Genetic Programming and Evolvable Machines (2023) 24:26
https://doi.org/10.1007/s10710-023-09474-y

REPLY

Response to comments on "Jaws 30"

W. B. Langdon[1]

Published online: 22 November 2023

I would like to thank Leonardo Vanneschi and Leonardo Trujillo for the opportunity to lead their peer commentary on the thirtieth anniversary of John R. Koza's book "Genetic Programming: On the programming of computers by means of natural selection" [1] and the colleagues who took the time to read my initial article [2] and kindly comment upon it. They raise important points which I should like to reply to.

In their wittily titled "Veni, Vidi, Evolvi" (I came, I saw, I evolved)[1] *Giovanni Squillero and Alberto Tonda* [3] point to the success of GP at producing better than human results and give pointers to a number of GP tools. Although they suggest GP could be used to a greater extent in the real world, particularly in future highly automated industry, they list many areas where GP is competitive. For example, the design of ensembles of other artificial intelligence (AI) generated models [4], such as deep neural networks [5], and recent successes in our own software industry. Indeed search based software engineering [6] is often GP based and has lead to successes such as automatic bug fixing [7, 8] and genetic improvement of software [9–11], including industrial use [12–14]. Interestingly Squillero and Tonda include areas where they feel that current large language models (using deep neural networks) will never be able to compete with GP [15]. Indeed their reasoning for this, based on the availability of training data, may apply to many other special circumstances. They also suggest sometimes GP will hybridise well with other AI approaches.

Mauro Castelli [16] looks forward 30 years and stresses GP's ability to support multidisciplinary research, particularly with Biology. Although there is some theoretical underpinning for genetic programming [17], he points out both GP and other forms of AI, such as artificial neural networks, are largely empirical, with

[1] Cf. Veni, vidi, vici (I came; I saw; I conquered) Julius Caesar, 47 BC.

This reply refers to the articles available online at https://doi.org/10.1007/s10710-023-09467-x; 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] University College London, London, UK

If glucose **exceeds** 155 **then diabetes**

Fig. 1 Interpretation of evolved 3 node GP tree which performs approximately as well as sophisticated machine learning techniques [21, Fig. 1]

considerable progress being made by skilled researchers following what works in practice. Castelli also makes a plea for more thoughtful consideration about what makes a solution human-interpretable, i.e. meaningful to GP's customers. If computer based systems are understood perhaps this can help companies convince their employees and users that they are fair [18]. There is considerable interest in ways to measure and improve how comprehensible software is [19, 20]. Possibly GP researchers can find insight in software engineering's readability metrics. Indeed in future, perhaps genetic improvement could use automatic comprehensibility measures to make software easier to maintain: after all, if you can measure it, you can evolve it. Also could I add to Castelli's plea, and suggest where possible, especially in presentations, we put the (simplified) evolved model on a slide using the customer's language, e.g. "glucose" rather than "D1" Fig. 1 .[2]

Malcolm Heywood [22] follows up two points made by John Koza on his own work at his GECCO 2022 lecture: parallel GP and co-evolution. As Koza predicted, the thirty plus years since the first GP book [1] have been dominated by the exponential increase in available compute power [23]. It looks likely that this will continue into the near future. However CPU clock speeds may not rise much above the 3.6 GHz common today, instead silicon chip designers will spend the extra transistors available on more CPU cores and on more on chip RAM memory (e.g. for cache memory). Thus continuing today's trend for increasing parallelism and to architectures where compute power is plentiful but distributed and the true costs lie in getting data to each CPU fast enough to keep them all busy [24]. As Heywood points out GP is "embarrassingly parallel" as the algorithm can be readily split into independent work units which can proceed on independent processing units with only limited need for communication or synchronisation between them [25, 26].[3] In the case of GP we can imagine the traditional workload as being divided into a computational cube (Fig. 2) with 3 dimensions: across individuals in the population × across the test cases and × across the opcodes that form each GP program. The computational cube metaphor stresses that the work can be split up in many ways on parallel hardware. Even with a modest GP experiment with a population of 1000, 10 fitness cases, and programs of 10 instructions, that gives us 100 000 items to compute per generation, which in five years time (2028) might map well onto a field programmable gate array (FPGA) or graphics card with 100 000 processing units. (Fig. 2 shows a much more modest population of 4, with 5 test cases and programs containing up to 12 instructions.) Fukunaga et al. [27] showed GP could be run without an interpreter. Whereas Juille showed an imaginative way of running a GP interpreter

[2] At EuroGP 2000 I did not follow today's advice and the information about glucose went into the figure's caption rather than in the figure itself.

[3] At his GECCO 2023 invited keynote Kenneth De Jong said we should be careful to avoid designing our evolutionary computation algorithms with more synchronisation points than necessary.

