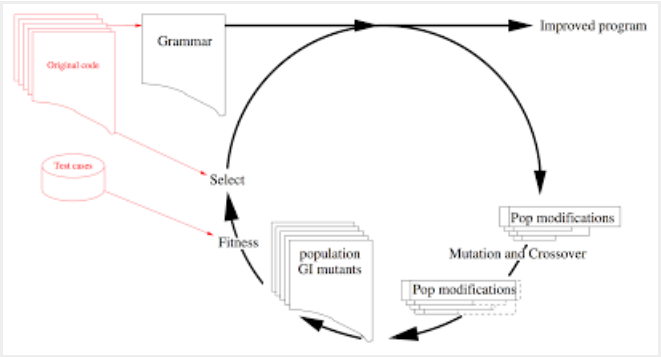
IEEE Software BLO

Wednesday, February 3, 2016

Genetic Improvement

by William B. Langdon & Justyna Petke
Associate Editor: Federica Sarro (@f_sarro)

Genetic improvement [1] has been demonstrated to be able to automatically repair real bugs in real programs [2] and give considerable speed up to real programs [3], e.g. by specialising them [4] or adapting to new hardware. There is now interest in using GI to improve non-functional properties such as extending battery life or reducing memory consumption and indeed in providing software designers with a range of trade-offs between different objectives: such as speed v. accuracy, speed v. memory.
Typically, in the case of bug fixing, a test case is needed which demonstrates the bug and a handful of tests which are used to ensure the putative fix has not repaired the bug at the expense of destroying other parts of the program. Usually the test comes from regression test suites developed as the program was written but they could be automatically generated. In GI it is common to run both the original code and the mutated code on the tests and compare their answers and performance. Effectively the old code becomes its own specification and even for new tests it can be used as the test oracle.



The figure shows the basics of Genetic Improvement. On the left hand side is the system to be improved and its test suite. On the right is the generational cycle of artificial evolution which optimises patches. Typically a patch deletes, copies or inserts an existing line of human written code. GI does not need to invent entirely new code. Each generational mutation and crossover create new patches. The patches are applied via the grammar to create patched versions of the software. These variants are tested on a small randomly selected part of the test suite and their answers and

IEEE Software

Blog Archive

▼ 2016 (7)
▼ February (2)
License comp free/open s
Genetic Impro
► January (5)
► 2015 (7)

Subscribe To

Posts
Comments

resource consumption are compared to those of the original system. Only patches responsible for better programs get children in the next generation. After a small number of generations the best patch in the last generation is cleaned up by removing unneeded changes and validated.
The grammar describes the original code and legal variations from it. In the case of automated bug repair, its role is typically replaced by abstract syntax trees ASTs. In the case of AST, perhaps about half the mutants compile and similarly the grammar ensures between 40% and 100% of patches compile. Typically, most patches that compile also run and produce an- swers. CPU limits, timeouts or loop iteration limits are imposed to ensure termination and typically some form of sandboxing is used to ensure badly behaved mutants cannot damage the GI system itself.
Whilst much GI work has been on source code, Darwinian evolution has been shown to be effective on Java byte code and even machine code. GI has been shown to offer Pareto tradeoffs in graphics applications and in improving Bioinformatics and other applications which use graphics cards as parallel computing accelerators (known as GPGPU).

Presenters at the First Genetic Improvement workshop, Madrid 2015

There is increasing interest in genetic improvement, with the number of papers and successful applications increasing. Last year saw the first international workshop and a GI special issue of the “Genetic Programming and Evolvable Machines” journal (publication due later this year). Whilst this year the workshop will be repeated in Denver and there will be a special session on GI at the IEEE World Congress on Computational Intelligence in Vancouver.

References

[1] William B. Langdon. Genetically improved software. In Amir H. Gandomi et al., editors, Handbook of Genetic Programming Applications, chapter 8, pages 181–220. Springer, 2015.

[2] 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.

[3] William B. Langdon and Mark Harman. Optimising existing software with genetic programming. IEEE Transactions on Evolutionary Computation, 19(1):118–135, February 2015.

[4] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. Using genetic improvement and code transplants to specialise aC++ program to a problem class. In Miguel Nicolau et al., editors,

17th European Conference on Genetic Programming, LNCS Springer v. 8599.

You might also enjoy reading:

- James Temperton. Code 'transplant' could revolutionise programming. Wired.co.uk, 30 July 2015.
- John R. Woodward, Justyna Petke, and William Langdon. How computers are learning to make human software work more efficiently. The Conversation, page 10.08am BST, June 25 2015.
- Justyna Petke. Revolutionising the process of software development. DAASE project blog, August 7 2015.
- Example automatic bug repair system using GI: Le Goues' bug fixing system [2] is at http://genprog.cs.virginia.edu/

Posted by Federica Sarro at 5:44 PM

No comments:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)