Genetic Programming and Evolvable Machines (2023) 24:19
https://doi.org/10.1007/s10710-023-09467-x

Jaws 30

W. B. Langdon[1]

Published online: 22 November 2023

Abstract
It is 30 years since John R. Koza published "Jaws", the first book on genetic programming [Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)]. I recount and expand the celebration at GECCO 2022, very briefly summarise some of what the rest of us have done and make suggestions for the next thirty years of GP research.

Keywords Genetic programming · Genetic improvement · Modularity · Scaling · Parallel computing

1 Introduction

An evening at the 2022 GECCO conference was devoted to celebrating the thirtieth anniversary of the publication of John Koza's book "Genetic Programming: On the programming of computers by means of natural selection" [1].[1] Indeed that is the purpose of this special issue of Genetic Programming and Evolvable Machines. I hope to put my own spin on and fill out points raised in that panel discussion (which

[1] Named after Charles Darwin's 1859 foundational book "On the Origin of Species by Means of Natural Selection" [2] which contains huge volumes of evidence (for example gathered on his five year voyage around the world [3]) in support of his scientific theory of evolution, which after a struggle was eventually accepted as the explanation for biology.

Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection.

Commentaries to this article can be found at: https://doi.org/10.1007/s10710-023-09468-w; https://doi.org/10.1007/s10710-023-09469-9; https://doi.org/10.1007/s10710-023-09470-2; https://doi.org/10.1007/s10710-023-09471-1; https://doi.org/10.1007/s10710-023-09472-0; https://doi.org/10.1007/s10710-023-09473-z.

W. B. Langdon
w.langdon@cs.ucl.ac.uk

1 London, UK

was recorded and is available on line[2]). I should stress this is not a survey of GP and that many valuable contributions are omitted. Similarly many digressions are placed in footnotes and there are hyper links to online articles in Wikipedia etc.

1.1 The book

Dr. Amy Brand, Director of The MIT Press, was clearly delighted that John Koza had chosen MIT Press to publish the first book on genetic programming [1] (see Figs. 1 and 2). She says it ”was one of the seeds from which sprang a whole ecosystems of books and journals at the intersection of computer and biological sciences for the MIT Press.” Adding it “is still available and selling in print-on-demand. That’s quite solid for a specialized and ground-breaking work in computer science from 1992.”[3]

John Koza said that the motivation for the book was his team in the preceding five years had published GP solutions to 81 diverse problems common to artificial intelligence, machine learning and knowledge based systems. They had shown that instead of, as had previously been done, using a solution technique devoted to each benchmark, a single evolutionary computing technique (now named Genetic Programming[4]) could solve them all[5][6]. However the GP solutions were published in widely disperse conference venues. The goal of the book was to convince everyone that 1) a single technique could solve many diverse problems and 2) they could all be recast as the problem of searching for (and finding) a computer program. Whereas previous solutions had often used (non-evolutionary) search but used a representation, e.g. graph, grammar, network, often purpose built for each benchmark. The size of the book[7] stems from the need to convince people that GP is a general solution. Whereas everyone who first comes to programming knows that programming languages are exceedingly picky about insisting they get everything, every comma, every semicolon, in the right place: so how could random stand a hope? Hence a

[2] A Conversation with John Koza, 30 years after the publication of Genetic Programming Sunday, July 10, 18:00-20:00 2022 https://whova.com/portal/webapp/gecco_202207/Agenda/2516377

[3] John Koza’s publications have been at the top of the list of publications downloaded via the genetic programming bibliography since 2006, when download statistics were first gathered.

[4] The name Genetic Programming was suggested by David E. Goldberg. John Koza said he was originally reluctant to use the name but came to realise it was a brilliant choice.

[5] John Koza has previously likened GP’s success with early machine learning benchmarks with Sherman’s march through Georgia in 1864, which helped end the four year USA civil war.

[6] In the late 1990s Peter Nordin reported similar success with his linear genetic programming on the UCI machine learning benchmarks.

[7] The first genetic programming book was colloquially known as “Jaws” after the 1975 Hollywood movie of the same name, were the shark appears to get progressively bigger throughout the film. In a similar way Koza remarked that as each new GP experiment was covered, the book got bigger, eventually exceeding 800 pages. The three successing GP books, are similarly known as Jaws 2 [4], Jaws 3 [5] and Jaws 4 [6], all four are each accompanied by an hour long video [7–10] (now available on YouTube and www.human-competitive.org). In 2009, John Koza gave a seminar at Stanford summarising his GP work which was recorded and is also available on YouTube [11].

substantial book, backed by a video, would be necessary to convince a skeptical public.[8]

1.2 The man

John R. Koza was born 1944 and did both his undergraduate degree and PhD at the University of Michigan in Ann Arbor, studying mathematics and, the then newfangled, computer science. He reports great interest in playing games including computer games, with students and faculty, for example, John H. Holland. As with John Holland's other students[9], he was well versed in John Holland's genetic algorithms.

1.3 The millionaire

John Koza graduated from the University of Michigan in December 1972, and using his mathematical skills in combinatorics, probability and game playing he joined a lottery company which printed games on paper which were sold at petrol stations and supermarkets. In 1974 he and a colleague formed their own company, Scientific Games Inc., to exploit John Koza's invention of a secure way of printing scratch off lottery tickets. They successfully lobbied various USA states to allow them to run the state's lottery[10]. By 1978 the technology of printing had moved on and they jettisoned their own technique in favour of more flexible computer based printing. In 1987, having made his fortune, he returned to research.

1.4 The researcher

From about 1987 until 2005, John Koza devoted himself to research, applying genetic algorithms to the discovery of computer programs (GP). He published some 208 items, predominately papers but also book chapters, technical reports, proceedings, etc. and of course Jaws [1] and the three follow-up up door stoppers [4–6] and the four accompanying videos [7–10]. Initially the genetic programming systems were written in Lisp, although later implementations where in C, e.g. [22].

There were GP workshops associated with the International Conference on Genetic Algorithms, ICGA-93[11] and again in the summer of 1995 at ICGA-95 and ICML-95. In the fall, John R. Koza and Eric V. Siegel organised a GP event with

[8] There is a growing body of work, such as automatic bug fixing [12] and genetic improvement [13], that shows ordinary programs are not fragile [14–21]. The misplaced semicolon problem refers to the source code syntax as understood by the language compiler (another computer program). Since the syntax is formally defined, computer generated mutations can be automatically written to be syntactically correct. If mutated code compiles, it often runs and produces an answer which can be fed into a fitness function.

[9] John Holland's PhD students include: Stephanie Forrest, Tommaso F. Bersano-Begey, Melanie Mitchell, Tom Westerdale, Lashon Booker, Ted Codd, Clare Congdon, Dave Goldberg, Annie Wu, Ken De Jong, Leeann Fu, Rick Riolo, Chris Langton, Robert Reynolds, Bernie Zeigler and John Koza.

[10] By 2009 the combined profits to the USA state governments which permitted lotteries had reached $17.6 billion.

[11] ICGA had strong links with John Holland's students.

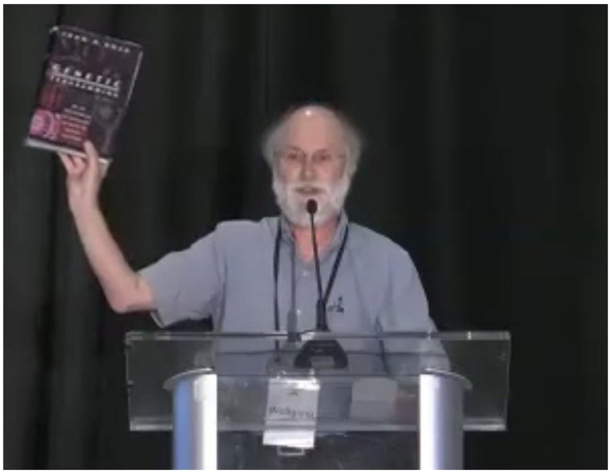

Fig. 1 Prof. Dr. Wolfgang Banzhaf **holding his copy of "Genetic Programming: On the programming of computers by means of natural selection" (Jaws) 834 pages [1] at the GECCO 2022 celebration of 30 years after its publication (Wolfgang says he was told that his copy was the first one sold by the bookshop in Boston.)**

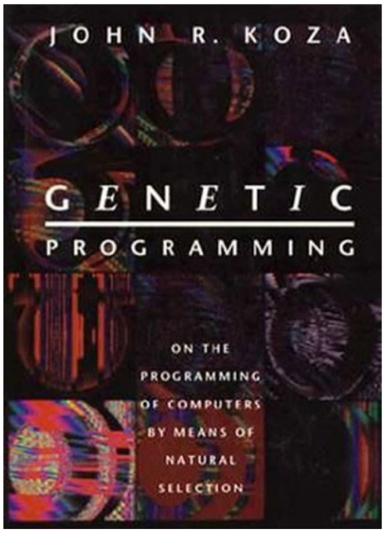
JOHN R. KOZA
GENETIC
PROGRAMMING
ON THE PROGRAMMING OF COMPUTERS BY MEANS OF NATURAL SELECTION

Fig. 2 At 834 pages, the first genetic programming book [1] weighs in at 4lb 2oz

the 1995 Fall Symposium of the AAAI in MIT. In 1994 Kim Kinnear had launched the "Advances in Genetic Programming" edited book series published by MIT Press [23–25]. But, since ICGA was a biannual conference, there was no ICGA conference in 1996, and instead it was the right time to launch the first GP conference [26]. One of the rules laid down at GP-96, was the absolute need for independent peer review.

July 1997 saw the return of ICGA-97, carefully scheduled a few days after the second GP conference, GP-97 [27], so attendance at both was encouraged. Again there was no ICGA in 1998, instead at GP-98 [28] there were serious discussions about combining the growing number of evolutionary computing conferences. John Koza in particular felt that the separate EC events were splitting EC into separate communities, and that the balkanisation of EC, did not make sense to people outside, particularly to funding bodies. And that this divergence was hurting the field. So at GP-98 there were negotiations about unifying, particularly: the Evolutionary Programming Society conference (EP), the IEEE's WCCI/ICEC, GP, ICGA, and the International Workshop on Learning Classifier Systems (IWLCS). These were only partially successful, leading in 1999 to the formation of the duopoly of CEC 1999 [29] and GECCO 1999 [30]. Of the european evolutionary computing conferences, only the IEE's Galesia elected to join CEC. PPSN,[12]ICANNGA and the newly established EuroGP[13] [31] continued as before[14].

Again John Koza's organisational skills came to the for, with him helping to draft the byelaws for GECCO. These ensure it has a federal "big tent" structure, whereby none of its constituent groups would feel left out or put down by the others.

Having progressed genetic programming to the point were it could be described as a routine invention machine [6, 32, 33], John Koza turned to public service and electoral reform and in 2006 founded National Popular Vote.

1.5 The public benefactor

In 2004 John Koza started the annual "Humies" awards for human-competitive results produced by genetic and evolutionary computation. He continues to fund the cash prizes. The finals are held each year as part of the GECCO conference.

Since 2016 he has endowed Michigan State University with the first chair in genetic programming in the United States (held by Prof. Dr. Wolfgang Banzhaf).

1.6 Pre-history

At GECCO-2022 the question of research before genetic programming was raised. John Koza pointed out that by 1987 the field of Genetic Algorithms was already well established[15]. There had been early experiments on machine learning in Columbia

[12] Parallel Problem Solving from Nature (PPSN) had started in Germany in 1990. It is a also a biannual conference on evolutionary computing and, although based in Europe, it was held on alternating years with ICGA. Like Genetic Programming, PPSN was also named by Dave Goldberg.

[13] The First European Workshop on Genetic Programming had been held in 1998 in Paris, with the help of EvoNet, the EU Network of Excellence in Evolutionary Computing.

[14] In 2003 John Koza listed 25 international conferences and workshops primarily devoted to the various forms of evolutionary computation. Many are still held annual or biannually, and some have combined. In most cases the proceedings are still available, often on line.

[15] In addition to genetic algorithms, there is early work on evolutionsstrategie in Germany by Ingo Rechenberg and Hans-Paul Schwefel, and in the USA on evolutionary programming by Larry Fogel.

[34] and Manchester [35][16] universities. However John Koza traced Evolutionary Computing back to Alan Turing. He said Turing's 1948 paper on machine intelligence [36, 37] suggested three routes to machine intelligence: 1) knowledge based, 2) based on logic (as would be expected of a mathematician), but John Koza highlighted the third: 3) in which machine intelligence was based on evolution. Although he pointed out it did not use crossover (which was added by John Holland).

1.7 Advice for the future

Another question raised at GECCO-2022 was did John Koza have advice for new researchers. His answer was researchers must keep current, i.e., keep up to date with research, but not just in your area but with research in general. Take an interdisciplinary approach. He stressed be open to ideas from elsewhere, particularly from Biology.

John Koza's heuristic (perhaps common to all John Holland's students) was to ask himself "What would John Holland do?" to which the answer was often: John Holland would respond with his own question, "What does Nature do?" John Koza's particular example was: how did Nature evolve from microscopic organisms (like bacteria) which have genes for creating may be about 500 proteins to multicellural organisms (e.g. us) which have genes for creating about 20 000 proteins. He reported asking this question around the Stanford School of Medicine.

The example John Koza quoted was the evolution of Myoglobin and Hemog lobin, which is thought to have occurred via gene duplication and subsequent specialisation. The idea being: "accidental" copying of parts of DNA sequences is common.[17] Once a species has two copies of a vital gene, it may be free to tinker with one. Since the other gene remains functional, the children with the duplicated gene remain viable and so some can survive long enough to carry both the working gene and the tinkered copy to the grand children. Over subsequent generations the two genes may diverge allowing the species to find new proteins which may help it survive. Susumu Ohno in his 1970 book [40] suggested that such gene duplication is a powerful mechanism in natural evolution. Indeed John Koza used it as inspiration [41] for his architecture-altering operations. These GP operations allow, not just the code within automatically defined functions (ADFs) [4] to evolve, but also their structure (e.g. which ADF calls which ADF) evolves [5, 42, 9, minute 10]. In terms of traditional AI, this can be thought of as dividing the whole problem into subcases and having an evolvable representation which facilitates not just the solution of the sub-problems but also their subsequent combination into a complete solution. Some form (or indeed many forms of) automatic problem decomposition is essential if any AI technique is to scale.