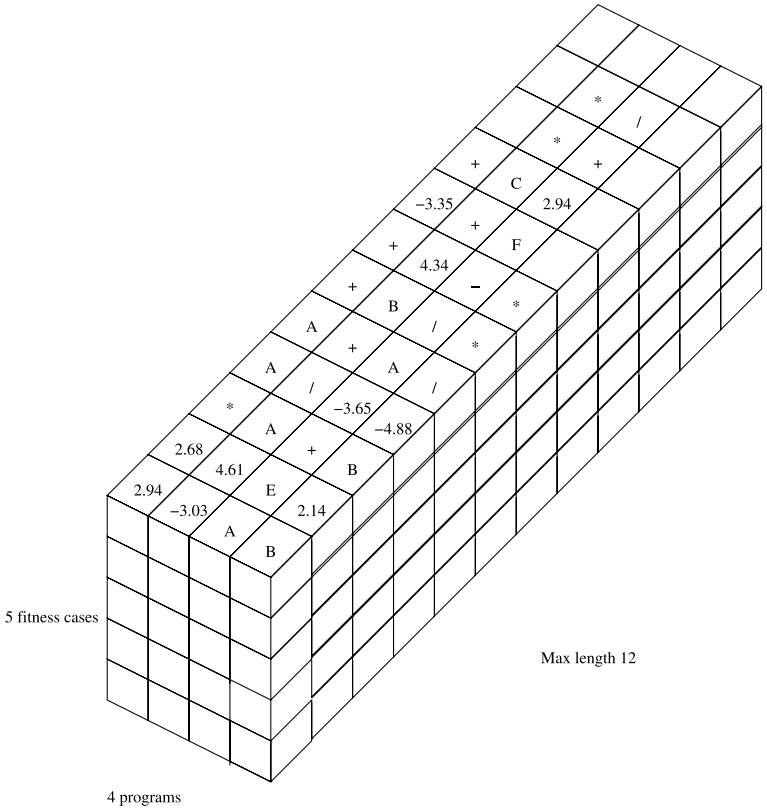

Fig. 2 Evaluating a GP population of four individuals each on the same five fitness cases. There are upto $4 \times 5 \times 12$ GP operations to be performed by, in principle, 240 GPU threads. Each cube needs the opcode to be interpreted, the fitness test case (program inputs) and the previous state of the program (i.e. the stack). Taken from [35, Fig. 19]

on a highly parallel computer [28], which inspired more recent work on running GP on computer gaming or graphics cards (GPUs) [29] (see also iCUBE's EASEA platform which supports GPU computing [30]). Heywood also mentions exploiting parallel hardware in the form of FPGA [31, 32]. Another area, which every one hopes will become available soon, is Quantum Computing [33]. Although Quantum Computing is at present limited in terms of number of Q-bits, evolution has already been shown to be able to help improve the reliability of existing quantum algorithms [34].

Heywood [22] also talks of the many cases, since Koza's first book [1], where GP has contributed to the exciting area of co-evolution. Including both competitive "red queen" coevolution [36], and co-operative coevolution (such as the evolution of multiple tree individuals [37] and ADFs [38]). He gives competitive coevolution "arms race" [39] examples, such as simultaneously evolving a program and its test suite [40], where programs are evolved to pass tests but the tests are being evolved to find bugs in the evolving programs. Heywood also describes co-operative coevolution. Cooperative evolution covers many approaches, such as: evolving separate

programs so that they work as an ensemble [41, 42], as a team [43], part of a complete solution [44] or as a member of a multi-agent simulation [45, 46].[4] He also points out that compared to current large language models created by deep learning artificial neural networks, GP is not slow.

Alberto Bartoli, Luca Manzoni and Eric Medvet [50] caution us against accepting too rosy a picture of GP. They are right to point to the diffuse evidence of industrial GP take up and lack of a dominant GP package[5]. Whilst a few GP tools are now firmly in the industrial domain: Eureqa [51–53] and HeuristicLab [54], they are right to point to the diversity of available GP tools. It is unfair to pick out examples from the many available, nevertheless a few come to mind: DEAP [55] EASEA [30, 56] ECJ [57] GeneticEngine [58] GenProg [59] Gin [60, 61] GPLAB [62] gplearn, GPTIPS [63] Inspyred [64] Magpie [65] Pony GE2 [66] PushGP [67] and TensorGP [68]. I have already pointed to a few examples of GP take up in the software industry, more can be found in the water industry [69] and civil engineering [70], indeed they celebrate the success of TPOT [71, 72] in Bioinformatics. Generally Bioinformatics and medical research has embraced open science and is more than happy to cite GP tools, such as TPOT, or tools enhanced by evolution [11] when they use them [73]. However, as David Andre at his invited keynote at GPTP-2021 [74] pointed out, generally companies in competitive industries (particularly in finance) are very wary about talking openly about any tool or technique that gives them an edge. Bartoli et al. point to GP's continued success in symbolic regression, citing Bill La Cava and team's work, which was published at the top neural networks conference [75] and their tool, SRBench, which is available on GitHub. Bartoli et al. make important points and suggest ways the GP community should do better. They point to the recent success of deep neural networks, but in some ways perhaps we should take heart from this. In the popular press deep networks are "AI" and yet they, like GP, are empirical rather than theory driven, and both are firmly based on learning. The idea that AI requires someone to patiently code all human knowledge into a rule base is nowhere to be seen. Deep neural networks demand huge compute resources, that is, they are not efficient, and they are far from error free. Perhaps we should be happy to let our GP systems consume resources and tolerate some errors. As Stephanie Forrest said [76] what could we do if we allowed our evolutionary system the same resources that the mega corporations have spent.

