SYNASC 2014 (post proceedings), Franz Winkler *et al.*, Eds., p14–19, Timisoara, 22-25 September, IEEE.

Genetic Improvement of Programs

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

***Abstract*— Genetic programming can optimise software, including: evolving test benchmarks, generating hyper-heuristics by searching meta-heuristics, generating communication protocols, composing telephony systems and web services, generating improved hashing and C++ heap managers, redundant programming and even automatic bug fixing. Particularly in embedded real-time or mobile systems, there may be many ways to trade off expenses (such as time, memory, energy, power consumption) vs. functionality. Human programmers cannot try them all. Also the best multi-objective Pareto trade off may change with time, underlying hardware and network connection or user behaviour. It may be GP can automatically suggest different trade offs for each new market. Recent results include substantial speed up by evolving a new version of a program customised for a special case.**

***Index Terms*— GI, genetic programming (GP), Automatic software re-engineering, Bowtie2^{GP}, multiple objective exploration, search based software engineering (SBSE), GPGPU.**

I. Introduction

Genetic programing [Koza, 1992; Poli *et al.*, 2008] has been very widely applied[1]. For example in

- modelling [Kordon, 2010],
- prediction [Langdon and Barrett, 2004; Podgornik *et al.*, 2011; Kovacic and Sarler, 2014],
- classification [Freitas, 1997],
- design [Lohn and Hornby, 2006],
- creating art [Reynolds, 2011; Jacob, 2001; Langdon, 2004; Romero *et al.*, 2013],
- the generation of hyper-heuristics [Burke *et al.*, 2013],
- configuring intelligent telephony networks [Martin, 2000] and
- Web mashups [Rodriguez-Mier *et al.*, 2010],
- Hashing [Hussain and Malliaris, 2000],
- Heap managers [Risco-Martin *et al.*, 2014],
- multiplicity computing [Feldt, 1998; Cadar *et al.*, 2010]
- and even to create benchmarks which demonstrate the relative strengths and weaknesses of optimisers [Langdon and Poli, 2005].

Recently genetic programming has been applied to the production of programs itself, however so far relatively small programs have been evolved. Nonetheless GP has had some great successes when applied to existing programs. Perhaps the best known work is that on automatic bug fixing [Arcuri and Yao, 2008]. Particularly the Humie award winning work of Westley Weimer and Stephanie Forrest [Forrest *et al.*, 2009]. This has received multiple awards and best paper prizes [Weimer *et al.*, 2009; Weimer *et al.*, 2010]. GP has been used repeatedly to automatically correct most (but not all) real bugs in real programs [Le Goues *et al.*, 2012]. Weimer and Le Goues have now shown GP based automatic software correction to be effective on several millions of lines of C++ programs. Their GenProg [Le Goues *et al.*, 2012b] approach is based on re-using existing human written code to patch the source code defect. A recent study [Barr *et al.*, 2014] showed many updates to Java code made by people are not totally novel but could have been made by re-using existing code. Indeed, baring layout and identifier names, most human written code of up to five lines has already been written somewhere by someone else [Gabel and Su, 2010].

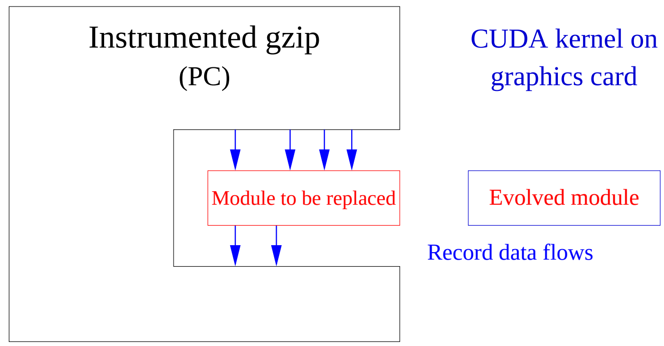

Fig. 1. The original code is instrumented to record the inputs and outputs (blue arrows) of the target function (red) every time it is called. These become the fitness function and test suite for the automatically evolved replacement module running on novel hardware (actually GPUs). The CUDA code generated by GP is functionally identical to the C code inside gzip [Langdon and Harman, 2010].

