*To be presented at* CEC 2016 Key Challenges and Future Directions of Evolutionary Computation Workshop Yun Li, *et al., Eds.*, Vancouver, 25-29 July, IEEE.

# Genetic Improvement: A Key Challenge for Evolutionary Computation

William B. Langdon
Gabriela Ochoa

***Abstract*—Automatic Programming has long been a sub-goal of Artificial Intelligence (AI). It is feasible in limited domains. Genetic Improvement (GI) has expanded these dramatically to more than 100 000 lines of code by building on human written applications. Further scaling may need key advances in both Search Based Software Engineering (SBSE) and Evolutionary Computation (EC) research, particularly on representations, genetic operations, fitness landscapes, fitness surrogates, multi objective search and co-evolution.**

## I. Introduction

Genetic Algorithms were invented right at the start of Artificial Intelligence research in the hope they could harness the power of Darwin's natural evolution [Darwin, 1859] so that we can get "computers to do what is needed to be done, without being told exactly how to do it" [Koza, 1992, page 1]. Natural selection and inheritable genetic variation acting over billions of years has seen life diversify and new species emerge and colonise every conceivable niche on the planet. Indeed geographic [Owen *et al.*, 1990] or even man made environmental changes[1] have seen the evolution or diversification of species within a few hundred years. The hope that such rapid evolution might also take place within computer populations has in many cases been vindicated. Today Evolutionary Computing has been successfully applied to finding acceptable solutions (even optimal solutions) to a wide range of numerical problems within the computer and to the evolution of engineering structures and even art and music in the physical world. However, now more than twenty years after [Koza, 1992], Evolutionary Computing has had less success at evolving the programs which control our computers.

The success of Genetic Programming [Koza, 1992] is well known. It has evolved predictive models and classifiers [Pappa and Freitas, 2004; Bhowan *et al.*, 2013] over a huge range of applications, from predicting human endeavours (finance [Neely and Weller, 1999; Dempster and Jones, 2000; Tsang and Li, 2002], insurance [Langdon, 1999]) to the outcome of breast cancer [Langdon and Buxton, 2004] and drug research [Langdon and Barrett, 2004]. In engineering it has been used to save energy in steel manufacture [Kovacic and Sarler, 2014] and modelling microscopic particulate pollution [Kovacic *et al.*, 2013]. It has been used to design electronic circuits [Koza *et al.*, 1999; Koza and Bennett III, 1999] radio antennas [Hornby *et al.*, 2011; Baker *et al.*, 2010], to aid building architects [O'Reilly and Hemberg, 2007] and in civil engineering [Baykasoglu *et al.*, 2008; Najafzadeh and Barani, 2011; Mirzahosseini *et al.*, 2011] and the environment [Savic *et al.*, 1999; Brumby *et al.*, 2001].

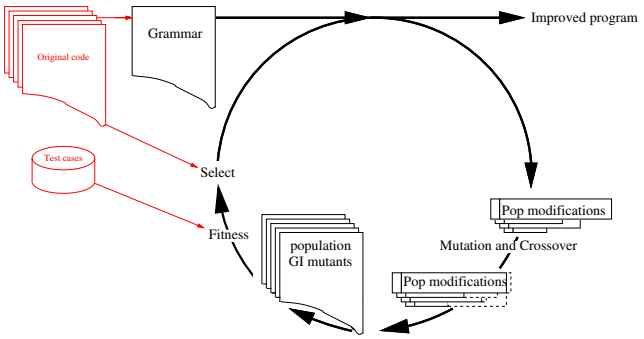


Fig. 1. Evolutionary Computation cycle adapted for Genetic Improvement (GI) of manually written C code (left). The grammar (or AST) tries to ensure many mutants compile, run, and terminate. Fitness is given by comparing with the original code run on the same test cases. (Figure from [Langdon and Petke, 2016].)

The next section introduces Genetic Improvement (GI) whilst Section III gives more details on how it uses Evolutionary Computing. Section IV asks how EC might further improve GI and we challenge EC to lead further down the road to Artificial Intelligence in general and to Automatic Programming in particular. Section V discusses the existing strengths and weakness of GI, whilst Section VI suggests ways that EC might improve it. Finally Section VII mentions some ways that GI researchers might help EC before we conclude (Section VIII).

## II. Perspective: What Genetic Improvement has done so far

Although Evolutionary Computing has had many successes in software engineering [Harman, 2007], it has had less success generating new software. However it has recently started to be used for improving existing programs written by people. Genetic Programming has been used to automatically fix bugs [Arcuri, 2011; Weimer *et al.*, 2010]. Although still controversial in some software engineering circles, Weimer and Forrest [Forrest *et al.*, 2009] kicked down the door and showed automatic bugfixing could be possible, whereas the software engineering community appear to have ignored the possibility until it was shown Evolutionary Computing could fix real bugs in real programs. (Albeit some bugs, not necessarily all of them.) Given this impetuous by EC,

Computer Science, University College London, London, WC1E 6BT, UK.
Computer Science, University of Stirling, Stirling, FK9 4LA, UK

[1] E.g. Melanism in the Peppered Moth during the $19^{th}$ and $20^{th}$ centuries in England.

automatic bug repair [Kessentini *et al.*, 2011] is now a very active research topic in software engineering.

