Genetic Improvement of Data gives double precision invsqrt

W. B. Langdon
Department of Computer Science, University College London Gower Street, WC1E 6BT, UK
w.langdon@cs.ucl.ac.uk

ABSTRACT

CMA-ES plus manual code changes rapidly transforms 512 Newton-Raphson start points from a GNU C library table driven version of sqrt into a double precision reciprocal square root function. The GI $x^{-\frac{1}{2}}$ is far more accurate than Quake's InvSqrt, Quare root.

CCS CONCEPTS

• **Software and its engineering → Search-based software engineering**;

KEYWORDS

evolutionary computing, Evolution Strategies, GGGP, search based software engineering, SBSE, software maintenance of literals, data transplantation, glibc, Newton's method

ACM Reference Format:
W. B. Langdon. 2019. Genetic Improvement of Data gives double precision invsqrt. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3319619.3326800

1 QUAKE

Quake was a first person shooter video game from the end of the second millennium. It was designed to run on single user consumer electronics, such as personal computers and home video game consoles. For our purposes, it is noteworthy because it made extensive use of the reciprocal square root function ($x^{-\frac{1}{2}}$) in computer illumination calculations (see Figure 1) at a time when lack of hardware support made $x^{-\frac{1}{2}}$ expensive to compute. To produce high quality video displays many such calculation are needed. And yet there is very little time for calculation if the software is to achieve satisfactory interactive real-time response and refresh the user's video display at an acceptable rate. Quake solved this computational bottle neck by using a fast approximation to the reciprocal square root[1]. Today graphics cards (GPUs) provide hardware support for $x^{-\frac{1}{2}}$ specifically for interactive video games like Quake.

The GNU C library provides many maths functions, including sqrt. The current version, glibc 2.29, does not include invsqrt ($x^{-\frac{1}{2}}$).

[1] https://en.wikipedia.org/wiki/Fast_inverse_square_root

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326800

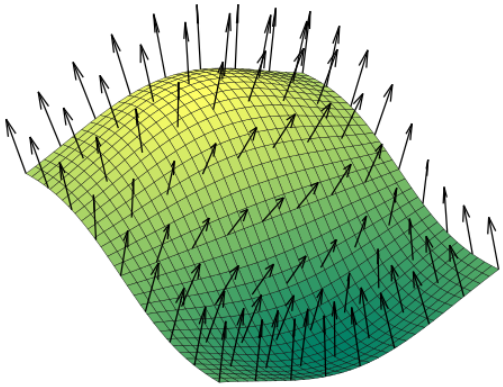
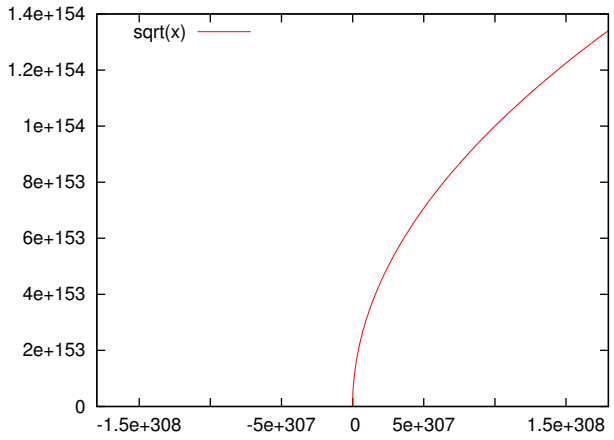


Figure 1: In 3D computer graphics the function $x^{-\frac{1}{2}}$ may be used many times to normalise vectors at right angles to surfaces. Such surface normal vectors are used in light and shade illumination calculations. https://commons.wikimedia.org/wiki/File:Surface_normals.svg

Although it does include sqrt and invsqrt can be readily calculated by taking the reciprocal $1/x$. Nonetheless even on powerful modern processors typically division is more expensive than multiplication. Not all processors provide hardware support for sqrt. Although hardware support for division is common, it need not be universal and may be provided by expensive software emulation. As computing becomes ubiquitous, with the internet-of-things and in particular mote computing, there will be a demand to run increasingly sophisticated algorithms (such as computer vision and machine learning) which may require vector normalisation on non-standard hardware lacking support for some maths functions and basic facilities we usually take for granted (such as a power supply). Hence there may be a demand for non-conventional maths implementations possibly providing unconventional trade-offs with energy consumption [18] or accuracy [25, 29] and a software based solution may be preferred to hardware circuits, such as [7].