Once GP has been used to *do the impossible* (i.e. automatic bug fixing) it was improved [Kessentini *et al.*, 2011] and people felt brave enough to try other techniques, e.g. [Nguyen *et al.*, 2013].

Andrea Arcuri was again in at the start of inspirational work on showing GP can create real code from scratch. Although the programs remain small, David White, he and John Clark [White *et al.*, 2011] evolved programs to accomplish real tasks such as creating pseudo random numbers for ultra tiny computers where they showed a trade off between "randomness" and energy consumption. Such tradeoffs are vital if RFID based nano-computing devices are to be programmed.

To accompany keynote at the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2014)

[1] Genetic programming bibliography http://www.cs.bham.ac.uk/~wbl/biblio/ gives details of more than nine thousand articles, papers, books, etc.

II. Auto Porting Functionality

The Unix compression utility gzip was written in C in the days of Digital Equipment Corp.'s mini-computers. It is largely unchanged. It contains a procedure (of about two pages of code) which is so computationally intensive that it has been re-written in assembler for the Intel 86X architecture (i.e. Linux). The original C version of gzip has been retained and is distributed as part of Software-artifact Infrastructure Repository sir.unl.edu [Hutchins *et al.*, 1994]. SIR also contains a test suite for gzip. In *Genetic Improvement*, as with Le Goues' bug-fixing work, we start with an existing program and a small number of test cases. In the case of the gzip function, we showed genetic programming could evolve a parallel implementation for an architecture not even dreamt of when the original program was written [Langdon and Harman, 2010]. Whereas Le Goues uses the original program's AST (Abstract Syntax Tree) to ensure that many of the mutated programs produced by GP compile, we have used a BNF grammar. In the case of [Langdon and Harman, 2010] the grammar was derived from generic code written by the manufacture of the parallel hardware. Note that it had nothing special to do with gzip. The original function in gzip was instrumented to record its inputs and its outputs each time it was called (see Figure 1). When gzip was run on the SIR test suite, this generated more than a million test cases, however only a few thousand were used by the GP[2]. Essentially GP was told to create parallel code from the BNF grammar which when given a small number of example inputs returned the same answers. The resulting parallel code is functionally the same as the old gzip code.

III. Bowtie2^{GP} Improving 50 000 lines of C++

As Figure 2 shows, genetic programming produces populations of programs which may have different abilities on different scales. While Figure 2 shows speed versus quality, other tradeoffs have been investigated. For example it may be impossible to simultaneously minimise execution time, memory foot print and energy consumption. Yet, conventionally human written programs choose one trade-off between multiple objectives and it becomes infeasible to operate the program with another trade-off. For example, consider approximate string matching.

Finding the best match between (noisy) strings is the life blood of Bioinformatics. Huge amounts of people's time and computing resources are devoted every day to matching protein amino acid sequences against databases of known proteins from all forms of life. The acknowledge gold standard is the BLAST program [Altschul *et al.*, 1997] which incorporate heuristics of known evolutionary rates of change. It is available via the web and can lookup a protein in every species which has been sequences in a few minutes. Even before the sequencing of the human genome, the volume of DNA sequences was exploding exploding at a rate like Moore's Law [Moore, 1965]. With modern NextGen sequencing machines throwing out 100s of millions (even billions) of (albeit very noisy) DNA base-pair sequences, there is no way that BLAST can be used to process this volume of data. This has lead to human written look up tools for matching NextGen sequences against the human genome. Wikipedia lists more than 140 programs (written by some of the brightest people on the planet) which do some form of Bioinformatics string matching.

[2]Later work used even fewer tests.

The authors of all this software are in a quandary. For their code to be useful the authors have to chose a point in the space of tradeoffs between speed, machine resources, quality of solution and functionality, which will: 1) be important to the Bioinformatics community and 2) not be immediately dominated by other programs. In practise they have to choose a target point when they start as once basic design choices (e.g. target data sources and computer resources) have been made, few people or even research teams have the resources to discard what they have written and start totally from scratch. Potentially genetic programming offers them a way of exploring this space of tradeoffs [Harman *et al.*, 2012]. GP can produce many programs across the trade-off space and so can potentially say "look here is a trade-off which you had not considered". This could be very useful to the human, even if they refuse to accept machine generated code and insist on coding the solution themselves.