Le Goues' [Le Goues *et al.*, 2012] prize winning work on bug fixing is far from the only prize winning work using genetic programming to improve existing code. Last year saw the first international workshop on genetic improvement and this year will see the second and also genetic improvement represented at this conference, as well as various local GI events. As well as fixing errors, GI has been demonstrated speeding up code [Langdon and Harman, 2015b] generating [Langdon and Harman, 2010] and improving parallel code [Langdon and Harman, 2014; Langdon *et al.*, 2014; Langdon *et al.*, 2015; Langdon and Harman, 2015a], automatically specialising programs [Petke *et al.*, 2014], reducing energy [Schulte *et al.*, 2014a; Bruce, 2015] and memory consumption [Wu *et al.*, 2015]. Indeed there is great interested in multi-objective approaches where evolution might generate a Pareto front [Harman *et al.*, 2012] allowing the designer to trade-off different objectives (e.g. quality v. memory) indeed we might see the user being given the option of trading objectives according to circumstances. E.g. someone might want a mobile application's full functionality whilst their phone was plugged in at home but be prepared to accept lower response in return for longer battery life when disconnected from a power supply during the day.

## III. How Genetic Improvement Works

Although there are many approaches in detail, currently (see Figure 1) the general approach is to automatically pre-process the program source [2] to give either a simple grammar or description at the AST level and use that description to constrain the genetic operations to ensure children in the next generation do not have syntax errors. The individuals in the evolving population are typically changes to the software to be evolved. This makes them more compact. This is a bit like seeding the population [Langdon and Nordin, 2000] with the human written code rather than starting with a totally random population. Although typically the syntax is correct, a sizeable fraction of the mutants may not compile. This is almost always due to variables being moved out of scope. In some cases the mutations may be restricted to respect variable scope limits, allowing 100% of children to compile.

Typically how good a child is (i.e. its fitness) is found by compiling it, running it and then comparing its results with those of the original program.

Great progress has been made with using Evolutionary Computation to generate test suite for programs [Fraser and Arcuri, 2011]. It is common to be able to generate test cases that will cause the software under test to execute most of the paths within it (NB. not all combinations). However in the classic software engineering case, the automatic tests say how to run the program but have no way of knowing if it produced the right answer. (In software engineering this is known as the "Oracle Problem"). But notice how Evolutionary Computing side-steps the Oracle Problem. We have an oracle. We have the original code. Admittedly in some cases (e.g. bug fixing) the original code's answer may need some adjustment. But in many cases it is sufficient. We just want the improved code to give the same answer but be faster, take less resources, etc. As long as the fitness function can automatically qualitatively say if the code is better or not we can in principle automatically evolve code in the right direction. Even in the case of multi-objective evolution (MOEA), it may be possible to automatically score several objectives. E.g., as well as the program's run time, it may be possible to calculate the quality of a mutant's answer and compare it with the quality of the original code's answer. Thus allowing a MOEA to automatically evolve mutants of the original hand written code.

Running the mutated code is like the well established software engineering tool of mutation testing [Langdon *et al.*, 2010] (also called mutation analysis). As with mutation testing, typically you need to protect your system against badly behaved mutants. Therefore it is common to use either CPU or elapsed time limits to force termination. (Aborted programs seldom get high fitness). You may want to use some form of protection against mutants accessing data outside the bounds of arrays. Depending upon the range of allowed mutations, you may want to use virtual machines or some form of sand boxing to prevent damage by rogue mutants.

As with mutation testing, and indeed Evolutionary Algorithms (EAs), typically GI run time is dominated by fitness testing. Also sometimes the time taken to compile the mutants is also important. You may want to use Unix' `make` to ensure only the modified code is recompiled, pre-compile libraries (e.g. C `.h` files) [Langdon and Harman, 2015b], compile the whole population of mutants in one operation [Langdon *et al.*, 2015], or compile the population using multiple computers [Harding and Banzhaf, 2009].

## IV. The Automatic Programming Challenge to Evolutionary Computing

Artificial Intelligence has started to achieve impressive results. Twenty years ago IBM's deep blue showed traditional AI can play chess better than every human on the planet. In the last few years neural network based deep learning has made great strides and we now have cars driving themselves. Although improving, these still suffer from the the AI being hand built for a task. Evolutionary Computing roots are mixed with those of AI [Fogel, 1994, page 9], but will these advances leave EC behind? Even though single sequential processor CPU core speeds have stagnated, Moore's Law [Moore, 1965] continues to pump out computation. Does this mean the traditional complaint that EC is compute heavy is not relevant. Like neural networks, EC can readily use parallelism. Perhaps it is time to start to revisit EC's primary goal: the evolution of machine intelligence. This was GP's goal but two decades on it risks getting stuck in a symbolic regression cul-de-sac. If we could improve software and then make improvements on top of those and make more

[2] Evolution at the level of Java byte code [Lukschandl *et al.*, 1998; Orlov and Sipper, 2011] or indeed machine code binaries [Schulte *et al.*, 2010; Schulte *et al.*, 2014b; Schulte *et al.*, 2015] may also be possible.

improvements on those and so on, we might escape from the symbolic regression local attractor. However realistically what GI gives you is more like trying to evolve a human from an ape, rather than starting with a single celled amoeba. Genetic Improvement does not give human intelligence but it has been demonstrated on a few examples to be able to create better programs than people have. This is not to say that people could not have done better themselves but that the GI did better stating from where the people had got to and where they ran out of time/money/enthusism and stopped.

Possibly GI has been too ambitious so far, in that it has compared its artefacts with code generated by some of the brightest programmers on the planet on non-trivial tasks. A more mundane goal might be to compare with the bread-and-better tedious tasks which we now expect people to code. If the machines can do this, they will get cheaper and so free millions of human programmers for more interesting challenges. Should EC aim to evolve simple cheap boiler plate code better than the average jobbing programmer?