Jason Moore [77] points to the impact of Koza's first GP book [1] in artificial intelligence (AI), artificial life, machine learning, art, biology, economics and engineering but questions if Darwin's fitness driven evolution is helpful. (You may remember the full title is "Genetic Programming: On the programming of computers by means of natural selection", which was inspired by that of Charles Darwin's 1859 revolutionary book "On the Origin of Species by Means of Natural Selection"

[4] Since it is known that disruptions caused by mutation or crossover often fail to propagate up deeply nesting programs to impact fitness [47–49], another argument for splitting up programs is to ensure each member of the team or ensemble is exposed to the fitness testing environment, whilst each remains relatively shallow.

[5] A miscellaneous collection of GP tools can be found at http://www.cs.ucl.ac.uk/staff/W.Langdon/homepages.html#6.

[78].) Instead Moore focuses on the importance of the representation used to define the programs, the variation operators used to modify them and suggests perhaps GP needs a name change. Certainly the last 30+ years have seen an expansion in GP representations from interpretable Lisp trees (including ADFs [38]) to linear GP [79], grammatical evolution [80], and graph based GP, such as cartesian genetic programming [81]. Also genetic improvement [82, 83] has reinforced the idea that existing computer programs are not fragile [84] and can also be evolved, with genetic representations as diverse as: lines of C++ code [85], Java [86], XML based abstract syntax trees [65],[6] SQL [88, 89], Java byte code [90–92], assembler [93], Clang intermediate code [94] and even binary machine code [95]. Moore is correct in saying that not only are trees not the only representation but also genetic search is not the only game in town. Already people have (in addition to genetic algorithms [1]) been successful with: hill climbing [96], simulated annealing [97], novelty search [98] and Monte-Carlo Tree Search [99]. As he points out we need to be cautious about re-naming, for example, often people do not like "random" but the equivalent "stochastic" sounds more scientific. But in the end he points out whatever we call GP our goal must be to continue to help people and help society.

Colin Johnson [100] points to the "unreasonable effectiveness" of GP fitness functions but nevertheless suggests several ways to improve them including information theory [101], and suggests the possibility of a universal fitness function, perhaps derived from existing examples, e.g. using his Learned Guidance Function LGF [102]. Similarly we can regard GP as providing a universal representation [103]. He also mentions current work on using deep neural network based large language models (LLMs) in natural language processing (NLP) text generation applications. Perhaps LGFs with LLMs could act as surrogate fitness functions [104] and may be give multitudes of fitness test points? He also suggests we abandon static fitness functions, and instead use dynamic fitness functions, whose role and target change during the run as the GP population evolves. (In some ways dynamic fitness functions might emulate the often hoped for role of co-evolution of continuously stretching the populations, by adapting the direction of fitness selection. Thus preventing any species in the ecosystem from stagnating near a local optimum. See also *Heywood* [22], page 3 above.) Johnson highlights work by Krzysztof Krawiec [105], which perhaps has already taken a step in the direction advocated by *Jason Moore* [77] (see previous paragraph), where instead of fitness being applied blindly, "black box", to the whole organism (i.e. the whole program) "search drivers" consider components within the program [106] and try to improve the whole by improving its parts. This is very much in the mode of recent "white box, blind no more" work by Darrell Whitely [107], where Whitely uses variable interaction graphs (VIGs) to find the natural components of combinatorial problems and uses them with crossover to effectively search vast spaces. See also Zaidi's Value State Flow Graphs (VSFGs) for describing data flows inside programs [108]. Perhaps VIGs or VSFGs

[6] Abstract syntax trees are often used by high level language compilers during syntax analysis. ASTs written in XML typically contain many diverse types and strongly typed genetic programming, STGP [87], crossover and mutation are readily implemented with XML [65].

could be used in GP? Perhaps with a degree of fuzziness to eliminate potential weak connections between program components? Perhaps approximate VIGs could form part of the inherited genotype? Perhaps they could themselves be subject to mutation or other genetic operations, with new programs (epigenomes) being stochastically generated from VIGs?

I would like again to thank the contributors to this peer commentary and especially the two editorial "Lions" for assembling such a diverse set of skilled peer commentators with such good ideas about how to pull GP forward for the next 30 years.

