Genetic Programming Convergence [Hot of the Press]

William B. Langdon W.Langdon@cs.ucl.ac.uk
Department of Computer Science, University College London, London, UK

ABSTRACT

We study both genotypic and phenotypic convergence in GP floating point continuous domain symbolic regression over thousands of generations. Subtree fitness variation across the population is measured and shown in many cases to fall. In an expanding region about the root node, both genetic opcodes and function evaluation values are identical or nearly identical. Bottom up (leaf to root) analysis shows both syntactic and semantic (including entropy) similarity expand from the outermost node. Despite large regions of zero variation, fitness continues to evolve and near zero crossover disruption suggests improved GP systems. W. B. Langdon. 2022. Genetic Programming Convergence. *GP & EM* 23,1, 71–104.

KEYWORDS

genetic programming, evolutionary computation, stochastic search, diversity, bottom up incremental evaluation, PIE, propagation, Failed Disruption Propagation, FDP, infection, and execution, SIMD parallel processing, AVX vector instructions

ACM Reference Format:
William B. Langdon W.Langdon@cs.ucl.ac.uk . 2022. Genetic Programming Convergence [Hot of the Press]. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3520304.3534063

MOTIVATION

Our goal is to push the limits of GP and report what we find there. At more than 2 billion nodes, we have evolved the largest ever GP trees. At up to a million generations, we have evolved GP populations far longer than ever before. To do this we have built the fastest ever GP system (equivalent to up to 1.1 trillion GP operation per second) [13]. In the process we have found thousands of fitness improvements [13], confirmed early theoretical predictions[1], mapped GP convergence in multiple dimensions (see next section) and demonstrated new information theoretical results: Failed Disruption Propagation, which prevents deep mutations impacting fitness and which has implications for the evolution of computer based complex systems [10] and software engineering [16].

To support this we have made a series of traditional and automatic software improvements [4–9, 12] such that the run shown in [11, Fig. 3] which was cut short after 5 weeks due to a neigbourhood wide external power failure, can now be repeated in 5 days. (Code is available via my home pages GPinc.tar.gz.)

SUMMARY

Much Genetic Programming work is aimed at applications where there is a need for a quick solution and so GP runs tend to be short, e.g. no more than fifty generations. But the goal of GP should also be to solve problems which cannot be solved by other methods. Recently Rich Lenski [15] has confounded the Biological establishment and overturned conventional wisdom by showing that natural evolution can continue to produce fitter organisms even after tens of thousands of generations (see Section 1.5 in [11]). Perhaps a way to open up GP to more adventurous applications, will require that the GP population evolves for far longer?

We have made a start with long term evolution experiments in GP. These have shown, even in fixed environments, GP can continue to find improvements. The GP&EM paper shows what is going on in these highly evolved populations. E.g., the GP fitness landscape is far smoother than is commonly assumed, with crossover becoming less phenotypically disruptive as tree grow larger and Section 9.6 suggesting increasing the rigour of the fitness function will only slowly increase crossover's effect.

Section 3 investigates genetic convergence. Section 4 shows that phenotypic convergence lags behind genetic convergence of the trees. Section 5 shows although operations like multiplication or division by zero can render large parts of trees ineffective, in the continuous domain such obviously ineffective code can be a small (≈0.5%) or a large (91%) part of highly evolved programs. Thus explaining why automatic intron removal may not always improve GP execution time. Section 6 shows in converged populations many subtrees have identical phenotypes. In Section 7 we study information flow within evolved trees and we find on average entropy rises monotonically from the inputs (the leafs) towards the the output (the tree's root node) and quickly reaches a maximum, $\log_2$ |test suite size|. This means large parts of evolved programs have identical entropy. In contrast, Section 8 shows typically the phenotypic disruption of crossover has a limited effect, which tends to be damped in the region above the crossover point towards the root node. This opens the way to the implementation of new efficient GP interpreters for large evolving programs.