We have made a start by showing GP can transform human written DNA sequence matching code, moving it from one tradeoff point to another. The overall frame work is shown in Figure 3. In our example, the new program is specialised to a particular data source and sequence problem for which it is on average more than 70 times faster. Indeed on this particular problem, we were fortunate that not only is the variant faster but indeed it gives a slight quality improvement on average [Langdon and Harman,].

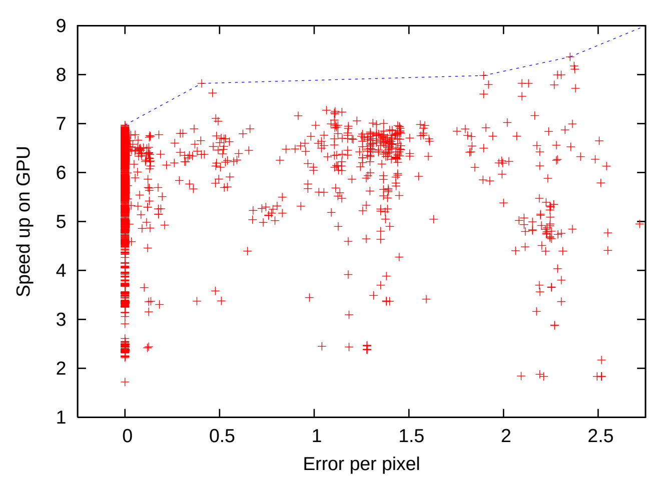

Fig. 2. Example of automatically generated Pareto tradeoff front. Genetic programming used to improve 2D Stereo Camera code [Stam, 2008] for modern nVidia GPU [Langdon and Harman, 2014]. Left (above 0) many programs are faster than the original code written by nVidia's image processing expert (human) and give exactly the same answers. Many other automatically generated programs are also faster but give different answers. Some (cf. dotted blue line) are faster than the best zero error program.

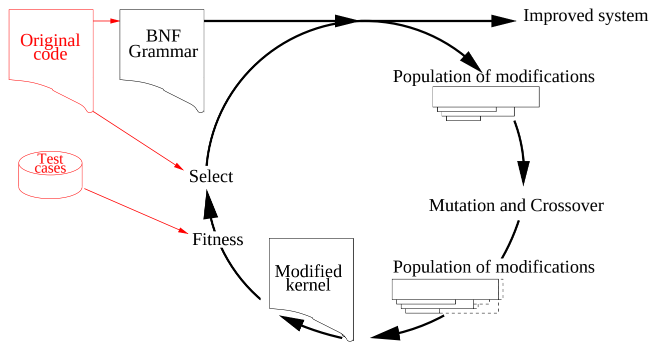

Fig. 3. Genetic Improvement cycle extends traditional GA/GP evolutionary cycle. GI starts with human written code (left, colour). It is automatically converting into a BNF grammar, which is used to create the initial generation. The GP evolves small patches, which are converted back into C++ code by effectively reversing the grammar. Finally the fitness of the mutated code is found by compiling and running it and then comparing its answers with those given by the original code on the same test cases. A small number of test cases (e.g. 5) are randomly chosen each generation from a much large stock, which may have been supplied with original code as its regression test suite.

IV. Improving Parallel Processing CUDA Code Written by Experts

In other examples we returned to computer graphics hardware. In the first GP was able to automatically update for today's GPUs software written specifically by nVidia's image processing expert to show off the early generations of their graphics cards [Stam, 2008]. Genetic improvement lead (on the most powerful modern Tesla GPU, see Figure 4) to almost a seven fold speed up relative to the original code on the same GPU [Langdon and Harman, 2014]. In another example a combination of manual and automated changes to production 3D medical image processing code lead to the creation of a version of a performance critical kernel which (on a Tesla K20c) is more than 2000 times faster than the production code running on an 2.67GHz CPU [Langdon *et al.*, 2014].