References

1. J.R. Koza, Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA (1992), http://mitpress.mit.edu/books/genetic-programming
2. W.B. Langdon, Jaws 30. Genetic programming and evolvable machines peer commentary on the thirtieth anniversary of genetic programming: on the programming of computers by means of natural selection
3. G. Squillero, A. Tonda, Veni, vidi, evolvi. Genetic Programming and Evolvable Machines Peer Commentary on the Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection
4. W.B. Langdon et al., Comparison of AdaBoost and genetic programming for combining neural networks for drug discovery. In: Raidl, G.R., et al. (eds.) Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM. LNCS, vol. 2611, pp. 87–98. Springer-Verlag, University of Essex, UK (14-16 Apr 2003), https://doi.org/10.1007/3-540-36605-9_9
5. F. Assuncao et al., DENSER: deep evolutionary network structured representation. Genet. Program. Evol. Mach. **20**(1), 5–35 (2019). https://doi.org/10.1007/s10710-018-9339-y
6. M. Harman, B.F. Jones, Search based software engineering. Inf. Softw. Technol. **43**(14), 833–839 (2001). https://doi.org/10.1016/S0950-5849(01)00189-6
7. S. Forrest et al., A genetic programming approach to automated software repair. In: Raidl, G., et al. (eds.) GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 947–954. ACM, Montreal (8-12 Jul 2009), https://doi.org/10.1145/1569901.1570031, gECCO 2019 10-Year Most Influential Paper Award, Best paper
8. C. Le Goues et al., Automated program repair. Commun. ACM **62**(12), 56–65 (2019). https://doi.org/10.1145/3318162
9. W.B. Langdon, Genetic improvement of programs. In: Matousek, R. (ed.) 18th International Conference on Soft Computing, MENDEL 2012. Brno University of Technology, Brno, Czech Republic (27-29 Jun 2012), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Langdon_2012_mendel.pdf, invited keynote
10. W.B. Langdon et al., Genetic improvement of GPU software. Genet. Program. Evol. Mach. **18**(1), 5–44 (2017). https://doi.org/10.1007/s10710-016-9273-9
11. W.B. Langdon, R. Lorenz, Improving SSE parallel code with grow and graft genetic programming. In: Petke, J., et al. (eds.) GI-2017. pp. 1537–1538. ACM, Berlin (15-19 Jul 2017), https://doi.org/10.1145/3067695.3082524
12. S.O. Haraldsson et al., Fixing bugs in your sleep: How genetic improvement became an overnight success, in Petke, J., et al. (eds.) GI-2017. pp. 1513–1520. ACM, Berlin (15-19 Jul 2017), https://doi.org/10.1145/3067695.3082517, best paper
13. N. Alshahwan, Industrial experience of genetic improvement in Facebook, in Petke, J., et al. (eds.) GI-2019, ICSE workshops proceedings. p. 1. IEEE, Montreal (28 May 2019), https://doi.org/10.1109/GI.2019.00010, invited Keynote

14. 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
15. G. Squillero, Artificial evolution in computer aided design: from the optimization of parameters to the creation of assembly programs. Computing **93**(2–4), 103–120 (2011). https://doi.org/10.1007/s00607-011-0157-9
16. M. Castelli, Commentary for the GPEM peer commentary special section on W. B. Langdon's "Jaws 30". Genetic Programming and Evolvable Machines Peer Commentary on the Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection
17. W.B. Langdon, R. Poli, Foundations of genetic programming. Springer-Verlag (2002). https://doi.org/10.1007/978-3-662-04726-2
18. M. Hort et al., Multi-objective search for gender-fair and semantically correct word embeddings. Appl. Soft Comput. **133**, 109916 (2023). https://doi.org/10.1016/j.asoc.2022.109916
19. E. Daka et al., Modeling readability to improve unit tests. In: Nitto, E.D., et al. (eds.) Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015. pp. 107–118. ACM (2015), https://doi.org/10.1145/2786805.2786838
20. A. Panichella et al., Revisiting test smells in automatically generated tests: Limitations, pitfalls, and opportunities. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 523–533. Adelaide (2020), https://doi.org/10.1109/ICSME46990.2020.00056
21. W.B. Langdon, J.P. Nordin, Seeding GP populations, in Poli, R., et al. (eds.) Genetic Programming, Proceedings of EuroGP'2000. LNCS, vol. 1802, pp. 304–315. Springer-Verlag, Edinburgh (15-16 Apr 2000), https://doi.org/10.1007/978-3-540-46239-2_23
22. M.I. Heywood, W. B. Langdon "JAWS 30". Genetic Programming and Evolvable Machines Peer Commentary on the Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection
23. G.E. Moore, Cramming more components onto integrated circuits. Electronics **38**(8), 114–117 (1965)
24. W.B. Langdon et al., Genetically improved software with fewer data caches misses. In: Proceedings of the 2023 Genetic and Evolutionary Computation Conference. GECCO '23, Association for Computing Machinery, Lisbon, Portugal (15-19 Jul 2023), https://doi.org/10.1145/3583133.3590542, forthcoming
25. D. Andre, J.R. Koza, Parallel genetic programming on a network of transputers. In: Rosca, J.P. (ed.) Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications. pp. 111–120. Tahoe City, California, USA (9 Jul 1995), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/andre_1995_parallel.pdf
26. D. Andre, J.R. Koza, A parallel implementation of genetic programming that achieves super-linear performance. Inf. Sci. **106**(3–4), 201–218 (1998). https://doi.org/10.1016/S0020-0255(97)10011-1
27. A. Fukunaga, et al., A genome compiler for high performance genetic programming. In: Koza, J.R., et al. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference. pp. 86–94. Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA (22-25 Jul 1998), http://metahack.org/gp98-compiler.pdf
28. H. Juille, J.B. Pollack, Massively parallel genetic programming. In: Angeline, P.J., Kinnear, Jr., K.E. (eds.) Advances in Genetic Programming 2, chap. 17, pp. 339–357. MIT Press, Cambridge, MA, USA (1996), https://doi.org/10.7551/mitpress/1109.003.0023
29. W.B. Langdon, W. Banzhaf, A SIMD interpreter for genetic programming on GPU graphics cards. In: O'Neill, M., et al. (eds.) Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008. Lecture Notes in Computer Science, vol. 4971, pp. 73–85. Springer, Naples (26-28 Mar 2008), https://doi.org/10.1007/978-3-540-78671-9_7
30. O. Maitre, Genetic programming on GPGPU cards using EASEA. In: Tsutsui, S., Collet, P. (eds.) Massively Parallel Evolutionary Computation on GPGPUs, chap. 11, pp. 227–248. Natural Computing Series, Springer (2013), https://doi.org/10.1007/978-3-642-37959-8_11
31. C. Ortega-Sanchez et al., Embryonics: a bio-inspired cellular architecture with fault-tolerant properties. Genet. Program. Evol. Mach. **1**(3), 187–215 (2000). https://doi.org/10.1023/A:1010080629099

