Research Note

RN/18/05

Evolving Square Root into Binary Logarithm

27 June 2018

W. B. Langdon

Abstract

Genetic improvement can adapt numbers to create new functionality. Expanding prior work in which Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES) converted a GNU C library sqrt into cbrt, we show (with suitable objective function) the same GGGP framework can find a double precision binary logarithm log2 for C by adjusting 512 floats

Keywords genetic algorithms, genetic programming, genetic improvement, search based software engineering, SBSE, software maintenance of empirical constants, data transplantation, glibc, exponential doubling function, exp2

1 Creating New Functionality with Data Changes

Previously [1] we applied Grow and Graft Genetic Programming [2] 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. However most Genetic Improvement [3; 4; 5; 6] has applied evolution to program code. Usually source code, but also byte code [7] assembler [8] and even machine code [9] have been optimised. A suitable fitness function is essential for such directed evolution. The fitness function may consider functionality (as here) and/or non-functional properties, such as run-time, energy [10; 11; 12; 13] or even memory [14] consumption. In [15] we considered the problems of using physical measurements as part of the fitness measure for GI. As in [1], here we apply search based techniques [16] directly to data values embedded inside the source code, with a view to create new functionality rather than to improve existing functionality. See [1] for a quick survey of work on software maintenance related to maintaining numbers within programs.

In [1] 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. The framework could be adapted to the trigonometric mathematics functions again by using the inverse function as the objective but to be useful the objective must either be computationally cheap or readily available (ideally both).

Unlike the deep parameter tuning work by Wu et al. [14], our goal is not to improve existing functionality by modifying existing functionality, but to use the framework to create new functionality. Also it is related to data transplanting as almost all the changes are related to numbers rather than to code (c.f. program transplanting [17; 18]) as data are transferred from the square root function to log2. However they are heavily adapted by CMA-ES [19] once in their new home.

We have strayed a little from the original motivation with RNAfold [20][1]. 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. In the case of converting sqrt to cbrt and (here) sqrt to log2, we use evolutionary computation in the form of CMA-ES.

We follow the structure of [1]: in the next section we describe the experiment which evolved log2 from the GNU C library sqrt and then conclude in Section 3.

2 Automated Parameter Tuning for Evolving log2

For completeness, this section repeats some of the details of the framework already given in [1, Sect. 2].

In [1] we used an existing implementation of the square root function and used genetic improvement to evolve a cube root function. This was achieved by mutating the constant values in the existing code. Here we use the same approach and starting point. The current release of the GNU C library (glibc-2.27) 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 [21] was selected for use as a model for a table-based version of the logarithm to base 2 function (log2).

2.1 Manual changes

We are primarily concerned with adjusting data values. However, some changes to the existing powerPC sqrt code were made by hand so that it could support log2: 1) Various powerPC optimisations were dis-

[1]The GI parameters have been distributed with the ViennaRNA package, which includes RNAfold, since version 2.4.7.

abled. 2) For simplicity the trap for negative numbers is implemented by failing an assert[2]. 3) Division of the exponent part of double precision numbers by two is replaced by simply adding its value to the final result. 4) As with cbrt, the top nine bits of the fractional part are used as the table index. Thus numbers in the range 1 to 2 are mapped onto the table's 512 entries. See also Figure 1. 5) The Newton-Raphson objective function for sqrt (i.e. x^2) is replaced by 2^x. Since the derivative of log2(x) can be calculated exactly from x (using the existing constant M_LOG2El), it is not necessary to store it in the table or to estimate it or update the current estimate. Hence the table need only contain 512 values (rather than 512 pairs of values) and so the lines of code to maintain sy, sy2 and e can be removed. Similarly the constant almost_half can also be removed. 6) The infrequently used recursive code to deal with denormalised numbers (denorm:) is similarly simplified to multiply by the existing constant two108 and subtract 108 from the result.

2.2 *Automatic changes to data table using CMA-ES*

The __t_sqrt table contains 512 pairs of floats. The top 256 correspond to numbers in the range 1 to 2. The value estimates, rather than the estimates of the derivatives, i.e. the first of each pair, were used as start points when evolving the 512 floats in the new table __t_log2.

The Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES [19]) 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. As with cbrt [1], 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.

2.2.1 CMA-ES parameters

The CMA-ES defaults (cmaes_initials.par) were used, except: the problem size (N 1), the initial values and mutation sizes are loaded from __t_sqrt (see previous section) and various small values concerned with run termination were set to zero (stopFitness, stopTolFun, stopTolFunHist, stopTolX). The initial seed used for pseudo random numbers was also set externally.

2.2.2 Fitness function

Each time CMA-ES proposes a value (N=1), 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. However, in both cases, to prevent the sub-fitness being negative, log of the smallest feasible (i.e. encountered) non-zero difference is subtracted (i.e. 40.0 is added).

CMA-ES will stop when the difference on all three test points is zero.