V. MiniSAT: Improving Boolean Satisfiability Code Written by Experts

The basic GI technique has also been used to create an improved version of C++ code from multiple versions of a program written by different authors. Boolean Satisfiability (SAT) is a problem which appears often. MiniSAT is a popular SAT solver. The satisfiability community has advanced rapidly since the turn of the century. This has been due in part to a series of competitions. These include the "MiniSAT hack track", which is specifically designed to encourage humans to make small changes to the MiniSAT code. The new code is available after each competition. MiniSAT and a number of human variants were given to GI and it was asked to evolve a new variant specifically designed to work better on a software engineering problem (interaction testing) [Petke *et al.*, 2014b]. At GECCO 2014 it received a Human Competitive award (HUMIE) [Petke *et al.*, 2014a].

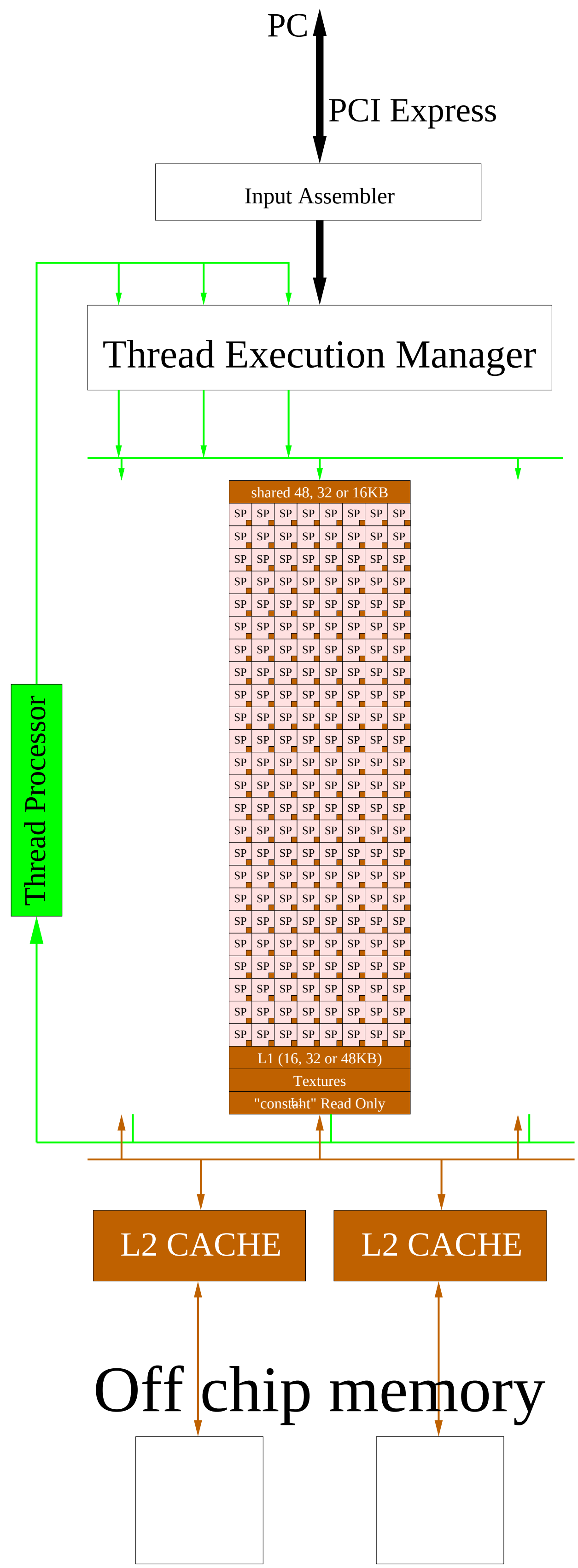

Fig. 4. Tesla K20c contains 13 SMX multiprocessors, a PCI interface to the host PC, thread handling logic and 4800 MBytes of on board memory. Each SMX contains 192 stream processors (only one SMX shown).