32. C. Pedraza et al., Genetic algorithm for Boolean minimization in an FPGA cluster. The Journal of Supercomputing 58(2), 244–252 (2011), https://doi.org/10.1007/s11227-010-0401-7, special issue on HPC in computational Science and Engineering. Part I
33. L. Spector, Automatic Quantum Computer Programming: A Genetic Programming Approach, Genetic Programming, vol. 7. Kluwer Academic Publishers, Boston/Dordrecht/New York/London (Jun 2004), https://doi.org/10.1007/978-0-387-36791-0
34. G. O'Brien, J. Clark, Using genetic improvement to retarget quantum software on differing hardware. In: Petke, J., et al. (eds.) GI @ ICSE 2021. pp. 31–38. IEEE, internet (30 May 2021), https://doi.org/10.1109/GI52543.2021.00015, winner Best Presentation
35. W.B. Langdon, Large scale bioinformatics data mining with parallel genetic programming on graphics processing units. In: Fernandez de Vega, F., Cantu-Paz, E. (eds.) Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence, vol. 269, chap. 5, pp. 113–141. Springer (Jan 2010), https://doi.org/10.1007/978-3-642-10675-0_6
36. M. Ridley, The Red Queen, Sex and the Evolution of Human Nature. Penquin (1993), https://en.wikipedia.org/wiki/The_Red_Queen:_Sex_and_the_Evolution_of_Human_Nature
37. W.B. Langdon, Genetic programming and data structures: genetic programming + data structures = automatic programming! Genet. Program. (1998). https://doi.org/10.1007/978-1-4615-5731-9
38. J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge Massachusetts (May 1994), http://www.genetic-programming.org/gpbook2toc.html
39. M. Ebner, Coevolution and the red queen effect shape virtual plants. Genet. Program. Evol. Mach. **7**(1), 103–123 (2006). https://doi.org/10.1007/s10710-006-7013-2
40. A. Arcuri, X. Yao, Coevolving programs and unit tests from their specification, in IEEE International Conference on Automated Software Engineering (ASE). Atlanta, Georgia, USA (Nov 5-9 2007), https://doi.org/10.1145/1321631.1321693
41. W.B. Langdon, B.F. Buxton, Genetic programming for combining classifiers, in Spector, L., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). pp. 66–73. Morgan Kaufmann, San Francisco, California, USA (7-11 Jul 2001), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL_gecco2001_roc.pdf
42. M. Virgolin, Genetic programming is naturally suited to evolve bagging ensembles. In: Chicano, F., et al. (eds.) Proceedings of the 2021 Genetic and Evolutionary Computation Conference. pp. 830–839. GECCO '21, Association for Computing Machinery, internet (Jul 10-14 2021), https://doi.org/10.1145/3449639.3459278
43. M. Brameier, W. Banzhaf, Evolving teams of predictors with linear genetic programming. Genet. Program. Evol. Mach. **2**(4), 381–407 (2001). https://doi.org/10.1023/A:1012978805372
44. J. Louchet, Using an individual evolution strategy for stereovision. Genet. Program. Evol. Mach. **2**(2), 101–109 (2001). https://doi.org/10.1023/A:1011544128842
45. F.H. Bennett III, Emergence of a multi-agent architecture and new tactics for the ant colony foraging problem using genetic programming, in: Maes, P., et al. (eds.) Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4. pp. 430–439. MIT Press, Cape Code, USA (9-13 Sep 1996), https://doi.org/10.7551/mitpress/3118.003.0044
46. M. Georgiev et al., Performance analysis and comparison on heterogeneous and homogeneous multi-agent societies in correlation to their average capabilities, in 2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE). pp. 674–679. Nara, Japan (11-14 Sep 2018), https://doi.org/10.23919/SICE.2018.8492713
47. J. Petke et al., Software robustness: A survey, a theory, and some prospects, in Avgeriou, P., Zhang, D. (eds.) ESEC/FSE 2021, Ideas, Visions and Reflections. pp. 1475–1478. ACM, Athens, Greece (23-28 Aug 2021), https://doi.org/10.1145/3468264.3473133
48. W.B. Langdon, Genetic programming convergence. Genet. Program. Evol. Mach. **23**(1), 71–104 (2022). https://doi.org/10.1007/s10710-021-09405-9
49. W.B. Langdon, A trillion genetic programming instructions per second. ArXiv (6 May 2022), https://arxiv.org/abs/2205.03251
50. A. Bartoli et al., Commentary on "Jaws 30", by W. B. Langdon. Genetic Programming and Evolvable Machines Peer Commentary on the Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection
51. M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data. Science **324**(5923), 81–85 (2009). https://doi.org/10.1126/science.1165893
52. N. Savage, Automating scientific discovery. Commun. ACM **55**(5), 9–11 (2012). https://doi.org/10.1145/2160718.2160723