## V. Current Weaknesses of GI and Ways Forward

A number of issues have already been raised

- Is the new code legible?
  Is it maintainable?
  These are important points, which may make some reluctant to take up GI. [Fry *et al.*, 2012] try to answer them to some extent. In their study, they showed that the provision of automatically generated comments to accompany the source code changes made automatically generated bug repairs more maintainable, rather than less.

  As will be mentioned on at the end of Section VI-F, it is common to minimise the size of the code change. For example, in bug fixing delta debugging [Zeller, 1999] is often used to remove unneeded changes. Reducing the volume of code is often assumed to make it more comprehensible.

  Of course if GI is operating on machine code binaries (Section III) then we assume that we do not care about understanding the program (perhaps we do not even have the source code) even before it is debugged or its performance is automatically improved. Hence we need only try to understand the mutations from the academic point of view of understanding our evolutionary process.
- Does the evolved program work?
  Is it correct?
  It is possible to run the full gamete of software validation tools post evolution. It seems reasonable that these tools will work on artificial code as well as on hand written code.

  The original code remains available and the new code can be automatically compared with it. We had one case where the new and old code were run together and their answers automatically compared [Langdon and Harman, 2010]. We did this more than a million times. No difference was ever found.

  Do you ever test your code a million times? And check it gives the right answer? Every time?
- Who is liable if (when?) something goes wrong?
  Hmm its not clear that this is worse for GI than for any other part of the software tool chain.

  Compilers are not formally verified. They certainly can do unexpected things. However these days they are sufficiently good, that their behaviour has become the de facto definition of the language they compile.
- User acceptability.
  It might be suggested that people will not want to trust code that has been automatically generated. However Microsoft claim their Flash Fill (end of Section VII) has more than a hundred million potential users. In fact this insert of practical AI into the user experience has been widely welcomed.
- Benchmarks
  From a practical point of view, experts in Evolutionary Computing will want an easy route into genetic improvement. Part of this aught to be a set of simple to understand benchmark problems [Ang and Li, 2002] and their associated tools so that they can quickly get useful results without an unnecessarily steep learning curve. Claire Le Goues has made a start on this by making available her set of 105 bugs to be repaired as part of her GenProg tool. Something similar is needed for other forms of program improvement.

## VI. Stepping Stones on the Route Forward

There are five steps we need to go though before running a genetic programming system [Poli *et al.*, 2008, page 19]. A key challenge for EC is to decide if they are really suitable for genetic improvement. And if not to investigate alternatives.

### A. Representation

Traditionally in tree based GP the first two steps (choice of terminals, leaf nodes, and functions, internal tree nodes) define the problem representation. However we should also include considering the genetic operations. In EC we have a wealth of experience in devising representations.

An obvious requirement is that the representation should include at least one acceptable solution. So far with the programs modified and their new requirements it has not been difficult to ensure the existence of solutions. However it may be that some bug repairs have been less successful because they have restricted the range of modification they allow too much. Unfortunately the existence of a solution is not sufficient. We need the representation, fitness function and genetic operations to conspire together to make a fitness landscape which makes it practical to find a solution.

Most GI work has represented members of the population as changes to the target program's source code. This has the advantage that changes may be human readable but is it the best for evolution? Should we be looking at:

- Is the source the right target? Would Evolutionary Algorithms do better trying to modify the program trace or the sequence of instructions it executes?

- Much GI work has focused on industrial strength code written in C or C++. Would other languages better better? Is the source code the right target? Would intermediate levels like Java or .net byte code be more evolvable?
- What of genotype-phenotype mappings? Should EC use an intermediate mapping, perhaps based on Gruau's embryology [Gruau and Whitley, 1993; Gruau, 1996]?
- New mutation operators.
  GI has so far been restricted to operations like deleting a line of the target program and copying a line of code and and pasting it elsewhere.
- New crossover operators.
- Provably correct transformations.
  Right at the beginning [Ryan and Walsh, 1995], the then conservative nature of the parallel computing community effectively mandated only provably semantics preserving transformations be used to convert sequential to parallel code. The risk of mutated code doing something unwanted is still very much with us. Perhaps with now much faster ways to check for semantic equivalence GI should re-consider its fast and loose ways?
  Although SAT technology has progressed in leaps and bounds in the last ten years, in practice Evolutionary Computing might want to consider hybrid approaches in which only the most likely mutants are validated. Or indeed, formal methods are only used after evolution has finished.

### *B. Improving in Multiple Ways: EMOs*

Traditionally people have been able to optimise code for one objective (typically speed). It appears they are less able to optimise programs for non-traditional objectives like extending battery life. However machines may be able to automatically optimise non-traditional objectives provided suitable measurements (e.g. energy consumption) can be incorporated into the fitness function.

It is typical in engineering to seek a good trade-off between multiple conflicting objectives. Evolutionary Multiobjective Optimization (EMO), e.g. [Deb *et al.*, 2002], has been widely used (e.g. [Coello Coello and Cruz Cortes, 2005; Xue *et al.*, 2013]) and are increasingly being used in Search Based Software Engineering (e.g. [Langdon *et al.*, 2009]). Although very asymmetric objectives may be problematic [Langdon and Harman, 2014], EMOs offer the prospect of automatically optimising code for several objectives [Harman *et al.*, 2012], which may be difficult for manual coders.

### *C. Improving Code and its Validation: Coevolution*

Coevolution [Darwen and Yao, 2001] of code to pass the current test suite and simultaneous evolution of the tests to stretch the code [Hillis, 1992] has been considered but with little progress. However successful applications in financial modelling, presented at the recent UCL workshop on Genetic Improvement [Hemberg *et al.*, 2015], may encourage more research into using coevolution within GI.