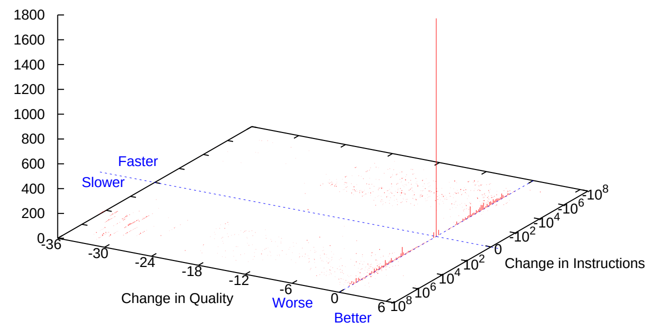

Fig. 5. Histogram of impact on speed and solution quality made by single mutations to Bowtie2 when running a random test case (Section III). Although many mutations cause Bowtie2 to fail (not plotted) and others cause it to produced very poor solutions (e.g. reducing by 36, left) others have less dramatic impact. Some slow Bowtie2 and others make it faster. However many changes have no impact on quality (although they may charge Bowtie2's speed, plotted along x=0). Indeed a large number do not change its speed either (note spike at the origin). There are even a few mutations which give better quality solutions. It is from these GP evolves a seventy fold speed up. Note non-linear scales.

VI. Babel Pidgin: Creating and Incorporating New Functionality

Another prize winning genetic programing based technique has recently been demonstrated to be able to extend the functionality of existing code [Harman *et al.*, 2014]. GP, including human hints, was able to evolved new functionality externally and then search based techniques [Harman, 2011] were used to graft the new code into an existing program (pidgin) of more than 200 000 lines of C++.

VII. Conclusion: Software is Not Fragile

There has been a tremendous fear of making random changes to programs. It was felt that any unthinking change must damage the software. Indeed a single random change may do so. However software engineers have long been familiar with mutation testing [DeMillo and Offutt, 1991; Langdon *et al.*, 2010], in which bugs are deliberately seeded into programs in order to gauge the effectiveness of test methods at finding bugs. One of the lessons of mutation testing has been that there are some "stubborn" mutants which are very hard to detect by testing [Yao *et al.*, 2014]. In other words some mechanically introduced changes to the code have little effect on its operation. That is, not all changes damage the code. Figure 5 shows an experiment (from Section III) in which the random changes made by genetic programming in the initial generation (i.e. before selection) were done thousands of times. For each mutation the program (Bowtie2) was run and the difference made by the mutation was recorded. Figure 5 plots both the change in solution quality and speed for each run. Notice (left hand side) some changes do indeed cause Bowtie2 to fail or generate junk results. However Figure 5 is dominated by a large spike at the origin corresponding to mutations which changes neither speed nor solution quality. There are even some mutants which produce slightly better answers.

[Schulte *et al.*, 2014] recently investigated the software mutational robustness of twenty two diverse programs and found consistently about a third of mutations do not cause the program to fail under testing. Whilst most investigations have mutated source code, similar robustness has been reported at assembler code end even binaries [Schulte *et al.*, 2013]. So yes, a single random change may break code, but if you are prepared to create a population of mutated programs, some programs in it may be broken but others may run ok. Evolutionary techniques select the better ones, the fitter ones, and create further changes to them. Using survival of the fitness [Darwin, 1859] over time the population can evolve to contain highly fit programs.

Genetic programming aims to tackle, what John Koza called the "S word" in AI, the *Scaling* problem. Recently there has been considerable progress not so much by evolving complete system from scratch but either by evolving modest code to glue large systems together from existing components or by evolving small changes to existing programs which make large improvements to them.

References

[Altschul *et al.*, 1997] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[Arcuri and Yao, 2008] Andrea Arcuri and Xin Yao. A novel co-evolutionary approach to automatic software bug fixing. In Jun Wang, editor, *2008 IEEE World Congress on Computational Intelligence*, pages 162–168, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press.

[Barr *et al.*, 2014] Earl T. Barr, Yuriy Brun, Premkumar Devanbu, Mark Harman, and Federica Sarro. The plastic surgery hypothesis. In Alessandro Orso, Margaret-Anne Storey, and Shing-Chi Cheung, editors, *22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, Hong Kong, 16 2014.