[16] Kilburn, Grimsdale and Sumner ran their experiments in machine learning and thinking on the world's first digital stored program computer the Manchester Mark 1.

[17] The evolution of repeating patterns in DNA due to crossover is common. Indeed crossover in GP can readily produce huge volumes of repeated code in trees [38, 39].

John Koza felt that in the 1960s the University of Michigan had had a wide ranging curriculum. He said computer scientists need to know about biology, language processing, psychology, information theory, electronic circuits, etc. However, this breadth has been lost from modern computer science curricula. Instead people should seek ideas from many places. He cited successful start ups in silicon valley, such as Adobe, which had come from co-working between two people with experience of newspaper publishing and another with a computer science background. Often in silicon valley success had come from partnerships of individuals with different experience. Alternatively, success may arise when different experience or many odd ideas are held by *one* person.

I would like to add, be ambitious in the problems you tackle. John Koza's impact, the impact of his book [1], stems from showing something widely viewed as *impossible* could be done. Before his work, the idea of automatically evolving a computer program was clearly ludicrous. Similarly, the idea of a computer fixing computer bugs was clearly impossible, until Stephanie Forrest et al. showed GP could do it [43]. Readers may remember Lewis Carroll's Alice and the White Queen [44] (Fig. 3), Alice reproaches the White Queen for some nonsense, saying it is clearly impossible, to which the White Queen responds that Alice should practice believing the impossible. My suggestion would be to an ambitions researcher that she should *do* the impossible. Claire Le Goues was a PhD student in 2009 [45, 46]. Fortunately her adviser did not tell her her idea was impossible. And so She and the team are famous, not because they completely solved the problem, but because they took something impossible and partially solved it. So that today the argument is not if it can be done, but what is the best way [12] to solve the previously impossible problem [47–49].

1.8 The ones that got away: missing gaps

John Koza was asked to muse on his less successful experiments. Two came to mind: FPGAs and GPUs.

1.8.1 Genetic programming and field programmable gate arrays, FPGAs

John Koza had hope to create a field programmable gate array (FPGA), which had all the likely to be useful program operations pre-loaded. An ultra fast evolved GP program would then simply be an evolvable way of linking these together.

In some ways this seams similar to Juille's [50] way of running a GP interpreter on the hugely parallel MasPar MP-2 computer. Although it had thousands of processing units, they each did the same one thing at the same time. Juille's brainwave was to say: since computing is cheap, we will discard most of it. (Simplifying), Juille built a tiny interpreter which ran on all processing elements one of a handful of GP operations. The different members of the GP population were spread across the processing elements. Each with its own program counter. If the interpreter was currently executing a GP op code that was not the one the GP individual wanted, it did nothing but wait. However the interpreter cycled round all possible GP op codes.

When it did reach the desired op code, that processor executed it and moved that GP individual's programme counter on by one. (The right hand side of Fig. 4 shows the same idea in the context of GPUs.)

It sounds hideously inefficient, but bear in mind the GP is getting useful works done, whereas mostly human programmers could not handle the MasPar MP-2's SIMD architecture efficiently at all. Secondly often in many high performance computers (HPCs), most of the time the processing elements are waiting for data to arrive and so spend most of their time spinning in idle loops. This turns on its head our common conception of computers. In HPC (and indeed GPUs, see Sect. 1.8.2), computing is often cheap compared to moving data. Indeed sometimes it can be more efficient to compute a value a second time, rather than store it and retrieve it later when it is needed[18].

In many cases FPGAs form the bed rock of evolvable hardware (EHW) [51, 52]. As well as offering a cheap and flexible alternative to dedicated integrated circuits (also known as application-specific integrated circuits, ASICs) they can be cost effective, particularly when only a limited number of chips will be needed. There are several examples where FPGAs have been used to run GP, e.g. [53–55].

1.8.2 Genetic programming and graphics cards, GPUs

In the early 2000s it was noticed that the graphics cards (GPUs) used to drive computer screens were becoming increasingly powerful parallel computing devices in their own right and so people started using them for other things.

Initially GPUs were designed just to rapidly render images on the computer's screen. To do this quickly (in real time) they comprised many parallel components all doing the same thing but for different parts of the screen. As the computer video games market took off, GPUs rapidly ramped up their processing abilities and power. Each parallel component became a fully functional processor, often with special support for operations common in graphics applications (such as reciprocal square root [56]). This was so that more of the parallel aspects of generating, rather than simply displaying, real time video could be devolved from the (serial) CPU to the (parallel) graphics card. As GPUs were often somewhat independent of the end users' computer mother board, keen video gamers could easily upgrade their GPU. This promoted rapid technological improvement, as rival GPU manufactures sought sales by offering better and/or cheaper hardware than their rivals. However even today, GPUs essentially (like the SIMD MasPar, page 8) require their parallel processing elements, to do the same thing at the same time.

Initially GPUs were very hard to program and their support software was only designed to be used by dedicated programmers employed by video game companies. However the abundant and cheap parallel processing the GPUs offered was taken up by scientific programming, leading to the field of General-Purpose Computing on

[18] A second recommendation to the novice computer scientist, do not assume that a very old paper has no merit. Computer science is littered with examples of old ideas which returned, e.g. virtual memory, virtual machines and Maurice Wilkes' microcode.

Fig. 3 When I was your age I could think of six impossible things before breakfast

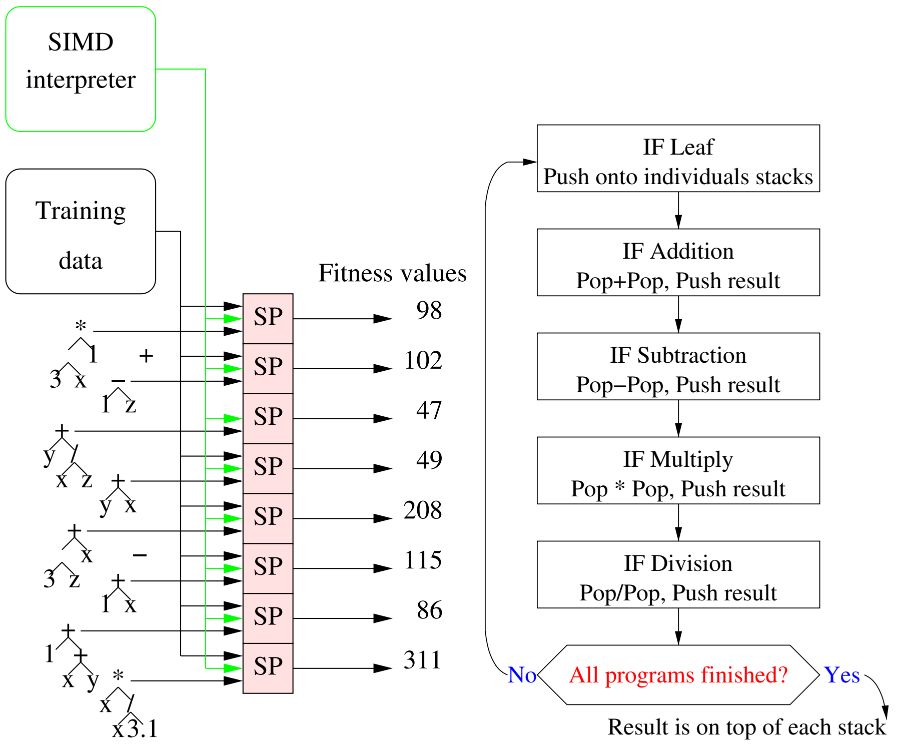

Fig. 4 Left: Avoid compilation overhead by interpreting GP trees. Run single SIMD interpreter on GPU's stream processors (SP) on many trees. Right: Programs wait for the interpreter to offer an instruction they need evaluating. For example an addition. When the interpreter wants to do an addition, everyone in the whole population who is waiting for addition is evaluated. The operation is ignored by everyone else. The interpreter moves on to its next operation. The interpreter runs round its loop until the whole population has been interpreted. Fitness values can also be calculated in parallel

GPUs (GPGPU) [57]. As GPGPU became more popular, the GPU manufactures, particularly nVidia provided much better software support.

At first in genetic programming GPUs were only used to speed up fitness evaluation, e.g. work by Simon Harding [58]. and Darren Chitty [59]. Indeed it was said that, due to the GPUs peculiar SIMD architecture, running the GP interpreter on the GPU was *impossible* (cf. Fig. 3). Of course this was not true, and inspired by Juille's work with the MasPar SIMD supercomputer [60] (page 8), I built a SIMD interpreter for nVidia's GPUs (see Fig. 4) [61, 62][19]. See also [64–70].[20]

As the memory available on the GPU cards increased, it became possible to work with huge populations of small GP trees. In [71] I used a cascade of GP populations to winnow useful bioinformatic data from more than a million GeneChip features. The top level GP populations contained more than five million individuals trees. This GPU application could scale from a $50 GPU to a top 500 super computer [72]. Figure 1 in [73] shows the dramatic improvement in nVidia GPU speed (2003 to 2012, which still continues), whilst Table 3 in [74] shows some high performance parallel GP implementations, almost all running on GPUs.[21]

1.8.3 Deep learning and accelerators: GPUs and TPUs

Due to the availability of internet scale data sets and GPGPU processing power, since 2010 the field of deep learning has taken off [77]. It is generally accepted that researchers need a GPU (possibly a whole cluster of GPUs) to do any form of competitive deep neural net learning. Even with the availability of cloud computing, this may soon have the effect of "pricing out" individual academic researchers from the future of deep learning [78].

Sometimes the whole notion of using a GPU to drive a computer's screen (also called the computer's monitor) may be disregarded. Often called "headless" GPUs, to save space and power, some GPUs dispensed with the screen interface altogether. An extreme examples of this is Google's TPU, which is totally specialised to Artificial Neural Network (ANN) processing.

As gaming and now AI have become more important, the notion of a GPU as a cheap alternative to the computer's CPU has also faded, and now a top end GPU can cost more than a CPU.

[19] People also said that it was impossible to create random numbers on GPU. Again not true [63]. These days pseudo random number generators PRNG, (e.g. CuRAND) are supplied by nVidia with its CUDA software.

[20] The metric "Genetic Programming Operations per Second" (GPops) permits easy comparison of performance, e.g. across different implementations and hardware.

[21] Recent extremely high performance on Intel multi-core CPU SIMD hardware [75] has been achieved, with unchanging fitness functions, in large trees, in converged populations [76], that do not have side effects (and so can be evaluated in any order). This is because a lot of work can be omitted, if it can be proved that a child has the same fitness as its parents.

1.9 Other gaps: memory, theory, bloat

John Koza mentioned that even though Jaws [1] did not include much work on evolving memory, he regarded it as important because it provides another route to allow re-use. Since a value stored in memory can be re-used, potentially many times, without the code for it having to be evolved more than once. He mentioned my book [79], although using indexed memory in GP is due to Teller [80]. Surprisingly, there has been a steady stream of research on evolving memory within GP [81–133].

Genetic programming theory has a variety of forms [134]. Jaws [1] starts with adapting the then current explanations of how linear bit string genetic algorithms work, due to John Holland and Dave Goldberg. Such schema theories were also analysed by Una-May O'Reilly [135], Justin Rosca [136] and most notably by Riccardo Poli [137]. Another popular thread is to take ideas from biology about how evolution works and use them to understand GP [138], e.g. Price's theorem [139, 140], population convergence [76, 141, 142] and neutral networks (plateaus) [143] in fitness landscapes [144–148]. Similarly biology has been an inspiration for other search operators, such as homologous crossover [149]. In recent years there has been a flowering of formal or rigorous run time analysis in evolutionary computing and some success applying mathematical techniques to GP problems [150–155]. Of course it is difficult to make such theorems widely applicable and when using results we must remember the inevitable assumptions they require. For example, SAT has been proved to be NP-complete. Nevertheless in the last decade considerable progress has been made with practical SAT solvers and they are now routinely applied, e.g. in software engineering. Similarly, the No Free Lunch theorem [156] applies to GP (as with all optimisers) but fortunately (as in other branches of AI) that has not inhibited development of the field. Although, as noted above, there are exceptions, but genetic programming as a whole remains a deeply empirical endeavour with many new ideas being reported. However it is difficult to persuade authors to carefully analyse their evolving populations of programs so as to be able to explain why their experiment succeeded (or even why it failed).

Although John Koza reports [1] bloat[22] from the start of genetic programming, the tendency, indeed the name, for programs to be bigger than necessary is not unique to GP. Bloated human written programs are common. Indeed people writing computer programs with unnecessary instructions goes back to the very beginning of electronic digital computers, with bloat reported in programs run on the first stored program digital computer, the Manchester Mark I [35]. This human tendency is rampant, with some Internet code bases having grown to over a billion lines of code in less than 20 years. Bloat continues to be a well studied topic in GP with 426 entries in the GP bibliography mentioning it.

Although there are potential ways of mitigating bloat's impact on runtime [157] and reducing its memory requirements with DAGs [158] (indeed bloated trees produced by crossover [159] should be highly compressible), in practice bloated populations can quickly overwhelm the available computer resources and so the common

[22] Bloat is the tendency for programs to grow in size without commensurate increase in performance.

approach is to shut bloat down. For example, by enforcing either depth or size limits on the evolving programs. However this is not risk free [160] and more sophisticated approaches may be wanted. For example, controls on selection, such as using multiple fitness objectives (e.g. a size versus performance Pareto trade-off [161, 162]) or tighter controls on offspring generation [163–166]. In many cases bloat appears to be an unexpected aspect of early (even premature) convergence and so has some similarity with overfitting sometimes seen with artificial neural networks (ANNs), where prolonged search drives locally improved performance on the training data. This gives a more convoluted mapping between the ANN's inputs and outputs but at the possible expense of the ANN's ability to generalise to unseen data. Where the goal is to explain or predict, such complexity or overfitting is clearly unhelpful. In ANN anti-over fitting techniques are essential. These include stopping training early (i.e. in GP terms using fewer generations), regularization [167–169], changing the training data during training [170, 171] and even expression simplification [172], either during evolution [173] or to increase comprehensibility and explainability, cf. XAI, after GP has finished [174]. Whilst Dale Hopper [173] and other authors, ensure their automatic rewrite of GP individuals gives a semantically equivalent but smaller replacement, in many cases this is not wanted. When a 100% correct program is not realistic, e.g. on many prediction tasks, it may be better to accept (or allow evolution to find) a similar but much simpler program, rather than spending a lot of effort creating an exactly equivalent program to what is essentially only an approximation.