Starting from an open source table driven glibc sqrt C source code specifically written for a processor lacking hardware support for sqrt, we have shown evolution (plus manual changes) can create double precision implementations for the cube root ($\sqrt[3]{x}$) [15] and the logarithm to the base 2 ($\log_2 x$) [16]. We have claimed that the frame work is somewhat general and here demonstrate it further by evolving a table driven double precision implementation of the inverted square root function.

The next section summarises using genetic improvement (GI) [27][1][12][8][9][11][13][22][10][21][14][26] to find better software by updating numeric values within it (rather than modifying the



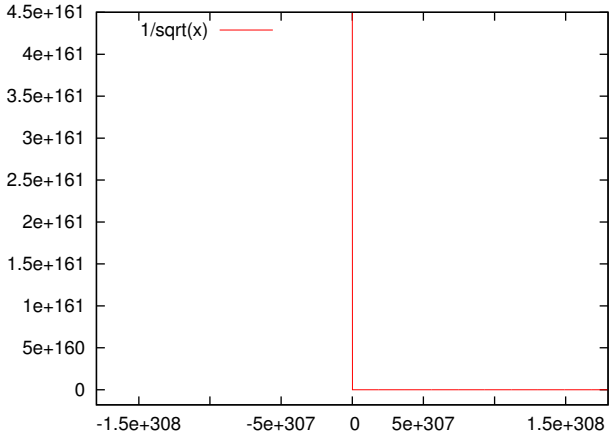


Figure 2: Left: Double precision square root Right: Double precision inverted square root

code itself). Section 3 explains how an existing open source C code calculates sqrt $\sqrt{x}$ and the following section (4) explains how to convert it to $1/\sqrt{x}$ using manual changes (Section 4.2) and evolution (4.3). We finish with a comparison with Quake (Section 5) and discussion (Section 6) before (Section 7) concluding that evolutionary computation in the form of CMA-ES [5] can rapidly produce a double precision implementation of the desired reciprocal square root function.

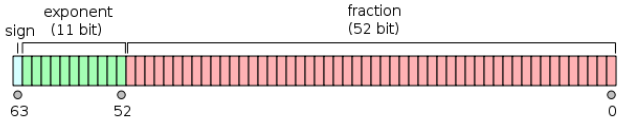


Figure 3: IEEE 754 Double-precision floating-point format (Wikipedia).

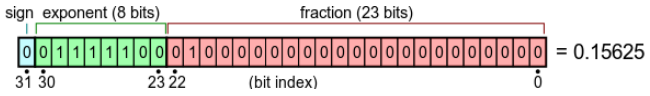


Figure 4: Example of IEEE 754 single precision floating-point format (Wikipedia).

2 GI FOR NEW FUNCTIONALITY

There is just a tiny bit of work on using genetic improvement (GI) to create new functionality. In [17] we used custom mutation and crossover operators to evolve compile time constants within C source code. Evolution took about five days to find a new program which on thousands of real examples gave predictions which were on average 11% more accurate. (Although some were worse, most were unchanged or better.) Notice that there were no changes to the code. Only data were changed. The new data have been distributed with the ViennaRNA package since Version 2.4.6 (released 19 April 2018). (See also Section 6.)

The GNU C library contains more than a million constants. One file of particular interest holds a table driven implementation of the double precision square root function, sqrt (see Figure 2). This specifically targets a processor, an IBM powerPC, without hardware support for sqrt. In [15] we showed that standard CMA-ES with default parameters, could evolve in a few seconds several hundred compile time parameters to give the cube root function, cbrt, which does not exist in the GNU C library. Although manual changes to the code were required, they were (as here Section 4.2) primarily to deal with the exponent part of double precision numbers (green in Figure 3). At the time we also claimed that the approach was general and could be widely applied to continuous maths functions. Since we have also shown evolution can produce a double precision implementation of $\log_2$ [16].

3 EXISTING NEWTON-RAPHSON CALCULATION OF SQRT

The current version of the GNU C library, glibc-2.29, was downloaded[2]. The table driven powerPC version of sqrt.c uses the format of double precision numbers (Figure 3). To find the square root of the exponent part of x, it uses a shift left operation to divide the exponent by two. Secondly it treats the least significant bit of the exponent plus the fractional part as the normalised version of x lying between 0.5 and 2.0.

It uses the top nine bits of the normalised number as an index into a table of 512 pairs of floating point numbers. The first of each pair is used as a start point for a fixed number of iterations of the Newton-Raphson method to find a root (crossing point for y=0) of a continuous differentiable function.