### *D. Fitness Measure. Can GI use Surrogates?*

As mentioned in Section III, existing GI work performs selection by creating and running the mutant code and comparing its performance with that of the original code. This is computationally demanding and typically the end result of all this work is condensed into a single bit: does this mutant get children or not.

To reduce computational overhead and so allow bigger populations, usually GI uses random subsets selected from the complete test suite every generation. Assessing fitness on dynamic randomised subsets goes back to Gathercole [Gathercole and Ross, 1994]. (His DSS is used commercially [Foster, 2001] [Poli *et al.*, 2008, page 84].) [Langdon, 1998; Teller and Andre, 1997] advanced statistical arguments for choosing how many tests to use. However in practise, it appears feasible to use just a handful of tests provided they are randomly redrawn frequently.

The problem of computationally demanding fitness functions has frequently been encountered in Evolutionary Computation when dealing with real problems [Forrester *et al.*, 2008]. For example designing an aircraft to withstand lightening strikes might need running a complete three dimensional electrodynamic simulation of the aircraft. However some success has been reported by replacing expensive fitness functions with cheaper surrogates [Jin, 2011]. Surrogates may be applied when the underlying system has boundaries and discontinuities. Which gives hope that they may be applied to program spaces. As will be seen in Section VI-H, program spaces may be better behave than common prejudices suggest.

### *E. Setting key parameters values*

In EC it is well known that parameters like population size and mutation and crossover rates can make a huge difference to how successful a run will be. There has been no published studies of how parameters affect GI. Surely there is EC theory [Bäck, 1996] or experience [Ribeiro Filho *et al.*, 1994] which could be applied to the problem of evolving better programs?

### *F. Termination, Who is the result*

As with many non-trivial EC problems, the choice of when to terminate evolution is often dominated by the available compute resources. But again, perhaps there is EC theory and practise to be applied here. Should we be looking at restart strategies when the population (genotype or phenotype) appears to have converged? How can we reliable recognise premature convergence? There has been only a little work in GI on preventing re-exploration of the same solutions (via some form of tabu list [Langdon *et al.*, 2015]).

In EC (including GI) it is common to simply use the best individual in the last generation as the result of evolution. However, it is entirely feasible to store every mutant program and its associated fitness. Perhaps an earlier mutant might be chosen. We might want to use other criteria as well as fitness to choose the final mutant. For example we might opt for the

mutant which makes the fewest changes to the original code. In GI it is common to chose the mutant with the best fitness and then minimise it after evolution by removing changes one at a time and retaining only those essential for its improved performance.

### G. The Search Space

The global structure of program search spaces is little understood, partly due to the lack of tools for analysing their complex structure. A recent model, *local optima networks* [Ochoa *et al.*, 2014; Verel *et al.*, 2011] helps to fill this gap by providing a way of expressing search spaces as graphs where nodes are local optima under a given mutation operator; and edges represent probabilistic transitions with an explorative operator, such as a stronger perturbation or crossover [Ochoa *et al.*, 2015a]. Modelling landscapes as networks brings a new set of tools and metrics for analysing search spaces and the possibility of visualising them (see Figure 2). The global structure of several combinatorial spaces, such as the travelling salesman problem, has been thought to contain a big-valley or central-massif where many local optima exist. That is, the local optima are not randomly distributed, instead good solutions tend to cluster around the global optimum. However, recent studies have observed that, for solutions close to the global optimum, this structure breaks down into multiple valleys [Hains *et al.*, 2011; Ochoa *et al.*, 2015b; Ochoa and Veerapen, 2016] (see Figure 2). In the study of energy surfaces in theoretical chemistry these have been called multiple funnels [Doye *et al.*, 1999]. Multiple funnels implies that local optima are organised into clusters. We suggest that local optima networks can be used to analyse the global structure of program spaces. Important aspects to study are the distribution of local optima and their connectivity pattern. Do program spaces conform a big-valley? Do they divide into multiple valleys or funnels? Answering these questions will help to design more effective algorithms for traversing program search spaces.

### H. Neutral Networks

Traditionally the space of program mutations is regarded as very disjointed with few good programs. However actual experience with sizeable real-world programs [Schulte *et al.*, 2014b] (it may be small toy program are less robust) in equivalent mutants in mutation testing [Yao *et al.*, 2014] automatic bug repair and genetic improvement [Langdon and Petke, 2015] suggests that many changes do not affect programs at all. Indeed it may be that program spaces may not be as hard to search as expected. Neutral Networks have been studied in GP [Langdon and Poli, 1998; Banzhaf and Leier, 2005], Evolvable Hardware [Vassilev *et al.*, 2000], GI [Schulte, 2014], Artificial Life [Standish, 2003] as well as in Nature [Babajide *et al.*, 1997; van Nimwegen *et al.*, 1999].

A key challenge to EC is to consolidate prior theory on landscapes riddled with fitness neutral pathways and usefully apply it to real world programs.

