UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note

RN/19/02

The State and Future of Genetic Improvement

25 June 2019

William B. Langdon Westley Weimer Christopher Timperley Oliver Krauss
Zhen Yu Ding Yiwei Lyu Nicolas Chausseau Eric Schulte
Shin Hwei Tan Kevin Leach Yu Huang Gabin An

When I was your age I could think of six impossible things before breakfast

Abstract

We report the discussion session at the sixth international Genetic Improvement workshop, GI-2019 @ ICSE, which was held as part of the 41^{st} ACM/IEEE International Conference on Software Engineering on Tuesday 28^{th} May 2019. Topics included GI representations, the maintainability of evolved code, automated software testing, future areas of GI research, such as co-evolution, and existing GI tools and benchmarks.

1 Introduction

The sixth Genetic Improvement workshop (GI 2019) was held as part of the forty-first International Conference on Software Engineering (ICSE 2019) in Montreal on Tuesday 28$^{\text{th}}$ May 2019. Although there is a formal proceedings [1] for peer-reviewed research, the workshop also included significant discussions of current and future topics and challenges relevant to researchers and practitioners in genetic improvement. Because the workshop included participants representing industrial and academic interests, our hope is that the ideas and patterns identified may serve as one angle on the current state of the art and this technical report will serve as recorded evidence that certain problems and topics are important.

The next section reports the discussion at the GI 2019 workshop. Section 3 lists tools that might be useful to genetic improvement researchers. Finally in the Appendix A Nicolas Chausseau suggests (post event) ways for actively continuing the GI discussion on line and suggests more biological inspirations.

2 Discussion

2.1 Underexplored Research Techniques — Evolutionary Computing

There is a need to investigate other forms of genetic operators, i.e. other forms of mutation and crossover, and indeed other forms of search. such as Tabu search and the use of anti-patterns to direct search away from unproductive areas [2].

There is a need to measure what is happening inside our GI populations and to study the course of evolutionary search within them.

How can GI explain what it has done? As an aside perhaps "explain" is being taken seriously. Perhaps we should not start aiming at a gold-plated Rolls-Royce "explanation." It appears to be adequate in some legal circumstances, rather than quoting details of statutes or common law, to simply say we did X in your case because previously in a similar case to yours we did X.

There was a discussion about taking more inspiration from Biology. Others felt simple search (e.g. random search) could be good enough, particularly if we used localisation tools to focus the search. E.g. focus just on buggy code. Concern about epistatic interactions between parts of code, crossover etc. might be more appropriate when considering full program synthesis. There was also concern for the lack of Biological knowledge amongst software engineers and the wide and ever growing range of junk search techniques [3] which claimed Nature as their sole justification.

2.2 Underexplored Research Techniques — General

To what extent can we extract beneficial non-AST information from source code (or other sources), such as comments and variable naming conventions. How can this second information channel assist code or test transplantation, code repair or improvement?

Perhaps there is a need to investigate if GI can maintain code modified by GI. This provoked a discussion of how to define "Alien" code, i.e. code generated by machines or AI. It was suggested alien code might be defined by using a group of software engineers to review the code and numerically score its comprehensibility, if the code received a low overall score then it is alien.

Not all code is long-lived and there may be a role for GI to generate small bits of short-lived ephemeral code. In particular to rapidly automatically generate a patch to a system to keep it going (cf. resilient system). This might be required for remote systems, or simply to keep a system running over a weekend until it could be "properly" manually repaired. Such *opaque* patches need not be comprehensible, whereas patches that are expected to persist in the codebase alongside code produced by (human) developers, are known as *transparent* patches [4].

To what extent can a co-evolutionary virtuous circle, such as starting with a system as its test suite, be established? Could mutation testing [5] highlight deficiencies in the test suite, followed by evolution (perhaps based on EvoSuite [6]) to generate new tests to extend the test suite and so cover the now exposed gaps in testing. Extended testing might expose bugs which automatic program repair might generate patches for. Is this one of our 6 impossible things for GI to do next?

2.3 Underexplored Application Areas

There has already been some interest in using GI to reduced energy consumption. It appears some feel that this is likely only to give small gains, whereas others were more enthusiastic.

Can A/B testing be incorporated into GI?

Others felt that software engineering should concentrate upon software maintenance. Potentially evolving source code helps with software maintenance. For example, the first GIed code in use (BarraCUDA [7]) contains evolved C code which had been accepted by the owners of BarraCUDA and is now maintained in the usual way. The same is also true of RNAfold [8, 9] and two bugfixing systems [10, 11] where GI code is being maintained by people using the same tools that they use to maintain human written code.

