SSBSE-2018. Thelma Elita Colanzi and Phil McMinn *Eds.*, LNCS 11036, p363–369, Montpellier. 8-9 Sept. 2018. Springer. doi:10.1007/10.1007/978-3-319-99241-9_22
*Preprint*

# Evolving Better Software Parameters

William B. Langdon and Justyna Petke

CREST, Computer Science, UCL, London, WC1E 6BT, UK

**Abstract.** Genetic improvement might be widely used to adapt existing numerical values within programs. Applying GI to embedded parameters in computer code can create new functionality. For example, CMA-ES can evolve 1024 real numbers in a GNU C library square root to implement a cube root routine for C.
**Keywords** genetic improvement, SBSE, GGGP, software maintenance of empirical constants, data transplantation, glibc, sqrt, cbrt

## 1 Literature on Maintaining Numbers within Code

Many programs contain embedded parameters. Typically these are numeric values (often float or double, but also integers, e.g. the GNU C library contains more than a million integer constants, see Figure 1, also [1]). In many cases these parameters relate to the software itself or to simple facts which are unlikely to change during the program's lifetime or period of active use. However, many others aught to be updated. This maintenance problem has been known for a long time (Martin and Osborne, 1983 [2, Section 6.8, page 24, Hard Coded Parameters Which Are Subject To Change]).

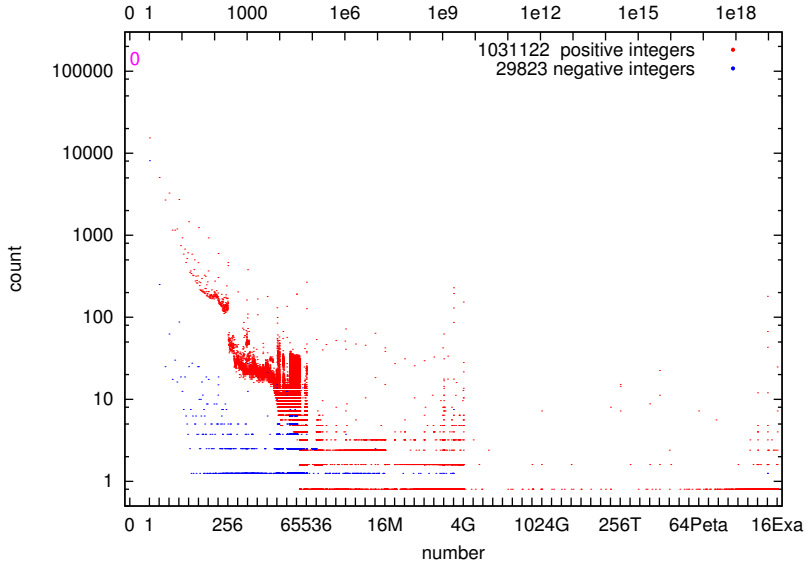


**Fig. 1.** The GNU C library version 2.27 (excluding test suite) contains 1 202 711 integer constants. Zero is the most common, occurring a total of 141,874 times, followed by 1 (19 203) and -1 (6 479). Every integer between -28 and 40 956 occurs at least once.

Parameters may relate to heuristics within the code, which the developer chose before contact with real users. Their values perhaps should have been updated shortly after first release, or values (e.g. those relating to memory or array sizes) may need updating due to operating on new hardware, as well as to changes in patterns of use. Other parameters can relate to the problem itself. For example, chemical reaction rate constants in ozone layer simulations [3]. In some cases the exact numerical values are critical [3]. Some physical values are known with very high precision, but for others the state of scientific knowledge can improve over the operational life of the program. For example, the ViennaRNA package [4] contains more than 50 000 binding energy values. These are derived from scientific measurements of RNA molecules. Even so, during the relatively short life of this suite of C programs, knowledge has moved on and various newer versions of these parameters are available. Recently [5], we showed genetic improvement could be used to adapt these 50 000 int values. (The GI values have been distributed with ViennaRNA since version 2.4.5.)