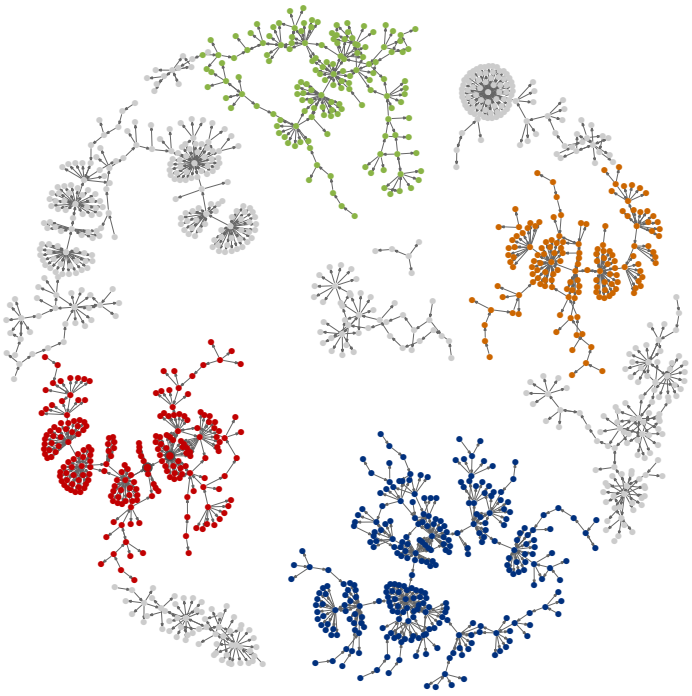

Fig. 2. Local optima network of a travelling salesman (TSP) instance with 666 cites. Nodes are local optima according to Lin-Kernighan, and edges represent probabilistic transitions with the L-K double-bridge perturbation operator. Colours identify the four largest connected components, which are related to the dominant funnels in the landscape. The red component (8 O'Clock) contains the global optima.

### I. Semantic Search of Open Source code (GitHub etc.)

Software engineers are now used to vast repositories of less than perfect but freely available program source code. At present this is only exploited manually. However we are starting to see the use of Evolutionary Computing both to automatically fix bugs by reusing free code [Ke *et al.*, 2015] and to transplant new functionality into existing code [Barr *et al.*, 2015; Marginean *et al.*, 2015]. Indeed EC can evolve new functionality for sizeable applications [Harman *et al.*, 2014; Jia *et al.*, 2015].

## VII. Benefits to Evolutionary Computing

As mentioned in the previous section (VI), there are several key challenges which are already core to the on going success of Evolutionary Computing. GI research contains many important practical real world problems which EC has already made substantial progress on and also Automatic Programming lies at the roots of Evolutionary Computing as a practical Artificial Intelligence technique. Whilst evolutionary routes to true AI are probably some way off, genetic improvement of existing code is here and now. Evolutionary Algorithms techniques offer the prospect of substantial progress both in the short term and towards more distance goals.

By sidestepping the Software Engineering Oracle Problem (mentioned in Section III) and using existing automatic test case generation tools, EC is close to having automated fitness functions. By substantially automating program modification and by re-using open source code, it may be that EC can

lift millions of programmers from their current error prone grind of mundane programming to a higher level, which is more like them saying what needs to be done without having to tell the computer how to do it [Langdon and Poli, 2002]. When the computer gets it wrong, the future response might be to update the test suite, rather than the code. Indeed we are already seeing user level programming based solely on examples [Gulwani *et al.*, 2012]. For example three years ago, Microsoft released "Flash Fill" within their excel 2013 spreadsheet. Flash Fill allows people to program excel purely from examples within their spreadsheet.

Surely Evolutionary Computing can do more!

## VIII. Conclusions

Today Automatic Programming, in restricted domains, is a reality for millions of users (see previous section). Genetic Improvement [Langdon, 2015] is firmly rooted in Evolutionary Computing and already offers a general way of extending sizeable existing programs by using genetic programming [Poli *et al.*, 2008] to evolve not complete programs but patches to them. In Section VI we have listed many deficiencies of current GI and hopes that the Evolutionary Computing experts may help. Perhaps the most urgent are the related problems of representation and the fitness landscape and also GA expertise in fitness surrogates may help radically reduce fitness evaluation effort. Being the second best way of solving any problem [Eiben and Smith, 2015] makes Evolutionary Computing very general but it is always at risk of being usurped in any domain by algorithms developed exclusively for that domain. To survive EC must keep conquering new challenges. Solving problems no one else can, or simply no one has been brave enough to try. The principle payment to EC (Section VII) may simply be the opportunity to work on truly challenging problems, relating back to the AI roots of EC and moving Automatic Programming towards the sort of programs that are well within the scope of manual methods.

## References


[Ang and Li, 2002] Kiam Heong Ang and Yun Li. An overview of benchmarking techniques for multi-objective evolutionary algorithms. In *Soft Computing and Industry: Recent Applications*. Springer, 2002.

[Arcuri, 2011] Andrea Arcuri. Evolutionary repair of faulty software. *Applied Soft Computing*, 11(4):3494–3514, 2011.

[Babajide *et al.*, 1997] Aderonke Babajide, Ivo L. Hofacker, Manfred J. Sippl, and Peter F. Stadler. Neutral networks in protein space, a computational study based on knowledge-based potentials of mean force. *Folding & Design*, 2:261–269, 20 August 1997.

[Bäck, 1996] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York, 1996.

[Baker *et al.*, 2010] James Baker, Nuri Celik, Nobutaka Omaki, Jill Kobashigawa, Hyoung-Sun Youn, and Magdy F. Iskander. On the design of integrated HF radar systems for homeland security applications. In *2010 IEEE International Conference on Wireless Information Technology and Systems (ICWITS)*, 28 October-September 3 2010.

[Banzhaf and Leier, 2005] Wolfgang Banzhaf and Andre Leier. Evolution on neutral networks in genetic programming. In Tina Yu et al., editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 14, pages 207–221. Springer, 12-14 May 2005.

[Barr *et al.*, 2015] Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke. Automated software transplantation. In Tao Xie and Michal Young, editors, *International Symposium on Software Testing and Analysis, ISSTA 2015*, pages 257–269, Baltimore, USA, 14-17 July 2015. ACM. ACM SIGSOFT Distinguished Paper Award.

