Genetic Improvement of Data gives Binary Logarithm from sqrt

W. B. Langdon
Department of Computer Science, UCL

Justyna Petke
Department of Computer Science, UCL

ABSTRACT

Automated search in the form of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), plus manual code changes, transforms 512 Newton-Raphson floating point start numbers from an open source GNU C library, glibc, table driven square root function to create a new bespoke custom mathematical implementation of double precision binary logarithm log2 for C in seconds.

CCS CONCEPTS

• **Software engineering → Search-based software engineer**;

KEYWORDS

genetic programming, GI, search based software engineering, SBSE, software maintenance of empirical constants, data transplantation

ACM Reference Format:
W. B. Langdon and Justyna Petke. 2019. Genetic Improvement of Data gives Binary Logarithm from sqrt. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3321954

1 NEW FUNCTIONALITY VIA DATA UPDATE

Despite all the successes of genetic improvement GI [7], [3], [6], even more than forty years after the advent of software engineering, we are still faced with an industry which is reliant on human labour. Indeed there has been relativity little effort in automating non-coding aspects of software development, such as updating data values. We wish to find a radically new approach to maintaining the numeric and data components of software. Such automation we believe will yield a dramatic increase in human productivity, giving rise to both cost savings and also significantly reducing delays in introducing new system components.

We provide how this can be done for converting existing mathematical functions into new ones, and thus, for example, extending existing mathematical libraries.

Previously we applied Grow and Graft Genetic Programming [4] to one of the existing implementations of the double precision square root function within the GNU C library and so converted sqrt into a double precision cube root function [4] and the normalising function $\frac{1}{\sqrt{x}}$. We apply search based techniques directly to data values embedded inside the source code, with a view to create new functionality rather than to improve existing functionality.

In [4] we claimed that the framework could support other double precision mathematical functions provided there was a suitable objective function. In the case of sqrt and cbrt the inverse function (i.e. square and cube) readily provide such an objective. Here we show log2 can be easily evolved from sqrt using exp2 as the objective function. It should be pointed out that log is a very well known function and there are existing computationally efficient ways to calculate it. Here we use it only as an example.

We have strayed a little from the original motivation with RNAfold [5]. Instead of adapting existing programs to new circumstances, new hardware, new users or new knowledge, by evolving empirical constants within them, we show that there is an interesting class of programs that can be created from existing programs primarily by automatically changing data embedded within them using search based optimisation techniques. We use evolutionary computation in the form of CMA-ES. Full details can be found in technical report RN/18/05 [2].

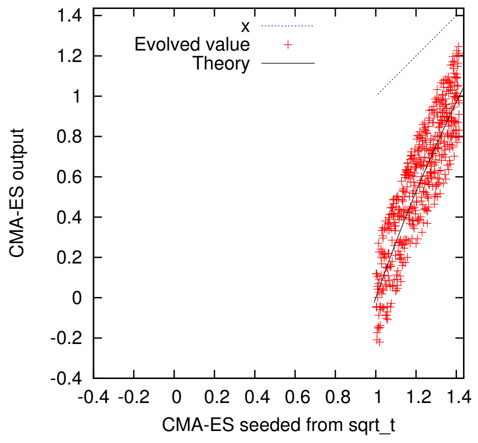

Figure 1: modification from sqrt table values to give corresponding log2 table value (vertical axis). 512 CMA-ES runs.

2 EVOLVING LOG2 DATA TABLE VIA CMA-ES

We use an existing table driven implementation of the square root function sqrt and mutate the constant values in sqrt's table to give a table driven implementation of log2. The open source C implementation of sqrt is provided by the free software foundation (GNU). Release glibc-2.27 of the GNU C library was downloaded from https://www.gnu.org/s/libc/. It contains multiple implementations of the square root function. One (../powerpc/fpu), which uses table lookup, was selected for use as a model for a table-based version of the logarithm to base 2 function (log2). See RN/18/05 [2] for details of manual code changes.

The __t_sqrt table contains 512 pairs of floats. In sqrt, each pair is the start point and starting derivative for the fixed iteration (3) Newton-Raphson algorithm. Notice only float precision is needed for the start points. Newton-Raphson will convergence rapidly to double precision accuracy. By exploiting the IEEE754 floating point representation, the sqrt code converts all input x values into one of 512 possible starting bins. Each corresponding to an entry in __t_sqrt. For the new log2 code we do not need start values for the derivative, so the new table __t_log2 becomes 512 floats, which we are going to evolve using CMA-ES [1].

The top 256 __t_sqrt pairs correspond to numbers in the range 1 to 2. They become CMA-ES's initial (seed) value when it evolved the new table __t_log2. (CMA-ES is run 512 times, see Figure 1).

The Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES [1]) is the state-of-the-art evolutionary strategy tool for solving continuous domain problems. It was downloaded from https://github.com/cma-es/c-cmaes/archive/master.zip It was set up to fill the table of floats one at a time. Each value is initially set to either the corresponding value in __t_sqrt or the mean of two adjacent pairs. The initial mutation step size used by CMA-ES was set to 3.0 times the standard deviation calculated from the first of each of the 512 pairs of numbers in __t_sqrt.

The CMA-ES defaults (cmaes_initials.par) were used.