However, bear in mind that evolution is a hacker. It builds on what was there before. In biology evolution overfits. Classic example include: 1) the Giraffe's left laryngeal nerve, which runs the whole length of its neck from its head, round the aorta in its chest and then returns to its throat at the top of its neck, because evolution did not find a shorter path, 2) the male peacock's heavy tail which helps secure a mate but impedes flight and 3) the human brain which consumes 20% of our food [175] but made our ancestors more appealing as mates to other members of their tribe [176].

2 A brief selection of other genetic programming work

In addition to continuing with evolving Lisp like trees, major branches of genetic programming include: linear genetic programming [177] cartesian genetic programming (CGP) [178] and grammatical evolution (GE) [179], all of which use a linear chromosome. Following John Koza's automatically defined functions, ADFs, see page 7, there were several attempts to encourage the evolution of modular programs using individuals with multiple trees or libraries of subtrees [180–183]. However, these seem not to have taken hold.

As with evolutionary computation in general, the major computational cost of GP is usually evaluating fitness [1, p783]. In tree GP this is usually the cost of interpreting the trees. When members of the population are going to be run many times.[23] it may be worth the cost of compiling the population and then running the compiled programs.[24] rather than interpreting them [188]. However, as Ronald Crepeau showed [189], for GP, it is not essential to run a full blown compiler, instead knowing the restricted set of primitives used by GP, he constructed a dedicated fast compiler which converted the evolved code into machine code and ran that directly. Peter Nordin eliminated the compilation step entirely by using GP to evolve firstly Sun 32 bit SPARC RISC architecture machine code [85] and later Intel x86 binaries [190] (which in turn later became Discipulus [191]). He used tailored mutation operations which respected the layout of the machine code. Although perhaps first motivated by speed and simplicity, the idea of evolving variable length linear programs has taken off [192, 193].

Grammatical Evolution (GE) [194, 195] shows the virtues of trying ideas out. Michael O'Neill and Conor Ryan took the idea of a variable length linear chromosome, simplified it to become just an ordered list of byte sized integers (0..255) and married it to another favourite of computer scientists: the Backus-Naur form grammar (BNF). Pretty much anything which can be run on a computer can be expressed in a BNF grammar. They disregarded that BNF is essentially tree shaped and trusted in evolution to find a way of putting them together. The linear stream of bytes is mapped using modulus to say which branch to take next in the grammar. If there are not enough bytes, we simply wrap round and start again from the first. If there are too many, we ignore the excess. The resulting grammar is treated as the individual's phenotype and in a problem dependent way converted into a trial solution with a fitness value. The sloppiness of the mapping from genotype to phenotype offended some and provoked wide discussion in a peer commentary issue of "Genetic Programming and Evolvable Machines" [196]. But as Conor Ryan says "GE works" [197]. Indeed the separation of genotype from BNF grammar makes grammatical evolution flexible and has been widely used. (The GP bibliography contains well over seven hundred entries relating to grammatical evolution.)

With Cartesian Genetic Programming (CGP) [178, 198–201], Julian Miller turned to a fixed representation, more a kin to traditional bit string genetic algorithms (GAs). However the chromosome is a fixed sized two dimensional rectangle, rather than a single string, where each cell contains a digital computational unit, such as an XOR gate. Both the contents of the cells and crucially the connections between them are evolvable.[25] Notice, like linear GP (but unlike GE), evolution

[23] Considerable saving are sometimes possible by accepting fitness selection will be somewhat random and using a cheaper and approximate fitness function. After all, the goal of a fitness function, is not to measure performance (that can be done after the run) but to guide search. Why run thousands of fitness cases, when fitness will ultimately be reduced to a single bit: does this individual get a child or not? Even then, we typically add noise to this bit, e.g. via tournament selection, [184–186], see also [187, Sect. 10.1].

[24] Lisp provides compilation as an alternative to interpreting programs [1, p785].

[25] Nowadays in cartesian genetic programming people often set the width of the FPGA like rectangle of components to 1, making it effectively a string. Similarly it is common to allow only feed forward connections, so preventing recurrent loops.

directly sets the contents and connections of each cell (i.e. evolution acts directly on the phenotype). Also there is no explicit left-right flow of control. In CGP the chromosome is treated as a circuit and so its evaluation has to take note of where data enters and leaves. It is also not necessary to evaluate cells which are not connected. Cartesian GP has been widely used, including in the evolution of approximate computing [202, 203], where evolution can be well suited to finding good trade-offs between conflicting objectives, such as fidelity, size, number of components, power consumption and speed.

2.1 Inspired by computer science

In order for subtree crossover to freely mix subtrees from parents to create children, John Koza required the components of his GP trees to have closure [1, Sect. 6.1.1]. Meaning 1) any leaf or function in the tree can be an argument to any other function. Since components typically communicate via function return values, this often means GP trees use a single type, often float. 2) To ensure each function can deal with any combination of inputs, many functions have protected GP versions. Such as protected log RLOG [1, p83], which returns a defined value (rather than raising an exception) even if its input is zero or negative. Alternatives might be to allow evolution to deal with the exception, or simply assign poor fitness to individuals with illegal combinations. However notice that ruling it out prevents GP exploring not only this tree but all the trees that might have evolved from it.

Perhaps the most famous extensions to closure are Dave Montana's strongly typed GP [204] and Tina Yu's polymorphic GP [205, 206] which allow multiple types but ensure evolution explores only type safe expressions. Another approach is to use various types of grammar to try and keep evolution in the most productive parts of the search space [207]. For example, using context free grammars [208, 209], using grammars to ensure the evolution of expressions which are dimensionally consistent [210], using tree-adjunct grammars to guide GP (TAG3) [211] and using GP with Lindenmayer Systems (often abbreviated to L-Systems) [212–215].

Whereas Lisp and most GP systems implicitly use the system stack, programs which explicitly use a stack [216, 217], e.g. to pass vectors and matrices [218], are also possible. An explicit stack allows the evolution of Reverse Polish Notation (RPN) [62] and even infix expressions [219]. In PushGP [220] there are multiple stacks, one per type. These may include a code stack, so allowing GP to manipulate code, thus permitting GP to evolve its own genetic operators.

2.2 Non genetic GP

John Koza's GP [1] is clearly strongly influenced by his PhD supervisor, John Holland, and GP [1] is essentially the application of John Holland's genetic algorithms to the evolution of Lisp s-expressions, i.e. tree shaped programs. But, as we have seen, the programs need not be trees, and similarly the search algorithm does not have to be a genetic algorithm. Other techniques include: local search, Simulated Annealing [221, 222], Differential Evolution [223], Bayesian probability search [224], Estimation-of-Distribution Algorithms (EDAs) [225, 226] Ken Stanley's Neat [227–229] and even deterministic search, e.g. Trent McConaghy's FFX [230]. Indeed search does not have to be guided only by fitness but can "look inside" the program [231] and its execution [232]. SRbench [233] compares many GP and non-GP approaches to symbolic regression, including MRGP [234], M3GP [235], FEW [236] and Operon [237].

2.3 Less explored

2.3.1 Assembly code, byte code

In human terms assembly code is usually viewed as intermediate between high level languages and machine code. Offering the potential advantage of machine code (speed and compactness), and ease of use and readability of high level source code. There has been very little GP work on evolving assembly code. Exceptions include microcontroller assembly [238], nVidia GPU PTX [239, 240] and the intermediate (IR) code used by LLVM [241], and again on GPUs [242].

Java, and some other interpreted languages, compile the source code into byte code which they then interpret. Eduard Lukschandl showed it is possible to run GP at the level of Java byte code [243].

2.3.2 Modularity, recursion, loops

Some of the work on encouraging the evolution of modular code was mentioned on page 13. In Jaws, John Koza described GP solving the Fibonacci problem [1, pp473–477] as an example requiring the evolution of recursion and several examples where GP evolved do-until loops and other forms of iteration, but again there has been relatively little work on either by others. Again a few exceptions. These include work by Peter Whigham [244, 245] and Tom Castle [246].

2.3.3 Coevolution

As with many topics, there are examples of co-evolution [247, 248] in Jaws [1] and many elsewhere in genetic programming [81], for example in agent learning [249]. However, it does feel like coevolution has not yet fulfilled its potential. In deep artificial neural networks there is interest in antagonistic adversarial learning and so perhaps this will stimulate renewed interest in coevolution in genetic programming.

3 The future

At GECCO 2022 Erik Goodman asked if there we any applications of GP that had surprised John Koza. Amongst the many human competitive [6] results, perhaps one of the most encouraging is quantum computing. As with quantum physics, quantum computing has a deserved reputation for being difficult for people. However, the rules about quantum computing gates can be coded for GP to use without being an expert quantum physicist, and then GP can be left to evolve novel quantum circuit designs incorporating them [250–252]. Riccardo Poli, Leonardo Vanneschi and others have previously reported on the state of GP and in particular what remains to be done [253, 254].

In genetic improvement [13] existing (human written) software is optimised (typically by using GP). Notice genetic improvement does not start from primordial ooze [1]. Instead search automates the potentially labour intensive, tedious and error prone task of find modifications. For example, to repair bugs [12, 43, 47, 49, 255], including energy bugs [256], reducing memory consumption [257], reduce run time [174, 258–265] improve existing functionality (e.g. to give better predictions [266]), porting to new hardware [267] including improving GPU applications [242, 262–265, 268] or even to incorporate existing functionality from outside the existing code base [269].

The idea of mixing evolutionary computing (including GP) with other optimisation tools to give hyperheuristics [270] has a long history. In particular, with the recent explosion of interest in deep artificial neural networks, combining evolutionary learning and artificial neural networks seems set to continue. One particularly encouraging trend is AutoML tools such as TPOT [271, 272] which automatically tune existing machine learning pipelines.

In GP, as in most optimisation problems, most of the computation effort is spent on evaluating how good the proposed solutions are. Various ideas for speeding up fitness evaluation have been proposed, for example surrogate fitness functions [273]. Colin Johnson's Learned Guidance Functions [274] seem a particularly elegant approach to making best used of previously gained knowledge. It would be interesting to see Learned Guidance Functions applied to genetic programming or when using genetic improvement to adapt existing human written programs.

Since all digital computing progressively loses information, information about crossover and mutation gets progressively washed out the further it has to travel. In nested functions without side effects, deep genetic changes become invisible to the fitness function. Thus to evolve complex programs, they must remain shallow and so I propose that to evolve large complex programs, they be composed of many shallow trees, within a strong low entropy-loss data interconnect to and from the environment. This should ensure that the good and bad effects of most genetic code changes are externally measurable [275].

At GECCO John Koza pointed out that in both biology and in human design, modularity and reuse are ever present. Biology scales from a single cell to individuals containing billions of cells. It does this, like human engineers, not by solving many billions of individual problems but by reusing existing designs. We need to revisit the scaling problem.

4 Conclusions

We have seen that in the thirty years since John Koza published his first GP book, the field has blossomed. The genetic programming bibliography contains some 16 367 entries by 16 342 authors[26]. Many of the genetic and evolutionary computation papers judged to be the best human competitive work of each year have used genetic programming. Clearly GP is doing well in its mission to help the world.

As mentioned at the end of the last section, although GP continues to flourish, perhaps we need to tackle the scaling problem. Are we evolving small things? Do we need to be more ambitious? Following Stephanie Forrest's recent questions [276]: what could GP do with Google Deep AI scale resources?

As John Koza foresaw, 30 years of Moore's law [277] (with component count doubling every 18 months) means 20 lots of doubling ($2^{20} = 1\ 048\ 576$). That is, since the genetic programming field started, the computer power available to us has increased a million fold. What of the next 30 years? Perhaps Moore's Law will end? Certainly the death of Moore's Law has been confidently predicted many times. What seems certain is we will not see dramatic increases in silicon computing's clock speeds. Instead we anticipate the future of computing will be ever more parallel. But as John Koza says GP is *embarrassingly parallel*. Indeed the use of distributed parallel GP populations, not only makes good use of current and future compute resources but is in keeping with Sewall Wright's [278] model of natural evolution and as John Koza reports by keeping population diversity, the distributed population demes of the island model, improve GP results as well as speeding it up.

In 2052 will genetic programming researchers be using computers a million times faster than they use today? Certainly GP seems well placed to exploit them.

Acknowledgements I would like to thank Sean Luke, and Andrew and Claire, and my anonymous reviewers. No competing interests. Funded by the Meta (formerly Facebook) Oops project (Award number 181551).

[26] The GP bibliography was started by John Koza. Although in recent years it has undoubtedly missed some work, in the five years before the pandemic (i.e. 2015–19) there were 3340 new entries and 5177 authors published at least one GP paper.