There was discussion about the need to maintain programs written in COBOL, for which human skills are in short supply.

2.4 Tool Selection and Integration

How to get the best when working with other tools? Can tools, like Daikon [12] and other program analysis tools, help the GI search or do they simply add too much noise?

Facebook intend to make open source its automatic bugfixing tool SapFix [13], perhaps in mid 2020.

It was suggested that compiler writers are not comfortable with incorporating machine learning into their compilers, particularly if code patches are incomprehensible. However others felt this was overblown and, for example, the University of Edinburgh compiler research group have actively considered heuristics and learning as part of ways to generate better optimised (typically meaning faster) machine code. Indeed genetic programming has been proposed to give better register allocation [14] and machine learning for branch prediction [15] and for Java garbage collection [16].

2.5 Benchmark Selection and Generality

Is there a danger of working on programs that are too small? Are small benchmark program atypical? Are they more fragile [17] than programs that people care about?

Although GI has been demonstrated with assembler, byte code and binary machine code, most GI work has operated on the level of source code. The two favoured approaches have been to use the abstract syntax tree (AST) generated by the compiler and representing the source code via a grammar. What other ways of representing the genetic material should be considered? E.g. Jhe-Yu Liou et al. [18] suggested using compiler intermediate representation (IR), as generated by the LLVM compiler. Whilst control flow graphs [15] might also be adopted by GI as an evolvable representation [19].

It was felt important to make results freely available. The workshop web pages http://geneticimprovementofsoftware.com/faq/ already points to some open source tools and benchmarks. Web sites such as GitHub and tools like Docker containers might help and also reduce the impact of “bit rot” on older tools.

There was discussion of the importance of benchmarks. Some felt it made life easier, e.g. to reproduce others results and therefore they aided good science. Others, however, felt that the point of evolutionary search was to try (and hopefully succeed) in solving problems which had previously not been considered at all or where simply regarded as being impossible. A counter argument was put that we can never get enough tests to define precisely what the program being tested will do in all circumstances. In other words the semantics of the program have to be defined. Can we treat testing as sampling the programs behaviour and in someway say how we expect the program to behave between samples? I.e., how should it interpolate or extrapolate from test cases?

2.6 Industrial Interest

Employment prospects. Student and post-doctoral internships are available at Facebook and elsewhere

In some programs an approximate answer may be sufficient, i.e. good enough.

3 Tools and Resources

Some free tools that were mentioned during the GI-2019 workshop (extended by the co-authors in the three weeks after GI-2019).

1. http://geneticimprovementofsoftware.com/faq/
2. Wikipedia Genetic improvement (computer science)
3. Wikipedia Automatic bug fixing
4. Google VirusTotal www.virustotal.com
5. LLVM, especially LLVM's C compiler intermediate representation https://github.com/eschulte/llvm-mutate
6. Rodinia benchmarks https://rodinia.cs.virginia.edu
7. ThunderSVM machine learning https://github.com/Xtra-Computing/thundersvm
8. Sleuth2 pattern mining https://cran.r-project.org/web/packages/Sleuth2
9. GraalVM https://www.graalvm.org/ and truffle (truffle 3)
10. Defects4J https://github.com/rjust/defects4j
11. Tiny CC, tcc may give faster compilation. I.e. trade compilation time against execution time. Similarly we may not want all compiler optimisation switched on during GI evolution.
12. EvoSuite http://www.evosuite.org
13. ManyBugs and IntroClass, Benchmark of small buggy programs for C https://repairbenchmarks.cs.umass.edu/
14. IntroClassJava, Benchmark of small buggy programs for Java https://github.com/Spirals-Team/IntroClassJava
15. GenProg (Automatic repair tool for C) https://github.com/squaresLab/genprog-code
16. GenProg4Java, (Automatic repair tool for Java) https://github.com/squaresLab/genprog4java
17. BugZoo is a container-based platform for studying historical bugs https://github.com/squaresLab/BugZoo [20]
18. Darjeeling is a language-agnostic tool for search-based program repair https://github.com/squaresLab/Darjeeling
19. GIN (lightweight micro-framework for the Genetic Improvement of Java code) https://github.com/gintool [21]
20. PyGGI (Python General Framework for Genetic Improvement) https://github.com/coinse/pyggi [22]
21. SEL, Software Evolution Library (which works with llvm-mutate, item 5 above) https://github.com/grammatech/sel
22. Blue (simple blue example of grammar based Genetic Improvement) http://www.cs.ucl.ac.uk/staff/W.Langdon/gggp/#code (see Free Code heading) [23]

A Post Workshop Suggestions by Nicolas Chausseau

Turn this article into an ongoing collaborative website online it could:

- Allow ongoing discussion in a more informal online forum setting
- Help discover people working on same or similar problems,
- And potentially enable collaboration, or branching out of other fruitful, more specific conversations on those topics.
- ... In other words it could be the starting point for new research, to be discussed, collaborators found, etc. Even paper drafts reviewed and commented on. (Nicolas Chausseau volunteered to help set up such a website).

