UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note

RN/18/06

Genetic Improvement GISMOE Blue Software Tool Demo

22 September 2018

W. B. Langdon

Abstract

The GISMO Backus-Naur Form BNF grammar based Genetic Improvement (GI) system demonstration shows artificial evolutionary computation reducing run time of blue.cpp, a noddy C program to count how many blue pixels there are in an image.
http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/opencv_gp.tar.gz

Keywords genetic algorithms, genetic programming, genetic improvement, search based software engineering, SBSE

7 990 272 pixels

2 760 641 blue pixels

Figure 1: blue.cpp simply counts the number of blue pixels in the image. The demo shows using BNF grammar based genetic improvement to reduce elapsed run time. blue.cpp is deliberately inefficient.

1 Evolving Code

Genetic programming [1; 2; 3] has recently been applied to existing human written code rather than starting from primordial ooze. The use of search based engineering techniques [4], principally genetic programming, to existing software is known as genetic improvement [5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17]. GI has been used to automatically fix coding bugs [18; 19]. Whilst still an active research topic (e.g. there were multiple technical sessions on Software Repair at 40^{th} International Conference on Automatic Software Engineering, ICSE-2018) automatic program repair (APR) is now in industrially use[1]. As well as bug fixing, GI has been used to improve the performance of programs, often by reducing run time [20; 21; 22] or multi-objective run time tradeoffs [23; 24; 25; 26; 27] but also better energy consumption [28; 12; 29], particularly for mobile computing (e.g. smart phone) applications [30; 31], improving memory efficiency [32] and indeed porting to new hardware, typically for auto-parallelisation or improved parallel performance [33; 34; 35; 36; 37; 38; 39; 40; 41]. As in many cases, we will be concerned with applying artificial evolution to human written source code, but GI has also been applied to assembler [42], byte code [43] and indeed machine code [44; 45]. Mostly, so far, GI has been applied to C/C++ or Java but [46] evolves Python and [47] Scala.

Almost all genetic improvement work so far has been empirical, however there show been some theoretical investigations: [48; 49; 50].

Essentially the GISMOE system represents the program source code as a sequence of lines of code. The idea being programmers care about the layout of their code. In particular a "line of source code" has a meaning to them. Perhaps there is something in "line of code" which will apply to automated changes to the program? GISMO represents the source code as a sequence of BNF grammar rules. Note it is a very specific grammar. It represents just the program. Not, for example, every program that could be written in C. GISMOE defines several ways to make changes (i.e. mutate) the grammar. Each evolved individual is simply an ordered variable length list of such mutations. There are also crossover operations for combining (parts of) variable lists from different individuals to create children with (some of) the characteristics of their (two) parents.

The GISMOE chromosomes are interpreted in left right order and applied to the original grammar. The new grammar is then expanded into source code to give a child program, which we attempt to compile and run (Figure 2). Notice we work on text files, and so (in principle) the program can be written in any languages, whereas other tools (e.g. GIN [51] and PyGGI [52]) are specific to a given programming language (Java and Python, respectively).

[1] In 2018 Facebook announce that they were routinely using their SapFix tool on their own code.