References

1. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
2. C. Darwin, *On the Origin of Species by Means of Natural Selection*, 1985th edn. (John Murray, Penguin classics, London, 1859)
3. C. Darwin, *Voyage of the Beagle*, 1989th edn. (Henry Colburn, Penguin classics, London, 1839)
4. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge, 1994)
5. J.R. Koza et al., *Genetic Programming III: Darwinian Invention and Problem Solving* (Morgan Kaufmann, Cambridge, 1999)
6. J.R. Koza et al., *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (Kluwer Academic Publishers, Dordrecht, 2003). https://doi.org/10.1007/0-387-26417-5_1
7. J.R. Koza, J.P. Rice, *Genetic Programming: The Movie* (MIT Press, Cambridge, 1992)
8. J.R. Koza, *Genetic Programming II Videotape: The next generation* (MIT Press, Cambridge, 1994)
9. J.R. Koza et al., *Genetic Programming III Videotape: Human Competitive Machine Intelligence* (Morgan Kaufmann, San Francisco, 1999)
10. J.R. Koza et al., *Genetic Programming IV Video: Human-Competitive Machine Intelligence* (Kluwer Academic Publishers, Dordrecht, 2003)
11. J. Koza, Automated design using Darwinian evolution and genetic programming. Stanford University, EE380: Computer Systems Colloquium (18 Feb 2009). https://www.youtube.com/watch?v=xIoytwJWJP8
12. C. Le Goues et al., Automated program repair. Commun. ACM **62**(12), 56–65 (2019). https://doi.org/10.1145/3318162
13. J. Petke et al., Genetic improvement of software: a comprehensive survey. IEEE Trans. Evol. Comput. **22**(3), 415–432 (2018). https://doi.org/10.1109/TEVC.2017.2693219
14. W.B. Langdon, J. Petke, Software is not fragile. in *Complex Systems Digital Campus E-conference*, ed. by P. Parrend et al. CS-DC'15. Proceedings in Complexity, Springer (Sep 30-Oct 1 2015), pp. 203–211. https://doi.org/10.1007/978-3-319-45901-1_24, invited talk
15. W.B. Langdon et al., Efficient multi-objective higher order mutation testing with genetic programming. J. Syst. Softw. **83**(12), 2416–2430 (2010). https://doi.org/10.1016/j.jss.2010.07.027
16. N. Harrand et al., A journey among Java neutral program variants. Genet. Program Evolvable Mach. **20**(4), 531–580 (2019). https://doi.org/10.1007/s10710-019-09355-3
17. E. Schulte et al., Software mutational robustness. Genet. Program Evolvable Mach. **15**(3), 281–312 (2014). https://doi.org/10.1007/s10710-013-9195-8
18. R. Abou Assi et al., Coincidental correctness in the Defects4J benchmark. Softw. Testing, Verif. Reliab. **29**(3), e1696 (2019). https://doi.org/10.1002/stvr.1696
19. B. Danglot et al., Correctness attraction: a study of stability of software behavior under runtime perturbation. Empir. Softw. Eng. **23**(4), 2086–2119 (2018). https://doi.org/10.1007/s10664-017-9571-8
20. M. Monperrus, Principles of antifragile software. in *Companion to the First International Conference on the Art, Science and Engineering of Programming*. Programming '17, ACM, New York, NY, USA (2017), pp. 32:1–32:4. https://doi.org/10.1145/3079368.3079412
21. J. Petke et al., Software robustness: A survey, a theory, and some prospects, in *ESEC/FSE 2021, Ideas, Visions and Reflections*. ed. by P. Avgeriou, D. Zhang (ACM, Athens, 2021), pp.1475–1478. https://doi.org/10.1145/3468264.3473133
22. D. Andre, J.R. Koza, Parallel genetic programming on a network of transputers. in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, ed. by J.P. Rosca. Tahoe City, California, USA (9 Jul 1995), pp. 111–120. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/andre_1995_parallel.pdf
23. K.E. Kinnear Jr. (ed.), *Advances in Genetic Programming* (MIT Press, Cambridge, 1994)
24. P.J. Angeline, K.E. Kinnear Jr. (eds.), *Advances in Genetic Programming 2* (MIT Press, Cambridge, 1996). https://doi.org/10.7551/mitpress/1109.001.0001
25. L. Spector et al., Quantum computing applications of genetic programming, chap. 7, in *Advances in Genetic Programming 3*. ed. by L. Spector. et al. (MIT Press, Cambridge, 1999), pp.135–160. https://doi.org/10.7551/mitpress/1110.003.0010

26. J.R. Koza et al., (eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference*. MIT Press, Stanford University, CA, USA (28–31 Jul 1996). http://www.genetic-programming.org/gp96proceedings.html
27. J.R. Koza et al., (eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*. Morgan Kaufmann, Stanford University, CA, USA (13-16 Jul 1997). http://www.amazon.com/Genetic-Programming-2nd-Conference-Author/dp/1558604839
28. J.R. Koza et al., (eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann, University of Wisconsin, Madison, WI, USA (22-25 Jul 1998)
29. P.J. Angeline et al, (eds.), *Proceedings of the 1999 Congress on Evolutionary Computation*, CEC 1999. IEEE Press, Washington, DC, USA (July 6-9 1999). https://dblp.org/rec/conf/cec/1999.bib
30. W. Banzhaf et al., (eds.), *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, Orlando, Florida, USA (13-17 Jul 1999). http://www.amazon.com/exec/obidos/ASIN/1558606114/qid%3D977054373/105-7666192-3217523
31. W. Banzhaf et al., (eds.), *Genetic Programming*, LNCS, vol. 1391. Springer-Verlag, Paris (14-15 Apr 1998). https://doi.org/10.1007/BFb0055923
32. J.R. Koza et al., Evolving inventions. Sci. Am. **288**(2), 52–59 (2003)
33. J.R. Koza, Human-competitive results produced by genetic programming. Genet. Programm. Evolvable Mach. **11**(3/4), 251–284 (2010). https://doi.org/10.1007/s10710-010-9112-3
34. R.M. Friedberg, A learning machine: I. IBM J. Res. Dev. **2**(1), 2–13 (1958)
35. T. Kilburn et al., Experiments in machine learning and thinking. in *Information Processing, Proceedings of the 1st International Conference on Information Processing*. UNESCO, Paris (15-20 Jun 1959), pp. 303–308. https://dblp.org/rec/conf/ifip/KilburnGS59.bib
36. A.M. Turing, Intelligent machinery (1948), https://www.npl.co.uk/getattachment/about-us/History/Famous-faces/Alan-Turing/80916595-Intelligent-Machinery.pdf, report for National Physical Laboratory. Reprinted in Ince, D. C. (editor). 1992. Mechanical Intelligence: Collected Works of A. M. Turing. Amsterdam: North Holland. Pages 107127. Also reprinted in Meltzer, B. and Michie, D. (editors). (1969). Machine Intelligence 5. Edinburgh: Edinburgh University Press [278]
37. A.M. Turing, Intelligent machinery, chap. 1, in *Machine Intelligence*, vol. 5, ed. by B. Meltzer, D. Michie (Edinburgh University Press, Edinburgh, 1969), pp.3–23. https://doi.org/10.1109/GI52543.2021.00008
38. W.B. Langdon, W. Banzhaf, Repeated patterns in tree genetic programming. in*Proceedings of the 8th European Conference on Genetic Programming*. Lecture Notes in Computer Science, vol. 3447, ed.by M. Keijzer et al. Springer, Lausanne, Switzerland (30 Mar–1 Apr 2005), pp. 190–202. https://doi.org/10.1007/978-3-540-31989-4_17
39. W.B. Langdon, W. Banzhaf, Repeated patterns in genetic programming. Nat. Comput. **7**(4), 589–613 (2008). https://doi.org/10.1007/s11047-007-9038-8
40. S. Ohno, *Evolution by Gene Duplication* (Springer, Berlin, 1970). https://doi.org/10.1007/978-3-642-86659-3
41. J.R. Koza, D. Andre, A case study where biology inspired a solution to a computer science problem, in *Pacific Symposium on Biocomputing '96*. ed. by L. Hunter, T.E. Klein (World Scientific, Singapore, 1996), pp.500–511
42. J.R. Koza, Architecture-altering operations for evolving the architecture of a multipart program in genetic programming. Technical Report STAN-CS-94-1528, Dept. of Computer Science, Stanford University, Stanford, California 94305, USA (Oct 1994). http://www.genetic-programming.com/jkpdf/tr1528.pdf
43. S. Forrest et al., A genetic programming approach to automated software repair. in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ed. by G. Raidl et al. ACM, Montreal (8-12 Jul 2009), pp. 947–954. https://doi.org/10.1145/1569901.1570031, gECCO 2019 10-Year Most Influential Paper Award, Best paper
44. L. Carroll, *Through the Looking-Glass, and What Alice Found There* (Macmillan, London, 1871)
45. W. Weimer et al., Automatically finding patches using genetic programming. in *International Conference on Software Engineering (ICSE) 2009*, ed. by S. Fickas. Vancouver (May 16-24 2009), pp. 364–374. https://doi.org/10.1109/ICSE.2009.5070536
46. C. Le Goues, *Automatic Program Repair Using Genetic Programming*. Ph.D. thesis, Faculty of the School of Engineering and Applied Science, University of Virginia, USA (May 2013). http://www.cs.virginia.edu/~weimer/students/claire-phd.pdf

47. S.O. Haraldsson et al., Fixing bugs in your sleep: how genetic improvement became an overnight success, in *GI-2017*. ed. by J. Petke et al. (ACM, Berlin, 2017), pp.1513–1520. https://doi.org/10.1145/3067695.3082517
48. N. Alshahwan, Industrial experience of genetic improvement in Facebook. in GI-2019, ed. by J. Petke, et al. ICSE workshops proceedings. IEEE, Montreal (28 May 2019), p. 1. https://doi.org/10.1109/GI.2019.00010, invited Keynote
49. S. Kirbas et al., On the introduction of automatic program repair in Bloomberg. IEEE Softw. **38**(4), 43–51 (2021). https://doi.org/10.1109/MS.2021.3071086
50. H. Juille, J.B. Pollack, Massively parallel genetic programming, chap. 17, in *Advances in Genetic Programming 2*. ed. by P.J. Angeline, K.E. Kinnear Jr. (MIT Press, Cambridge, 1996), pp.339–357. https://doi.org/10.7551/mitpress/1109.003.0023
51. A. Thompson, *Hardware Evolution Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution* (Springer, Berlin, 1998). https://doi.org/10.1007/978-1-4471-3414-5
52. T.G.W. Gordon, *Exploiting Development to Enhance the Scalability of Hardware Evolution*. Ph.D. thesis, University College, London, UK (Jul 2005). https://discovery.ucl.ac.uk/id/eprint/1444775/
53. P.N. Martin, *Genetic Programming in Hardware*. Ph.D. thesis, University of Essex, University of Essex, Wivenhoe Park, Colchester, UK (Mar 2003). http://www.naiadhome.com/HardwareGeneticProgramming.pdf
54. L. Sekanina, Z. Vasicek, CGP acceleration using field-programmable gate arrays, chap. 7, in *Cartesian Genetic Programming. Natural Computing Series*. ed. by J.F. Miller. (Springer, Berlin, 2011), pp.217–230. https://doi.org/10.1007/978-3-642-17310-3_7
55. C. Goribar-Jimenez et al., Towards the development of a complete GP system on an FPGA using geometric semantic operators, in *2017 IEEE Congress on Evolutionary Computation (CEC)*. ed. by J.A. Lozano (IEEE, Donostia, 2017), pp.1932–1939. https://doi.org/10.1109/CEC.2017.7969537
56. W.B. Langdon, O. Krauss, Genetic improvement of data for maths functions. ACM Trans. Evolut. Learn. Optim. **1**(2), 7 (2021). https://doi.org/10.1145/3461016
57. J.D. Owens et al., A survey of general-purpose computation on graphics hardware. Comput. Gr. Forum **26**(1), 80–113 (2007). https://doi.org/10.1111/j.1467-8659.2007.01012.x
58. S. Harding, W. Banzhaf, Fast genetic programming on GPUs. in *Proceedings of the 10th European Conference on Genetic Programming*. Lecture Notes in Computer Science, vol. 4445, ed. by M. Ebner et al. Springer, Valencia, Spain (11-13 Apr 2007), pp. 90–101. https://doi.org/10.1007/978-3-540-71605-1_9
59. D.M. Chitty, A data parallel approach to genetic programming using programmable graphics hardware. in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. vol. 2, ed. by D. Thierens et al. ACM Press, London (7-11 Jul 2007), pp. 1566–1573. https://doi.org/10.1145/1276958.1277274
60. H. Juille, J.B. Pollack, Parallel genetic programming and fine-grained SIMD architecture. in *Working Notes for the AAAI Symposium on Genetic Programming*, ed. by E.V. Siegel, J.R. Koza. AAAI, MIT, Cambridge, MA, USA (10–12 Nov 1995), pp. 31–37. http://www.aaai.org/Papers/Symposia/Fall/1995/FS-95-01/FS95-01-005.pdf
61. W.B. Langdon, *A SIMD interpreter for genetic programming on GPU graphics cards*. Tech. Rep. CSM-470, Department of Computer Science, University of Essex, Colchester, UK (3 Jul 2007). http://cswww.essex.ac.uk/technical-reports/2007/csm_470.pdf
62. W.B. Langdon, W. Banzhaf, A SIMD interpreter for genetic programming on GPU graphics cards. in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*. Lecture Notes in Computer Science, vol. 4971, ed. by M. O'Neill et al. Springer, Naples (26-28 Mar 2008), pp. 73–85. https://doi.org/10.1007/978-3-540-78671-9_7
63. W.B. Langdon, A fast high quality pseudo random number generator for graphics processing units. in *2008 IEEE World Congress on Computational Intelligence*, ed. by J. Wang. IEEE, Hong Kong (1-6 Jun 2008), pp. 459–465. https://doi.org/10.1109/CEC.2008.4630838
64. D. Robilliard et al., Genetic programming on graphics processing units. Genet. Program Evolvable Mach. **10**(4), 447–471 (2009). https://doi.org/10.1007/s10710-009-9092-3
65. L.A. Baumes et al., EASEA: a generic optimization tool for GPU machines in asynchronous island model. Comput. Methods Mater. Sci. **11**(3), 489–499 (2011)
66. J. Vitola et al., Parallel algorithm for evolvable-based boolean synthesis on gpus. in *Third IEEE Latin American Symposium on Circuits and Systems (LASCAS 2012)* (29 Feb-2 Mar 2012). https://doi.org/10.1109/LASCAS.2012.6180339

