Co-evolutionary Software Development and Testing Revisited

Timo Mantere
Department of Electrical Engineering and Automation
University of Vaasa
P.O. Box 700, FIN-65101 VAASA

Abstract

This paper introduces the idea of simultaneous software testing and development with the co-evolutionary evolution algorithms. Software development in this context could be software correction, setting of software parameters, classification software boundary search or software bug fixing. We introduce what we have done within this framework and also discuss the research done by others. Everyone working with this idea has obtained promising results, which proofs that this concept has potential, but we are still far from the practical real world applications.

1. Introduction

In 2002 we proposed a method of using co-evolutionary genetic algorithms for the simultaneous software development and testing [1], [2], [3], [4], [5]. Soon after that our embedded software testing project ended and we move on to study other things. Over the years we have proposed [6] this co-evolutionary method for further development and study in couple of software testing research plans and proposals, but none of them has ever materialized. So, this interesting and promising method was then mostly forgotten.

Lately, similar approaches as ours have been re-invented and proposed separately by some researchers [7], [8], [9]. In this paper we revisited our original approach, and review the newer similar approaches presented by others, and compare their approaches and findings to ours.

We will also briefly present our tentative co-evolutionary design and testing application concerning cooperative (slightly parasitic) co-evolutionary development of near-infrared based skin imaging device and its software testing.

1.1 Software Testing

Software testing is an essential task when trying to achieve high software quality. Testing is both technically and economically an essential part in high quality software production. It has been estimated [10] that testing causes about half of the expenses related to software production. Much of the testing is done manually or using other labor-intensive methods. It is thus vital for the software industry to develop efficient, cost effective, and automatic means and tools for this task. According to the US Trade Ministry [11] software errors cost the U.S. economy $59.5 billion a year, approximately 0.6 percent of gross domestic product. They estimate that more than 30% of these costs could be eliminated by earlier and more effective identification and removal of software defects with an improved testing infrastructure.

1.2 Genetic Algorithms

Evolutionary algorithms (EA) belong to a branch of evolution inspired heuristic optimization methods, the most well known being: evolution strategies, genetic algorithms, and genetic programming. Genetic algorithms (GA) [12] are a group of evolutionary algorithms, which use evolution principles, like selection, mutation, and recombination. Genetic algorithms form a kind of electronic population that fights for survival, adapting to its environment as well as possible, which is an optimization problem. Surviving and crossbreeding possibilities depend on how well individuals fulfill the target function. GAs are used to solve complex optimization tasks, they do not require the optimized function to be continuous or derivable, or even be a mathematical formula, and that is perhaps the most important factor why they are gaining more and more popularity in practical technical optimization.

1.3 Co-evolution

Co-evolution (CE) means that an evolutionary algorithm (EA) is composed of several species with different types of individuals, while a standard evolutionary algorithm has only one single population of individuals. In CE the genetic operations, crossover and mutation are applied to only one species, while selection can be performed among individuals of one or more species. When we deal with an optimization problem, the environmental conditions are either stochastic or immeasurable. We can then try to develop the environmental conditions concurrently with the problem. Change in one population causes an environmental change in the other population. Trial solutions implied by one species are evaluated in the environment implied by another species. The goal is to accomplish an upward spiral, an arms race, where both species would achieve ever better results. This corresponds to the nature, where evolutionary developments in pray species also causes evolutionary changes in predators, and vice versa.

1.4 Evolutionary Algorithms in Software Testing

It is important to observe that a GA does not usually find any single error from the software with any higher probability than random search. However, if the error situation is composed of a combination of input parameters or a sequence of operations by these inputs, then it is possible that GA gains advantage. In practice this means that a GA tester generates trials with a several parameter combinations that cause minor faults and constructs a new input sequence which causes more errors than pure random tests.

When testing nondeterministic time critical software with a temporal fitness function, GA learns to favor inputs that cause delays and generates input sequence that causes longer response delays. Note that the extreme execution times found in this way are not necessarily globally maximal/minimal.

In software development and testing, the co-evolution can be applied so that another EA try to generate harder and harder test cases for the software, and the other EA tries to evolve, either software, or its parameter setting, etc., so that the software under test would manage the test cases ever better and better.

2. Literary Review

Automatic generation of computer programs has long been a dream of evolutionary algorithms researchers, but in reality no real working large software has ever been generated this way. Main successes have been generating some simple functions or game planning rules. Using co-evolution for software development was first suggested by Koza [13], but in this study he actually co-evolves game strategies of two players against each other, not the software against the test cases.