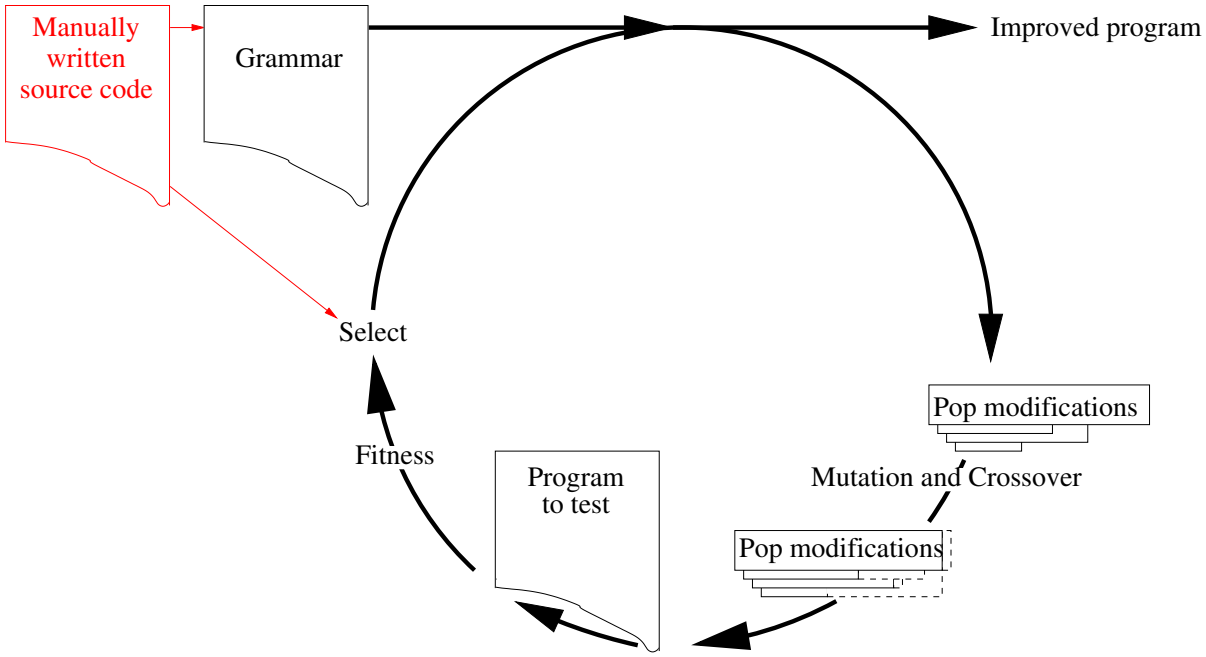


Figure 2: Evolutionary Computation cycle adapted for Genetic Improvement (GI) of manually written C code (left). The grammar tries to ensure many mutants compile, run, and terminate. For blue, fitness is given by comparing with the original code's answer on a random image (see Figure 3) and run time.

The GISMO framework has been repeatedly used [21; 53; 54; 55; 56; 57; 34; 35; 38; 39; 37; 58; 38; 40; 59; 60]. The plan was to use the most recent version which did not require specialise hardware, i.e. not CUDA or SSE instructions. This was optimising OpenCV [58]. However OpenCV is huge and installing it is a task in its own right. Hence a noddy graphics program was devised, which does not require special hardware. This is blue.cpp, which counts blue pixels in an image. To keep it simple almost all OpenCV dependency and input and output were striped out.

In Figure 1 one of the blue sky images used to validate the GI version of OpenCV's SEEDS image segmentation algorithm [58] is used to demonstrate blue.cpp counting blue pixels. Non-blue pixels are replaced by black pixels.

The following sections run trough installing the system under Linux and an example of running it. After the references, some common error messages and how to put them right are given Appendix A.

2 Installing

GISMOE does not need system privileges. It can be installed and run with a standard Unix user account.

First, download and unpack the compressed tar ball: http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/opencv_gp.tar.gz. It is perhaps best to use a new empty directory. (To reduce measurement noise, for blue.cpp, you might want to put GISMOE on a local disk with minimal non-GISMOE load.)

```
gunzip -c opencv_gp.tar.gz | tar xvf
```

opencv_gp.tar refers to the whole of the OpenCV experiment [58] however part of the README.txt file refers specifically to blue.cpp

opencv_gp.tar contains the code used to create the BNF grammar but, for simplicity, we will just use the grammar, blue.bnf (see tcsh and gawk scripts in sub-directory create_bnf for details of how to create the BNF file).

3 Running Evolution

The GISMO framework is essentially a collection of Linux tcsh and gawk scripts. It defines a directory hierarchy. It is best to stick with this tree structure. Changing it will probably mean changes to many parts of the framework.

Most of the scripts lie at the top of the directory tree. The sources subdirectory contains files used during evolution but not modified by it. On the other hand, the sources_xxx subdirectory is updated during each evolutionary run. You might like to use a new "xxx" for each run. Although it is possible to run multiple GISMOE times in parallel by giving them each their own "xxx", in the case of blue.cpp, parallel runs may interfere with accurate measurement of run time.