67. A. Maghazeh et al., General purpose computing on low-power embedded GPUs: has it come of age? in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, ed. by H. Jeschke. IEEE, Samos, Greece (15-18 Jul 2013). https://doi.org/10.1109/SAMOS.2013.6621099
68. D.M. Chitty, Faster GPU-based genetic programming using a two-dimensional stack. Soft. Comput. **21**(14), 3859–3878 (2017). https://doi.org/10.1007/s00500-016-2034-0
69. K. Ono, Y. Hanada, Self-organized subpopulation based on multiple features in genetic programming on GPU. J. Adv. Comput. Intell. Intell. Inform. **25**(2), 177–186 (2021). https://doi.org/10.20965/jaciii.2021.p0177
70. L. Trujillo et al., GSGP-CUDA - a CUDA framework for geometric semantic genetic programming. SoftwareX **18**, 101085 (2022). https://doi.org/10.1016/j.softx.2022.101085
71. W.B. Langdon, A.P. Harrison, GP on SPMD parallel graphics hardware for mega bioinformatics data mining. Soft. Comput. **12**(12), 1169–1183 (2008). https://doi.org/10.1007/s00500-008-0296-x
72. W.B. Langdon, Distilling GeneChips with genetic programming on the Emerald GPU supercomputer. SIGEVOlution Newsl. ACM Spec. Interest Group Genet. Evolut. Comput. **6**(1), 15–21 (2012). https://doi.org/10.1145/2384697.2384699
73. W.B. Langdon, Large scale bioinformatics data mining with parallel genetic programming on graphics processing units, chap. 15, in *Massively Parallel Evolutionary Computation on GPGPUs. Natural Computing Series*. ed. by S. Tsutsui, P. Collet. (Springer, Berlin, 2013), pp.311–347. https://doi.org/10.1007/978-3-642-37959-8_15
74. W.B. Langdon, Large scale bioinformatics data mining with parallel genetic programming on graphics processing units, chap. 5, in *Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence*, ed. by F. Fernandez de Vega, E. Cantu-Paz., vol. 269 (Springer, Berlin, 2010), pp.113–141. https://doi.org/10.1007/978-3-642-10675-0_6
75. W.B. Langdon, W. Banzhaf, Long-term evolution experiment with genetic programming. Artif. Life **28**(2), 173–204 (2022). https://doi.org/10.1162/artl_a_00360
76. W.B. Langdon, Genetic programming convergence. Genet. Program Evolvable Mach. **23**(1), 71–104 (2022). https://doi.org/10.1007/s10710-021-09405-9
77. I. Goodfellow et al., *Deep Learning* (MIT Press, Cambridge, 2016)
78. W. Weimer, *From deep learning to human judgments: Lessons for genetic improvement*. GI @ GECCO 2022 (9 Jul 2022), http://geneticimprovementofsoftware.com/slides/gi2022gecco/weimer-keynote-gi-gecco-22.pdf, invited keynote
79. W.B. Langdon, Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!, Genetic Programming, vol. 1. Kluwer, Boston (1998), https://doi.org/10.1007/978-1-4615-5731-9
80. A. Teller, The evolution of mental models, chap. 9, in *Advances in Genetic Programming*. ed. by K.E. Kinnear Jr. (MIT Press, Cambridge, 1994), pp.199–219
81. J. Jannink, Cracking and co-evolving randomizers, chap. 20, in *Advances in Genetic Programming*. ed. by K.E. Kinnear Jr. (MIT Press, Cambridge, 1994), pp.425–443
82. D. Andre, Evolution of mapmaking ability: Strategies for the evolution of learning, planning, and memory using genetic programming. in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*. vol. 1, IEEE Press, Orlando, Florida, USA (27-29 Jun 1994), pp. 250–255. https://doi.org/10.1109/ICEC.1994.350007
83. H. Iba et al. Temporal data processing using genetic programming. in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, ed. by L.J. Eshelman. Morgan Kaufmann, Pittsburgh, PA, USA (15-19 Jul 1995), pp. 279–286. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/iba_1995_tdpgp.pdf
84. T.D. Haynes, R.L. Wainwright, A simulation of adaptive agents in hostile environment. in *Proceedings of the 1995 ACM Symposium on Applied Computing*, ed. by K.M. George et al. ACM Press, Nashville, USA (1995), pp. 318–323. https://doi.org/10.1145/315891.316007
85. P. Nordin, W. Banzhaf, Evolving Turing-complete programs for a register machine with self-modifying code. in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, ed. by L.J. Eshelman. Morgan Kaufmann, Pittsburgh, PA, USA (15-19 Jul 1995), pp. 318–325. http://www.cs.mun.ca/~banzhaf/papers/icga95-2.pdf
86. S. Brave, Evolving recursive programs for tree search, chap. 10, in *Advances in Genetic Programming 2*. ed. by P.J. Angeline, K.E. Kinnear Jr. (MIT Press, Cambridge, 1996), pp.203–220. https://doi.org/10.7551/mitpress/1109.003.0015

87. A.I. Esparcia Alcazar, K.C. Sharman, Some applications of genetic programming in digital signal processing. in *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, ed. by J.R. Koza. Stanford Bookstore, Stanford University, CA, USA (28–31 Jul 1996), pp. 24–31.http://www.iti.upv.es/~anna/papers/someappsgp96.ps
88. W.S. Bruce, *The Application of Genetic Programming to the Automatic Generation of Object-Oriented Programs*. Ph.D. thesis, School of Computer and Information Sciences, Nova Southeastern University, 3100 SW 9th Avenue, Fort Lauderdale, Florida 33315, USA (Dec 1995). https://nsuworks.nova.edu/gscis_etd/430/
89. A. Ronge, M.G. Nordahl, Genetic programs and co-evolution developing robust general purpose controllers using local mating in two dimensional populations. in *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*. LNCS, vol. 1141, ed. by : H.M. Voigt et al. Springer Verlag, Berlin, Germany (22-26 Sep 1996), pp. 81–90. https://doi.org/10.1007/3-540-61723-X_972
90. L. Spector, S. Luke, Cultural transmission of information in genetic programming. in *Genetic Programming 1996: Proceedings of the First Annual Conference*, ed. by J.R. Koza et al. MIT Press, Stanford University, CA, USA (28–31 Jul 1996), pp. 209–214. http://www.cs.gmu.edu/~sean/papers/culture-gp96.pdf
91. S.E. Raik, D.G. Browne, Evolving state and memory in genetic programming, in *Simulated Evolution and Learning*. ed. by X. Yao, J.H. Kim, T. Furuhashi (Springer, Berlin, 1997). https://doi.org/10.1007/BFb0028523
92. B. Edmonds, S. Moss, Modelling of boundedly rational agents using evolutionary programming techniques, in *Evolutionary Computing, LNCS*, vol. 1305, ed. by D. Corne, J.L. Shapiro (Springer-Verlag, Berlin, 1997), pp.31–42. https://doi.org/10.1007/BFb0027164
93. F.H. Bennett III, A multi-skilled robot that recognizes and responds to different problem environments. in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, ed. by J.R. Koza et al. Morgan Kaufmann, Stanford University, CA, USA (13-16 Jul 1997), pp. 44–51. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/gp1997/bennet_1997_msrrrdpe.pdf
94. P.J. Angeline, An alternative to indexed memory for evolving programs with explicit state representations. in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, ed. by J.R. Koza et al. Morgan Kaufmann, Stanford University, CA, USA (13-16 Jul 1997), pp. 423–430
95. I.S. Lim, D. Thalmann, Indexed memory as a generic protocol for handling vectors of data in genetic programming. in *Fifth International Conference on Parallel Problem Solving from Nature*. LNCS, vol. 1498, ed. by A.E. Eiben et al. Springer-Verlag, Amsterdam (27-30 Sep 1998), pp. 325–334.https://doi.org/10.1007/BFb0056875
96. A. Trenaman, *The Evolution of Autonomous Agents Using Concurrent Genetic Programming*. Ph.D. thesis, Department of Computer Science, National University of Ireland, Maynooth, Ireland (Oct 1999), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/trenaman/at_thesis1.ps.gz
97. A. Silva et al., Evolving controllers for autonomous agents using genetically programmed networks. in *Genetic Programming, Proceedings of EuroGP'99*. LNCS, vol. 1598, ed. by R. Poli et al. Springer-Verlag, Goteborg, Sweden (26-27 May 1999), pp. 255–269. https://doi.org/10.1007/3-540-48885-5_22
98. B. Andersson et al., Reactive and memory-based genetic programming for robot control. in *Genetic Programming, Proceedings of EuroGP'99*. LNCS, vol. 1598, ed. by R. Poli et al. Springer-Verlag, Goteborg, Sweden (26-27 May 1999), pp. 161–172. https://doi.org/10.1007/3-540-48885-5_13
99. P. Martin, Genetic programming for service creation in intelligent networks. in *Genetic Programming, Proceedings of EuroGP'2000*. LNCS, vol. 1802, ed. by R. Poli et al. Springer-Verlag, Edinburgh (15-16 Apr 2000), pp. 106–120. https://doi.org/10.1007/978-3-540-46239-2_8
100. K. Bearpark, *Learning and memory in genetic programming*. Ph.D. thesis, School of Engineering Sciences, University of Southampton, UK (2000). http://eprints.soton.ac.uk/45930/
101. R. Karlsson et al., Sound localization for a humanoid robot using genetic programming, in *Real-World Applications of Evolutionary Computing, LNCS*, vol. 1803, ed. by S. Cagnoni et al. (Springer-Verlag, Edinburgh, 2000), pp.65–76. https://doi.org/10.1007/3-540-45561-2_7
102. M.C. Martin, Visual obstacle avoidance using genetic programming: First results. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, ed. by L. Spector et al. Morgan Kaufmann, San Francisco, California, USA (7-11 Jul 2001), pp. 1107–1113. http://www.martincmartin.com/Dissertation/VisualObstacleAvoidanceGP.pdf
103. S.P. Brumby et al., Evolving forest fire burn severity classification algorithms for multi-spectral imagery. in *In Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VII,*

Proceedings of SPIE. vol. 4381, ed. by S.S. Shen, M.R. Descour, (2001), pp. 236–245. https://doi.org/10.1117/12.437013

104. D. Howard et al., The boru data crawler for object detection tasks in machine vision. in *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim/EvoPLAN*. LNCS, vol. 2279, ed. by S. Cagnoni et al. Springer-Verlag, Kinsale, Ireland (3-4 Apr 2002), pp. 222–232.https://doi.org/10.1007/3-540-46004-7_23
105. K. Imamura et al., *N*-version genetic programming via fault masking. in *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*. LNCS, vol. 2278, ed. by J.A. Foster et al. Springer-Verlag, Kinsale, Ireland (3-5 Apr 2002), pp. 172–181. https://doi.org/10.1007/3-540-45984-7_17
106. M. Johnson, Sequence generation using machine language evolved by genetic programming. in *Procceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)*, ed. by L. Wang et al. Orchid Country Club, Singapore (18-22 Nov 2002), p. #1251. http://www.worldcat.org/title/seal02-proceedings-of-the-4th-asia-pacific-conference-on-simulated-evolution-and-learning-november-18-22-2002-orchid-country-club-singapore/oclc/51951214
107. M. O'Neill, C. Ryan, Investigations into memory in grammatical evolution. in *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, ed. by A.M. Barry. AAAI, New York (8 Jul 2002), pp. 141–144. http://www.grammatical-evolution.org/gews2002/oneill.ps
108. N. Pillay, Using genetic programming for the induction of novice procedural programming solution algorithms. in *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*. ACM Press, Madrid, Spain (Mar 2002), pp. 578–583. https://doi.org/10.1145/508791.508903
109. M.I. Quintana et al., Morphological algorithm design for binary images using genetic programming. Genet. Program Evolvable Mach. **7**(1), 81–102 (2006). https://doi.org/10.1007/s10710-006-7012-3
110. M. Segond et al., Iterative filter generation using genetic programming. in *Proceedings of the 9th European Conference on Genetic Programming*. Lecture Notes in Computer Science, vol. 3905, ed. by P. Collet et al. Springer, Budapest, Hungary (10 - 12 Apr 2006), pp. 145–153. https://doi.org/10.1007/11729976_13
111. D. Kim, A quantitative analysis of memory usage for agent tasks, chap. 14, in *Frontiers in Evolutionary Robotics*. ed. by H. Iba (IntechOpen, Rijeka, 2008), pp.247–274. https://doi.org/10.5772/5458
112. E. Frias-Martinez, F. Gobet, Automatic generation of cognitive theories using genetic programming. Mind. Mach. **17**(3), 287–309 (2007). https://doi.org/10.1007/s11023-007-9070-6
113. N.F. McPhee, R. Poli, Memory with memory: Soft assignment in genetic programming. in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ed. by M. Keijzer et al. ACM, Atlanta, GA, USA (12-16 Jul 2008), pp. 1235–1242. https://doi.org/10.1145/1389095.1389336
114. G. Katz, D. Peled, Genetic programming and model checking: Synthesizing new mutual exclusion algorithms, in *Automated Technology for Verification and Analysis. Lecture Notes in Computer Science*, vol. 5311, ed. by S. Cha, J.Y. Choi, M. Kim, I. Lee, M. Viswanathan (Springer, Berlin, 2008), pp.33–47. https://doi.org/10.1007/978-3-540-88387-6_5
115. M.S. Withall et al., An improved representation for evolving programs. Genet. Program Evolvable Mach. **10**(1), 37–70 (2009). https://doi.org/10.1007/s10710-008-9069-7
116. G.C. Wilson, W. Banzhaf, Soft memory for stock market analysis using linear and developmental genetic programming. in *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ed. by G. Raidl et al. ACM, Montreal (8-12 Jul 2009), pp. 1633–1640. https://doi.org/10.1145/1569901.1570119
117. K. Wolfson, M. Sipper, Efficient list search algorithms. in *9th International Conference, Evolution Artificielle, EA 2009*. Lecture Notes in Computer Science, vol. 5975, ed. by P. Collet et al. Springer, Strasbourg, France (Oct 26-28 2009), p. 158–169. https://doi.org/10.1007/978-3-642-14156-0_14, revised Selected Papers
118. M. Hyde, *A genetic programming hyper-heuristic approach to automated packing*. Ph.D. thesis, School of Computer Science, University of Nottingham, UK (Mar 2010). http://etheses.nottingham.ac.uk/1625/1/mvh_corrected_thesis.pdf
119. M. Suchorzewski, Extending genetic programming to evolve perceptron-like learning programs. in *10th International Conference Artificial Intelligence and Soft Computing, ICAISC 2010, Part*