Newton's method requires the repeated division by the derivative. The derivative of $\sqrt{x}$ is $\frac{1}{2}\frac{1}{\sqrt{x}}$ but since $\sqrt{x}$ is unknown, sqrt.c uses the second of each pair of entries in the table to store the first estimate of the reciprocal of the derivative. By using the reciprocal,

[2] https://ftp.gnu.org/gnu/glibc/glibc-2.29.tar.gz

the division can be replaced by a (faster) multiplication. Newton-Raphson is then also applied to the estimate of the reciprocal of the derivative as well.

Newton-Raphson converges quadratically fast in ideal circumstances. By starting with an 8 or 9 bit approximation each iteration improves the accuracy: 16, 32, and finally 64 bits. Thus for double precision (52 bits, Figure 3) only three iterations are needed. Also for speed sqrt.c does not check if that it has reached the right answer at each iteration but proceeds to do all three iteration in an unrolled loop. The final step is to restore the adjusted exponent.

4 EVOLVING INVSQRT $x^{-\frac{1}{2}}$

4.1 Newton-Raphson solves $x^{-\frac{1}{2}}$

The function whose root we want is $f(x) = \frac{1}{x^2} - \text{input}$. (I.e. what value of $x = \frac{1}{\sqrt{\text{input}}}$?) The derivative of $f(x)$ is $f'(x) = \frac{-2}{x^3}$.

The first iteration of Newton-Raphson is:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_1 = x_0 + \left(\frac{1}{x_0^2} - \text{input}\right) \times \frac{x_0^3}{2}$$

$$x_1 = x_0 + \frac{(x_0 - x_0^3 \text{input})}{2}$$

Notice this formulation means we do not need to estimate $x^{-\frac{1}{2}}$ for the derivative and so we can avoid maintaining an estimate of its reciprocal but at the cost of three multiplications.

4.2 Manual changes

As with cbrt and log2 a small number of similar changes are needed before running evolution on the data table.

- The construction of the nine bit indexing operation is essentially unchanged but must take into account the table contains 512 floats not 512 pairs of floats.
- The code to maintain the estimate of the reciprocal of the derivative can be commented out.
- The new formula (Section 4.1) for the Newton-Raphson step is used (three times).
- The exponent part of the original floating point number must not only be divided by two (as in sqrt.c) but also must be negated. Since the IEEE 754 standard uses 11 unsigned bits to represent the exponent, this is accomplished by a left shift and then subtracting from the mid point ($1023 = 2^{(11-1)} - 1$).
- Dealing with denormalised double precision numbers is accomplished as in sqrt.c except the constant 2^{-54} is replaced by 2^{54}.

4.3 CMA-ES evolves $x^{-\frac{1}{2}}$ data table

The __t_sqrt table contains 512 pairs of floats, corresponding to numbers in the range 0.5 to 2. The first of each pair was used as the start point when evolving the 512 floats in the new table (see Figures 5 and 6).

The Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES [5]) was downloaded from https://github.com/cma-es/c-cmaes/archive/master.zip CMA-ES is going to be run at least 512 times. Each time it is initialised to the data value (i.e. first of each pair) from the corresponding value in __t_sqrt. The initial mutation

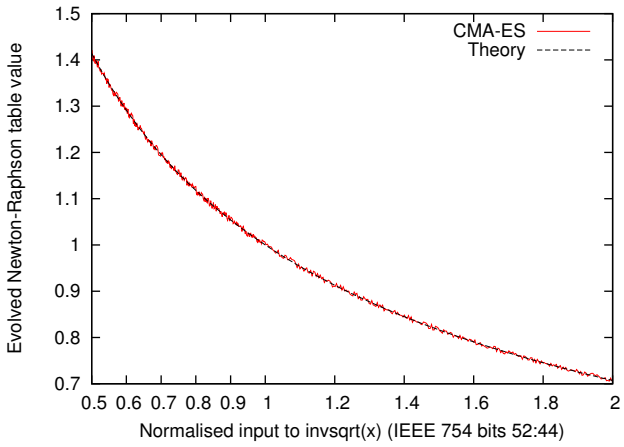


Figure 5: 512 GI table values. Starts for 3 iterations of Newton-Raphson calculation of $x^{-\frac{1}{2}}$.