4 RE_gp.bat

RE_gp.bat is the top level tcsh batch file. It invokes many other tcsh and gawk scripts. Its parameters are described by comments at the top of RE_gp.bat (Details vary between the different versions of GISMOE but follow the same trend. Next we describe the system specifically for blue.cpp)

Before starting create a suitable sources_xxx sub-directory. E.g.
```
mkdir sources_blue
```

Then (on a quiet system) start GISMOE. E.g.:
```
./RE_gp.bat 142605 100 10 blue 11 blue.bnf
```

These parameters are:

1. $1 seed. The fitness function includes real elapse time, since this is noisy, it is impossible to re-run evolution and get exactly the same output. Nevertheless argument $1 allows you to control all the pseudo random seeds used. It should not be zero or too big. In practice a six digit number, e.g. the current time[2] can be used.
2. $2 number of individuals in the population. The length of time evolution takes is roughly proportional to $popsize \times generations$. With modern machines, it is often possible to set $2 to a huge number. Your computer can probably cope, but it does not ensure the evolution is better than with a smaller population but will usually take longer.
   100 seems to be ok for the blue.cpp demo
3. $3 generations. Again you can experiment, but a small number of generations (i.e. ten) seems to be ok for blue.cpp (Exceeding 999 will probably cause something to fail...)
4. $4 sources_blue. Note RE_gp.bat requires the active sub-directory name to start with sources_ so you only supply the training part of the name.
5. $5 fitness repeats. Parameters five and six are specific to the blue.cpp version of GISMO. Run time is a very noisy measure. Also it is asymmetric, with many cases of long tails. To get a reliable way of comparing fitness measures, well follow Affymtrix' practice and take an average of a large number of elapse time measurements using use a robust median rather than the mean. Effectively we discard measurements from the longest half of the runs and then take the median of the shorter half.
   Increasing the number of repeats will increase the accuracy of the fitness measure. Also initially, given overhead elsewhere, increasing $5 makes little difference to how long evolution takes. However there does not seem to be a big benefit to getting increasing accuracy. For blue.cpp something in the range 10 to 100 seems to be ok.
   Although the "median of the lower half" calculation should be robust, perhaps help it by choosing an odd number for $5.
6. $6 grammar. This should be blue.bnf

[2] 142605 corresponds to about half past two in the afternoon.

RE_gp.bat starts by creating the initial random population. At the end of each generation it will display the best individual in that generation before moving on to create the next generation.

In addition to aborting RE_gp.bat in the usual ways, e.g. via the command line, evolution can be stopped by creating a file called STOP either in the top directory or the active sub-directory. (Remember to remove STOP before trying to use RE_gp.bat again.)

5 Files in the active sub-directory

It can be instructive to watch evolution taking place. Whilst your RE_gp.bat is active, in another terminal window, move into the active sub-directory and look at the contents of the current pop.nnn.fit as it is generated. E.g.

```
cd sources_blue
tail -f pop.000.fit
```

All the output from the current generation (e.g. generation 000, Table 1) is appended to each pop.nnn.fit file. This includes using blue.bnf to create each mutated version of blue.cpp and the output of the gcc C compiler (including compilation error messages). It also includes the output generated by blue.cpp when it is run. The code is instrumented to display both the number of blue pixels (see Figure 3) and the times it took to run.

Once everyone in the current generation has been evaluated, RE_gp1.bat analyses pop.nnn.fit looking for successfully run mutants. It creates a second file pop.nnn.fit2 which is formally structured, with a line for each mutant. Each line contains the mutants fitness score.

RE_gp1.bat then takes the pop.nnn.fit2 it has just generated and sorts it by fitness to give pop.nnn.fit3

Finally it takes the top half of pop.nnn.fit3 and extracts just the mutant itself (i.e. remove fitness etc.) to create pop.nnn_select

RE_gp1.bat is also responsible for showing the best of each generation onto the screen.