  GI might follow ML4SE's example and set up a Google forum. He wondered if others interested in GI would participate in such a forum (e.g. ask questions, make comments).

A.1 Biological input to GP

Turn this article into an ongoing collaborative website online it could:

- I would be curious to know which recent papers discuss approaches inspired from biology / nature, if any (since there are no references for this section)

  In particular I wanted to read more about these very specific topics: Hierarchical structure of DNA (e.g. junk DNA, which we know now, enables or disables vast libraries of SNPs, and creates epistatic interactions) – what are the implications for GAs? Is it fruitful at all to store genetic information in hierarchical form?
- Hierarchical structure of layers of neural networks (e.g. bottom layers evolve to become multi-purpose feature detectors, during backpropagation) – we know in the case of neural networks this hierarchical structure is what allowed them to solve hard problems, by having lower layers learn these ”multipurpose features”.
- Hierarchical structure of human codebases (object-oriented code, where a first layer of reusable building block is constantly combined and recombined into new applications) – can GAs use these ”pre-evolved APIs” more efficiently, to speed up convergence? Can we also store a program's ”DNA information” in a more hierarchical form somehow? Or is human representation already optimal for building-block reuse, for enabling and disabling pre-evolved building blocks?
- ... It seems all 3 solutions (DNA, deep NNs and human codebases) present a hierarchical structure, a structure that allows for ”building-block discovery” and ”building-block reuse”.
  - I found only one article so far that examined this question a bit, it made interesting observations in the ”Modularity” section https://www.hindawi.com/journals/jaea/2010/568375/#B30, from 2010
  - Is it possible that the way information is stored (as a hierarchy) affects the speed of convergence of a population? Has this been examined before? Did we observe the formation of building blocks in GAs, has this been measured?
- Concerning search and GA hybrids, any additional link is welcome! [5] is very interesting.

References

[1] Petke, J., Tan, S.H., Langdon, W.B., Weimer, W. (eds.): Proceedings 2019 ACM/IEEE 6th International Genetic Improvement Workshop, GI 2019. Montreal (28 May 2019), http://www.cs.ucl.ac.uk/staff/W.Langdon/icse2019/GI_2019_frontmatter.pdf

[2] Tan, S.H., Yoshida, H., Prasad, M.R., Roychoudhury, A.: Anti-patterns in search-based program repair. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016. pp. 727–738. ACM, Seattle, WA, USA (2016), https://www.comp.nus.edu.sg/~abhik/pdf/FSE16.pdf

[3] Weyland, D.: A rigorous analysis of the harmony search algorithm: How the research community can be misled by a ”novel” methodology. Int. J. Appl. Metaheuristic Comput. 1(2), 50–60 (Apr 2010), http://dx.doi.org/10.4018/jamc.2010040104

[4] Afzal, A., Lacomis, J., Le Goues, C., Timperley, C.S.: A Turing test for genetic improvement. In: Petke, J., Stolee, K., Langdon, W.B., Weimer, W. (eds.) GI-2018, ICSE workshops proceedings. pp. 17–18. ACM, Gothenburg, Sweden (2 Jun 2018), http://dx.doi.org/10.1145/3194810.3194817

[5] Howden, W.E.: Weak mutation testing and completeness of test sets. IEEE Transactions on Software Engineering 8, 371–379 (1982)

[6] Fraser, G., Arcuri, A.: Evosuite: automatic test suite generation for object-oriented software. In: 8^{th} European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE ’11). pp. 416–419. ACM, Szeged, Hungary (September 5th - 9th 2011), http://doi.acm.org/10.1145/2025113.2025179

[7] Langdon, W.B., Lam, B.Y.H., Modat, M., Petke, J., Harman, M.: Genetic improvement of GPU software. Genetic Programming and Evolvable Machines 18(1), 5–44 (Mar 2017), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Langdon_2016_GPEM.pdf