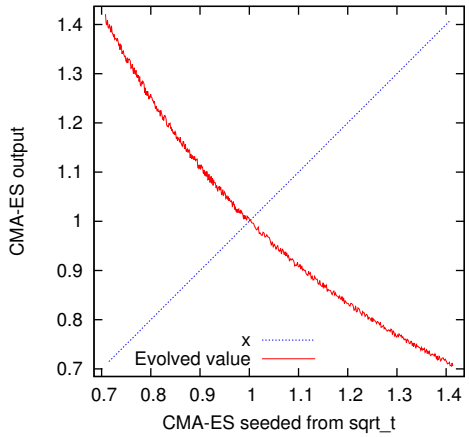


Figure 6: Evolved change from sqrt table values (horizontal axis) to corresponding invsqrt table value (vertical axis). 512 successful CMA-ES runs. (Diagonal blue line shows y=x, i.e. no change.)

step size used by CMA-ES was set to 3.0 times the standard deviation calculated from the 512 data values in __t_sqrt. (Thus all CMA-ES runs start with the same mutation step size.)

4.3.1 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.

4.3.2 Fitness function. Each time CMA-ES proposes a value (N=1), it is converted from a double into a float and loaded into the table at the location that CMA-ES is currently trying to optimise. The fitness function uses three test double values in the range 0.5 to 2.0. These are: the lowest value for the table entry, the mid point

and the top most value. The manually written code is called (using the updated table) 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 manual code, and reverses it (i.e. squares it and the takes the reciprocal) and takes the difference between this and the corresponding test value. Of course if everything is working then they are the same. If they are the same, the sub-fitness is 0, otherwise it is positive. Since when invsqrt 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.

Since all the calculations are done as double precision approximations a certain degree of rounding inaccuracy can be expected. If the difference is really small, it may be treated as zero. To decide if the difference is small enough, the reverse operation is repeated for the double slightly bigger (i.e. larger = multiplied by 1+DBL_EPSILON) and slightly smaller (i.e. smaller = divided by 1+DBL_EPSILON) to give two new differences. The original answer may be treated as close as possible to the right answer, i.e. fitness is zero, if either: 1) the original difference was zero or smaller than both the absolute difference of either smaller or larger. Or 2) the original difference is no more than 2×DBL_EPSILON and it lies between d_smaller and d_larger.

4.3.3 Avoiding Negative Values. Notice all the steps in the fitness function (previous section) do not require comparison with an existing implementation. They take a purest approach of taking the inverse function ($f^{-1}(x) = x^{-2}$) and seeing how closely $f^{-1}(y)$ resembles the initial input. But notice $f^{-1}(x)$ is not monotonic. In particular $f^{-1}(-x) = f^{-1}(x)$. Thus from a mathematical perspective if y is a solution, then so too is $-y$. In one run evolution found such negative solutions to $1/\sqrt{x}$. Although mathematically sound, programming standards would not allow negative values. Therefore the fitness function was adjusted, so that negative numbers appear to be distant from $1/\sqrt{x}$, by adding $2x$ to the fitness objective. (Remember $0.5 \le x < 2$ during testing and CMA-ES is minimising.)

4.3.4 Restart Strategy. If CMA-ES cannot find a suitable value (i.e. either zero or a really small difference on all three test points), it is restarted. That is, CMA-ES is run again with a different pseudo random seed but with the same initial starting position and mutation size. In 494 of 512 cases CMA-ES found a suitable value in one run, but in 16 cases it was run twice, and in 2 it was run three times. To run CMA-ES 532 times took six seconds in total. Figures 7 and 8 shows the effort reported by CMA-ES for each evolutionary run.

4.4 Testing the evolved invsqrt

In all cases invsqrt produced a correct double precision answer.

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 invsqrt always came within a relative error of DBL_EPSILON (i.e. 2.22 10^{-16}) of the best possible answer.

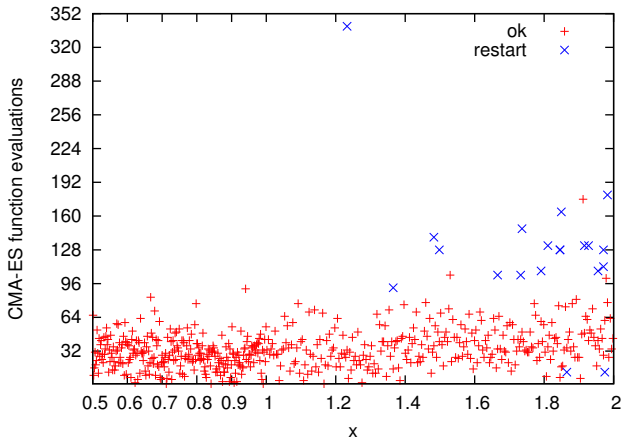


Figure 7: CMA-ES finds it very easy to evolve 512 new start points for $x^{-\frac{1}{2}}$ when starting with data for $x^{\frac{1}{2}}$. (Mean 38.8 fitness evaluation.) No strong correlation with change to initial seed value (Figure 6). However all 20 of the runs which were restarted (×) are for x>1.