2.2.3 Restart Strategy

Although a restart strategy was provided (as [1, Sect. 2.2]), in all 512 runs CMA-ES found a value for which all three test cases passed. Again the first reported solution was used.

[2]The GNU C library provides a sophisticated but complicated mechanism for raising exception, which would have to be used if the GI version of log2 were to be incorporated back into glibc, but it is not necessary at this stage, since we only wish to demonstrate our approach.

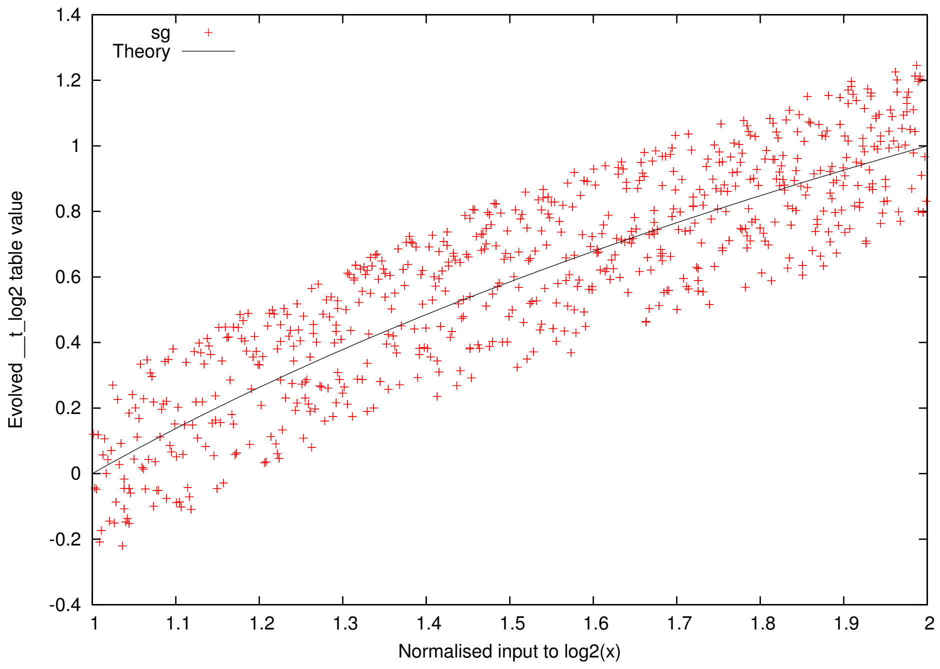

Figure 1: 512 (*sg*) numbers found by CMA-ES for use as start points for Newton-Raphson three step calculation of the binary logarithm, i.e. the values in the float array __t_log2. Horizontal axis is the normalised input to log2 corresponding to each *sg* value in __t_log2.

2.3 Testing the evolved log2 function

The float values found by CMA-ES, called *sg* in the system, are shown in Figure 1. Figure 1 makes plain, that for a function as smooth as logarithm, no great care is needed in starting Newton-Raphson 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. $5\ 10^{-12}$) 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 IEEE754 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 in the traditional way, i.e. by hand, see Section 2.1.

The sqrt code uses a fast fixed three step Newton-Raphson approach. In addition to an estimate of the derivative of the square root function, Newton-Raphson requires an objective measure, x^2, to tell it how far it has overshot. Notice, although sqrt uses a table of values, it is not interpolating. Instead it is actually calculating the true square root to double precision accuracy in a small number of steps. For speed, it uses the table of starting values to limit the number of Newton-Raphson iterations needed before it converges on the right answer.

We have shown it is possible to use the same approach with the binary logarithm, log2. However Newton-Raphson still requires an objective function. We have chosen to use the GNU C library exp2. Naturally this limits the usefulness of the Newton-Raphson approach for log2. However logarithm has the advantage that its derivative can be easily calculated, whereas in sqrt and (our cbrt) Newton-Raphson had to maintain and refine an estimate of the derivative at each step. Removing the code to do this greatly simplifies our GI log2. Further Figure 1 suggests that CMA-ES had almost no work to do and any reasonable estimate of log2, such as $x - 1$, would perhaps be sufficient. This would also allow the removal of the whole of the __t_log2 table of start values and deleting even more of the sqrt source code.

There are potentially other mathematical functions where our approach might be beneficial. For example, functions without direct hardware support, where there is a fast objective measure available to Newton-Raphson and where the function is sufficiently non-linear to justify evolving a table of constants for a fixed iteration Newton-Raphson approach. Perhaps $\sqrt[5]{}$, $\sqrt[7]{}$, $\sqrt[11]{}$ ... (with objective functions x^5, x^7, x^{11}...). Also the availability of small (e.g. 4K bytes) of fast memory, e.g. within core, might suit this approach.

Our previous work [20; 1] has shown it is not uncommon for programs to contain, sometimes large numbers of, embedded constants. These may be critical to the software's performance and although the problem has been known for a long time [22]) there is as yet very little research on automatically maintaining them.