[Baykasoglu *et al.*, 2008] Adil Baykasoglu, Hamza Gullu, Hanifi Canakci, and Lale Ozbakir. Prediction of compressive and tensile strength of limestone via genetic programming. *Expert Systems with Applications*, 35(1-2):111–123, 2008.

[Bhowan *et al.*, 2013] Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Trans. EC*, 17(3):368–386.

[Bruce, 2015] Bobby R. Bruce. Energy optimisation via genetic improvement A SBSE technique for a new era in software development. In William B. Langdon et al., editors, *Genetic Improvement 2015 Workshop*, pages 819–820, Madrid, 11-15 July 2015. ACM.

[Brumby *et al.*, 2001] S. P. Brumby, J. J. Bloch, N. R. Harvey, J. Theiler, S. Perkins, A. C. Young, and J. J. Szymanski. Evolving forest fire burn severity classification algorithms for multi-spectral imagery. In Sylvia S. Shen and Michael R. Descour, editors, *In Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VII, Proceedings of SPIE*, volume 4381, pages 236–245, 2001.

[Coello Coello and Cruz Cortes, 2005] Carlos A. Coello Coello and Nareli Cruz Cortes. Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6(2):163–190, June 2005.

[Darwen and Yao, 2001] Paul J. Darwen and Xin Yao. Why more choices cause less cooperation in iterated prisoner's dilemma. In *CEC-2001*, volume 2, pages 987–994. IEEE, 27-30 May 2001.

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

[Deb *et al.*, 2002] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr 2002.

[Dempster and Jones, 2000] M. A. H. Dempster and C. M. Jones. A real-time adaptive trading system using genetic programming. *Quantitative Finance*, 1:397–413, 2000.

[Doye *et al.*, 1999] J P K Doye, M A Miller, and D J Wales. The double-funnel energy landscape of the 38-atom Lennard-Jones cluster. *Journal of Chemical Physics*, 110(14):6896–6906, 1999.

[Eiben and Smith, 2015] Agoston E. Eiben and Jim Smith. From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482.

[Fogel, 1994] David B. Fogel. An introduction to simulated evolutionary optimization. *IEEE trans. on Neural Networks*, 5(1):3–14, Jan 1994.

[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 et al., editors, *GECCO '09*, pages 947–954, Montreal, 8-12 July 2009. ACM. Best paper.

[Forrester *et al.*, 2008] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering Design via Surrogate Modelling: A Practical Guide*. Wiley, 2008.

[Foster, 2001] James A. Foster. Review: Discipulus: A commercial genetic programming system. *Genetic Programming and Evolvable Machines*, 2(2):201–203, June 2001.

[Fraser and Arcuri, 2011] Gordon Fraser and Andrea Arcuri. 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)*, pages 416–419. ACM, September 5th - 9th 2011.

[Fry *et al.*, 2012] Zachary P. Fry, Bryan Landau, and Westley Weimer. A human study of patch maintainability. In Zhendong Su, editor, *Proceedings of the 2012 International Symposium on Software Testing and Analysis, ISSTA 2012*, pages 177–187, Minneapolis, MN, USA, 15-20 July 2012. ACM.

[Gathercole and Ross, 1994] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor et al., editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321, Jerusalem, 9-14 October 1994. Springer-Verlag.

[Gruau and Whitley, 1993] Frederic Gruau and Darrell Whitley. Adding learning to the cellular development process: a comparative study. *Evolutionary Computation*, 1(3):213–233, 1993.

[Gruau, 1996] Frederic Gruau. On using syntactic constraints with genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors,

*Advances in Genetic Programming 2*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, USA, 1996.

[Gulwani *et al.*, 2012] Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105, August 2012.

[Hains *et al.*, 2011] D R Hains, L D Whitley, and A E Howe. Revisiting the big valley search space structure in the TSP. *Journal of the Operational Research Society*, 62(2):305–312, 2011.

[Harding and Banzhaf, 2009] Simon L. Harding and Wolfgang Banzhaf. Distributed genetic programming on GPUs using CUDA. In Ignacio Hidalgo et al., editors, *Workshop on Parallel Architectures and Bioinspired Algorithms*, pages 1–10, Raleigh, NC, USA, 13 September 2009. Universidad Complutense de Madrid.

[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, 2007] Mark Harman. The current state and future of search based software engineering. In Lionel Briand and Alexander Wolf, editors, *Future of Software Engineering 2007*, pages 342–357, 2007.

[Hemberg *et al.*, 2015] Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May O'Reilly. Tax non-compliance detection using co-evolution of tax evasion risk and audit likelihood. In Katie Atkinson and Ted Sichelman, editors, *Proceedings of the 15th International Conference on Artificial Intelligence and Law, ICAIL-2015*, pages 79–88, San Diego, USA, 2015. ACM.

[Hillis, 1992] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Christopher G. Langton et al., editors, *Artificial Life II*, volume X of *Sante Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, 1992.

[Hornby *et al.*, 2011] Gregory. S. Hornby, Jason D. Lohn, and Derek S. Linden. Computer-automated evolution of an X-band antenna for NASA's space technology 5 mission. *Evolutionary Computation*, 19(1):1–23, Spring 2011.

[Jia *et al.*, 2015] Yue Jia, Mark Harman, William B. Langdon, and Alexandru Marginean. Grow and serve: Growing Django citation services using SBSE. In Shin Yoo and Leandro Minku, editors, *SSBSE 2015 Challenge Track*, volume 9275 of *LNCS*, pages 269–275, Bergamo, Italy, 5-7 September 2015.

[Jin, 2011] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[Ke *et al.*, 2015] Yalin Ke, Kathryn T. Stolee, Claire Le Goues, and Yuriy Brun. Repairing programs with semantic code search. In Lars Grunske and Michael Whalen, editors, *30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, Lincoln, Nebraska, USA, November 9-13 2015.