As computing is now mature, maintaining software has become the dominant cost. Marounek [6, page 51] quotes figures of more than 90% of total cost. Moreover, software maintenance routinely requires highly skilled experts [7, page 65]. Yet a forthcoming survey [8] starts by saying "a relatively small amount [of SBSE research] is related to software maintenance", whilst [9] does not give a break down of the SBSE literature on software maintenance. Indeed it appears that maintaining embedded constants within existing packages has received little attention so far. For example, [10] considers the maintenance impact of names given to constants in Java source code, but not how to maintain their values. Similarly, [11] consider how to hide constant values, but not how to update them.

There is some research on parameter tuning. For example, ParamILS[1] or irace[2] tools. However, there is scarcely any on updating parameters in the code that are not specifically exposed to the user for tuning. The deep parameter tuning work by Wu et al. [12] being the first known example, where they optimised for runtime and memory consumption. Unlike Wu et al. [12], we focus on adapting numerical values only. Previous work on evolving new features using GI dealt with transplantation of portions of one program to another [13], or evolving functionality separately and then adding them to existing code using automated software transplantation [14] (so-called 'grow-and-graft'). Our approach does not require additional code, just changes within the existing code base.

In the next section we continue exploring automated parameter tuning by taking existing code which relies on ≈1 thousand embedded constants from the GNU C library to create a function, cbrt, which is not implemented by the library. Section 2.3 shows its accuracy is typically better than $2\ 10^{-16}$ and not worse than $10^{-15}$. (I.e. typically within one bit in the IEEE 754 double precision representation.) Finally, in Section 3, we suggest there is a great need for research into both automated data update and data transplantation.

[1] `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/`
[2] `http://iridia.ulb.ac.be/irace/`

## 2 Example of Automated Parameter Tuning for Evolving New Functionality

We use an existing implementation of the square root function and use genetic improvement to evolve a cube root function. This is achieved by mutating the constant values in the chosen code for square root.

The current release of the GNU C library (glibc-2.27, 1 Feb 2018, 851 080 lines of non-test code) was downloaded from `https://www.gnu.org/s/libc/`. It contains multiple implementations of the square root function (sqrt). One (../powerpc/fpu) which uses table lookup [15] was selected for use as a model for a table-based version of the cube root function (cbrt).

### 2.1 Manual changes

We are primarily concerned with adjusting data values. However, a few changes to the existing powerPC sqrt code were made by hand so that it could support cbrt. Whilst in [5] no code changes were needed, we envision that such changes may be required. For cbrt: 1) Various powerPC optimisations were disabled. 2) Replaced the trap for negative numbers by returning $-\sqrt[3]{-x}$ if $x$ is negative. 3) Division of the exponent part of double precision numbers by three is rather more tricky than division by two. Keeping track of the remainder required the multiplication or division by $\sqrt[3]{2}$ or $\sqrt[3]{\sqrt[3]{2}}$ (Section 2.3). The existing constants CBRT2 and SQR_CBRT2 were used. 4) sysdeps/powerpc/fpu/e_sqrt.c uses a right shift to do two operations. Firstly to divide the exponent by two. And secondly to combine the least significant bit of the exponent with the top eight bits of the fractional part, forming a nine bit index into the table. Effectively mapping numbers in the range 0.5 to 2 onto the table. The more tricky division by three led to the decision to exclude the exponent and to just use the top nine bits of the fractional part as the table index. So numbers in the range 1 to 2 are mapped onto the table, see also Figure 2. 5) The constant almost_half was replaced by new constant almost_third = 0.3333333333333334.

### 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. These were used as start points when evolving the 512 pairs of floats in the new table __t_cbrt.

The Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES [16]) was downloaded from `https://github.com/cma-es/c-cmaes/archive/master.zip` It was set up to fill the table of floats one pair at a time. Each pair being initially set to either the corresponding pair of values in __t_sqrt or the mean of two adjacent pairs. The initial mutation step sizes used by CMA-ES were set (pairwise) to 3.0 times the standard deviation calculated from the 512 pairs of numbers in __t_sqrt.