53. R. Dubcakova, Eureqa: software review. Genet. Program. Evol. Mach. **12**(2), 173–178 (2011). https://doi.org/10.1007/s10710-010-9124-z
54. A. Elyasaf, M. Sipper, Software review: the heuristiclab framework. Genet. Program. Evol. Mach. **15**(2), 215–218 (2014). https://doi.org/10.1007/s10710-014-9214-4
55. J. Kim, S. Yoo, Software review: DEAP (distributed evolutionary algorithm in python) library. Genet. Program. Evol. Mach. **20**(1), 139–142 (2019). https://doi.org/10.1007/s10710-018-9341-4
56. U. Abdulkarimova et al., The PARSEC machine: a non-Newtonian supra-linear super-computer. Azerbaijan J. High Perf. Comput. **2**(2), 122–140 (2019). https://doi.org/10.32010/26166127.2019.2.2.122.140
57. D.R. White, Software review: the ECJ toolkit. Genet. Program. Evol. Mach. **13**(1), 65–67 (2012). https://doi.org/10.1007/s10710-011-9148-z
58. G. Espada et al., Data types as a more ergonomic frontend for grammar-guided genetic programming, in Scholz, B., Kameyama, Y. (eds.) 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 2022). pp. 86–94. ACM, Auckland, New Zealand (Dec 6-7 2022), https://doi.org/10.1145/3564719.3568697
59. C. Le Goues et al., GenProg: a generic method for automatic software repair. IEEE Trans. Softw. Eng. **38**(1), 54–72 (2012). https://doi.org/10.1109/TSE.2011.104
60. D.R. White, GI in no time, in Petke, J., et al. (eds.) GI-2017. pp. 1549–1550. ACM, Berlin (15-19 Jul 2017), https://doi.org/10.1145/3067695.3082515
61. M. Watkinson, A. Brownlee, Updating Gin's profiler for current java, in Nowack, V., et al. (eds.) 12th International Workshop on Genetic Improvement @ICSE 2023. pp. 23–28. IEEE, Melbourne, Australia (20 May 2023), https://doi.org/10.1109/GI59320.2023.00015
62. I. Atmosukarto, GPLAB: software review. Genet. Program. Evol. Mach. **12**(4), 457–459 (2012). https://doi.org/10.1007/s10710-011-9142-5
63. A.H. Gandomi, E. Atefi, Software review: the GPTIPS platform. Genet. Program. Evol. Mach. **21**(1–2), 273–280 (2020). https://doi.org/10.1007/s10710-019-09366-0
64. A. Tonda, Inspyred: bio-inspired algorithms in python. Genet. Program. Evol. Mach. **21**(1–2), 269–272 (2020). https://doi.org/10.1007/s10710-019-09367-z
65. A. Blot, J. Petke, MAGPIE: Machine automated general performance improvement via evolution of software. arXiv (4 Aug 2022), https://doi.org/10.48550/arxiv.2208.02811
66. T.M. Vu, Software review: Pony GE2. Genet. Program. Evol. Mach. **22**(3), 383–385 (2021). https://doi.org/10.1007/s10710-021-09409-5
67. L. Spector, A. Robinson, Genetic programming and autoconstructive evolution with the push programming language. Genet. Program. Evol. Mach. **3**(1), 7–40 (2002). https://doi.org/10.1023/A:1014538503543
68. F. Baeta et al., TensorGP - genetic programming engine in TensorFlow, in Castillo, P., Jimenez-Laredo, J. (eds.) 24th International Conference, EvoApplications 2021. LNCS, vol. 12694, pp. 763–778. Springer Verlag, virtual event (7-9 Apr 2021), https://doi.org/10.1007/978-3-030-72699-7_48
69. A. Danandeh Mehr et al., Genetic programming in water resources engineering: a state-of-the-art review. J. Hydrol. **566**, 643–667 (2018). https://doi.org/10.1016/j.jhydrol.2018.09.043
70. Q. Zhang et al., Genetic programming in civil engineering: advent, applications and future trends. Artif. Intell. Rev. **54**, 1863–1885 (2021). https://doi.org/10.1007/s10462-020-09894-7
71. R.S. Olson et al., Automating biomedical data science through tree-based pipeline optimization. In: Squillero, G., Burelli, P. (eds.) Proceedings of the 19th European Conference on Applications of Evolutionary Computation, EvoApplications 2016, Part I. LNCS, vol. 9597, pp. 123–137. Springer, Porto, Portugal (Mar 30 - Apr 1 2016), https://doi.org/10.1007/978-3-319-31204-0_9, best paper, EvoBio track
72. R.S. Olson et al., A system for accessible artificial intelligence. In: Banzhaf, W., et al. (eds.) Genetic Programming Theory and Practice XV. pp. 121–134. Genetic and Evolutionary Computation, Springer, University of Michigan in Ann Arbor, USA (May 18–20 2017), https://doi.org/10.1007/978-3-319-90512-9_8
73. R.J. Andrews et al., A map of the SARS-CoV-2 RNA structurome. NAR Genom. Bioinf. **3**(2), lqab043 (2021). https://doi.org/10.1093/nargab/lqab043
74. W. Banzhaf et al., (eds.): Genetic Programming Theory and Practice XVIII. Genetic and Evolutionary Computation, Springer, East Lansing, USA (19-21 May 2021), https://doi.org/10.1007/978-981-16-8113-4

