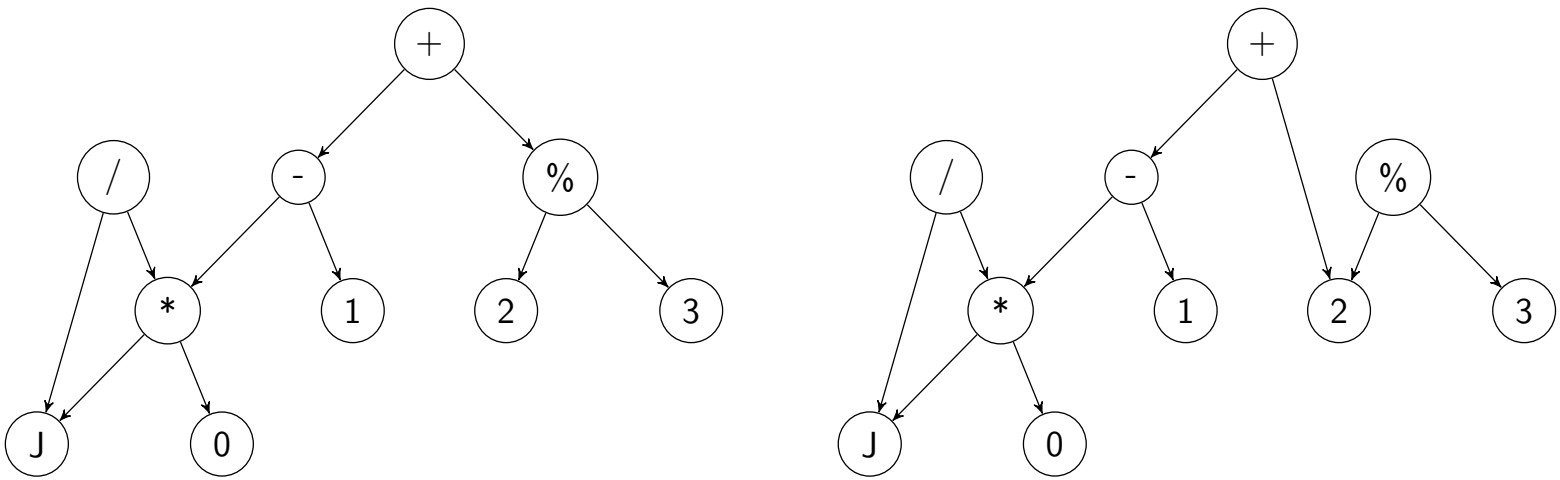
Inital Generation

The initial population (of the size defined by the user) is generated by adding all the elements in the terminal set exactly once, and then populating the rest with random nodes in the function set. The two successors for a function node are randomly selected from the existing population and added to S_i. P_i is then updated in the successor nodes. The outputs and fitnesses are calculated as the nodes are being added.

Successor Mutation

SNGP makes use of one evolutionary operator, Successor Mutation or *smut*. This is where one element in S_i in a node is randomly changed to another (deeper) node in the population.

Figure : Successor Mutation



Before: $\{J\}$, $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{*\ J\ 0\}$, $\{/\ J\ *\ J\ 0\}$, $\{-\ *\ J\ 0\ 1\}$, $\{\%\ 2\ 3\}$, $\{+\ -\ *\ J\ 0\ 1\ \%\ 2\ 3\}$.

After: $\{J\}$, $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{*\ J\ 0\}$, $\{/\ J\ *\ J\ 0\}$, $\{-\ *\ J\ 0\ 1\}$, $\{\%\ 2\ 3\}$, $\{+\ -\ *\ J\ 0\ 1\ 2\}$

Outputs and Termination

The outputs for SNGP are the exact same as those for GP. This will make comparing and evaluating both approaches simpler.

If the current generation count is equal to the user defined maximum, or the sum of all the fitness values for all nodes ($\sum r_i$) is greater than or equal to the user defined target, the program terminates. Otherwise process continues, until one of these criteria is met.

Evaluation Design

GP vs SNGP. To compare GP and SNGP, we must form test cases for matching inputs of both methodologies. There are three matching inputs as shown in the figure below, and the other inputs will remain as constant for all test cases as in the figure below that.

Figure : Matching parameters

Data Type	Input description
Integer	Maximum number of generations / Length of a run (number of *smut* operations)
Integer	Population size; number of trees (GP) or number of nodes (SNGP)
Double	Target fitness value (GP), Target fitness value for sum of all node targets (SNGP)

Figure : Remaining constant parameters for GP

Data Type	Input description	Values
Integer	Maximum initial program depth	4
Integer	Maximum crossover program depth	15
Double	Fitness proportionate reproduction probability	0.1
Double	Crossover operation probability	0.9
Double	Internal node crossover probability	0.9
Double	External node crossover probability	0.1

Test Cases

I formed 8 test cases, two different values for each of the three inputs over all combinations;

	1	2	3	4	5	6	7	8
Max Gen/*smut*	51	51	101	101	101	101	51	51
Pop size	500	1000	500	1000	500	1000	500	1000
Targ Fit	27.990	28.0	27.990	28.0	28.0	27.990	28.0	27.990

Test Criteria

Test criteria for GP vs SNGP

1) Average entropy fitness of fittest member over 100 runs
2) Average solution rate over 100 runs
3) Average minimum and maximum solution size over 100 runs
3) Average run time over 100 runs
4) Generation run time over one run
5) Average solution size over one run
6) Entropy climb over one run
7) Scalar entropies over one run

Other Random Number Generators

- Test the fittest RNGs produced by GP and SNGP against the Pseudo Random Number Generator in the C programming language. Using the same entropy calculation in the fitness function.

