ABSTRACT

Genetic Improvement (GI) can be used to give better quality software and to create new functionality.

We show that GI can evolve the PowerPC open source GNU C runtime library square root function into cube root, binary logarithm log2 and reciprocal square root. The GI cbrt is competitive in run-time performance and our inverted square root $x^{-\frac{1}{2}}$ is far more accurate than the approximation used in the Quake video game. We use CMA-ES to adapt constants in a Newton-Raphson table, originally from glibc’s sqrt, for other double precision mathematics functions. Such automatically customised math libraries might be used for mobile or low resource, IoT, mote, smart dust, bespoke cyber-physical systems.

Evolutionary Computing (EC) can be used to not only adapt source code but also data, such as numerical constants, and could enable a new way to conduct software data maintenance. This is an exciting opportunity for the GECCO and optimisation communities.

CCS CONCEPTS

- Software and its engineering → Search-based software engineering.

KEYWORDS

evolutionary computing, software engineering, search based software engineering, SBSE, computer-aided software engineering, CASE, autorepair, software maintenance of empirical constants, data transplantation, glibc, vector normalisation, Newton’s method

ACM Reference Format:


1 INTRODUCTION

We hope to alert the optimisation community to a new opportunity to apply their existing tools and techniques to an exciting and important potential application: optimising and maintaining existing software.

“Genetic Improvement of Data for Maths Functions” [8] is the first journal publication in which we optimise numbers in source code to generate new functionality. These mathematical functions use a table containing start values for the Newton-Raphson method. Newton’s method is often used to reduce runtime of complex arithmetic operations. We evolve values in an existing table to give new functions. This method is not just for mathematical functions: we have also used genetic improvement (GI) to improve the answers given by a program by tuning 50,000 parameters buried in its source code [6]. We used CMA-ES [3] on 1024 floating point numbers within glibc’s implementation of square root for the PowerPC and manually adapted the function and derivative to give a C implementation of cube root. Not only does the new $\sqrt[3]{x}$ code give double precision accuracy but, subsequently we were able to show, it is competitive with standard implementations for C++ and indeed it is faster than Java [4]. The journal article also extends this to other double precision mathematical functions (see Table 1).

Glibc is the GNU C runtime library. It is a key component of the Free Software Movement’s support for the widely used C programming language and core to GNU and GNU/Linux systems. Note glibc does not support cube root.

Table 1: GI accuracy [8, 9] and total time time for CMA-ES

<table>
<thead>
<tr>
<th>Start</th>
<th>Evolved</th>
<th>accuracy</th>
<th>seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt</td>
<td>cbrt()</td>
<td>$\sqrt[3]{x}$</td>
<td>double precision, i.e. $\leq 6.7 \times 10^{-16}$</td>
</tr>
<tr>
<td>sqrt</td>
<td>log2()</td>
<td>$\log_2 x$</td>
<td>double precision, i.e. $\leq 2.2 \times 10^{-16}$</td>
</tr>
<tr>
<td>sqrt</td>
<td>invsqrt()</td>
<td>$x^{-1/2}$</td>
<td>double precision, i.e. $\leq 2.2 \times 10^{-16}$</td>
</tr>
<tr>
<td>sqrt</td>
<td>reciproc()</td>
<td>$x^{-1}$</td>
<td>double precision, i.e. $\leq 2.2 \times 10^{-16}$</td>
</tr>
</tbody>
</table>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

GECCO ’21 Companion, July 10–14, 2021, Lille, France
© 2021 Copyright held by the owner/author(s).
https://doi.org/10.1145/3449726.3462730
Almost all software contains numbers, e.g. the glibc source code contains more than a million integers. These tend to be set when the code is written. Many are indeed fixed, but some of them represent tunable parameters whose values are often chosen by the initial software developer before first contact with the users, and never subsequently updated. The Evolutionary Computation (EC) community has the opportunity to develop new tools for automatic update of source code parameters. Uses include: 1) first initial tuning, and also returning in response to 2) increased user load, 3) changing user expectations or user behaviour and 4) to support new hardware. (E.g. increased/decreased RAM, GPU, multi-core, mobile computing, internet-of-things (IoT) and smart dust or mote computing.)

2 SQUARE ROOT ⇒ CUBE ROOT VIA DATA

If there is no hardware support for square root, glibc uses the Newton-Raphson method to iteratively solve $x^2 - a = 0$ (i.e. to find $x = a^{1/2}$) [13]. To get double precision accuracy quickly, Newton-Raphson needs to start with a good approximation of $\sqrt{a}$. To achieve this, glibc divides the double precision number range into 512 bins (Figure 2) and stores good start values for Newton-Raphson in each.

To convert the open source glibc C code to support cube root, we need a little manual code manipulation, e.g. to remove the trap for negative values of $a$ and to divide the exponent (11 green bits in Figure 2) by 3 rather than 2. However our article shows it is easy to use evolutionary computation to mutate the “fixed” sqrt numbers into numbers for the new cube root function. The evolved $\sqrt[3]{a}$ function has been tested many thousands of times to show it does indeed give the true double precision cube root.

The CMA-ES fitness function (Figure 1) uses the values in the table to try and calculate the cube root. It cubes the calculated value. This is compared to the original number and the absolute difference becomes the fitness value guiding CMA-ES. (See the journal paper [8] for details.)

3 CONCLUSIONS

Genetic Improvement (GI) [15] has been widely used to automatically fix bugs [12], speed up programs [7], reduce energy consumption [2] and transplant code [14] [1]. In our ACM TELO paper [8], we show GI can be applied, not just to code, but also to data. Giving functionally improved software primarily by evolving floating point numbers in its source code.

The method gives in all cases correct double precision answers. It was able to rapidly evolve hundreds of C source code constants. (See Table 1.)

In a world addicted to software, software maintenance is a huge cost. Most research concentrates on maintaining source code and, as yet, there is little research on maintaining numbers within programs. Therefore both automated data maintenance and data transplantation could be vital new areas for optimisation research.

Acknowledgements

We are grateful for the assistance of Justyna Petke and Roy Longbottom and thank Simon Tatham for PuTTY. Funded by EPSRC GGGP and InfoTestSS grants EP/M025853/1 EP/P005888/1.

Code. Our code is available via https://github.com/oliver-krauss/Replication_GI_Division_Free_Division or http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/gi_cbrt.tar.gz

REFERENCES