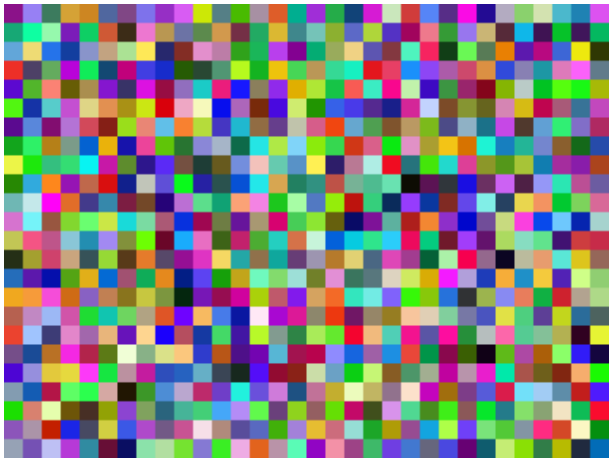
6 Fitness Function: gip_fit1.bat

The tcsh script gip_fit1.bat is invoked by RE_gp1.bat to create each blue mutant and attempt to compile it. If it compiles without errors, gip_fit1.bat attempts to run it.

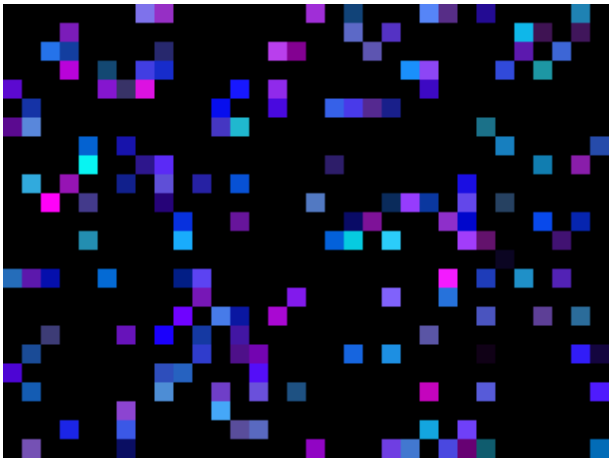
As mentioned above (page 3), in order to get a robust estimate of the mutant code's speed, the C code runs the function to count the number of blue pixels in a small fixed image (Figure 3) multiple times. The time taken by each is recorded at the maximum clock resolution (nanoseconds, i.e. 1 gigahertz precision). The code prints out both the blue pixel count and the time taken to a file (pop.nnn.fit) for analysis by RE_gp1.bat (see Section 5).

Table 1: Files in sources_blue/ Each generation five files are created. "nnn" is a three digit generation number. The initial generation is 000. Even though they are not needed after evolution proceeds to the next generation, to ease debugging and post-evolution analysis, these files are not automatically deleted. (You can of course delete them manually.)

Population	pop.nnn
Raw output	pop.nnn.fit
Extract fitness	pop.nnn.fit2
Sort by fitness	pop.nnn.fit3
Parents of next generation	pop.nnn_select



768 pixels



157 blue pixels

Figure 3: Fixed random training image. Since it will be scanned many times during fitness calculation, the training image is deliberately small.

7 RE_next.bat

Once RE_gp1.bat has evaluated the current generation, RE_gp.bat either (if we have reached the required number of generations) stops or passes RE_gp1.bat's output (i.e. pop.nnn_select) to RE_next.bat RE_next.bat creates the next generation (nnn+1) by mutating each line in pop.nnn_select and then crossing each each member with another member. In principle, this doubles the number of lines. However, if the number of lines output is less than the population size, additional individuals are created by mutation.

Note by standard genetic programming terms, the selection pressure [61] is quite low (2). If an individual is lucky enough to get into the top half of the population, that is good enough, the best in the population gets no explicit additional benefit above the average guy.

8 Result

At the end of the run, RE_gp.bat will display the best individual as a list of mutations and its fitness.

As part of its usual processing gip_fit1.bat (Section 6), compares the mutated code with the original (in sub-directory sources). In other words it converts the list of mutations into C source code and displays the changes in pop.nnn.fit Given the information printed by RE_gp.bat at the end of the run, it is straightforward to find the fitness evaluation of the best individuals in the last sources_blue/pop.nnn.fit created.