**CMA-ES parameters** The CMA-ES defaults (cmaes_initials.par) were used, except: the problem size (N 2), 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.

**Fitness function** Each time CMA-ES proposes a pair (N=2) of double values, they are converted into floats and loaded into __t_cbrt 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_cbrt entry, the mid point and the top most value. The cbrt function is called (using the updated __t_cbrt) 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 cbrt, cubes it and takes the absolute difference between this and the corresponding test value. If they are the same, the sub-fitness is 0, otherwise it is positive. Since when cbrt is working well, the differences are very small, they are re-scaled for CMA-ES. If the absolute difference is less than one, its 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 non-zero difference DBL_EPSILON is subtracted.

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

**Restart Strategy** When CMA-ES failed to find a pair of values for which all three test cases pass, it was run again with the same initial starting position and mutation size, but a new pseudo random number seed. Mostly CMA-ES found a suitable pair in one run, but in 107 of 512 cases it was run more than once. (In no case was CMS-ES run more than 4 times on a particular pair.)

### 2.3 Testing the evolved cbrt function

The pairs of float values found by CMA-ES, called *sg,sy* in the system, are shown in Figure 2. 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) the largest discrepancy between (cbrt(x)**3) and x was three (i.e. 6.66 $10^{-16}$). In all tests, including those described in the rest of this section, this only arose when the exponent part of the double was not a multiple of 3. This requires the cbrt code to do an extra multiply or divide by $\sqrt[3]{2}$ or $\sqrt[\frac{2}{3}]{2}$ (i.e. CBRT2 or SQR_CBRT2, see Section 2.1), apparently resulting in additional loss of precision.

As well as ad-hoc testing, and the large positive integer tests mentioned in the previous paragraph, cbrt was tested with 5 120 random numbers uniformly distributed between 1 and 2 (the largest deviation was two[3] 5 120 random scientific notation numbers and 5 120 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 cbrt correctly returned NAN. In most cases cbrt returned a double, which when cubed was its input or within one bit of it. In some cases the cubed answer was two from the input. The maximum deviation was 3.

[3] 2 at the least significant part of IEEE754 double precision corresponds to 4.44 $10^{-16}$.

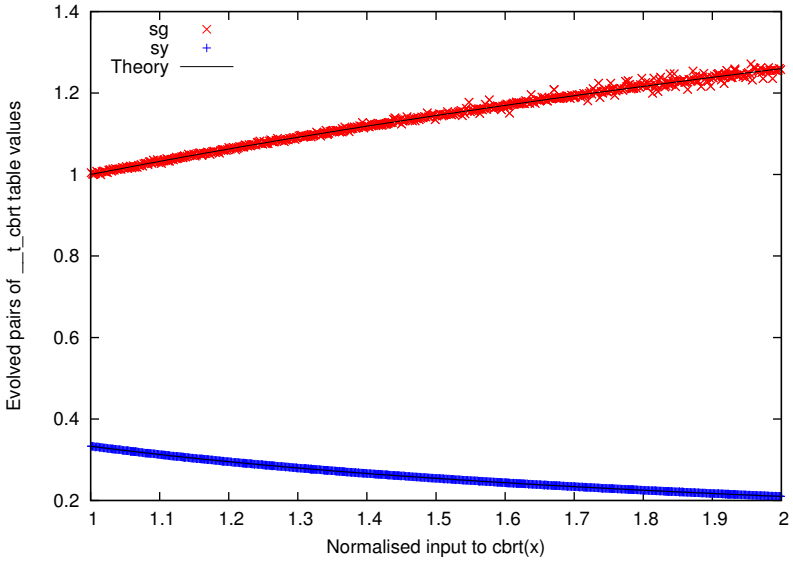


**Fig. 2.** 512 (*sg,sy*) pairs of numbers found by CMA-ES for __t_cbrt. Horizontal axis is the normalised argument of cbrt which corresponds to each pair in __t_cbrt.

## 3 The Importance of Automated Parameter Tuning

Section 1 has briefly covered the existing literature. It makes clear that, apart from our own recent work [5], the problem of automatic update of values embedded in existing software has been little studied. By page 2 it showed that the cost of software maintenance is staggering, yet there is little research on automatically adjusting software parameters, not exposed to the user for modification.