Wilkinson has written an interesting master thesis [7] where he tries to proof that the concept and idea of co-evolutionary software testing and correction works in practice. His original aim was to use genetic programming (GP) for changing the software under test, but unfortunately found that to be too difficult, and instead uses simpler approach to change software values (parameters) with evolutionary programming. In practice this means that his study is quite similar to ours [1], where we tried to evolve software parameters co-evolutionarily simultaneously with the testing. His results seems to proof the concept and confirm our findings too, since we found that the proposed system managed to correct the software 12 times out of 48 experiments. He plans to continue with the research topic and later actually try to correct software with GP.

Arcuri *et al.* [8] have chosen the more advanced and ambitious task of actually changing and correcting the software with using GP. They think that they can achieve this by initializing the software population with clones of the tested

software. The idea is that they start to correct the software from buggy version, since the buggy version is likely to be very close to the correct one, and GP does not need to change software that much. It is likely impossible to generate any working software from scratch with GP, but small corrections should be much easier task. They found out that simple bugs were correctly fixed most of the time, but the more difficult bugs never. The good news is that sometimes fairly easy bugs to fix might be very difficult for humans to detect. This occurs if the error is small, *i.e.* the code is very close to correct. This kind of bug might be relatively easily fixed with for EA based method; therefore the results are very interesting.

Koos *et al.* [9] have studied co-evolution of models and tests. In this study helicopter simulation model control system is developed co-evolutionarily against the test set. They use Pareto co-evolution, where they try to minimize the test set that forms a Pareto front. The control system is obviously a software based, so this study is clearly related to what we did in [1], where we co-evolutionarily optimize both the machine vision software and the whole filming setting together against the test cases (test objects to be measured), except that we did not use Pareto optimization. They also consider the gap between simulated model and reality, while their results where promising, they come up with a long list of problems and issues that need to be solved in order to develop the idea further.

3. The Proposed Method

When we are normally setting the software parameters we usually have a kind of improvement loop: set some values → test against some fixed situations → improve settings until we are satisfied with the software behavior.

In the proposed method (fig. 1) we do not manually set the parameters, but instead let the genetic algorithm (1st GA) to generate setting values, then the settings are tested against situations generated by 2nd GA, and then the 1st GA generates new parameter settings according the results measured against those situations generated by 2nd GA. This loop continues until any improvement in the settings stops. The 2nd GA generates more difficult test situations in each generation according the fitness value measured against the settings from 1st GA.

Co-evolution can be performed in three ways, with symbiotic, parasitic or predator-prey type. The predator-prey type is most the common type in co-evolutionary software testing and development, and also the one used in the system described above.

However, in our second system used in the result section 4.2 we have symbiotic co-evolution, which is normal for meta-GA systems, where all GAs have the same goal and they try to help each others and the overall system to achieve it. In our case we have two GAs, where lover level GA tries to generate feasible solutions to the problem, spectra's that fulfill the boundary constrains, and the

upper level GA tries to find spectra's that are as close to the opposite reference group as possible. One might argue that this system is somewhat parasitic also, because the upper level GA uses the trials that lover level GA generates, and lower level GA does not benefit anything from the upper level GA, so their gain relationship is one-way only.

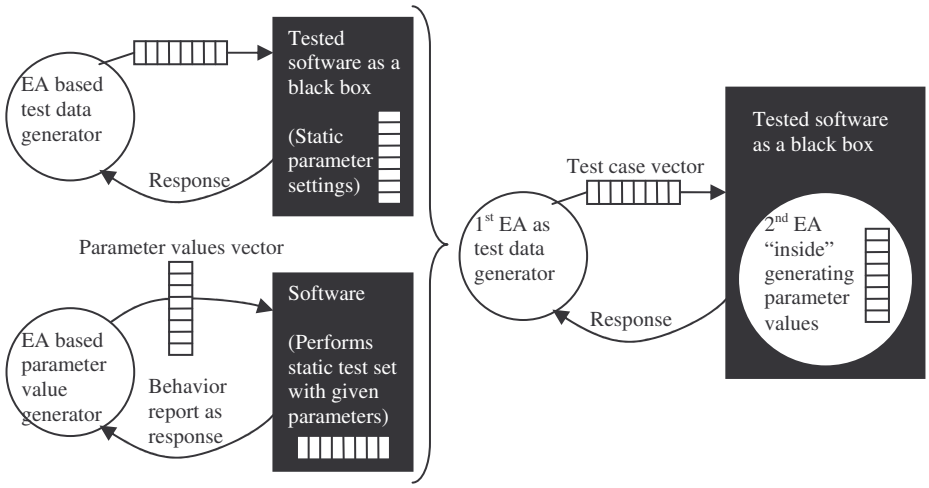

Figure 1: Example of how Evolutionary Algorithm based software testing (up left) and EA based software parameter setting (down left) can be combined to co-evolutionary simultaneous software development ant and testing (right). Note that in left, software's has either static parameter settings or static test set, but in right both are always dynamic and can change.