From pop.nnn.fit, it should be possible for you to check that the mutant code did indeed pass all tests and how fast it is.

8.1 Example Blue Mutant: Best in Generation Ten

```
Best generation 010 0 11 641165 SEEDS Revision:  1.4 ./GP.exe 010 88 ...24
by 32 fake_image PROG <_blue.cpp_98>+<_blue.cpp_100>
<_blue.cpp_91>x<_blue.cpp_92> <_blue.cpp_89>x<_blue.cpp_95> <_blue.cpp_92>
<_blue.cpp_113>+<_blue.cpp_98> <_blue.cpp_93>x<_blue.cpp_94>
<_blue.cpp_95>+<_blue.cpp_82> <_blue.cpp_90>x<_blue.cpp_82>
<_blue.cpp_72><_blue.cpp_80> <for1_blue.cpp_25><for1_blue.cpp_26>
<_blue.cpp_75>
```

Points to note:

- 0 (after "Best generation 010") means your run was successful, since 0 means that the blue pixel count returned by the mutant code, every time it was run, was exactly the same as the true answer.
- 11 is the number of repeats
- 641165 is the average time taken (nanoseconds, see section 6). That is, the mutant is 74% faster than the best random mutant in generation zero (1 115045).
- 010 is the generation
- 88 is the identifier of the best mutant in generation 010. You can use id 88 to locate it in pop.010.fit (see previous section).
- Everything after PROG is the actual individual. You can see it is a list of individual changes. E.g.:

  <_blue.cpp_92> means delete line 92 in file blue.cpp

  <_blue.cpp_72><_blue.cpp_80> means replace line 72 with a copy of line 80.

  <_blue.cpp_98>+<_blue.cpp_100> means insert a copy of line 100 before line 98.

  <_blue.cpp_91>x<_blue.cpp_92> means swap lines 91 and 92.

8.2 Example Blue Mutant

The expansion of the mutant from the list of instructions into their impact on the blue.cpp source code is shown in the last pop.nnn.fit file.

```
gip_fit1.bat Revision: 1.65 eden.cs.ucl.ac.uk 010 88
diff ./blue.cpp ../sources/blue.cpp
```

The best mutant in the last generation (as is typically the case) makes many changes. The unix diff utility shows all the changes made by mutation. These including deleting duplicate setting of red, green, blue arrays (i.e. removing duplicate calls to GP_R2, GP_G2 and GP_B2). Removing unnecessary code is why this mutant runs faster.

```
> GP_R2(width, height, input_image, red);
> GP_G2(width, height, input_image, green);
> GP_B2(width, height, input_image, blue);
```

Typically the evolved individual contains many changes some of which are ineffective. I.e. they may not change the source code, or even if they do change the source code, the change is syntactic only and the semantics of the change are identical to the original. You may beatify the mutant code before presenting it to the user. You might want to remove ineffective changes.

There is an optional tcsh script HC1.bat which removes the changes one at a time to help you decide which are really needed.

Evolution is good at finding weak spots. You may want to check that the code changes found by evolution are correct. For example, ensure that the mutated code will work on other images.

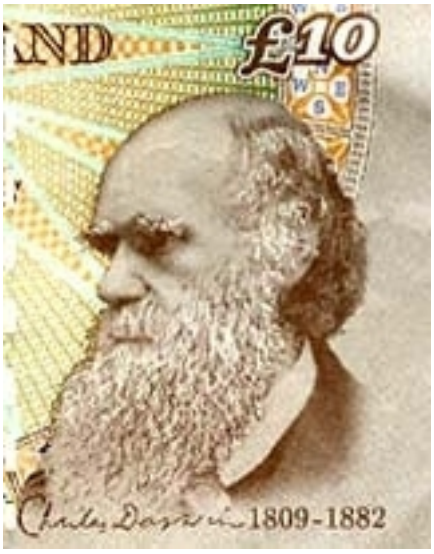


Figure 4: Charles Darwin (1809-1882) lived for a while (1838–1842) in a house on Gower Street. The site is now occupied by the UCL Biology department. Whilst CREST is in a modern tower block approximately where his garden would have been. From 2000 to 2018 his portrait was on the English ten pound note.