[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.

[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.

[Kovacic *et al.*, 2013] Miha Kovacic, Sandra Sencic, and Uros Zuperl. Genetic programming and artificial neural network modeling of PM10 emission close to a steel plant. *RMZ – Materials and Geoenvironment*, 60(1):9–16, July 2013.

[Koza and Bennett III, 1999] John R. Koza and Forrest H Bennett III. Automatic synthesis, placement, and routing of electrical circuits by means of genetic programming. In Lee Spector et al., editors, *Advances in Genetic Programming 3*, chapter 6, pages 105–134. MIT Press, Cambridge, MA, USA, June 1999.

[Koza *et al.*, 1999] John R. Koza, David Andre, Forrest H Bennett III, and Martin Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufman, April 1999.

[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 Buxton, 2004] W. B. Langdon and B. F. Buxton. Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257, September 2004.

[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 and Harman, 2014] William B. Langdon and Mark Harman. Genetically improved CUDA C++ software. In Miguel Nicolau et al., editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 87–99, Granada, Spain, 23-25 April 2014. Springer.

[Langdon and Harman, 2015a] William B. Langdon and Mark Harman. Grow and graft a better CUDA pknotsRG for RNA pseudoknot free energy calculation. In William B. Langdon et al., editors, *Genetic Improvement 2015 Workshop*, pages 805–810, Madrid, 11-15 July 2015. ACM.

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

[Langdon and Nordin, 2000] W. B. Langdon and J. P. Nordin. Seeding GP populations. In Riccardo Poli et al., editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 304–315, Edinburgh, 15-16 April 2000. Springer-Verlag.

[Langdon and Petke, 2015] William B. Langdon and Justyna Petke. Software is not fragile. In Paul Bourgine and Pierre Collet, editors, *Complex Systems Digital Campus E-conference, CS-DC'15*, Proceedings in Complexity, page Paper ID: 356. Springer, September 30-October 1 2015. Invited talk, Forthcoming.

[Langdon and Petke, 2016] William B. Langdon and Justyna Petke. Genetic improvement. IEEE Software Blog, February 3 2016.

[Langdon and Poli, 1998] W. B. Langdon and R. Poli. Why ants are hard. In John R. Koza et al., editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Langdon and Poli, 2002] W. B. Langdon and Riccardo Poli. Removal of the man-machine interface bottleneck "Do what I ment not what I said". In *Grand Challenges for Computing*, Edinburgh, 24-26 November 2002. Discussion paper.

[Langdon *et al.*, 2009] W. B. Langdon, Mark Harman, and Yue Jia. Multi objective higher order mutation testing with GP. In Guenther Raidl et al., editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, page 1945, Montreal, 8-12 July 2009. ACM.

[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 *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 et al., 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 *et al.*, 2015] William B. Langdon, Brian Yee Hong Lam, Justyna Petke, and Mark Harman. Improving CUDA DNA analysis software with genetic programming. In Sara Silva et al., editors, *GECCO '15: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 1063–1070, Madrid, 11-15 July 2015. ACM.

[Langdon, 1998] William B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston, 1998.

[Langdon, 1999] W. B. Langdon. Genetic programming approach to benelearn 99: I. In Peter van der Putten and Maarten van Someren, editors, *The Benelearn 1999 Competition*, page 3.5, Sociaal-Wetenschappelijke Informatica, Universiteit van Amsterdam, 2 November 1999.

[Langdon, 2015] 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.

[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.

[Lukschandl *et al.*, 1998] Eduard Lukschandl, Magus Holmlund, and Eirk Moden. Automatic evolution of Java bytecode: First experience with the Java virtual machine. In Riccardo Poli et al., editors, *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*, pages 14–16, Paris, France, 14-15 April 1998. CSRP-98-10, The University of Birmingham, UK.

[Marginean *et al.*, 2015] Alexandru Marginean, Earl T. Barr, Mark Harman, and Yue Jia. Automated transplantation of call graph and layout features into Kate. In Yvan Labiche and Marcio Barros, editors, *SSBSE*, volume 9275 of *LNCS*, pages 262–268, Bergamo, Italy, September 5-7 2015. Springer.

[Mirzahosseini *et al.*, 2011] Mohammad Reza Mirzahosseini, Alireza Aghaeifar, Amir Hossein Alavi, Amir Hossein Gandomi, and Reza Seyednour. Permanent deformation analysis of asphalt mixtures using soft computing techniques. *Expert Systems with Applications*, 38(5):6081–6100, 2011.

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

[Najafzadeh and Barani, 2011] M. Najafzadeh and Gh.-A. Barani. Comparison of group method of data handling based genetic programming and back propagation systems to predict scour depth around bridge piers. *Scientia Iranica*, 18(6):1207–1213, December 2011.

[Neely and Weller, 1999] Christopher J. Neely and Paul A. Weller. Technical trading rules in the european monetary system. *Journal of International Money and Finance*, 18(3):429–458, 1999.

[Ochoa and Veerapen, 2016] G. Ochoa and N. Veerapen. Deconstructing the big valley search space hypothesis. In *European Conference on Evolutionary Computation in Combinatorial Optimisation(EvoCOP 2016)*, volume 9595 of *LNCS*. Springer, 2016.

[Ochoa *et al.*, 2014] Gabriela Ochoa, Sebastien Verel, Fabio Daolio, and Marco Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. In Hendrik Richter and Andries Engelbrecht, editors, *Recent Advances in the Theory and Application of Fitness Landscapes*, volume 6 of *Emergence, Complexity and Computation*, pages 233–262. Springer, 2014.