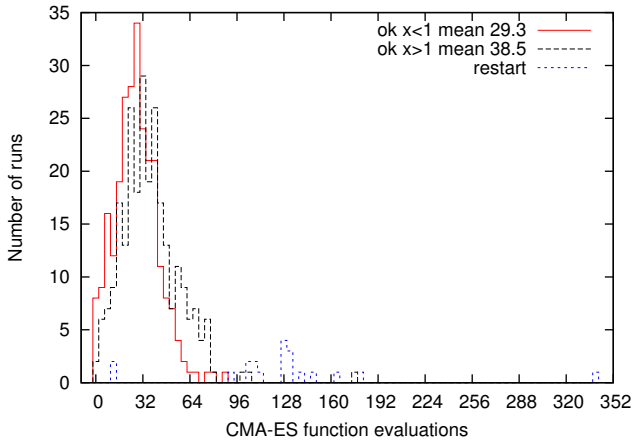


Figure 8: Histogram (bin size 4) of number fitness evaluations per run for successful runs and for 20 runs which did not find an acceptable solution. Data as Figure 7. Excluding where CMA-ES was restarted (blue dashed line), search time for $1 < x < 2$ (black dashed line) is similar to $0.5 < x < 1$ (red solid line).

As well as ad-hoc testing, and the large positive integer tests mentioned in the previous paragraph, invsqrt was tested with 5 120 random numbers uniformly distributed between 1.0 and 2.0. (The largest relative deviation was 1.5 DBL_EPSILON.) It was also tested on 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 invsqrt correctly returned NAN. Again, in

most cases invsqrt returned a double, which when squared and inverted was its input or within one bit of it. The maximum deviation from from the best possible answer was two in the fractional part, Figure 3 (i.e. a maximum relative error of DBL_EPSILON).

5 COMPARISON WITH QUAKE

The Quake (Section 1) fast inverted square root code, InvSqrt, was down loaded from stack overflow[3] and subject to the same tests as the evolved invsqrt (Section 4.4). In terms of functionality InvSqrt is far worse:

- InvSqrt gave the correct result to floating point precision in only 45 of the 16 903 tests. Often the result is up to 0.17% out. Unsurprisingly it never gave an answer correct to double precision accuracy.
- InvSqrt does not deal with negative numbers. It may return inf but there are cases where it returns an incorrect floating point number.
- Unsurprisingly InvSqrt does not deal with double precision numbers outside the range of floating point precision (cf. Figures 3 and 4). It has odd behaviour for numbers smaller than $1.5\ 10^{-37}$ and bigger than $3.3\ 10^{38}$.

6 AUTOMATED PARAMETER TUNING

There is some similarity with deep parameter tuning (also known as Deep Parameter Optimisation) [32][31] [24][30][4][3][28][2]. Wu's deep parameter tuning [31, 32] uses search based techniques to both expose and then tune values buried in existing source code. However Wu [32] deals with relatively few parameters, rather than five hundred. Also [32] seeks to tune existing functionality rather than as here to use data changes to create a new function, or to transplant existing functionality from one program to another [19][23][6].

Section 2 and the previous paragraph have briefly covered the existing literature. It makes clear that, apart from our own recent work [15–17], the problem of automatic update of values embedded in existing software has been little studied.

In these days where much of software development is performed as continuous integration (CI), it may not be clear in a CI environment where the boundary between development and bug-fixing and maintenance lies. Nonetheless the cost of software maintenance remains staggering.

Naturally most software engineering research concentrates upon source code. However as well as code, programs contain data and parameters. In the case of one well used scientific package (ViennaRNA [15]), we found 50 000 integers which represent scientific knowledge. They had been deliberately corralled to separate them in the source code so that they could be updated as knowledge of the science increases. However due to the expert manual effort involved they had been updated by hand only once in several years, during which the software has been in routine use around the world. As an initial demonstration, we were able to show search based techniques could automatically update these compile time constants [17].