9 Conclusions

Based on presentation given at TAROT 2018 (14^{th} Training And Research On Testing Summer School on Software Testing, Verification & Validation) 5^{th} July 2018 in Computer Science, University College, London.

The example code is deliberately simple minded. Commonly Darwinian evolution, in the form of GISMOE, is able to optimise it.

Please report any errors to W.Langdon@cs.ucl.ac.uk

Acknowledgements

Figure 1 and 446 other benchmark images http://www.cs.ucl.ac.uk/staff/W.Langdon/ssbse2016/acgi/evostar2014 were provided by Justyna Petke [55]. David R. White installed OpenCV.

References

[1] Koza, J.R.: Genetic Programming: On the Programming of Computers by Natural Selection. MIT press (1992)

[2] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, San Francisco, CA, USA (1998)

[3] Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk` (2008) (With contributions by J. R. Koza).

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

[5] 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.

[6] Langdon, W.B.: Genetic improvement of programs. In Winkler, F., et al., eds.: 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2014), Timisoara, IEEE (2014) 14–19 Keynote.

[7] Langdon, W.B.: Genetic improvement of software for multiple objectives. In Labiche, Y., Barros, M., eds.: SSBSE. Volume 9275 of LNCS., Bergamo, Springer (2015) 12–28 Invited keynote.

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

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

[10] 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

[11] Petke, J., Le Goues, C., Forrest, S., Langdon, W.B.: Genetic improvement of software: Report from dagstuhl seminar 18052. Dagstuhl Reports **8**(1) (2018) 158–182

[12] White, D.R., Arcuri, A., Clark, J.A.: Evolutionary improvement of programs. IEEE Transactions on Evolutionary Computation **15**(4) (2011) 515–538

[13] Weise, T., Tang, K.: Evolving distributed algorithms with genetic programming. IEEE Transactions on Evolutionary Computation **16**(2) (2012) 242–265

[14] Weimer, W.: Advances in automated program repair and a call to arms. In Ruhe, G., Zhang, Y., eds.: Symposium on Search-Based Software Engineering. Volume 8084 of Lecture Notes in Computer Science., Leningrad, Springer (2013) 1–3 Invited keynote.

[15] Woodward, J.R., Petke, J., Langdon, W.: How computers are learning to make human software work more efficiently. The Conversation (2015) 10.08am BST

[16] Wu, F., Harman, M., Jia, Y., Krinke, J.: HOMI: Searching higher order mutants for software improvement. In Sarro, F., Deb, K., eds.: Proceedings of the 8th International Symposium on Search Based Software Engineering, SSBSE 2016. Volume 9962 of LNCS., Raleigh, North Carolina, USA, Springer (2016) 18–33 best paper.

[17] Ye, M., Cohen, M.B., Srisa-an, W., Wei, S.: EvoIsolator: Evolving program slices for hardware isolation based security. In Colanzi, T.E., McMinn, P., eds.: SSBSE 2018. Volume 11036 of LNCS., Montpellier, France, Springer (2018) 377–382

[18] Arcuri, A., Yao, X.: A novel co-evolutionary approach to automatic software bug fixing. In Wang, J., ed.: 2008 IEEE World Congress on Computational Intelligence, Hong Kong, IEEE Computational Intelligence Society, IEEE Press (2008) 162–168

[19] Weimer, W., Forrest, S., Le Goues, C., Nguyen, T.: Automatic program repair with evolutionary computation. Communications of the ACM **53**(5) (2010) 109–116

[20] White, D.R., Tapiador, J.M.E., Hernandez-Castro, J.C., Clark, J.A.: Fine-grained timing using genetic programming. In Esparcia-Alcazar, A.I., et al., eds.: Proceedings of the 13th EuroGP 2010. Volume 6021 of LNCS., Istanbul, Springer (2010) 325–336

[21] Langdon, W.B., Harman, M.: Optimising existing software with genetic programming. IEEE Transactions on Evolutionary Computation **19**(1) (2015) 118–135