[Ochoa *et al.*, 2015a] G. Ochoa, F. Chicano, R. Tinos, and D. Whitley. Tunnelling crossover networks. In *GECCO 2015*, pages 449–456. ACM.

[Ochoa *et al.*, 2015b] G. Ochoa, N. Veerapen, D. Whitley, and E. K. Burke. The multi-funnel structure of TSP fitness landscapes: A visual exploration,. In *Artificial Evolution - 12th International Conference, Evolution Artificielle, EA*, 2015.

[O'Reilly and Hemberg, 2007] Una-May O'Reilly and Martin Hemberg. Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines*, 8(2):163–186, June 2007. Special issue on developmental systems.

[Orlov and Sipper, 2011] Michael Orlov and Moshe Sipper. Flight of the FINCH through the Java wilderness. *IEEE Transactions on Evolutionary Computation*, 15(2):166–182, April 2011.

[Owen *et al.*, 1990] R. B. Owen, R. Crossley, T. C. Johnson, D. Tweddle, I. Kornfield, S. Davison, D. H. Eccles, and D. E. Engstrom. Major low levels of Lake Malawi and their implications for speciation rates in cichlid fishes. *Proceedings of the Royal Society (B)*, 240(1299):519–553, 1990.

[Pappa and Freitas, 2004] Gisele L. Pappa and Alex A. Freitas. Towards a genetic programming algorithm for automatically evolving rule induction algorithms. In Johannes Furnkranz, editor, *ECML/PKDD 2004 Proceedings of the Workshop W8 on Advances in Inductive Learning*, pages 93–108, Pisa, Italy, 20-24 September 2004.

[Petke *et al.*, 2014] 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 Miguel Nicolau et al., editors, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 137–149, Granada, Spain, 23-25 April 2014. Springer.

[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).

[Ribeiro Filho *et al.*, 1994] Jose L. Ribeiro Filho, Philip C. Treleaven, and Cesare Alippi. Genetic-algorithm programming environments. *Computer*, 27(6):28, June 1994.

[Ryan and Walsh, 1995] Conor Ryan and Paul Walsh. Automatic conversion of programs from serial to parallel using genetic programming - the paragen system. In *Proceedings of ParCo'95*. North-Holland, 1995.

[Savic *et al.*, 1999] Dragan A. Savic, Godfrey A. Walters, and James W. Davidson. A genetic programming approach to rainfall-runoff modelling. *Water Resources Management*, 13(3):219–231, June 1999.

[Schulte *et al.*, 2010] Eric Schulte, Stephanie Forrest, and Westley Weimer. Automated program repair through the evolution of assembly code. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 313–316, Antwerp, 20-24 September 2010.

[Schulte *et al.*, 2014a] Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, and Westley Weimer. Post-compiler software optimization for reducing energy. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'14*, pages 639–652, Salt Lake City, Utah, USA, 1-5 March 2014. ACM.

[Schulte *et al.*, 2014b] 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.

[Schulte *et al.*, 2015] Eric Schulte, Westley Weimer, and Stephanie Forrest. Repairing COTS router firmware without access to source code or test suites: A case study in evolutionary software repair. In William B. Langdon et al., editors, *Genetic Improvement 2015 Workshop*, pages 847–854, Madrid, 11-15 July 2015. ACM. Best Paper.

[Schulte, 2014] Eric Schulte. *Neutral Networks of Real-World Programs and their Application to Automated Software Evolution*. PhD thesis, University of New Mexico, Albuquerque, USA, July 2014.

[Standish, 2003] Russell K. Standish. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(2):167–175, 2003.

[Teller and Andre, 1997] Astro Teller and David Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In John R. Koza et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Tsang and Li, 2002] Edward P. K. Tsang and Jin Li. EDDIE for financial forecasting. In Shu-Heng Chen, editor, *Genetic Algorithms and Genetic Programming in Computational Finance*, chapter 7, pages 161–174. Kluwer Academic Press, 2002.

[van Nimwegen *et al.*, 1999] Erik van Nimwegen, James P. Crutchfield, and Martijn Huynen. Neutral evolution of mutational robustness. *Proc. Natl. Acad. Sci*, 96:9716–9720, August 1999. USA.

[Vassilev *et al.*, 2000] Vesselin K. Vassilev, Dominic Job, and Julian F. Miller. Towards the automatic design of more efficient digital circuits. In Jason Lohn et al., editors, *The Second NASA/DoD workshop on Evolvable Hardware*, pages 151–160, Palo Alto, California, 13-15 July 2000. IEEE Computer Society.

[Verel *et al.*, 2011] S. Verel, G. Ochoa, and M. Tomassini. Local optima networks of NK landscapes with neutrality. *IEEE Transactions on Evolutionary Computation*, 15(6):783–797, 2011.

[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.

[Wu *et al.*, 2015] Fan Wu, Westley Weimer, Mark Harman, Yue Jia, and Jens Krinke. Deep parameter optimisation. In Sara Silva et al., editors, *GECCO '15: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 1375–1382, Madrid, 11-15 July. ACM.

[Xue *et al.*, 2013] Bing Xue, Mengjie Zhang, and Will N. Browne. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE Transactions on Cybernetics*, 43(6):1656–1671, Dec 2013.

[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 *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 919–930, Hyderabad. ACM.

[Zeller, 1999] Andreas Zeller. Yesterday, my program worked. today, it does not. why? In Oscar Nierstrasz and Michel Lemoine, editors, *ESEC/FSE '99*, pages 253–267, Toulouse, Sept. 6–10 1999. Springer.