Currently the task of keeping constants embedded in existing software up-to-date is labour-intensive and so there is great scope for automation.

Even parameters given by scientific measurement can be subject to change in just a few years [5]. Andronescu et al. [17] had tried to update parameters in RNAfold using constraint optimization. Nevertheless, our GI did better [5]. Section 2 expands this to the related task of creating new system software from existing functions via automated parameter tuning. In Section 2 we use CMA-ES to automatically adapt 1024 float constants, giving rise to cbrt, which does not currently exist in the C run time library. In addition to $\sqrt[3]{x}$, this framework could be readily adapted to provide new maths double functions [18] where there is an objective function, e.g. the inverse operation. It could also be used to port existing functions to different hardware.

Previously [5] we have demonstrated using SBSE to adapt 50 000 parameters to new scientific knowledge may be possible. Section 2 showed in less than five minutes it can adapt more than a thousand continuous values. We have used extensive testing to show the correctness of the automatically transplanted data. Additionally, e.g. following [15], it may be feasible to verify our GI cbrt.

These very early experiments hint, in a world addicted to software, both automated data maintenance and data transplantation could be vital new areas for search based software engineering.

**Code** see http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gi_cbrt.tar.gz

**Acknowledgements** My thanks to our EuroGP [5] anonymous reviewers.

## References

1. Langdon, W.B., Petke, J.: Software is not fragile. In Parrend, P., et al., eds.: CS-DC'15. Proceedings in Complexity, Springer (2015) 203–211 Invited talk.
2. Martin, R.J., Osborne, W.M.: Guidance on software maintenance. NBS Special Publication 500-106, National Bureau of Standards, USA (1983)
3. Cao, L., Sihler, H., Platt, U., Gutheil, E.: Numerical analysis of the chemical kinetic mechanisms of ozone depletion and halogen release in the polar troposphere. Atmospheric Chemistry and Physics **14**(7) (2014) 3771–3787
4. Lorenz, R., Bernhart, S.H., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P.F., Hofacker, I.L.: ViennaRNA package 2.0. AMB **6**(1) (2011)
5. 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
6. Marounek, P.: Simplified approach to effort estimation in software maintenance. Journal of Systems Integration **3**(3) (2012)
7. Dehaghani, S.M.H., Hajrahimi, N.: Which factors affect software projects maintenance cost more? Acta Informatica Medica **21**(1) (2013) 63–66
8. Mohan, M., Greer, D.: A survey of search-based refactoring for software maintenance. Journal of Software Engineering Research and Development **6**(1) (2018)
9. de Freitas, F.G., de Souza, J.T.: Ten years of search based software engineering: A bibliometric analysis. In Cohen, M.B., O Cinneide, M., eds.: SSBSE 2011 18–32
10. Butler, S.: Analysing Java Identifier Names. PhD thesis, Open University, UK
11. Tiella, R., Ceccato, M.: Automatic generation of opaque constants based on the K-clique problem for resilient data obfuscation. In: SANER 2017 182–192
12. 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
13. 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. LNCS 9275, Bergamo, Springer (2015) 262–268
14. Langdon, W.B., Harman, M.: Grow and graft a better CUDA pknotsRG for RNA pseudoknot free energy calculation. In Langdon, W.B., et al., eds.: Genetic Improvement 2015 Workshop, Madrid, ACM (2015) 805–810
15. Markstein, P.W.: Computation of elementary functions on the IBM RISC System/6000 processor. IBM J Res. Dev. **34**(1) (1990) 111–119
16. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2) (2001) 159–195
17. Andronescu, M., Condon, A., Hoos, H.H., Mathews, D.H., Murphy, K.P.: Efficient parameter estimation for RNA secondary structure prediction. Bioinformatics **23**(13) (2007) i19–i28
18. Langdon, W.B.: Evolving square root into binary logarithm. Technical Report RN/18/05, University College, London, London, UK (2018)