[Burke *et al.*, 2013] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, December 2013.

[Cadar *et al.*, 2010] Cristian Cadar, Peter Pietzuch, and Alexander L. Wolf. Multiplicity computing: a vision of software engineering for next-generation computing platform applications. In Kevin Sullivan, editor, *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 81–86, Santa Fe, New Mexico, USA, 7-11 November 2010. ACM.

[Darwin, 1859] Charles Darwin. *The Origin of Species*. John Murray, penguin classics, 1985 edition, 1859.

[DeMillo and Offutt, 1991] Richard A. DeMillo and A. Jefferson Offutt. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, 17(9):900–910, 1991.

[Feldt, 1998] Robert Feldt. Generating diverse software versions with genetic programming: an experimental study. *IEE Proceedings - Software Engineering*, 145(6):228–236, December 1998. Special issue on Dependable Computing Systems.

[Forrest *et al.*, 2009] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer, and Claire Le Goues. A genetic programming approach to automated software repair. In Guenther Raidl, Franz Rothlauf, Giovanni Squillero, Rolf Drechsler, Thomas Stuetzle, Mauro Birattari, Clare Bates Congdon, Martin Middendorf, Christian Blum, Carlos Cotta, Peter Bosman, Joern Grahl, Joshua Knowles, David Corne, Hans-Georg Beyer, Ken Stanley, Julian F. Miller, Jano van Hemert, Tom Lenaerts, Marc Ebner, Jaume Bacardit, Michael O'Neill, Massimiliano Di Penta, Benjamin Doerr, Thomas Jansen, Riccardo Poli, and Enrique Alba, editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 947–954, Montreal, 8-12 July 2009. ACM. Best paper.

[Freitas, 1997] Alex A. Freitas. A genetic programming framework for two data mining tasks: Classification and generalized rule induction. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 96–101, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Gabel and Su, 2010] Mark Gabel and Zhendong Su. A study of the uniqueness of source code. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 147–156, New York, NY, USA, 2010. ACM.

[Harman *et al.*, 2012] Mark Harman, William B. Langdon, Yue Jia, David R. White, Andrea Arcuri, and John A. Clark. The GISMOE challenge: Constructing the Pareto program surface using genetic programming to find better programs. In *The 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 12)*, pages 1–14, Essen, Germany, September 3-7 2012. ACM.

[Harman *et al.*, 2014] Mark Harman, Yue Jia, and William B. Langdon. Babel pidgin: SBSE can grow and graft entirely new functionality into a real world system. In Claire Le Goues and Shin Yoo, editors, *Proceedings of the 6th International Symposium, on Search-Based Software Engineering, SSBSE 2014*, volume 8636 of *LNCS*, pages 247–252, Fortaleza, Brazil, 26-29 August 2014. Springer. Winner SSBSE 2014 Challange Track.

[Harman, 2011] Mark Harman. Software engineering meets evolutionary computation. *Computer*, 44(10):31–39, October 2011. Cover feature.

[Hussain and Malliaris, 2000] Daniar Hussain and Steven Malliaris. Evolutionary techniques applied to hashing: An efficient data retrieval method. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, page 760, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann.

[Hutchins *et al.*, 1994] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria. In *Proceedings of 16th International Conference on Software Engineering, ICSE-16*, pages 191–200, May 1994.

[Jacob, 2001] Christian Jacob. *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, 2001.

[Kessentini *et al.*, 2011] Marouane Kessentini, Wael Kessentini, Houari Sahraoui, Mounir Boukadoum, and Ali Ouni. Design defects detection and correction by example. In *19th IEEE International Conference on Program Comprehension (ICPC 2011)*, pages 81–90, Kingston, Canada, 22-24 June 2011.

[Kordon, 2010] Arthur K. Kordon. *Applying Computational Intelligence How to Create Value*. Springer, 2010.

[Kovacic and Sarler, 2014] Miha Kovacic and Bozidar Sarler. Genetic programming prediction of the natural gas consumption in a steel plant. *Energy*, 66(1):273–284, 1 March 2014.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT press, 1992.

