How computers are learning to make human software work more efficiently

June 25 2015, by John R. Woodward, Justyna Petke And William Langdon

Need a computer doctor? Dial 100110011001. Credit: agsandrew

Computer scientists have a history of borrowing ideas from nature, such as evolution. When it comes to optimising computer programs, a very interesting evolutionary-based approach has emerged over the past five or six years that could bring incalculable benefits to industry and eventually consumers. We call it genetic improvement.

Genetic improvement involves writing an automated "programmer" who manipulates the source code of a piece of software through trial and error with a view to making it work more efficiently. This might include swapping lines of code around, deleting lines and inserting new ones – very much like a human programmer. Each manipulation is then tested against some quality measure to determine if the new version of the code is an improvement over the old version. It is about taking large software systems and altering them slightly to achieve better results.

The benefits

These interventions can bring a variety of benefits in the realm of what programmers describe as the functional properties of a piece of software. They might improve how fast a program runs, for instance, or remove bugs. They can also be used to help transplant old software to new hardware.

The potential does stop there. Because genetic improvement operates on source code, it can also improve the so-called non-functional properties. These include all the features that are not concerned purely with just the input-output behaviour of programs, such as the amount of bandwidth or energy that the software consumes. These are often particularly tricky for a human programmer to deal with, given the already challenging problem of building correctly functioning software in the first place.

We have seen a few examples of genetic improvement beginning to be recognised in recent years – albeit still within universities for the moment. A good early one dates from 2009, where such an automated "programmer" built by the University of New Mexico and University of Virginia fixed 55 out of 105 bugs in various different kinds of software, ranging from a media player to a Tetris game. For this it won $5,000 (£3,173) and a Gold Humie Award, which is awarded for achievements produced by genetic and evolutionary computation.

In the past year, UCL in London has overseen two research projects that have demonstrated the field's potential (full disclosure: both have involved co-author William Langdon). The first involved a genetic-improvement program that could take a large complex piece of software with more than 50,000 lines of code and speed up its functionality by 70 times.

The second carried out the first automated wholesale transplant of one piece of software into a larger one by taking a linguistic translator called Babel and inserting it into an instant-messaging system called Pidgin.

Nature and computers

To understand the scale of the opportunity, you have to appreciate that software is a unique engineering material. In other areas of engineering, such as electrical and mechanical engineering, you might build a computational model before you build the final product, since it allows you to push your understanding and test a particular design. On the other hand, software is its own model. A computational model of software is still a computer program. It is a true representation of the final product, which maximises your ability to optimise it with an automated programmer.

As we mentioned at the beginning, there is a rich tradition of computer scientists borrowing ideas from nature. Nature inspired genetic algorithms, for example, which crunch through the millions of possible answers to a real-life problem with many variables to come up with the best one. Examples include anything from devising a wholesale road distribution network to fine-tuning the design of an engine.

Though the evolution metaphor has become something of a millstone in this context, as discussed here, genetic algorithms have had a number of successes producing results which are either comparable with human programs or even better.

Evolution also inspired genetic programming, which attempts to build programs from scratch using small sets of instructions. It is limited, however. One of its many criticisms is that it cannot even evolve the sort of program that would typically be expected of a first-year undergraduate, and will not therefore scale up to the huge software systems that are the backbone of large multinationals.

This makes genetic improvement a particularly interesting deviation from this discipline. Instead of trying to rewrite the whole program from scratch, it succeeds by making small numbers of tiny changes. It doesn't even have to confine itself to genetic improvement as such. The Babel/Pidgin example showed that it can extend to transplanting a piece of software into a program in a similar way to how surgeons transplant body organs from donors to recipients. This is a reminder that the overall goal is automated software engineering. Whatever nature can teach us when it comes to developing this fascinating new field, we should grab it with both hands.

Source: The Conversation