75. W. La Cava et al., Contemporary symbolic regression methods and their relative performance. In: Vanschoren, J., Yeung, S. (eds.) Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks. vol. 1. Curran (2021), https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c0c7c76d30bd3dcaefc96f40275bdc0a-Abstract-round1.html
76. J. Petke et al., (eds.): 10th Genetic Improvement Workshop (GI 2021 @ ICSE) Chairs' Welcome. IEEE, virtual event (30 May 2021), http://www.cs.ucl.ac.uk/staff/W.Langdon/icse2021/gi2021_message_from_the_chairs.pdf
77. J.H. Moore, Is the evolution metaphor still necessary or even useful for genetic programming? Genetic Programming and Evolvable Machines Peer Commentary on the Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection
78. C. Darwin, On the Origin of Species by Means of Natural Selection. John Murray, penguin classics, 1985 edn. (1859)
79. W. Banzhaf et al., Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann, San Francisco, CA, USA (Jan 1998), https://www.amazon.co.uk/Genetic-Programming-Introduction-Artificial-Intelligence/dp/1558600510X
80. M. O'Neill, C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, Genetic programming, vol. 4. Kluwer Academic Publishers (2003), https://doi.org/10.1007/978-1-4615-0447-4
81. J.F. Miller, (ed.): Cartesian Genetic Programming. Natural Computing Series, Springer (2011), https://doi.org/10.1007/978-3-642-17310-3
82. J. Petke et al., Genetic improvement of software: a comprehensive survey. IEEE Trans. Evolut. Comput. **22**(3), 415–432 (2018). https://doi.org/10.1109/TEVC.2017.2693219
83. J. Petke et al., A survey of genetic improvement search spaces. In: Alexander, B., et al. (eds.) 7th edition of GI @ GECCO 2019. pp. 1715–1721. ACM, Prague, Czech Republic (Jul 13-17 2019), https://doi.org/10.1145/3319619.3326870
84. W.B. Langdon, J. Petke, Software is not fragile. In: Parrend, P., et al. (eds.) Complex Systems Digital Campus E-conference, CS-DC'15. pp. 203–211. Proceedings in Complexity, Springer (Sep 30-Oct 1 2015), https://doi.org/10.1007/978-3-319-45901-1_24, invited talk
85. W.B. Langdon, M. Harman, Optimising existing software with genetic programming. IEEE Trans. Evolut. Comput. **19**(1), 118–135 (2015). https://doi.org/10.1109/TEVC.2013.2281544
86. J. Petke, A. Brownlee, Software improvement with Gin: a case study. In: Nejati, S., Gay, G. (eds.) SSBSE 2019. LNCS, vol. 11664, pp. 183–189. Springer, Tallinn, Estonia (31 Aug - 1 Sep 2019), https://doi.org/10.1007/978-3-030-27455-9_14
87. D.J. Montana, Strongly typed genetic programming. Evolut. Comput. **3**(2), 199–230 (1995). https://doi.org/10.1162/evco.1995.3.2.199
88. C.Y. Ishida, A.T.R. Pozo, GPSQL miner: SQL-grammar genetic programming in data mining. In: Fogel, D.B., et al. (eds.) Proceedings of the 2002 Congress on Evolutionary Computation CEC2002. pp. 1226–1231. IEEE Press (12-17 May 2002), https://doi.org/10.1109/CEC.2002.1004418
89. J. Callan, J. Petke, Optimising SQL queries using genetic improvement. In: Petke, J., et al. (eds.) GI @ ICSE 2021. pp. 9–10. IEEE, internet (30 May 2021), https://doi.org/10.1109/GI52543.2021.00010
90. E. Lukschandl et al., Automatic evolution of Java bytecode: First experience with the Java virtual machine. In: Poli, R., et al. (eds.) Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming. pp. 14–16. CSRP-98-10, The University of Birmingham, UK, Paris, France (14-15 Apr 1998), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/csrp-98-10.pdf
91. M. Orlov, M. Sipper, FINCH: A system for evolving Java (bytecode). In: Riolo, R., et al. (eds.) Genetic Programming Theory and Practice VIII, Genetic and Evolutionary Computation, vol. 8, chap. 1, pp. 1–16. Springer, Ann Arbor, USA (20-22 May 2010), https://doi.org/10.1007/978-1-4419-7747-2_1
92. K. Yeboah-Antwi, B. Baudry, Embedding adaptivity in software systems using the ECSELR framework. In: Langdon, W.B., et al. (eds.) Genetic Improvement 2015 Workshop. pp. 839–844. ACM, Madrid (11-15 Jul 2015), https://doi.org/10.1145/2739482.2768425
93. E. Schulte, et al., Automated program repair through the evolution of assembly code. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. pp. 313–316. ACM, Antwerp (20-24 Sep 2010), https://doi.org/10.1145/1858996.1859059