4. The Results

4.1 Some Afterthoughts of the Old Results

In our study [1] we co-evolutionarily develop the machine vision software and the filming system settings together against the test cases that also test both the system and its software together. So, in that system we could not really distinguish what advantages where obtained by software improvement and what was due the system setting improvement. We should go back and test the software and system settings separately in order to see, if both improved or only the other one improved. It is also possible that only the other one improved and the other one weakened, while overall performance was improved.

In [4] we did a kind of mutation testing by generating nearly similar image filters, and try to detect faulty ones with co-evolutionary testing, the paper never mentioned that what we did was within the mutation testing framework.

Co-evolution has some common problems [14]. Obviously sometimes there is no co-evolution; the different species do not push each others. Sometimes the

improvement end prematurely to some kind of evolutionary equilibrium. Sometimes there is stagnation (co-evolving populations circulates some repeatable path). Sometimes only the other population evolves due the disproportion of their possible improvement windows.

Obviously in [4] we had these problems. Co-evolving image filters against test images, the filter population was not able to evolve much, but test images got much harder. There was also some path repetition present. There are ways to prevent these typical co-evolutionary problems [14], but back then we did not consider those. So, the new experiments with problem preventing methods might give us better and more reliable results.

4.2 Results of the New Experiments

One of our research interest lately have been simulating near-infrared (NIR) spectra's in order to test the NIR based melanoma (skin cancer) detector software under development. See more about the system and basic study in [15].

The problem was that there are only a small number of skin cancer spectra samples. The normal skin samples (moles) are easier to obtain. In order to test and develop the classification software and finding its classification boundaries we decided to simulate the NIR spectra's with the help of genetic algorithms (GA). This was done so that we calculate the envelope curves of 1) all skin samples (fig. 2a), 2) healthy skin samples and 3) melanoma samples. Then we calculate the envelope curves of the 1st and 2nd derivatives of the same sample groups. GA is used to generate simulated "fake" spectra's, so that we generate random number sequences and GA evolves them until they fit inside all the three envelope curves (measured from the real spectra, and its 1st and 2nd derivatives).

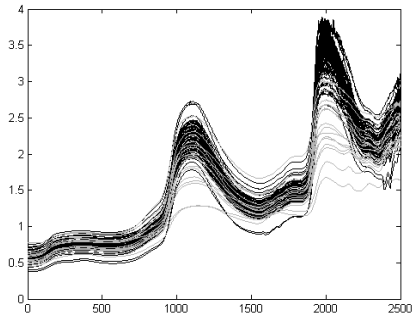

Figure 2a. The original spectra's, healthy skin (black) and cancer (gray).

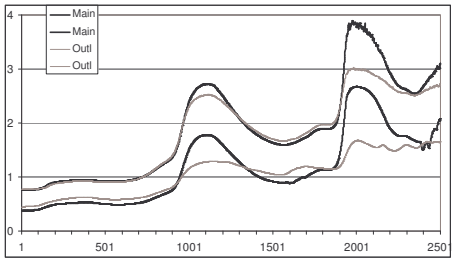

Figure 2b. The envelope curves of the original spectra's; the healthy skin (*black*), the cancer samples (*gray*).

The independent component analyses (ICA) based classification will then classify the simulated spectra either to as healthy skin or cancer groups. This was done in order to find classification boundaries and see if simulated data generated by

using its reference group envelope curves would always be classified correctly. Our first tests [15] showed that simulated spectra's were always classified correctly.

Now, the fact that simulated spectra's are always classified correctly with the classification system does not guarantee that classification boundaries are correctly set. It could also mean that simulated spectra's do not even have as much variation as real spectra's. In order to test this we decide to use cooperative co-evolutionary test setting.