Acknowledgements. I would like to thank the anonymous reviewers, Simon Tatham for PuTTY, and Dagstuhl Seminars 17191 on the theory of randomized heuristics and 18052 on Genetic Improvement of Software [17], for inspiring conversations.

Funded by EPSRC GGGP and InfoTestSS grants EP/M025853/1, EP/P005888/1.

[1] After many thousands of generations, GP populations are typically composed of trees of random shape, neither balanced and full nor deep and straggly but somewhat fractal, self similar, between the two extremes [2, 14]. We also see again sub-quadratic bloat [2] (e.g. [11, Fig. 6]) with trees growth following a power law near gens$^{2.0}$. Although in Boolean problems there can be cases of extreme fitness convergence where we see bloat falling off after thousands of generations [3].

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3534063

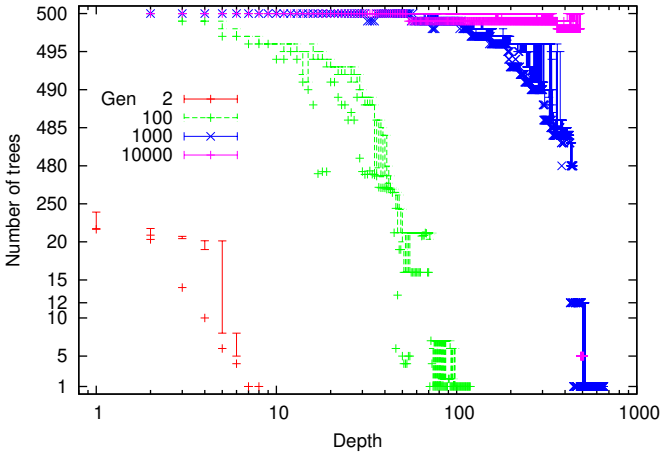

Figure 1: Genetic convergence of contents and shape of 500 GP trees in first run at generations 2, 100, 1000 and 10 000. At generation 1000 (blue) the whole population is identical around their root nodes to a depth of ≤57. Indeed most trees agree to more than 444 nested function calls deep. Note non-uniform scales. Also see https://youtu.be/TssAIo-vatE for an online video in which the population is plotted as a circular lattice [1].

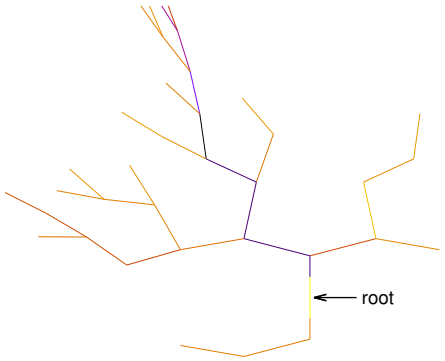

Figure 2: The consensus subfitness near the root changes little after generation 100 (see Figure 15). Yellow best, blue worse fitness. Movie https://youtu.be/_qz1_1AK1gw) shows the evolution of phenotypic convergence.

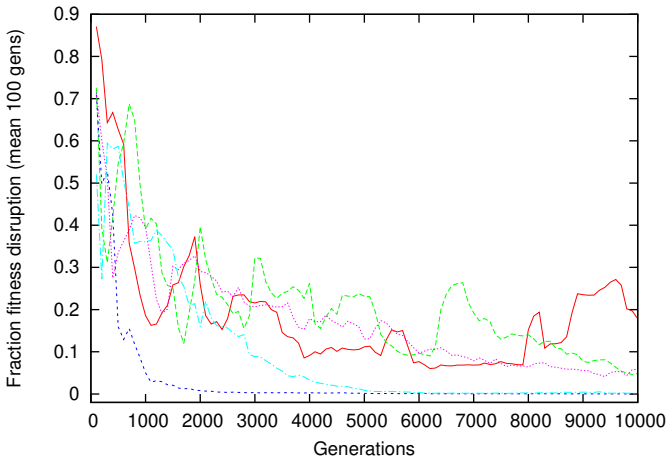

Figure 3: Evolution of fraction of children with fitness different from mum. Avoiding re-evaluation gives huge savings in evaluation of GP opcodes.