[3]https://stackoverflow.com/questions/268853/is-it-possible-to-write-quakes-fast-invsqrt-function-in-c See also Q_rsqr in https://en.wikipedia.org/wiki/Fast_inverse_square_root (11 March 2019).

It is often the case that software maintenance does not happen because because it requires specialised expertise and there is no one available, or that person is already over committed. Initially that expert may be busy working on more immediate problems. But overtime the expertise may be lost. In either case an automated approach may not need to be perfect to be acceptable. Something which improves on what we have could still be useful.

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

Section 2 expands this to the related task of creating new system software from existing functions via automated parameter tuning. In Section 4 we use CMA-ES to automatically adapt 512 float constants, giving rise to $1/\mathrm{sqrt}(x)$, which does not currently exist in the C run time library. By considering $\sqrt[3]{x}$, $\log_2$, and now $1/\mathrm{sqrt}(x)$, we have shown this framework can be readily adapted to provide new maths double functions [16] where there is an objective function, e.g. the inverse operation.[4] From a practical point of view perhaps it will prove most useful for porting existing functions to different hardware, possibly lacking direct maths support.

Previously [17] we have demonstrated using Genetic Improvement to adapt 50 000 parameters to new scientific knowledge may be possible. Section 4.3.4 showed in a few seconds it can adapt several hundred continuous values. (Although previously more complicated examples took far longer.) We have used extensive testing to show the correctness of the automatically transplanted data. Additionally, e.g. following [20], it may be feasible to verify our Genetic Improvement invsqrt.

This initial work hints that, in a world addicted to software, both automated data maintenance and data transplantation could be essential new areas for search based software engineering research.

7 CONCLUSIONS

In keeping with the GGGP [6] philosophy our approach combines minimal manual code changes (Section 4.2) and search. Starting from existing open source code, in a few seconds evolution was able to find 512 values to transform part of the GNU C library into the reciprocal square root function (invsqrt or $x^{-\frac{1}{2}}$). Uses of invsqrt include computer graphics and machine learning, e.g. to normalise vectors. In all cases the GI invsqrt produced a correct double precision answer.

Acknowledgements

Funded by EPSRC grant EP/M025853/1.

Unix scripts, source code (including CMA-ES) for the earlier cube root [15] and $\log_2$ [16] examples are available via http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gi_cbrt.tar.gz

REFERENCES

[1] B. J. Alexander and M. J. Gratton. 2009. Constructing an Optimisation Phase Using Grammatical Evolution. In *2009 IEEE Congress on Evolutionary Computation*, Andy Tyrrell (Ed.). IEEE Computational Intelligence Society, IEEE Press, Trondheim, Norway, 1209–1216. https://doi.org/doi:10.1109/CEC.2009.4983083

[2] Mahmoud A. Bokhari, Bobby R. Bruce, Brad Alexander, and Markus Wagner. 2017. Deep Parameter Optimisation on Android Smartphones for Energy Minimisation

[4]It may be that the framework is sufficiently robust, that the task of evolving another function e.g. $\sqrt[3]{x}$, (possibly without the requirement to normalise the double precision numbers) might be posed as an exercise for talented students.

- A Tale of Woe and a Proof-of-Concept. In *GI-2017*, Justyna Petke, David R. White, W. B. Langdon, and Westley Weimer (Eds.). ACM, Berlin, 1501–1508. https://doi.org/doi:10.1145/3067695.3082519

[3] Bobby R. Bruce. 2018. *The Blind Software Engineer: Improving the Non-Functional Properties of Software by Means of Genetic Improvement.* Ph.D. Dissertation. Computer Science, University College, London, UK. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/bruce_bobby_r_thesis.pdf

[4] Bobby R. Bruce, Jonathan M. Aitken, and Justyna Petke. 2016. Deep Parameter Optimisation for Face Detection Using the Viola-Jones Algorithm in OpenCV. In *Proceedings of the 8th International Symposium on Search Based Software Engineering, SSBSE 2016 (LNCS)*, Federica Sarro and Kalyanmoy Deb (Eds.), Vol. 9962. Springer, Raleigh, North Carolina, USA, 238–243. https://doi.org/doi:10.1007/978-3-319-47106-8_18

[5] 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

[6] Mark Harman, Yue Jia, and William B. Langdon. 2014. Babel Pidgin: SBSE Can Grow and Graft Entirely New Functionality into a Real World System. In *Proceedings of the 6th International Symposium, on Search-Based Software Engineering, SSBSE 2014 (LNCS)*, Claire Le Goues and Shin Yoo (Eds.), Vol. 8636. Springer, Fortaleza, Brazil, 247–252. https://doi.org/doi:10.1007/978-3-319-09940-8_20 Winner SSBSE 2014 Challange Track.