So, we programmed two genetic algorithms with the meta-GA style. The lower level GA tries to find the proper NIR spectra are that fulfills the boundary constraints (envelope curves). All the accepted simulated NIR spectra's are then sent to the upper level GA, that tries to find those simulated NIR spectra's that are the closest to the opposite reference group (healthy skin vs. cancer).

This means that simulated NIR spectra's are generated by using *e.g.* the healthy skin boundaries as reference, and when the accepted simulated NIR spectra are found, we calculate it's distance to the opposite cancer group members.

This distance is calculated by doing the ICA classification each time after 20 new accepted (30 in the first optimization round, initial population) NIR spectra's are generated by lower level GA. Then by using the first three components of ICA we calculate the sum of distances between each simulated NIR spectra to all the original spectra's in the opposite reference group. After that, 10 closest to the reference will survive in the upper level GA population.

The system is run 100 upper level GA classification rounds and after that the best 10 individuals are our boundary solutions. This means that in the upper level GA, population size is 30, and the elitism is 33%. The upper level GA does not have any other parameters; so it is not really a real GA in this case, but rather just a selection tool. The lower level GA parameters were: chromosome length 2500 floating point numbers between the envelope curve (fig. 2b), population size 30, elitism 16.7%*, uniform crossover rate 95%, Gaussian mutation probability 5%, and the stopping criteria was: the accepted NIR spectra found, *i.e.* fitness value = 0 = no envelope curve boundary violations. *When accepted spectra found, it was immediately moved from the lower lever GA population to the upper level GA population.

The fig. 3 shows the 10 best solutions find by using healthy skin spectra's and cancer spectra's as simulation references. We can see that both simulated NIR groups locates between the original groups, but they do not cross or touch each other. So, our conclusion was that our samples are too distinguishable. Main reason for this is obvious; the original samples are also too easily separated. Our cancer samples only include 17 specimens and it seems they were too easily separated from the healthy skin. Sample group obviously contains no border cases.

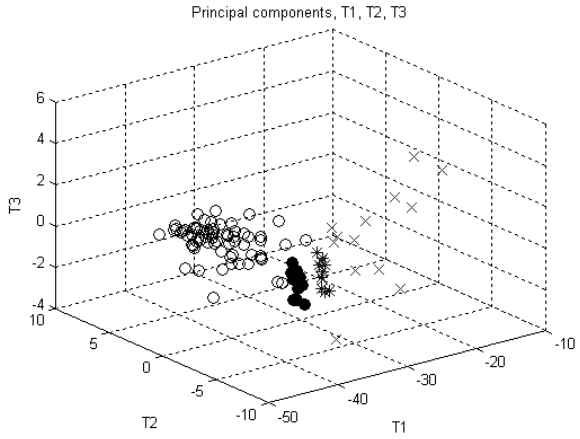

Figure 3: Searching the group boundaries. Original healthy skin spectra's (o), original cancer spectra's (×), spectra's simulated by healthy skin (•), and the spectra's simulated by cancer group (*).

However, the evolutionary border testing was able to generate samples that were between the two original groups, but there was still some unreachable "no mans land". It seems that since the used samples are so distinguishable the constraints do not allow the simulated data to go the area of opposite reference group.

The main conclusion of this testing was that our skin cancer sample group is way too small. While we can test the system with the simulated samples, they don't really reveal any software faults at this point, and do not help us to develop the classification software parameters any further. We must have more skin cancer samples before we can proceed.

5. Conclusion and Future

The results, both ours and referenced, seem to indicate that some software errors that might be difficult for humans to detect, could be relatively easy to fix for co-evolutionary based testing and correcting method, e.g., if error is small, i.e. the code is very close to correct. Also if the software has changeable parameters and values, to find the exactly right combination for them is very difficult for humans, but relatively easy for co-evolutionary based method.

In the results presented in chapter 4.2 suffer from the lack of real world data, so at the moment we could not improve the proposed cooperative co-evolutionary development & testing system before more real world measurements are available.

Our results in [1] showed that the more there are changeable parameters in the system, the more improvement the co-evolutionary testing and development system was able to achieve. This finding was important, because, the more there

are changeable parameters; the more difficult it is for human to find the optimal settings for them. Therefore co-evolutionary development could be helpful in the complex systems.

Wilkinson [7] and Arcuri [8] used sorting algorithms as test programs for their co-evolutionary software correction experiments. We also used GA based testing to detect bugs from the bubble-sort algorithm [16]. We all had to consider proper fitness function in order to measure how far from the correct software is. In evolutionary testing just the information if tested software is correct/incorrect is not enough, we need to have more variety fitness value to guide search.