[Langdon and Barrett, 2004] W. B. Langdon and S. J. Barrett. Genetic programming in data mining for drug discovery. In Ashish Ghosh and Lakhmi C. Jain, editors, *Evolutionary Computing in Data Mining*, volume 163 of *Studies in Fuzziness and Soft Computing*, chapter 10, pages 211–235. Springer, 2004.

[Langdon and Harman,] William B. Langdon and Mark Harman. Optimising existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*. Accepted.

[Langdon and Harman, 2010] W. B. Langdon and M. Harman. Evolving a CUDA kernel from an nVidia template. In Pilar Sobrevilla, editor, *2010 IEEE World Congress on Computational Intelligence*, pages 2376–2383, Barcelona, 18-23 July 2010. IEEE.

[Langdon *et al.*, 2014] William B. Langdon, Marc Modat, Justyna Petke, and Mark Harman. Improving 3D medical image registration CUDA software with genetic programming. In Christian Igel, Dirk V. Arnold, Christian Gagne, Elena Popovici, Anne Auger, Jaume Bacardit, Dimo Brockhoff, Stefano Cagnoni, Kalyanmoy Deb, Benjamin Doerr, James Foster, Tobias Glasmachers, Emma Hart, Malcolm I. Heywood, Hitoshi Iba, Christian Jacob, Thomas Jansen, Yaochu Jin, Marouane Kessentini, Joshua D. Knowles, William B. Langdon, Pedro Larranaga, Sean Luke, Gabriel Luque, John A. W. McCall, Marco A. Montes de Oca, Alison Motsinger-Reif, Yew Soon Ong, Michael Palmer, Konstantinos E. Parsopoulos, Guenther Raidl, Sebastian Risi, Guenther Ruhe, Tom Schaul, Thomas Schmickl, Bernhard Sendhoff, Kenneth O. Stanley, Thomas Stuetzle, Dirk Thierens, Julian Togelius, Carsten Witt, and Christine Zarges, editors, *GECCO '14: Proceeding of the sixteenth annual conference on genetic and evolutionary computation conference*, pages 951–958, Vancouver, BC, Canada, 12-15 July 2014. ACM.

[Langdon and Harman, 2014] William B. Langdon and Mark Harman. Genetically improved CUDA C++ software. In M. Nicolau, K. Krawiec, M. I. Heywood, M. Castelli, P. Garci-Sanchez, J. J. Merelo, V. M. R. Santos, and K. Sim, editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 87–99, Granada, Spain, 23-25 April 2014. Springer.

[Langdon and Poli, 2005] William B. Langdon and Riccardo Poli. Evolving problems to learn about particle swarm and other optimisers. In David Corne, Zbigniew Michalewicz, Marco Dorigo, Gusz Eiben, David Fogel, Carlos Fonseca, Garrison Greenwood, Tan Kay Chen, Guenther Raidl, Ali Zalzala, Simon Lucas, Ben Paechter, Jennifier Willies, Juan J. Merelo Guervos, Eugene Eberbach, Bob McKay, Alastair Channon, Ashutosh Tiwari, L. Gwenn Volkert, Dan Ashlock, and Marc Schoenauer, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 81–88, Edinburgh, UK, 2-5 September 2005. IEEE Press.

[Langdon *et al.*, 2010] William B. Langdon, Mark Harman, and Yue Jia. Efficient multi-objective higher order mutation testing with genetic programming. *Journal of Systems and Software*, 83(12):2416–2430, December 2010.

[Langdon, 2004] W. B. Langdon. Global distributed evolution of L-systems fractals. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming, Proceedings of EuroGP'2004*, volume 3003 of *LNCS*, pages 349–358, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[Le Goues *et al.*, 2012] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In Martin Glinz, editor, *34th International Conference on Software Engineering (ICSE 2012)*, pages 3–13, Zurich, June 2-9 2012.

[Le Goues *et al.*, 2012b] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, January-February 2012.

[Lohn and Hornby, 2006] Jason D. Lohn and Gregory S. Hornby. Evolvable hardware using evolutionary computation to design and optimize hardware systems. *IEEE Computational Intelligence Magazine*, 1(1):19–27, February 2006.