[22] White, D.R., Joffe, L., Bowles, E., Swan, J.: Deep parameter tuning of concurrent divide and conquer algorithms in Akka. In Squillero, G., Sim, K., eds.: 20th European Conference on the Applications of Evolutionary Computation. Volume 10200 of Lecture Notes in Computer Science., Amsterdam, Springer (2017) 35–48

[23] Sitthi-amorn, P., Modly, N., Weimer, W., Lawrence, J.: Genetic programming for shader simplification. ACM Transactions on Graphics **30**(6) (2011) article:152 Proceedings of ACM SIGGRAPH Asia 2011.

[24] 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

[25] Bruce, B.R., Petke, J., Harman, M., Barr, E.T.: Approximate oracles and synergy in software energy search spaces. (IEEE Transactions on Software Engineering)

[26] Vasicek, Z., Mrazek, V.: Trading between quality and non-functional properties of median filter in embedded systems. Genetic Programming and Evolvable Machines **18**(1) (2017) 45–82

[27] Burles, N., Bowles, E., Bruce, B.R., Srivisut, K.: Specialising Guava's cache to reduce energy consumption. In Labiche, Y., Barros, M., eds.: SSBSE. Volume 9275 of LNCS., Bergamo, Springer (2015) 276–281

[28] 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

[29] 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

[30] Valencia, P., Haak, A., Cotillon, A., Jurdak, R.: Genetic programming for smart phone personalisation. Applied Soft Computing **25** (2014) 86–96

[31] Bokhari, M.A., Bruce, B.R., Alexander, B., Wagner, M.: Deep parameter optimisation on Android smartphones for energy minimisation - a tale of woe and a proof-of-concept. In Petke, J., et al., eds.: GI-2017, Berlin, ACM (2017) 1501–1508

[32] 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

[33] Langdon, W.B., Harman, M.: Evolving a CUDA kernel from an nVidia template. In Sobrevilla, P., ed.: 2010 IEEE World Congress on Computational Intelligence, Barcelona, IEEE (2010) 2376–2383

[34] Langdon, W.B., Harman, M.: Genetically improved CUDA C++ software. In Nicolau, M., et al., eds.: 17th European Conference on Genetic Programming. Volume 8599 of LNCS., Granada, Spain, Springer (2014) 87–99

[35] Langdon, W.B., Modat, M., Petke, J., Harman, M.: Improving 3D medical image registration CUDA software with genetic programming. In Igel, C., et al., eds.: GECCO '14: Proceeding of the sixteenth annual conference on genetic and evolutionary computation conference, Vancouver, BC, Canada, ACM (2014) 951–958

[36] Langdon, W.B., Lam, Brian Y.H., Petke, J., Harman, M.: Improving CUDA DNA analysis software with genetic programming. In Silva, S., et al., eds.: GECCO, Madrid, ACM (2015) 1063–1070

[37] 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

[38] Langdon, W.B., Lam, Brian Y.H., Modat, M., Petke, J., Harman, M.: Genetic improvement of GPU software. Genetic Programming and Evolvable Machines **18**(1) (2017) 5–44

[39] Langdon, W.B., Lam, Brian Y.H.: Genetically improved BarraCUDA. BioData Mining **20**(28) (2017)

[40] 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

[41] Sohn, J., Lee, S., Yoo, S.: Amortised Deep Parameter Optimisation of GPGPU work group size for OpenCV. In Sarro, F., Deb, K., eds.: Proceedings of the 8th International Symposium on Search Based Software Engineering, SSBSE 2016. Volume 9962 of LNCS., Raleigh, North Carolina, USA, Springer (2016) 211–217

[42] 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

[43] Yeboah-Antwi, K., Baudry, B.: Online genetic improvement on the java virtual machine with ECSELR. Genetic Programming and Evolvable Machines **18**(1) (2017) 83–109

[44] Schulte, E., DiLorenzo, J., Weimer, W., Forrest, S.: Automated repair of binary and assembly programs for cooperating embedded devices. In: Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems. ASPLOS 2013, Houston, Texas, USA, ACM (2013) 317–328

[45] 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.