- Test the fittest RNGs produced by GP and SNGP against the True Random Number Generator at *www.random.org*, where up to 1,000,000 randomly generated bits can be sent in HTTP packets from a piece of hardware that measures atmospheric noise. From this bit sequence we can measure it's entropy and therefore compare results.

Plan Progress

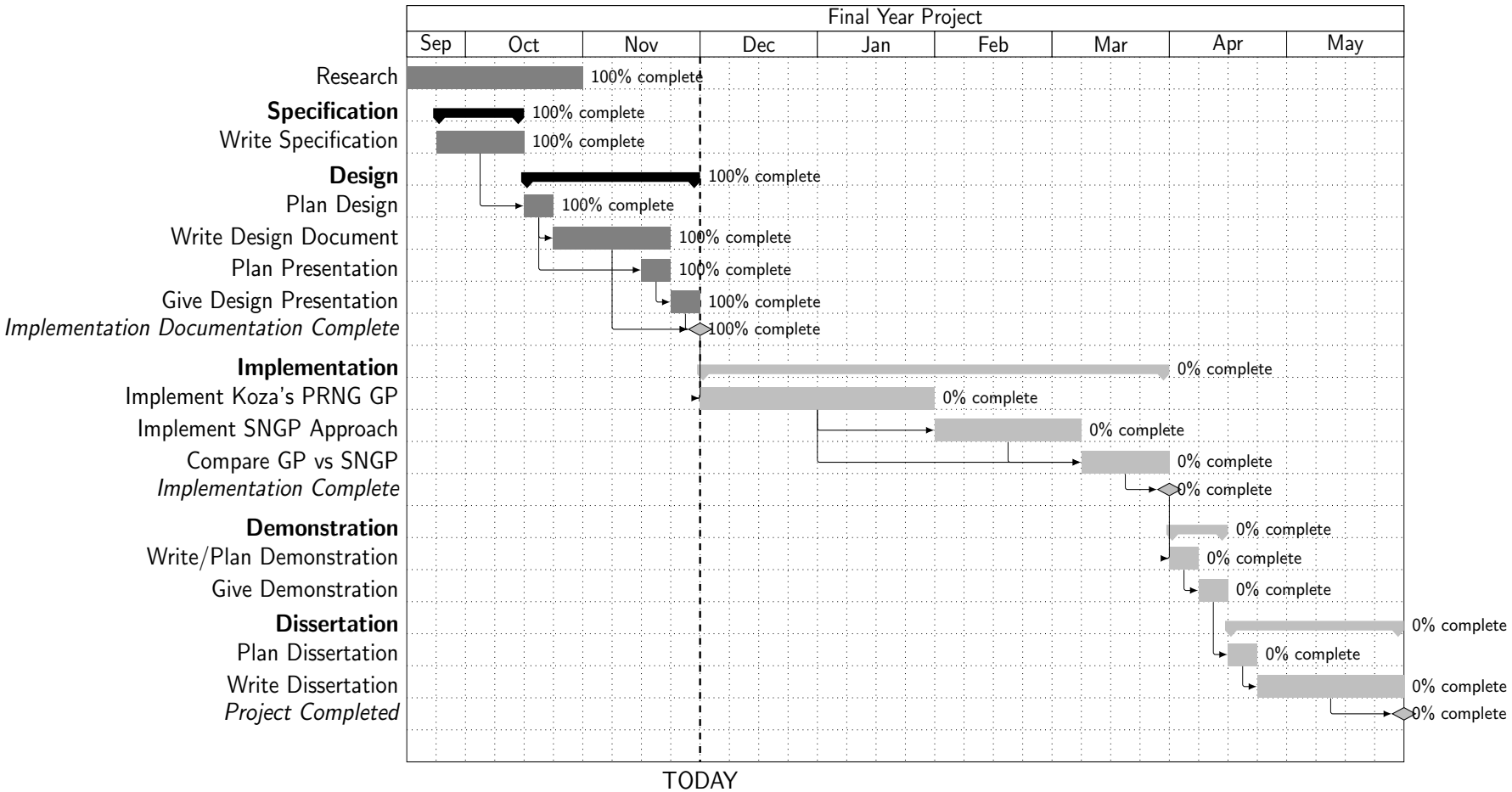


Figure : Gantt Chart - Work plan


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


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

Q&A