II. Lecture Notes in Computer Science, vol. 6114, ed. by L. Rutkowski et al. Springer, Zakopane, Poland (Jun 13-17 2010), pp. 221–228. https://doi.org/10.1007/978-3-642-13232-2

120. A. Agapitos et al., Learning environment models in car racing using stateful genetic programming. in *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games*. pp. 219–226. IEEE, Seoul, South Korea (31 Aug–3 Sep 2011). https://doi.org/10.1109/CIG.2011.6032010
121. T. Weise, K. Tang, Evolving distributed algorithms with genetic programming. IEEE Trans. Evol. Comput. **16**(2), 242–265 (2012). https://doi.org/10.1109/TEVC.2011.2112666
122. R. Kala, Multi-robot path planning using co-evolutionary genetic programming. Expert Syst. Appl. **39**(3), 3817–3831 (2012). https://doi.org/10.1016/j.eswa.2011.09.090
123. H. Yim, D. Kim, Evolving internal memory strategies for the woods problems. in *12th International Conference on Control, Automation and Systems (ICCAS 2012)*, (2012), pp. 366–369. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp= &arnumber=6393463
124. K. Igwe, N. Pillay, Automatic programming using genetic programming. in *Proceedings of the 2013 Third World Congress on Information and Communication Technologies (WICT 2013)*, ed. by L.T. Ngo et al. IEEE, Hanoi, Vietnam (15-18 Dec 2013), pp. 337–342. https://doi.org/10.1109/WICT.2013.7113158
125. O. Qadir et al., Hardware architecture of the protein processing associative memory and the effects of dimensionality and quantisation on performance. Genet. Program Evolvable Mach. **15**(3), 245–275 (2014). https://doi.org/10.1007/s10710-014-9217-1
126. P. Szczuko, Genetic programming extension to APF-based monocular human body pose estimation. Multimed. Tools Appl. **68**(1), 177–192 (2014). https://doi.org/10.1007/s11042-012-1147-4
127. X. Yuan et al., Making lock-free data structures verifiable with artificial transactions. in *Proceedings of the 8th Workshop on Programming Languages and Operating Systems, PLOS 2015*. ACM, Monterey, California, USA (4-7 Oct 2015), pp. 39–45. https://doi.org/10.1145/2818302.2818309
128. N. Chaumont, C. Adami, Evolution of sustained foraging in three-dimensional environments with physics. Genet. Program Evolvable Mach. **17**(4), 359–390 (2016). https://doi.org/10.1007/s10710-016-9270-z
129. R. Smith, M. Heywood, A model of external memory for navigation in partially observative visual reinforcement learning tasks. in *EuroGP 2019: Proceedings of the 22nd European Conference on Genetic Programming*. LNCS, vol. 11451, ed. by L. Sekanina et al. Springer Verlag, Leipzig, Germany (24-26 Apr 2019), pp. 162–177. https://doi.org/10.1007/978-3-030-16670-0_11
130. S. Kelly et al., Emergent tangled program graphs in partially observable recursive forecasting and ViZDoom navigation tasks. ACM Trans. Evolut. Learn. Optim. **1**(3), 1–41 (2021). https://doi.org/10.1145/3468857
131. E. Real et al., AutoML-zero: Evolving machine learning algorithms from scratch. in *Proceedings of the 37th International Conference on Machine Learning*. *Proceedings of Machine Learning Research*, vol. 119, ed. by H. Daume III, A. Singh, PMLR (13–18 Jul 2020), pp. 8007–8019. http://www.human-competitive.org/sites/default/files/automl_zero_humies_competition_entry.txt, winner 2021 HUMIES
132. C. Sulyok et al., Evolving the process of a virtual composer. Nat. Comput. **18**(1), 47–60 (2019). https://doi.org/10.1007/s11047-016-9561-6
133. M. Al Masalma, M. Heywood, Genetic programming with external memory in sequence recall tasks. in *Proceedings of the 2022 Genetic and Evolutionary Computation Conference Companion*, ed. by H. Trautmann et al. GECCO '22, Association for Computing Machinery, Boston, USA (9-13 Jul 2022), pp. 518–521. https://doi.org/10.1145/3520304.3528883
134. W.B. Langdon, R. Poli, *Foundations of Genetic Programming* (Springer-Verlag, Berlin, 2002). https://doi.org/10.1007/978-3-662-04726-2
135. U.M. O'Reilly, F. Oppacher, The troubling aspects of a building block hypothesis for genetic programming, in *Foundations of Genetic Algorithms 3*. ed. by L.D. Whitley, M.D. Vose (Morgan Kaufmann, Estes Park, 1994), pp.73–88. https://doi.org/10.1016/B978-1-55860-356-1.50008-X
136. J.P. Rosca, D.H. Ballard, Rooted-tree schemata in genetic programming, chap. 11, in *Advances in Genetic Programming 3*. ed. by L. Spector et al. (MIT Press, Cambridge, 1999), pp.243–271. https://doi.org/10.7551/mitpress/1110.003.0015
137. R. Poli, Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. Genet. Program Evolvable Mach. **2**(2), 123–163 (2001). https://doi.org/10.1023/A:1011552313821

138. C.R. Stephens, R. Poli, EC theory–" in theory": Towards a unification of evolutionary computation theory, chap. 7, in *Frontiers of Evolutionary Computation*, vol. 11, ed. by A. Menon (Kluwer, Boston, 2004), pp.129–155. https://doi.org/10.1007/1-4020-7782-3_7
139. G.R. Price, Selection and covariance. Nature **227**, 520–521 (1970). https://doi.org/10.1038/227520a0
140. L. Altenberg, The evolution of evolvability in genetic programming, chap. 3, in *Advances in Genetic Programming*. ed. by K.E. Kinnear Jr. (MIT Press, Cambridge, 1994), pp.47–74
141. C. Ryan et al., A competitive building block hypothesis, in *Genetic and Evolutionary Computation - GECCO-2004, Part II. Lecture Notes in Computer Science*, vol. 3103, ed. by K. Deb et al. (Springer-Verlag, Seattle, 2004), pp.654–665. https://doi.org/10.1007/978-3-540-24855-2_73
142. D.R. White et al., Modelling genetic programming as a simple sampling algorithm, in *Genetic Programming Theory and Practice XVII*. ed. by W. Banzhaf et al. (Springer, East Lansing, 2019), pp.367–381. https://doi.org/10.1007/978-3-030-39958-0_18
143. J. Miller, What bloat? cartesian genetic programming on Boolean problems. in *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, ed. by E.D. Goodman. San Francisco, California, USA (9-11 Jul 2001), pp. 295–302. http://www.elec.york.ac.uk/intsys/users/jfm7/gecco2001Late.pdf
144. T. Jones, One operator, one landscape. Tech. Rep. SFI TR 95-02-025, Santa Fe Institute (January 1995). http://www.santafe.edu/sfi/publications/Working-Papers/95-02-025.ps
145. U.M. O'Reilly, Using a distance metric on genetic programs to understand genetic operators. in *IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation*. vol. 5, Orlando, Florida, USA (12-15 Oct 1997), pp. 4092–4097. https://doi.org/10.1109/ICSMC.1997.637337
146. V.K. Vassilev et al., Smoothness, ruggedness and neutrality of fitness landscapes: from theory to application, in *Advances in Evolutionary Computing: Theory and Applications*. ed. by A. Ghosh, S. Tsutsui (Springer-Verlag, New York, 2003), pp.3–44. https://doi.org/10.1007/978-3-642-18965-4_1
147. W.B. Langdon, M. Harman, Fitness landscape of the Triangle program. in *PPSN-2016 Workshop on Landscape-Aware Heuristic Search*, ed. by N. Veerapen, G. Ochoa. Edinburgh (17 Sep 2016). http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research_Notes/rn1605.pdf, also available as UCL RN/16/05
148. W.B. Langdon et al., Dissipative polynomials. in *5th Workshop on Landscape-Aware Heuristic Search*, ed. by N. Veerapen et al. GECCO 2021 Companion, ACM, Internet (10-14 Jul 2021), pp. 1683–1691. https://doi.org/10.1145/3449726.3463147
149. F.D. Francone et al., Homologous crossover in genetic programming. in Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, ed. by W. Banzhaf et al. Morgan Kaufmann, Orlando, Florida, USA (13-17 Jul 1999), pp. 1021–1026. http://gpbib.cs.ucl.ac.uk/gecco1999/GP-463.pdf
150. G. Durrett et al., Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics, in *Foundations of Genetic Algorithms*. ed. by H.G. Beyer, W.B. Langdon (ACM, Schwarzenberg, 2011), pp.69–80. https://doi.org/10.1145/1967654.1967661
151. T. Koetzing et al., The Max problem revisited: the importance of mutation in genetic programming. Theoret. Comput. Sci. **545**, 94–107 (2014). https://doi.org/10.1016/j.tcs.2013.06.014
152. A. Nguyen et al., Single- and multi-objective genetic programming: new bounds for weighted order and majority, in *Foundations of Genetic Algorithms*. ed. by F. Neumann, K. De Jong (ACM, Adelaide, 2013), pp.161–172. https://doi.org/10.1145/2460239.2460254
153. A. Lissovoi, P.S. Oliveto, On the time and space complexity of genetic programming for evolving boolean conjunctions. J. Artif. Intell. Res. **66**, 655–689 (2019). https://doi.org/10.1613/jair.1.11821
154. B. Doerr et al., The impact of lexicographic parsimony pressure for ORDER/MAJORITY on the run time. Theoret. Comput. Sci. **816**, 144–168 (2020). https://doi.org/10.1016/j.tcs.2020.01.011
155. T. Koetzing et al., Destructiveness of lexicographic parsimony pressure and alleviation by a concatenation crossover in genetic programming. Theoret. Comput. Sci. **816**, 96–113 (2020). https://doi.org/10.1016/j.tcs.2019.11.036
156. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
157. W.B. Langdon, Incremental evaluation in genetic programming. in *EuroGP 2021: Proceedings of the 24th European Conference on Genetic Programming*. LNCS, vol. 12691, ed. by T. Hu et al.

Springer Verlag, Virtual Event (7-9 Apr 2021), pp. 229–246. https://doi.org/10.1007/978-3-030-72812-0_15
158. S. Handley, On the use of a directed acyclic graph to represent a population of computer programs. in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*. vol. 1, IEEE Press, Orlando, Florida, USA (27-29 Jun 1994), pp. 154–159. https://doi.org/10.1109/ICEC.1994.350024
159. W.B. Langdon, W. Banzhaf, Repeated sequences in linear genetic programming genomes. Complex Syst. **15**(4), 285–306 (2005)
160. T. Soule, J.A. Foster, Effects of code growth and parsimony pressure on populations in genetic programming. Evolut. Comput. **6**(4), 293–309 (1998). https://doi.org/10.1162/evco.1998.6.4.293
161. E.D. de Jong, J.B. Pollack, Multi-objective methods for tree size control. Genet. Program Evolvable Mach. **4**(3), 211–233 (2003). https://doi.org/10.1023/A:1025122906870
162. S. Bleuler et al., Multiobjective genetic programming: Reducing bloat using spea2. in *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*. IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea (27-30 May 2001), pp. 536–543. https://doi.org/10.1109/CEC.2001.934438
163. L. Panait, S. Luke, Alternative bloat control methods, in *Genetic and Evolutionary Computation - GECCO-2004, Part II. Lecture Notes in Computer Science*, vol. 3103, ed. by K. Deb et al. (Springer-Verlag, Seattle, 2004), pp.630–641. https://doi.org/10.1007/b98645
164. R. Poli, A simple but theoretically-motivated method to control bloat in genetic programming. in *Genetic Programming, Proceedings of EuroGP'2003*. LNCS, vol. 2610, ed. by C. Ryan et al. Springer-Verlag, Essex (14-16 Apr 2003), pp. 204–217. https://doi.org/10.1007/3-540-36599-0_19
165. S. Silva, *Controlling Bloat: Individual and Population Based Approaches in Genetic Programming*. Ph.D. thesis, Coimbra University, Portugal (Apr 2008). http://hdl.handle.net/10316/8542, full author name is Sara Guilherme Oliveira da Silva
166. S. Dignum, R. Poli, Operator equalisation and bloat free GP. in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*. Lecture Notes in Computer Science, vol. 4971, ed. by M. O'Neill et al. Springer, Naples (26-28 Mar 2008), pp. 110–121. https://doi.org/10.1007/978-3-540-78671-9_10
167. N.I. Nikolaev, H. Iba, Accelerated genetic programming of polynomials. Genet. Program Evolvable Mach. **2**(3), 231–257 (2001). https://doi.org/10.1023/A:1011949326249
168. I. Kushchu, Genetic programming and evolutionary generalization. IEEE Trans. Evol. Comput. **6**(5), 431–442 (2002). https://doi.org/10.1109/TEVC.2002.805038
169. T. Kowaliw, R. Doursat, Bias-variance decomposition in genetic programming. Open Math. **14**(1), 62–80 (2016). https://doi.org/10.1515/math-2016-0005
170. C. Gathercole, P. Ross, Dynamic training subset selection for supervised learning in genetic programming, in *Parallel Problem Solving from Nature III. LNCS*, vol. 866, ed. by Y. Davidor et al. (Springer-Verlag, Jerusalem, 1994), pp.312–321. https://doi.org/10.1007/3-540-58484-6_275
171. L. Spector et al., Relaxations of lexicase parent selection, in *Genetic Programming Theory and Practice XV. Genetic and Evolutionary Computation*. ed. by W. Banzhaf et al. (Springer, Cham, 2017), pp.105–120. https://doi.org/10.1007/978-3-319-90512-9_7
172. N. Javed et al., Simplification of genetic programs: a literature survey. Data Min. Knowl. Discov. **36**(4), 1279–1300 (2022). https://doi.org/10.1007/s10618-022-00830-7
173. D. Hooper, N.S. Flann, Improving the accuracy and robustness of genetic programming through expression simplification. in *Genetic Programming 1996: Proceedings of the First Annual Conference*, ed. by J.R. Koza et al. MIT Press, Stanford University, CA, USA (28–31 Jul 1996), p. 428. http://digital.cs.usu.edu/~flann/gp.pdf
174. W.B. Langdon, M. Harman, Optimising existing software with genetic programming. IEEE Trans. Evol. Comput. **19**(1), 118–135 (2015). https://doi.org/10.1109/TEVC.2013.2281544
175. M.E. Raichle, D.A. Gusnard, Appraising the brain's energy budget. Proc. Natl. Acad. Sci. **99**(16), 10237–10239 (2002). https://doi.org/10.1073/pnas.172399499
176. M. Ridley, The Red Queen, Sex and the Evolution of Human Nature. Penquin (1993). http://www.penguin.co.uk/Penguin/Books/0140167722.html
177. P. Nordin, A compiling genetic programming system that directly manipulates the machine code, chap. 14, in *Advances in Genetic Programming*. ed. by K.E. Kinnear Jr. (MIT Press, Cambridge, 1994), pp.311–331
178. J.F. Miller, An empirical study of the efficiency of learning Boolean functions using a cartesian genetic programming approach. in *Proceedings of the Genetic and Evolutionary Computation*