2.1 Fitness function

Each time CMA-ES proposes a value, it is converted into a float and loaded into __t_log2 at the location that CMA-ES is currently trying to optimise. The fitness function uses three fixed test double values in the range 1.0 to 2.0. These are: the lowest value for the __t_log2 entry, the mid point and the top most value. Our log2 function is called (using the updated __t_log2) for each and a sub-fitness value calculated with each of the three returned doubles. The sub-fitnesses are combined by adding them.

Each sub-fitness takes the output of our log2, and takes the absolute difference between this and the GNU log2 library function. If they are the same, the sub-fitness is 0, otherwise it is positive. Since when our log2 is working well, the differences are very small, they are re-scaled for CMA-ES. If the absolute difference is less than one, its (natural) log is taken, otherwise the absolute value is used. In both cases (i.e. as long as the difference is not zero), to prevent the sub-fitness being negative, a constant, 40.0, is added.

In all 512 runs CMA-ES found a value for which all three test cases passed. (Total run time 6 seconds).

2.2 Testing the evolved log2 function

The results makes plain, that for a function as smooth as logarithm, no great care is needed and more-or-less any reasonable value would do. Nonetheless we will continue to show that our evolved table driven version of the binary logarithm works as well as the GNU C library's log2. The glibc-2.27 powerPC IEEE754 table-based double sqrt function claims to produce answers within one bit of the correct solution. On 1 536 tests of large integers ($\approx 10^{16}$) designed to test each of the 512 bins 3 times (min, max and a randomly chosen point) our GI log2 always came within DBL_EPSILON (i.e. $2.2\ 10^{-16}$) of the correct answer.

As well as the large positive integer tests mentioned in the previous paragraph, our log2 was tested with 5120 random numbers uniformly distributed between 1 and 2 (the largest deviation was in the least significant part of IEEE 754 double precision corresponding to $4.44\ 10^{-16}$). It was also tested on 5120 random scientific notation numbers and 5120 random 64 bit patterns. Half the random scientific notation numbers were negative and half positive. Half were smaller than one and half larger. The exponent was chosen uniformly at random from the range 0 to |308|. In one case a random 64 bit pattern corresponded to NAN (Not-A-Number) and our log2 correctly returned NAN. In all other cases our log2 returned a double, which when fed into the GNU C library exp2 function gave its input exactly or gave a number within one bit of the closest value which could be inverted by exp2 to yield the original value.

3 CONCLUSIONS

Starting with a double precision implementation of sqrt from the GNU C library, we have used artificial evolution to find data values which give an accurate double precision implementation of log2. Modifications to the sqrt source code are needed and were made by hand, see RN/18/05 [2, sect. 2.1].

We have shown it is possible to use glibc's Newton-Raphson approach with the cube root [4], the reciprocal square root $x^{-1/2}$ and now the binary logarithm, log2. However Newton-Raphson still requires an objective function. We have chosen to use the GNU C library exp2.

In very limited computing environments (e.g. tiny, ≈1 millimetre, processors such as smart dust, mote computing) it may be impossible to host GLIBC and there may be very limited electrical power for computation. There could still be realtime requirements, which require a limited maths library. This limitation could take two forms. 1) restrictions on inputs. 2) Only a few combinations of functions are needed. This approach might be able to create an efficient implementation of such novel composite functions of no more than the required accuracy and still fit in a small read only memory (e.g. 4K bytes).

Source code and the evolved implementation are available via http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gi_cbrt.tar.gz

REFERENCES

[1] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (Summer 2001), 159–195. https://doi.org/doi:10.1162/106365601750190398

[2] W. B. Langdon. 2018. *Evolving Square Root into Binary Logarithm.* Technical Report RN/18/05. University College, London, London, UK. http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research_Notes/RN_18_05.pdf

[3] William B. Langdon and Mark Harman. 2015. Optimising Existing Software with Genetic Programming. *IEEE Transactions on Evolutionary Computation* 19, 1 (Feb. 2015), 118–135. https://doi.org/doi:10.1109/TEVC.2013.2281544

[4] William B. Langdon and Justyna Petke. 2018. Evolving Better Software Parameters. In *SSBSE 2018 Hot off the Press Track (LNCS)*, Thelma Elita Colanzi and Phil McMinn (Eds.), Vol. 11036. Springer, Montpellier, France, 363–369. https://doi.org/doi:10.1007/978-3-319-99241-9_22

[5] William B. Langdon, Justyna Petke, and Ronny Lorenz. 2018. Evolving better RNAfold structure prediction. In *EuroGP 2018 (LNCS)*, Mauro Castelli et al. (Eds.), Springer Verlag, 220–236. https://doi.org/doi:10.1007/978-3-319-77553-1_14

[6] Justyna Petke, Saemundur O. Haraldsson, Mark Harman, William B. Langdon, David R. White, and John R. Woodward. 2018. Genetic Improvement of Software: a Comprehensive Survey. *IEEE Transactions on Evolutionary Computation* 22, 3 (June 2018), 415–432. https://doi.org/doi:10.1109/TEVC.2017.2693219

[7] David Robert White, Andrea Arcuri, and John A. Clark. 2011. Evolutionary Improvement of Programs. *IEEE TEVC* 15, 4 (2011), 515–538.