[46] Yoo, S.: Amortised optimisation of non-functional properties in production environments. In Labiche, Y., Barros, M., eds.: SSBSE. Volume 9275 of LNCS., Bergamo, Springer (2015) 31–46

[47] Kocsis, Z.A., Swan, J.: Genetic programming + proof search = automatic improvement. Journal of Automated Reasoning **60**(2) (2018) 157–176

[48] Langdon, W.B., Harman, M.: Fitness landscape of the triangle program. In Veerapen, N., Ochoa, G., eds.: PPSN-2016 Workshop on Landscape-Aware Heuristic Search, Edinburgh (2016) Also available as UCL RN/16/05.

[49] Langdon, W.B., Veerapen, N., Ochoa, G.: Visualising the search landscape of the triangle program. In Castelli, M., et al., eds.: EuroGP 2017. Volume 10196 of LNCS., Amsterdam, Springer (2017) 96–113

[50] Veerapen, N., Ochoa, G.: Visualising the global structure of search landscapes: genetic improvement as a case study. Genetic Programming and Evolvable Machines **19**(3) (2018) 317–349 Special issue on genetic programming, Genetic improvement, Fitness landscape, Local optima network, Visualisation.

[51] White, D.R.: GI in no time. In Petke, J., et al., eds.: GI-2017, Berlin, ACM (2017) 1549–1550

[52] An, G., Kim, J., Lee, S., Yoo, S.: PyGGI: Python General framework for Genetic Improvement. In: Proceedings of Korea Software Congress. KSC 2017 (2017)

[53] Lopez-Lopez, V.R., Trujillo, L., Legrand, P.: Novelty search for software improvement of a SLAM system. In Alexander, B., et al., eds.: 5th edition of GI @ GECCO 2018, Kyoto, Japan, ACM (2018) wksp108s2

[54] Petke, J., Langdon, W.B., Harman, M.: Applying genetic improvement to MiniSAT. In Ruhe, G., Zhang, Y., eds.: Symposium on Search-Based Software Engineering. Volume 8084 of Lecture Notes in Computer Science., Leningrad, Springer (2013) 257–262 Short Papers.

[55] Petke, J., Harman, M., Langdon, W.B., Weimer, W.: Using genetic improvement and code transplants to specialise a C++ program to a problem class. In Nicolau, M., et al., eds.: 17th European Conference on Genetic Programming. Volume 8599 of LNCS., Granada, Spain, Springer (2014) 137–149

[56] Petke, J., Harman, M., Langdon, W.B., Weimer, W.: Specialising software for different downstream applications using genetic improvement and code transplantation. IEEE Transactions on Software Engineering **44**(6) (2018) 574–594

[57] Bruce, B.R.: The Blind Software Engineer: Improving the Non-Functional Properties of Software by Means of Genetic Improvement. PhD thesis, Computer Science, University College, London, UK (2018)

[58] Langdon, W.B., White, D.R., Harman, M., Jia, Y., Petke, J.: API-constrained genetic improvement. In Sarro, F., Deb, K., eds.: Proceedings of the 8th International Symposium on Search Based Software Engineering, SSBSE 2016. Volume 9962 of LNCS., Raleigh, North Carolina, USA, Springer (2016) 224–230

[59] Langdon, W.B.: Evolving better RNAfold C source code. Technical Report RN/17/08, University College, London, London, UK (2017) Also available as bioRxiv preprint 201640.

[60] 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

[61] Goldberg, D.E.: Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley (1989)

A Some Possible Errors

- Remember GISMOE assumes use of the "Tea Shell" tcsh. If you use bash, RE_gp.bat will fail.
- `mkdir: cannot create directory`
  `sources_blue: File exists`

  You can use existing directory created by tar (in which case you do not need to use mkdir to create it) or use another name, which must begin sources_ (see page 3).
- `./RE_gp.bat 142605 100 10 blue`
  `No BNF given`

  Not enough parameters for RE_gp.bat (see page 3).
- `Gen 000 ok 0 Zero!`

  Examine pop.000.fit to see why everyone in generation 000 failed.