Conference. vol. 2, ed. by W. Banzhaf et al. Morgan Kaufmann, Orlando, Florida, USA (13-17 Jul 1999), pp. 1135–1142. http://citeseer.ist.psu.edu/153431.html

179. C. Ryan et al., Grammatical evolution: Evolving programs for an arbitrary language. in *Proceedings of the First European Workshop on Genetic Programming*. LNCS, vol. 1391, ed. by W. Banzhaf et al. Springer-Verlag, Paris (14-15 Apr 1998), pp. 83–96. https://doi.org/10.1007/BFb0055930
180. P.J. Angeline, J.B. Pollack, The evolutionary induction of subroutines. in *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*. pp. 236–241. Lawrence Erlbaum, Bloomington, Indiana, USA (1992), http://www.demo.cs.brandeis.edu/papers/glib92.pdf
181. J. Rosca, Towards automatic discovery of building blocks in genetic programming. in *Working Notes for the AAAI Symposium on Genetic Programming*, ed. by E.V. Siegel, J.R. Koza. AAAI, MIT, Cambridge, MA, USA (10–12 Nov 1995), pp. 78–85. http://www.aaai.org/Papers/Symposia/Fall/1995/FS-95-01/FS95-01-011.pdf
182. L. Spector, Simultaneous evolution of programs and their control structures, chap. 7, in *Advances in Genetic Programming 2*. ed. by P.J. Angeline, K.E. Kinnear Jr. (MIT Press, Cambridge, 1996), pp.137–154. https://doi.org/10.7551/mitpress/1109.003.0012
183. G. Murphy, C. Ryan, Seeding methods for run transferable libraries. in *GECCO '07: Proceedings of the 9th annual conference on Genetic and Evolutionary Computation*. vol. 2,ed. by D. Thierens et al. ACM Press, London (7-11 Jul 2007), pp. 1755–1755. https://doi.org/10.1145/1276958.1277305
184. W.B. Langdon, *Data Structures and Genetic Programming*. Ph.D. thesis, University College, London, UK (27 Sep 1996), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/langdon.ps.gz
185. A. Teller, D. Andre, Automatically choosing the number of fitness cases: The rational allocation of trials. in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, ed. by J.R. Koza et al. Morgan Kaufmann, Stanford University, CA, USA (13-16 Jul 1997), pp. 321–328. http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/GR.ps
186. L. Spector, Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report, in *1st workshop on Understanding Problems (GECCO-UP)*. ed. by K. McClymont, E. Keedwell (ACM, Philadelphia, 2012), pp.401–408. https://doi.org/10.1145/2330784.2330846
187. R. Poli et al., *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008), http://www.gp-field-guide.org.uk, (With contributions by J. R. Koza)
188. S.L. Harding, W. Banzhaf, Distributed genetic programming on GPUs using CUDA, in *Workshop on Parallel Architectures and Bioinspired Algorithms*. ed. by I. Hidalgo et al. (Universidad Complutense de Madrid, Raleigh, 2009), pp.1–10
189. R.L. Crepeau, Genetic evolution of machine language software. in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, ed. by J.P. Rosca. Tahoe City, California, USA (9 Jul 1995), pp. 121–134. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/GEMS_Article.pdf
190. P. Nordin et al., Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover, chap. 12, in *Advances in Genetic Programming 3*. ed. by L. Spector et al. (MIT Press, Cambridge, 1999), pp.275–299. https://doi.org/10.7551/mitpress/1110.003.0017
191. F.D. Francone, *Discipulus Owner's Manual*. 11757 W. Ken Caryl Avenue F, PBM 512, Littleton, Colorado, 80127-3719, USA, version 3.0 draft edn. (2001). http://gpbib.cs.ucl.ac.uk/gp-html/francone_manual.html
192. W. Banzhaf et al., *Genetic Programming-An Introduction;On the Automatic Evolution of Computer Programs and its Applications* (Morgan Kaufmann, San Francisco, 1998)
193. M. Brameier, W. Banzhaf, *Linear Genetic Programming. No. XVI in Genetic and Evolutionary Computation* (Springer, Berlin, 2007)
194. M. O'Neill, C. Ryan, Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001). https://doi.org/10.1109/4235.942529
195. M. O'Neill, C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Genetic programming*, vol. 4 (Kluwer Academic Publishers, Dordrecht, 2003). https://doi.org/10.1007/978-1-4615-0447-4
196. L. Spector, Introduction to the peer commentary special section on "on the mapping of genotype to phenotype in evolutionary algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin. Genetic Programming and Evolvable Machines 18(3), 351–352 (Sep 2017). https://doi.org/

10.1007/s10710-017-9287-y, special Peer Commentary on Mapping of Genotype to Phenotype in Evolutionary Algorithms

197. C. Ryan, A rebuttal to whigham, dick, and maclaurin by one of the inventors of grammatical evolution: Commentary on "on the mapping of genotype to phenotype in evolutionary algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin. Genetic Programming and Evolvable Machines 18(3), 385–389 (Sep 2017). https://doi.org/10.1007/s10710-017-9294-z, special Peer Commentary on Mapping of Genotype to Phenotype in Evolutionary Algorithms
198. J.F. Miller et al., Principles in the evolutionary design of digital circuits-part I. Genet. Program Evolvable Mach. **1**(1/2), 7–35 (2000). https://doi.org/10.1023/A:1010016313373
199. J.F. Miller et al., Principles in the evolutionary design of digital circuits-part II. Genet. Program Evolvable Mach. **1**(3), 259–288 (2000). https://doi.org/10.1023/A:1010066330916
200. J.F. Miller (ed.), *Cartesian Genetic Programming. Natural Computing Series* (Springer, Berlin, 2011). https://doi.org/10.1007/978-3-642-17310-3
201. J.F. Miller, Cartesian Genetic Programming: its status and future. Genetic Programming and Evolvable Machines **21**(1–2), 129–168 (2020). https://doi.org/10.1007/s10710-019-09360-6
202. L. Sekanina, Z. Vasicek, Approximate circuit design by means of evolvable hardware. in *IEEE International Conference on Evolvable Systems (ICES 2013)*. (Apr 2013), pp. 21–28. https://doi.org/10.1109/ICES.2013.6613278
203. L. Sekanina et al., Approximate circuits in low-power image and video processing: The approximate median filter. Radioengineering **26**(3), 623–632 (2017). https://doi.org/10.13164/re.2017.0623
204. D.J. Montana, Strongly typed genetic programming. Evolutionary Computation **3**(2), 199–230 (1995). https://doi.org/10.1162/evco.1995.3.2.199
205. T. Yu, Structure abstraction and genetic programming. in *Proceedings of the Congress on Evolutionary Computation*. vol. 1, ed. by P.J. Angeline et al. IEEE Press, Mayflower Hotel, Washington D.C., USA (6-9 Jul 1999), pp. 652–659. https://doi.org/10.1109/CEC.1999.781995
206. T. Yu, Hierachical processing for evolving recursive and modular programs using higher order functions and lambda abstractions. Genet. Program Evolvable Mach. **2**(4), 345–380 (2001). https://doi.org/10.1023/A:1012926821302
207. R.I. McKay et al., Grammar-based genetic programming: a survey. Genet. Program. Evolvable Mach. **11**(3/4), 365–396 (2010). https://doi.org/10.1007/s10710-010-9109-y
208. P.A. Whigham, Grammatically-based genetic programming. in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, ed. by J.P. Rosca. Tahoe City, California, USA (9 Jul 1995), pp. 33–41. http://divcom.otago.ac.nz/sirc/Peterw/Publications/ml95.zip
209. P.A. Whigham et al., On the mapping of genotype to phenotype in evolutionary algorithms. Genet. Program. Evolvable Mach. **18**(3), 353–361 (2017). https://doi.org/10.1007/s10710-017-9288-x
210. A. Ratle, M. Sebag, A novel approach to machine discovery: Genetic programming and stochastic grammars. in *Proceedings of Twelfth International Conference on Inductive Logic Programming*. LNCS, vol. 2583, ed. by S. Matwin, C. Sammut. Springer Verlag, Sydney, Australia (Jul 9-11 2002), pp. 207–222. https://doi.org/10.1007/3-540-36468-4_14, revised Papers
211. X.H. Nguyen et al., Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. Aust. J. Intell. Inform. Process. Syst. **7**(3/4), 114–121 (2001)
212. C. Jacob, Evolution and coevolution of developmental programs. Comput. Phys. Commun. **121–122**, 46–50 (1999). https://doi.org/10.1016/S0010-4655(99)00277-5
213. C. Jacob, *Illustrating Evolutionary Computation with Mathematica* (Morgan Kaufmann, Cambridge, 2001). https://doi.org/10.1016/B978-155860637-1/50020-5
214. G.S. Hornby, J.B. Pollack, Evolving L-systems to generate virtual creatures. Comput. Graph. **25**(6), 1041–1048 (2001). https://doi.org/10.1016/S0097-8493(01)00157-1. (**artificial Life**)
215. M. Hemberg et al., Genr8: Architects' experience with an emergent design tool, in *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music, chap. 8*. ed. by J. Romero, P. Machado (Springer, 2008), pp.167–188. https://doi.org/10.1007/978-3-540-72877-1_8
216. T. Perkis, Stack-based genetic programming. in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*. vol. 1, pp. 148–153. IEEE Press, Orlando, Florida, USA (27-29 Jun 1994). https://doi.org/10.1109/ICEC.1994.350025

217. S. Openshaw, I. Turton, Building new spatial interaction models using genetic programming, in *Evolutionary Computing*. ed. by T.C. Fogarty (AISB workshop, Leeds, UK, 1994). https://doi.org/10.1007/3-540-58483-8
218. K. Holladay et al., Fifth: A stack based gp language for vector processing. in *Proceedings of the 10th European Conference on Genetic Programming*. Lecture Notes in Computer Science, vol. 4445, ed. by M. Ebner et al. pp. 102–113. Springer, Valencia, Spain (11-13 Apr 2007). https://doi.org/10.1007/978-3-540-71605-1_10
219. M. Oltean, C. Grosan, Solving classification problems using infix form genetic programming, in *Advances in Intelligent Data Analysis V. Lecture Notes in Computer Science*, vol. 2810, ed. by M.R. Berthold et al. (Springer, Berlin, 2003), pp.242–253. https://doi.org/10.1007/978-3-540-45231-7_23
220. L. Spector, A. Robinson, Genetic programming and autoconstructive evolution with the push programming language. Genet. Program Evolvable Mach. **3**(1), 7–40 (2002). https://doi.org/10.1023/A:1014538503543
221. U.M. O'Reilly, F. Oppacher, Program search with a hierarchical variable length representation: genetic programming, simulated annealing and hill climbing, in *Parallel Problem Solving from Nature - PPSN III. Lecture Notes in Computer Science*, vol. 866, ed. by Y. Davidor et al. (Springer-Verlag, Jerusalem, 1994), pp.397–406. https://doi.org/10.1007/3-540-58484-6_283
222. A.I. Esparcia-Alcazar, K.C. Sharman, Genetic programming techniques that evolve recurrent neural networks architectures for signal processing. in *IEEE Workshop on Neural Networks for Signal Processing*. IEEE, Seiko, Kyoto, Japan (4-6 Sep 1996), pp. 139–148. https://doi.org/10.1109/NNSP.1996.548344
223. A. Moraglio, S. Silva, Geometric differential evolution on the space of genetic programs. in *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*. LNCS, vol. 6021, ed. by A.I. Esparcia-Alcazar et al. Springer, Istanbul (7-9 Apr 2010), pp. 171–183. https://doi.org/10.1007/978-3-642-12148-7_15, best paper
224. B.T. Zhang, Bayesian methods for efficient genetic programming. Genet. Program Evolvable Mach. **1**(3), 217–242 (2000). https://doi.org/10.1023/A:1010010230007
225. K. Yanai, H. Iba, Estimation of distribution programming based on Bayesian network. in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, ed. by R. Sarker et al. IEEE Press, Canberra (8-12 Dec 2003), pp. 1618–1625. https://doi.org/10.1109/CEC.2003.1299866
226. P.A.N. Bosman, E.D. de Jong, Learning probabilistic tree grammars for genetic programming, in *Parallel Problem Solving from Nature - PPSN VIII. LNCS*, vol. 3242, ed. by X. Yao et al. (Springer-Verlag, Birmingham, 2004), pp.192–201. https://doi.org/10.1007/b100601
227. A. Rodriguez, *A Neat Approach To Genetic Programming*. Master's thesis, School of School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, USA (2007). https://stars.library.ucf.edu/etd/3323.pdf
228. Z. Buk et al., NEAT in HyperNEAT substituted with genetic programming. in *9th International Conference on Adaptive and Natural Computing Algorithms, ICANNGA 2009*. Lecture Notes in Computer Science, vol. 5495, ed. by M. Kolehmainen et al. Springer, Kuopio, Finland (23-25 Apr 2009), pp. 243–252. https://doi.org/10.1007/978-3-642-04921-7_25, revised selected papers
229. L. Trujillo et al., neat genetic programming: Controlling bloat naturally. Inf. Sci. **333**, 21–43 (2016). https://doi.org/10.1016/j.ins.2015.11.010
230. T. McConaghy, FFX: fast, scalable, deterministic symbolic regression technology, chap. 13, in *Genetic Programming Theory and Practice IX. Genetic and Evolutionary Computation*. ed. by R. Riolo et al. (Springer, Ann Arbor, 2011), pp.235–260. https://doi.org/10.1007/978-1-4614-1770-5_13
231. A. Moraglio et al., Geometric semantic genetic programming, in *Parallel Problem Solving from Nature, PPSN XII (part 1). Lecture Notes in Computer Science*, vol. 7491, ed. by C.A. Coello Coello et al. (Springer, Taormina, 2012), pp.21–31. https://doi.org/10.1007/978-3-642-32937-1_3
232. W.B. Langdon, *Directed crossover within genetic programming*. Research Note RN/95/71, University College London, Gower Street, London WC1E 6BT, UK (Sep 1995), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/directed_crossover.pdf
233. P. Orzechowski et al., Where are we now?: a large benchmark study of recent symbolic regression methods. in *GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference*,