94. W.B. Langdon et al., Genetic improvement of LLVM intermediate representation. In: Pappa, G., et al. (eds.) EuroGP 2023: Proceedings of the 26th European Conference on Genetic Programming. LNCS, vol. 13986, pp. 244–259. Springer Verlag, Brno, Czech Republic (12-14 Apr 2023), https://doi.org/10.1007/978-3-031-29573-7_16
95. E. Schulte et al., Automated repair of binary and assembly programs for cooperating embedded devices. In: Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems. pp. 317–328. ASPLOS 2013, ACM, Houston, Texas, USA (Mar 16-20 2013), https://doi.org/10.1145/2451116.2451151
96. H. Iba et al., Genetic programming with local hill-climbing. In: Davidor, Y., et al. (eds.) Parallel Problem Solving from Nature III. LNCS, vol. 866, pp. 334–343. Springer-Verlag, Jerusalem (9-14 Oct 1994), https://doi.org/10.1007/3-540-58484-6_274
97. A.I. Esparcia-Alcazar, Genetic Programming for Adaptive Signal Processing. Ph.D. thesis, Electronics and Electrical Engineering, University of Glasgow, UK (Jul 1998), http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/esparcia-alcazar/thesis.ps.gz
98. J. Lehman, K.O. Stanley, Novelty search and the problem with objectives. In: Riolo, R., et al. (eds.) Genetic Programming Theory and Practice IX, chap. 3, pp. 37–56. Genetic and Evolutionary Computation, Springer, Ann Arbor, USA (12-14 May 2011), https://doi.org/10.1007/978-1-4614-1770-5_3
99. H. Rakotoarison, et al., Automated machine learning with Monte-Carlo Tree Search. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019. pp. 3296–3303. ijcai.org, Macao, China (Aug 10-16 2019), https://doi.org/10.24963/ijcai.2019/457
100. C.G. Johnson, New directions in fitness evaluation: Commentary on Langdon's JAWS30. Genetic Programming and Evolvable Machines Peer Commentary on the Thirtieth Anniversary of Genetic Programming: On the Programming of Computers by Means of Natural Selection
101. S.W. Card, Towards an Information Theoretic Framework for Evolutionary Learning. Ph.D. thesis, Electrical Engineering and Computer Science, Syracuse University, USA (Aug 2011), https://surface.syr.edu/eecs_etd/307
102. C.G. Johnson, Solving the Rubik's cube with stepwise deep learning. Expert Syst. J. Knowl. Eng. (2021). https://doi.org/10.1111/exsy.12665
103. X. Yao, Universal approximation by genetic programming. In: Haynes, T., et al. (eds.) Foundations of Genetic Programming. pp. 66–67. Orlando, Florida, USA (13 Jul 1999), http://www.cs.ucl.ac.uk/staff/W.Langdon/fogp/yao.ps.gz
104. V. Parque, T. Miyashita, On vehicle surrogate learning with genetic programming ensembles. In: Cotta, C., et al. (eds.) GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1704–1710. ACM, Kyoto, Japan (2018), https://doi.org/10.1145/3205651.3208310
105. K. Krawiec, Behavioral Program Synthesis with Genetic Programming, Studies in Computational Intelligence, vol. 618. Springer International Publishing (2015), https://doi.org/10.1007/978-3-319-27565-9
106. 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
107. F. Chicano et al., Dynastic potential crossover operator. Evolutionary Computation 30(3) (Fall 2022), https://doi.org/10.1162/evco_a_00305
108. A.M. Zaidi, Accelerating control-flow intensive code in spatial hardware. Ph.D. thesis, Computer Laboratory, University of Cambridge (May 2015), https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-870.pdf, also available as Technical Report UCAM-CL-TR-870