[7] John R. Koza, Forrest H Bennett III, Jason Lohn, Frank Dunlap, Martin A. Keane, and David Andre. 1997. Automated Synthesis of Computational Circuits Using Genetic Programming. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*. IEEE Press, Indianapolis, 447–452. https://doi.org/doi:10.1109/ICEC.1997.592353

[8] W. B. Langdon. 2012. Genetic Improvement of Programs. In *18th International Conference on Soft Computing, MENDEL 2012* (2nd ed.), Radomil Matousek (Ed.). Brno University of Technology, Brno, Czech Republic. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Langdon_2012_mendel.pdf Invited keynote.

[9] William B. Langdon. 2014. Genetic Improvement of Programs. In *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2014)*, Franz Winkler, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen Watt, and Daniela Zaharie (Eds.). IEEE, Timisoara, 14–19. https://doi.org/doi:10.1109/SYNASC.2014.10 Keynote.

[10] W. B. Langdon. 2015. Genetic Improvement of Software for Multiple Objectives. In *SSBSE (LNCS)*, Yvan Labiche and Marcio Barros (Eds.), Vol. 9275. Springer, Bergamo, Italy, 12–28. https://doi.org/doi:10.1007/978-3-319-22183-0_2 Invited keynote.

[11] William B. Langdon. 2015. Genetically Improved Software. In *Handbook of Genetic Programming Applications*, Amir H. Gandomi, Amir H. Alavi, and Conor Ryan (Eds.). Springer, Chapter 8, 181–220. https://doi.org/doi:10.1007/978-3-319-20883-1_8

[12] W. B. Langdon and M. Harman. 2010. Evolving a CUDA Kernel from an nVidia Template. In *2010 IEEE World Congress on Computational Intelligence*, Pilar Sobrevilla (Ed.). IEEE, Barcelona, 2376–2383. https://doi.org/doi:10.1109/CEC.2010.5585922

[13] 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

[14] William B. Langdon, Brian Yee Hong Lam, Marc Modat, Justyna Petke, and Mark Harman. 2017. Genetic Improvement of GPU Software. *Genetic Programming and Evolvable Machines* 18, 1 (March 2017), 5–44. https://doi.org/doi:10.1007/s10710-016-9273-9

[15] 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

[16] W. B. Langdon and Justyna Petke. 2019. Genetic Improvement of Data gives Binary Logarithm from sqrt. In *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Manuel Lopez-Ibanez et al. (Eds.). ACM, Prague, Czech Republic. https://doi.org/doi:10.1145/3319619.3321954

[17] William B. Langdon, Justyna Petke, and Ronny Lorenz. 2018. Evolving better RNAfold structure prediction. In *EuroGP 2018: Proceedings of the 21st European Conference on Genetic Programming (LNCS)*, Mauro Castelli, Lukas Sekanina, and Mengjie Zhang (Eds.), Vol. 10781. Springer Verlag, Parma, Italy, 220–236. https://doi.org/doi:10.1007/978-3-319-77553-1_14

[18] Jie Lu, Hongyang Jia, Naveen Verma, and Niraj K. Jha. 2018. Genetic Programming for Energy-Efficient and Energy-Scalable Approximate Feature Computation in Embedded Inference Systems. *IEEE Trans. Comput.* 67, 2 (Feb. 2018), 222–236. https://doi.org/doi:10.1109/TC.2017.2738642

[19] Alexandru Marginean, Earl T. Barr, Mark Harman, and Yue Jia. 2015. Automated Transplantation of Call Graph and Layout Features into Kate. In *SSBSE (LNCS)*, Yvan Labiche and Marcio Barros (Eds.), Vol. 9275. Springer, Bergamo, Italy, 262–268. https://doi.org/doi:10.1007/978-3-319-22183-0_21

[20] P. W. Markstein. 1990. Computation of elementary functions on the IBM RISC System/6000 processor. *IBM Journal of Research and Development* 34, 1 (Jan 1990), 111–119. https://doi.org/doi:10.1147/rd.341.0111

[21] Justyna Petke. 2017. Preface to the Special Issue on Genetic Improvement. *Genetic Programming and Evolvable Machines* 18, 1 (March 2017), 3–4. https://doi.org/doi:10.1007/s10710-016-9280-x Editorial Note.