ed. by H. Aguirre et al. ACM, Kyoto, Japan (15-19 Jul 2018), pp. 1183–1190. https://doi.org/10.1145/3205455.3205539
234. I. Arnaldo et al., Multiple regression genetic programming. in *GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation*, ed. by C. Igel et al. ACM, Vancouver, BC, Canada (12-16 Jul 2014), pp. 879–886. https://doi.org/10.1145/2576768.2598291
235. L. Munoz et al., M3GP: multiclass classification with GP. in *18th European Conference on Genetic Programming*. LNCS, vol. 9025, ed. by P. Machado et al. Springer, Copenhagen (8-10 Apr 2015), pp. 78–91. https://doi.org/10.1007/978-3-319-16501-1_7
236. W. La Cava, J. Moore, A general feature engineering wrapper for machine learning using epsilon-lexicase survival. in *EuroGP 2017: Proceedings of the 20th European Conference on Genetic Programming*. LNCS, vol. 10196, ed. by M. Castelli et al. Springer Verlag, Amsterdam (19-21 Apr 2017), pp. 80–95. https://doi.org/10.1007/978-3-319-55696-3_6
237. B. Burlacu et al., Operon C++: An efficient genetic programming framework for symbolic regression. in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ed. by R. Allmendinger et al. GECCO '20, Association for Computing Machinery, internet (Jul 8-12 2020), pp. 1562–1570. https://doi.org/10.1145/3377929.3398099
238. D. Mota Dias et al., Automatic synthesis of microcontroller assembly code through linear genetic programming, in *Genetic Systems Programming: Theory and Experiences, Studies in Computational Intelligence*, vol. 13, ed. by N. Nedjah et al. (Springer, Germany, 2006), pp.193–227. https://doi.org/10.1007/3-540-32498-4_9
239. T.E. Lewis, G.D. Magoulas, TMBL kernels for CUDA GPUs compile faster using PTX, in *GECCO 2011 Computational Intelligence on Consumer Games and Graphics Hardware (CIGPU)*. ed. by S. Harding et al. (ACM, Dublin, 2011), pp.455–462. https://doi.org/10.1145/2001858.2002033
240. L.F. Cupertino et al., Evolving CUDA PTX programs by quantum inspired linear genetic programming, in *GECCO 2011 Computational intelligence on Consumer Games and Graphics Hardware (CIGPU)*. ed. by S. Harding et al. (ACM, Dublin, 2011), pp.399–406. https://doi.org/10.1145/2001858.2002026
241. M. Gregor, J. Spalek, Using LLVM-based JIT compilation in genetic programming. In: 2016 ELEKTRO. pp. 406–411. IEEE, Strbske Pleso, Slovakia (16-18 May 2016). https://doi.org/10.1109/ELEKTRO.2016.7512108
242. J.Y. Liou et al., GEVO: GPU code optimization using evolutionary computation. ACM Trans. Archit. Code Optim. **17**(4), 33 (2020). https://doi.org/10.1145/3418055
243. E. Lukschandl et al., Distributed java bytecode genetic programming. in *Genetic Programming, Proceedings of EuroGP'2000*. LNCS, vol. 1802, ed. by R. Poli et al. Springer-Verlag, Edinburgh (15-16 Apr 2000), pp. 316–325. https://doi.org/10.1007/978-3-540-46239-2_24
244. P.A. Whigham, R.I. McKay, Genetic approaches to learning recursive relations, in *Progress in Evolutionary Computation. Lecture Notes in Artificial Intelligence*, vol. 956, ed. by X. Yao (Springer-Verlag, Berlin, 1995), pp.17–27. https://doi.org/10.1007/3-540-60154-6_44
245. P.A. Whigham, A schema theorem for context-free grammars. in *1995 IEEE Conference on Evolutionary Computation*. vol. 1, pp. 178–181. IEEE Press, Perth, Australia (29 Nov - 1 Dec 1995). https://doi.org/10.1109/ICEC.1995.489140
246. T. Castle, C.G. Johnson, Evolving high-level imperative program trees with strongly formed genetic programming. in *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*. LNCS, vol. 7244, ed. by A. Moraglio et al. Springer Verlag, Malaga, Spain (11-13 Apr 2012), pp. 1–12. https://doi.org/10.1007/978-3-642-29139-5_1
247. W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, in *Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity*, vol. X, ed. by C.G. Langton et al. (Addison-Wesley, Santa Fe Institute, 1992), pp.313–324
248. E. Popovici et al., Coevolutionary principles, chap. 31, in *Handbook of Natural Computing*. ed. by G. Rozenberg et al. (Springer, Berlin, 2012), pp.987–1033. https://doi.org/10.1007/978-3-540-92910-9_31
249. B.T. Zhang, D.Y. Cho, Coevolutionary fitness switching: Learning complex collective behaviors using genetic programming, chap. 18, in *Advances in Genetic Programming 3*. ed. by L. Spector et al. (MIT Press, Cambridge, 1999), pp.425–445. https://doi.org/10.7551/mitpress/1110.003.0023
250. A. Leier, W. Banzhaf, Exploring the search space of quantum programs. in *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*. vol. 1, ed. by R. Sarker et al. IEEE Press, Canberra (8-12 Dec 2003), pp. 170–177. https://doi.org/10.1109/CEC.2003.1299571

251. L. Spector, *Automatic Quantum Computer Programming: A Genetic Programming Approach, Genetic Programming*, vol. 7 (Kluwer Academic Publishers, Boston, 2004). https://doi.org/10.1007/978-0-387-36791-0
252. G. O'Brien, J. Clark, Using genetic improvement to retarget quantum software on differing hardware. In: Petke, J., et al. (eds.) GI @ ICSE 2021. IEEE, internet (30 May 2021), pp. 31–38. https://doi.org/10.1109/GI52543.2021.00015, winner Best Presentation
253. R. Poli et al., Theoretical results in genetic programming: the next ten years? Genet. Program. Evolvable Mach. **11**(3/4), 285–320 (2010). https://doi.org/10.1007/s10710-010-9110-5
254. L. Vanneschi, R. Poli, Genetic programming: introduction, applications, theory and open issues, chap. 24, in *Handbook of Natural Computing*, vol. 2, ed. by G. Rozenberg et al. (Springer, Berlin, 2012), pp.709–739. https://doi.org/10.1007/978-3-540-92910-9_24
255. A. Marginean et al., SapFix: automated end-to-end repair at scale. in *41st International Conference on Software Engineering*, ed. by J.M. Atlee, T. Bultan, ACM, Montreal (25-31 May 2019), ACM, Montreal (25-31 May 2019), pp. 269-278. https://doi.org/10.1109/ICSE-SEIP.2019.00039
256. B.R. Bruce et al., Approximate oracles and synergy in software energy search spaces. IEEE Trans. Software Eng. **45**(11), 1150–1169 (2019). https://doi.org/10.1109/TSE.2018.2827066
257. F. Wu et al., Deep parameter optimisation. in *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ed. by S. Silva et al. ACM, Madrid (11-15 Jul 2015), pp. 1375–1382. https://doi.org/10.1145/2739480.2754648
258. W.B. Langdon, M. Harman, Genetically improved CUDA C++ software. in *17th European Conference on Genetic Programming*. LNCS, vol. 8599, ed. by M. Nicolau et al. Springer, Granada, Spain (23-25 Apr 2014), pp. 87–99. https://doi.org/10.1007/978-3-662-44303-3_8
259. W.B. Langdon et al., Improving 3D medical image registration CUDA software with genetic programming. in *GECCO '14: Proceeding of the sixteenth annual conference on genetic and evolutionary computation conference*, ed. by C. Igel et al. ACM, Vancouver, BC, Canada (12-15 Jul 2014), pp. 951–958. https://doi.org/10.1145/2576768.2598244
260. W.B. Langdon, M. Harman, Grow and graft a better CUDA pknotsRG for RNA pseudoknot free energy calculation, in *Genetic Improvement 2015 Workshop*. ed. by W.B. Langdon et al. (ACM, Madrid, 2015), pp.805–810. https://doi.org/10.1145/2739482.2768418
261. K. Yeboah-Antwi, B. Baudry, Embedding adaptivity in software systems using the ECSELR framework, in *Genetic Improvement 2015 Workshop*. ed. by W.B. Langdon et al. (ACM, Madrid, 2015), pp.839–844. https://doi.org/10.1145/2739482.2768425
262. W.B. Langdon et al., Improving CUDA DNA analysis software with genetic programming. in *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ed. by S. Silva et al. ACM, Madrid (11-15 Jul 2015), pp. 1063–1070. https://doi.org/10.1145/2739480.2754652
263. W.B. Langdon et al., Genetic improvement of GPU software. Genet. Program Evolvable Mach. **18**(1), 5–44 (2017). https://doi.org/10.1007/s10710-016-9273-9
264. W.B. Langdon, Genetically improved software, chap. 8, in *Handbook of Genetic Programming Applications*. ed. by A.H. Gandomi et al. (Springer, Berlin, 2015), pp.181–220. https://doi.org/10.1007/978-3-319-20883-1_8
265. W.B. Langdon, B.Y.H. Lam, Genetically improved BarraCUDA. BioData Mining (2017). https://doi.org/10.1186/s13040-017-0149-1
266. W.B. Langdon et al., Evolving better RNAfold structure prediction. in *EuroGP 2018: Proceedings of the 21st European Conference on Genetic Programming*. LNCS, vol. 10781, ed. by M. Castelli et al. Springer Verlag, Parma, Italy (4-6 Apr 2018), pp. 220–236. https://doi.org/10.1007/978-3-319-77553-1_14
267. W.B. Langdon, M. Harman, Evolving a CUDA kernel from an nVidia template. in *2010 IEEE World Congress on Computational Intelligence*, ed. by P. Sobrevilla. IEEE, Barcelona (18-23 Jul 2010), pp. 2376–2383. https://doi.org/10.1109/CEC.2010.5585922
268. J.Y. Liou et al., Genetic improvement of GPU code. in *GI-2019, ICSE workshops proceedings*, ed. by J. Petke et al. IEEE, Montreal (28 May 2019), pp. 20–27. https://doi.org/10.1109/GI.2019.00014, best Paper
269. E.T. Barr et al., Automated software transplantation. in *International Symposium on Software Testing and Analysis, ISSTA 2015*, ed. by T. Xie, M. Young. ACM, Baltimore, Maryland, USA (14-17 Jul 2015), pp. 257–269. https://doi.org/10.1145/2771783.2771796, ACM SIGSOFT Distinguished Paper Award

270. E.K. Burke et al., Exploring hyper-heuristic methodologies with genetic programming, chap. 6, in *Computational Intelligence, Intelligent Systems Reference Library*, vol. 1, ed. by C.L. Mumford, L.C. Jain (Berlin, Springer, 2009), pp.177–201. https://doi.org/10.1007/978-3-642-01799-5_6
271. R.S. Olson, J.H. Moore, TPOT: A tree-based pipeline optimization tool for automating data science. In: Hutter, F., et al. (eds.) AutoML 2016 workshop. New York City, USA (Jun 24 2016), https://docs.google.com/viewer?a=v &pid=sites &srcid=ZGVmYXVsdGRvbWFpbnxhdXRvbWwyMDE2fGd4OmFmYjMyNWU2NWI1YTBmZg, collocated with ICML
272. D. Radecic, Machine Learning Automation with TPOT. Packt Publishing (2021). https://www.amazon.com/Machine-Learning-Automation-TPOT-automated/dp/180056788X?asin=180056788X &revisionId= &format=4 &depth=1
273. K. Krawiec, P. Liskowski, Adaptive test selection for factorization-based surrogate fitness in genetic programming. Found. Comput. Decis. Sci. **42**(4), 339–358 (2017). https://doi.org/10.1515/fcds-2017-0017
274. C.G. Johnson, Solving the Rubik's cube with stepwise deep learning. Expert Syst.: J. Knowl. Eng. **38**(3), e12665 (2021). https://doi.org/10.1111/exsy.12665
275. W.B. Langdon, Evolving open complexity. SIGEVOlution Newsl. ACM Spec. Interest Group Genet. Evolut. Comput. **15**(1), 1–4 (2022). https://doi.org/10.1145/3532942.3532945
276. S. Forrest, Engineering and evolving software (2021). https://doi.org/10.1109/GI52543.2021.00008
277. G.E. Moore, Cramming more components onto integrated circuits. Electronics **38**(8), 114–117 (1965)
278. S. Wright, The roles of mutation, inbreeding, crossbreeding and selection in evolution. in *Proceedings of the Sixth Annual Congress of Genetics*. pp. 356–366 (1932). http://www.blackwellpublishing.com/ridley/classictexts/wright.pdf