[8] Langdon, W.B., Lorenz, R.: Improving SSE parallel code with grow and graft genetic programming. In: Petke, J., White, D.R., Langdon, W.B., Weimer, W. (eds.) GI-2017. pp. 1537–1538. ACM, Berlin (15-19 Jul 2017), http://doi.acm.org/10.1145/3067695.3082524

[9] Langdon, W.B., Lorenz, R.: Evolving AVX512 parallel C code using GP. In: Sekanina, L., Hu, T., Lourenco, N. (eds.) EuroGP 2019: Proceedings of the 22nd European Conference on Genetic Programming. LNCS, vol. 11451, pp. 245–261. Springer Verlag, Leipzig, Germany (24-26 Apr 2019), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon_2019_EuroGP.pdf

[10] Haraldsson, S.O.: Genetic Improvement of Software: From Program Landscapes to the Automatic Improvement of a Live System. Ph.D. thesis, Institute of Computing Science and Mathematics, University of Stirling, UK (May 2017), http://hdl.handle.net/1893/26007

[11] Alshahwan, N.: Industrial experience of genetic improvement in facebook. In: Petke, J., Tan, S.H., Langdon, W.B., Weimer, W. (eds.) GI-2019, ICSE workshops proceedings. p. 1. IEEE, Montreal (28 May 2019), http://www.cs.ucl.ac.uk/staff/W.Langdon/icse2019/Alshahwan_2019_GI.pdf, invited Keynote

[12] Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering 27(2), 99–123 (Feb 2001), http://dx.doi.org/doi:10.1109/32.908957

[13] Marginean, A., Bader, J., Chandra, S., Harman, M., Jia, Y., Mao, K., Mols, A., Scott, A.: SapFix: Automated end-to-end repair at scale. In: Atlee, J.M., Bultan, T. (eds.) 41st International Conference on Software Engineering. ACM, Montreal (25-31 May 2019), https://2019.icse-conferences.org/details/icse-2019-Software-Engineering-in-Practice/9/SapFix-Automated-End-to-End-Repair-at-Scale

[14] Stephenson, M., Amarasinghe, S., Martin, M., O'Reilly, U.M.: Meta optimization: improving compiler heuristics with machine learning. In: Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation (PLDI '03). pp. 77–90. ACM, San Diego, California, USA (2003), http://groups.csail.mit.edu/commit/papers/03/metaopt-pldi.pdf

[15] Muchnick, S.S.: Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers Inc. (1997)

[16] Andreasson, E., Hoffmann, F., Lindholm, O.: To collect or not to collect? machine learning for memory management. In: Java Virtual Machine Research and Technology Symposium. pp. 27–39. Citeseer (2002)

[17] Langdon, W.B., Petke, J.: Software is not fragile. In: Parrend, P., Bourgine, P., Collet, P. (eds.) Complex Systems Digital Campus E-conference, CS-DC'15. pp. 203–211. Proceedings in Complexity, Springer (Sep 30-Oct 1 2015), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon_2015_csdc.pdf, invited talk

[18] Liou, J.Y., Forrest, S., Wu, C.J.: Genetic improvement of GPU code. In: Petke, J., Tan, S.H., Langdon, W.B., Weimer, W. (eds.) GI-2019, ICSE workshops proceedings. pp. 20–27. IEEE, Montreal (28 May 2019), http://www.cs.bham.ac.uk/~wbl/biblio/gi2019/Liou_2019_GI.pdf

[19] Stadler, L., Würthinger, T., Simon, D., Wimmer, C., Mössenböck, H.: Graal IR : An extensible declarative intermediate representation (2013)

[20] Timperley, C.S., Stepney, S., Le Goues, C.: BugZoo: a platform for studying software bugs. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. pp. 446–447. ACM, Gothenburg, Sweden (2018), http://doi.acm.org/10.1145/3183440.3195050

[21] White, D.R.: GI in no time. In: Petke, J., White, D.R., Langdon, W.B., Weimer, W. (eds.) GI-2017. pp. 1549–1550. ACM, Berlin (15-19 Jul 2017), http://dx.doi.org/doi:10.1145/3067695.3082515

[22] An, G., Kim, J., Lee, S., Yoo, S.: PyGGI: Python General framework for Genetic Improvement. In: Proceedings of Korea Software Congress. pp. 536–538. KSC 2017, Busan, South Korea (20-22 Dec 2017), https://coinse.kaist.ac.kr/publications/pdfs/An2017aa.pdf

[23] Langdon, W.B.: Genetic improvement GISMOE blue software tool demo. Tech. Rep. RN/18/06, University College, London, London, UK (22 Sep 2018), http://www.cs.ucl.ac.uk/fileadmin/user_upload/blue.pdf