[22] 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

[23] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. 2014. Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class. In *17th European Conference on Genetic Programming (LNCS)*, Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo Garcia-Sanchez, Juan J. Merelo, Victor M. Rivas Santos, and Kevin Sim (Eds.), Vol. 8599. Springer, Granada, Spain, 137–149. https://doi.org/doi:10.1007/978-3-662-44303-3_12

[24] Jeongju Sohn, Seongmin Lee, and Shin Yoo. 2016. Amortised Deep Parameter Optimisation of GPGPU Work Group Size for OpenCV. In *Proceedings of the 8th International Symposium on Search Based Software Engineering, SSBSE 2016 (LNCS)*, Federica Sarro and Kalyanmoy Deb (Eds.), Vol. 9962. Springer, Raleigh, North Carolina, USA, 211–217. https://doi.org/doi:10.1007/978-3-319-47106-8_14

[25] Zdenek Vasicek and Vojtech Mrazek. 2017. Trading between quality and non-functional properties of median filter in embedded systems. *Genetic Programming and Evolvable Machines* 18, 1 (March 2017), 45–82. https://doi.org/doi:10.1007/s10710-016-9275-7

[26] David R. White. 2017. GI in No Time. In *GI-2017*, Justyna Petke, David R. White, W. B. Langdon, and Westley Weimer (Eds.). ACM, Berlin, 1549–1550. https://doi.org/doi:10.1145/3067695.3082515

[27] David R. White, Andrea Arcuri, and John A. Clark. 2011. Evolutionary Improvement of Programs. *IEEE Transactions on Evolutionary Computation* 15, 4 (Aug. 2011), 515–538. https://doi.org/doi:10.1109/TEVC.2010.2083669

[28] David R. White, Leonid Joffe, Edward Bowles, and Jerry Swan. 2017. Deep Parameter Tuning of Concurrent Divide and Conquer Algorithms in Akka. In *20th European Conference on the Applications of Evolutionary Computation (Lecture Notes in Computer Science)*, Giovanni Squillero and Kevin Sim (Eds.), Vol. 10200. Springer, Amsterdam, 35–48. https://doi.org/doi:10.1007/978-3-319-55792-2_3

[29] Michal Wiglasz and Lukas Sekanina. 2018. Cooperative Coevolutionary Approximation in HOG-based Human Detection Embedded System. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. Bangalore, India, 1313–1320. https://doi.org/doi:10.1109/SSCI.2018.8628910

[30] John R. Woodward, Colin G. Johnson, and Alexander E. I. Brownlee. 2016. Connecting Automatic Parameter Tuning, Genetic Programming as a Hyper-heuristic, and Genetic Improvement Programming. In *GECCO '16 Companion: Proceedings of the Companion Publication of the 2016 Annual Conference on Genetic and Evolutionary Computation*. ACM, Denver, Colorado, USA, 1357–1358. https://doi.org/doi:10.1145/2908961.2931728

[31] Fan Wu. 2017. *Mutation-Based Genetic Improvement of Software.* Ph.D. Dissertation. Department of Computer Science, University College, London, UK. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Thesis_Fan_v2.1.pdf

[32] Fan Wu, Westley Weimer, Mark Harman, Yue Jia, and Jens Krinke. 2015. Deep Parameter Optimisation. In *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, Sara Silva, Anna I Esparcia-Alcazar, Manuel Lopez-Ibanez, Sanaz Mostaghim, Jon Timmis, Christine Zarges, Luis Correia, Terence Soule, Mario Giacobini, Ryan Urbanowicz, Youhei Akimoto, Tobias Glasmachers, Francisco Fernandez de Vega, Amy Hoover, Pedro Larranaga, Marta Soto, Carlos Cotta, Francisco B. Pereira, Julia Handl, Jan Koutnik, Antonio Gaspar-Cunha, Heike Trautmann, Jean-Baptiste Mouret, Sebastian Risi, Ernesto Costa, Oliver Schuetze, Krzysztof Krawiec, Alberto Moraglio, Julian F. Miller, Pawel Widera, Stefano Cagnoni, JJ Merelo, Emma Hart, Leonardo Trujillo, Marouane Kessentini, Gabriela Ochoa, Francisco Chicano, and Carola Doerr (Eds.). ACM, Madrid, 1375–1382. https://doi.org/doi:10.1145/2739480.2754648