Code

Scripts, source code and the evolved implementation are available via anonymous FTP ftp.cs.ucl.ac.uk genetic/gp-code/gi_cbrt.tar.gz and from http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gi_cbrt.tar.gz

Acknowledgements

My thanks to our EuroGP [20] anonymous reviewers.

Funded by EPSRC grant EP/M025853/1.

References

[1] Langdon, W.B., Petke, J.: Evolving better software parameters. In Aleti, A., Panichella, A., eds.: SSBSE 2018 Hot off the Press Track, Montpellier, France (2018)

[2] Langdon, W.B., Lorenz, R.: Improving SSE parallel code with grow and graft genetic programming. In Petke, J., et al., eds.: GI-2017, Berlin, ACM (2017) 1537–1538

[3] Langdon, W.B.: Genetic improvement of programs. In Matousek, R., ed.: 18th International Conference on Soft Computing, MENDEL 2012, Brno, Czech Republic, Brno University of Technology (2012) Invited keynote.

[4] Langdon, W.B.: Genetically improved software. In Gandomi, A.H., et al., eds.: Handbook of Genetic Programming Applications. Springer (2015) 181–220

[5] Petke, J.: Preface to the special issue on genetic improvement. Genetic Programming and Evolvable Machines **18**(1) (2017) 3–4 Editorial Note.

[6] Petke, J., Haraldsson, S.O., Harman, M., Langdon, W.B., White, D.R., Woodward, J.R.: Genetic improvement of software: a comprehensive survey. IEEE Transactions on Evolutionary Computation **22**(3) (2018) 415–432

[7] Orlov, M., Sipper, M.: Flight of the finch through the Java wilderness. IEEE Transactions on Evolutionary Computation **15**(2) (2011) 166–182

[8] Schulte, E., Forrest, S., Weimer, W.: Automated program repair through the evolution of assembly code. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Antwerp, ACM (2010) 313–316

[9] Schulte, E., Weimer, W., Forrest, S.: Repairing COTS router firmware without access to source code or test suites: A case study in evolutionary software repair. In Langdon, W.B., et al., eds.: Genetic Improvement 2015 Workshop, Madrid, ACM (2015) 847–854 Best Paper.

[10] White, D.R., Clark, J., Jacob, J., Poulding, S.M.: Searching for resource-efficient programs: low-power pseudorandom number generators. In Keijzer, M., et al., eds.: GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA, ACM (2008) 1775–1782

[11] Schulte, E., Dorn, J., Harding, S., Forrest, S., Weimer, W.: Post-compiler software optimization for reducing energy. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'14, Salt Lake City, Utah, USA, ACM (2014) 639–652

[12] Bruce, B.R.: Energy optimisation via genetic improvement a SBSE technique for a new era in software development. In Langdon, W.B., et al., eds.: Genetic Improvement 2015 Workshop, Madrid, ACM (2015) 819–820

[13] Burles, N., Bowles, E., Brownlee, A.E.I., Kocsis, Z.A., Swan, J., Veerapen, N.: Object-oriented genetic improvement for improved energy consumption in Google Guava. In Labiche, Y., Barros, M., eds.: SSBSE. Volume 9275 of LNCS., Bergamo, Springer (2015) 255–261

[14] Wu, F., Weimer, W., Harman, M., Jia, Y., Krinke, J.: Deep parameter optimisation. In Silva, S., et al., eds.: GECCO, Madrid, ACM (2015) 1375–1382

[15] Langdon, W.B., Petke, J., Bruce, B.R.: Optimising quantisation noise in energy measurement. In Handl, J., et al., eds.: 14th International Conference on Parallel Problem Solving from Nature. Volume 9921 of LNCS., Edinburgh, Springer (2016) 249–259

[16] Harman, M., Jones, B.F.: Search based software engineering. Information and Software Technology **43**(14) (2001) 833–839

[17] Marginean, A., Barr, E.T., Harman, M., Jia, Y.: Automated transplantation of call graph and layout features into Kate. In Labiche, Y., Barros, M., eds.: SSBSE. Volume 9275 of LNCS., Bergamo, Springer (2015) 262–268

[18] Lu, Y., Chaudhuri, S., Jermaine, C., Melski, D.: Program splicing. In Chechik, M., Harman, M., eds.: 40th International Conference on Software Engineering, Gothenburg, Sweden, ACM (2018) 338–349

[19] Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2) (2001) 159–195

[20] Langdon, W.B., Petke, J., Lorenz, R.: Evolving better RNAfold structure prediction. In Castelli, M., et al., eds.: EuroGP. LNCS 10781, Springer (2018) 220–236

[21] Markstein, P.W.: Computation of elementary functions on the IBM RISC System/6000 processor. IBM J Res. Dev. **34**(1) (1990) 111–119

[22] Martin, R.J., Osborne, W.M.: Guidance on software maintenance. NBS Special Publication 500-106, National Bureau of Standards, USA (1983)