[Martin, 2000] Peter Martin. Genetic programming for service creation in intelligent networks. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 106–120, Edinburgh, 15-16 April 2000. Springer-Verlag.

[Moore, 1965] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 19 1965.

[Nguyen *et al.*, 2013] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. SemFix: program repair via semantic analysis. In Betty H. C. Cheng and Klaus Pohl, editors, *35th International Conference on Software Engineering (ICSE 2013)*, pages 772–781, San Francisco, USA, May 18-26 2013. IEEE.

[Petke *et al.*, 2014a] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. Using genetic improvement & code transplants to specialise a C++ program to a problem class. 11th Annual Humies Awards 2014, 14 July 2014. Winner Silver.

[Petke *et al.*, 2014b] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. Using genetic improvement and code transplants to specialise a C++ program to a problem class. In M. Nicolau, K. Krawiec, M. I. Heywood, M. Castelli, P. Garci-Sanchez, J. J. Merelo, V. M. R. Santos, and K. Sim, editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 137–149, Granada, Spain, 23-25 April 2014. Springer.

[Podgornik *et al.*, 2011] Bojan Podgornik, Vojteh Leskovsek, Miha Kovacic, and Josef Vizintin. Analysis and prediction of residual stresses in nitrided tool steel. *Materials Science Forum*, 681, Residual Stresses VIII:352–357, March 2011.

[Poli *et al.*, 2008] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *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).

[Reynolds, 2011] Craig Reynolds. Interactive evolution of camouflage. *Artificial Life*, 17(2):123–136, Spring 2011.

[Risco-Martin *et al.*, 2014] Jose L. Risco-Martin, J. Manuel Colmenar, J. Ignacio Hidalgo, Juan Lanchares, and Josefa Diaz. A methodology to automatically optimize dynamic memory managers applying grammatical evolution. *Journal of Systems and Software*, 91:109–123, 2014.

[Rodriguez-Mier *et al.*, 2010] Pablo Rodriguez-Mier, Manuel Mucientes, Manuel Lama, and Miguel I. Couto. Composition of web services through genetic programming. *Evolutionary Intelligence*, 3(3-4):171–186, 2010.

[Romero *et al.*, 2013] Juan Romero, Penousal Machado, and Adrian Carballal. Guest editorial: special issue on biologically inspired music, sound, art and design. *Genetic Programming and Evolvable Machines*, 14(3):281–286, September 2013. Special issue on biologically inspired music, sound, art and design.

[Schulte *et al.*, 2013] Eric Schulte, Jonathan DiLorenzo, Westley Weimer, and Stephanie Forrest. 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, pages 317–328, Houston, Texas, USA, March 16-20 2013. ACM.

[Schulte *et al.*, 2014] Eric Schulte, Zachary P. Fry, Ethan Fast, Westley Weimer, and Stephanie Forrest. Software mutational robustness. *Genetic Programming and Evolvable Machines*, 15(3):281–312, September 2014.

[Stam, 2008] Joe Stam. Stereo imaging with CUDA. Technical report, nVidia, V 0.2 3 Jan 2008.

[Weimer *et al.*, 2009] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In Stephen Fickas, editor, *International Conference on Software Engineering (ICSE) 2009*, pages 364–374, Vancouver, May 16-24 2009.

[Weimer *et al.*, 2010] Westley Weimer, Stephanie Forrest, Claire Le Goues, and ThanhVu Nguyen. Automatic program repair with evolutionary computation. *Communications of the ACM*, 53(5):109–116, June 2010.

[White *et al.*, 2011] David R. White, Andrea Arcuri, and John A. Clark. Evolutionary improvement of programs. *IEEE Transactions on Evolutionary Computation*, 15(4):515–538, August 2011.

[Yao *et al.*, 2014] Xiangjuan Yao, Mark Harman, and Yue Jia. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In Lionel Briand, Andre van der Hoek, and Pankaj Jalote, editors, *ICSE*, pages 919–930, Hyderbad, 31 May-7 June 2014. ACM.