In future we might go back to our bubble-sort testing [16] and try to fix it co-evolutionarily with Wilkinson's or Arcuri's ideas, and also try some new ideas of our own. Also the issues we raise in the chapter 4.1 are something we might retest in the future.

It is clear that the problem of co-evolutionary software testing and correction is still far from solved, and due to many promising experiments this method is worth of further study.

References

[1] Mantere T., Alander JT. 2002. Developing and testing structural light vision software by co-evolutionary genetic algorithm. *The Proceedings of the Second ASERC Workshop on Quantative and Soft Computing based Software Engineering (QSSE 2002)*, Feb 18-20 2002, Banff, Alberta, Canada. Alberta Software Engineering Research Consortium (ASERC): 31-37.

[2] Mantere, T. 2003. Software testing by evolutionary algorithms. In *Proceedings of Southeastern Software Engineering Conference*, April 1st - 3rd, 2003, Huntsville, Alabama, USA, CD and WWW (no longer available) Proceedings: 3 pages + 24 presentation slides.

[3] Mantere T. 2003. *Automatic Software Testing by Genetic Algorithms*. Ph.D. thesis, Acta Wasaensia 112, Computer Science 3, University of Vaasa: 151 pages.

[4] Mantere, T., Alander J.T. 2003. Testing digital halftoning software by generating test images and filters co-evolutionarily. In D. P. Casasent, E. L. Hall, and J. Röning, eds., *Proceedings of SPIE Vol. 5267 Intelligent Robots and Computer Vision XXI: Algorithms, Techniques, and Active Vision*, Providence, RI, 27-30 October 2003, SPIE, Bellingham, Washington, USA: 257-268.

[5] Mantere, T., Alander J.T. 2005. Evolutionary software engineering. In *Applied Soft Computing* **5**(3): 315-331.

[6] Koljonen J., Mantere T., Alander J.T. 2006. Soft computing in quality control of embedded systems: applications and future visions. Presented by invitation in *The 3rd annual NEXT (New EXploratory Technologies) conference*, Salo (Finland), Oct. 5-6, 2006: presentation only.

[7] Wilkerson, JL. 2008. *Co-Evolutionary Automated Software Correction: A Proof of Concept*. M.Sc. thesis, Missouri University of Science and Technology: 65 pages.

[8] Arcuri A., Yao X. 2008. A novel co-evolutionary approach to automatic software bug fixing. *2008 IEEE World Congress on Computational Intelligence (WCCI 2008)*, 1-6 June, Hong Kong, China, pages 162-168.

[9] Koos S., Mouret JB., Doncieux S. 2009. Automatic system identification based on coevolution of models and tests. *IEEE Congress on Evolutionary computation – CEC2009*, 18-21 May, Trondheim, Norway: 560-567.

[10] Edward K. 1995. *Software testing in the real world – improving the process*. New York, NY, USA: Addison-Wesley.

[11] NIST 2002. *The Economic Impacts of Inadequate Infrastructure for Software Testing, US Department of Commerce, National Institute of Standards and Technology (NIST), Planning Report 02–3, May 2002*, 309 pages [cited 8.4.2003]. Available: http://www.nist.gov/director/prog-ofc/report02-3.pdf.

[12] Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, University of Michigan Press, Reissued by The MIT Press, 1992.

[13] Koza J. R. 1991. Genetic evolution and co-evolution of computer programs. In *Artificial Life II*, Addison-Wesley: 603-629.

[14] Cartlidge J.P. 2004. *Rules of Engagement: Competitive Coevolutionary Dynamics in computational Systems*. PhD thesis, University of Leeds: 210 pages.

[15] Mantere, T., Välisuo P., Alander J.T. 2009. Testing NIR Based Skin Spectra Analyzer System and Software with the Simulated Data Generated by Genetic Algorithm. Accepted to be published in *NIR 2009, The 14th International Conference on Near Infrared Spectroscopy*, 7-16 November 2009, Bangkok, Thailand, 4 pages.

[16] Alander J.T., Mantere T. 2000. Genetic algorithms in automatic software testing - analysing a faulty bubble sort routine. In H. Hyötyniemi, ed., *SteP 2000 – Millennium of Artificial Intelligence, The 9th Finnish Artificial Intelligence Conference*, Helsinki University of Technology, Espoo, 28-31 August 2000, Publications of the Finnish Artificial Intelligence Society – 16, SteP 2000, vol. 2, Espoo, Finland, pp. 23-32.