REFERENCES

[1] Jason M. Daida, Adam M. Hilss, David J. Ward, and Stephen L. Long. 2005. Visualizing Tree Structures in Genetic Programming. *Genetic Programming and Evolvable Machines* 6, 1 (March 2005), 79–110. http://dx.doi.org/10.1007/s10710-005-7621-2

[2] W. B. Langdon. 2000. Quadratic Bloat in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer (Eds.). Morgan Kaufmann, Las Vegas, Nevada, USA, 451–458. http://gpbib.cs.ucl.ac.uk/gecco2000/GA069.pdf

[3] William B. Langdon. 2017. Long-Term Evolution of Genetic Programming Populations. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, Berlin, 235–236. http://dx.doi.org/10.1145/3067695.3075965

[4] W. B. Langdon. 2019. Parallel GPQUICK. In *GECCO '19 Companion*, Carola Doerr (Ed.). ACM, Prague, Czech Republic, 63–64. http://dx.doi.org/10.1145/3319619.3326770

[5] William B. Langdon. 2020. *Fast Generation of Big Random Binary Trees.* Technical Report RN/20/01. Computer Science, University College, London, Gower Street, London, UK. https://arxiv.org/abs/2001.04505

[6] W. B. Langdon. 2020. Genetic Improvement of Genetic Programming. In *GI @ CEC 2020 Special Session*, Alexander (Sandy) Brownlee, Saemundur O. Haraldsson, Justyna Petke, and John R. Woodward (Eds.). IEEE Computational Intelligence Society, IEEE Press, internet, paper id24061. http://dx.doi.org/10.1109/CEC48606.2020.9185771

[7] W. B. Langdon. 2020. Multi-threaded Memory Efficient Crossover in C++ for Generational Genetic Programming. ArXiv. arXiv:2009.10460 [cs.NE] http://arxiv.org/abs/2009.10460

[8] William B. Langdon. 2021. Fitness First. In *Genetic Programming Theory and Practice XVIII (Genetic and Evolutionary Computation)*, Wolfgang Banzhaf, Leonardo Trujillo, Stephan Winkler, and Bill Worzel (Eds.). Springer, East Lansing, MI, USA, 143–164. http://dx.doi.org/10.1007/978-981-16-8113-4_8

[9] William B. Langdon. 2021. Incremental Evaluation in Genetic Programming. In *EuroGP 2021: Proceedings of the 24th European Conference on Genetic Programming (LNCS, Vol. 12691)*, Ting Hu, Nuno Lourenco, and Eric Medvet (Eds.). Springer Verlag, Virtual Event, 229–246. http://dx.doi.org/10.1007/978-3-030-72812-0_15

[10] W. B. Langdon. 2022. Evolving Open Complexity. *SIGEVOlution newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation* 15, 1 (March 2022). http://arxiv.org/abs/2112.00812

[11] W. B. Langdon. 2022. Genetic Programming Convergence. *Genetic Programming and Evolvable Machines* 23, 1 (March 2022), 71–104. http://dx.doi.org/10.1007/s10710-021-09405-9

[12] W. B. Langdon and W. Banzhaf. 2019. *Faster Genetic Programming GPquick via multicore and Advanced Vector Extensions.* Technical Report RN/19/01. University College, London, London, UK. http://www.cs.ucl.ac.uk/fileadmin/user_upload/avx_rn1901.pdf

[13] William B. Langdon and Wolfgang Banzhaf. 2022. Long-Term Evolution Experiment with Genetic Programming. *Artificial Life* 28, 2 (2022). http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/Langdon_2022_ALJ.pdf Invited submission to Artificial Life Journal special issue of the ALIFE'19 conference.

[14] W. B. Langdon and Riccardo Poli. 2002. *Foundations of Genetic Programming.* Springer-Verlag. http://dx.doi.org/10.1007/978-3-662-04726-2

[15] Richard E. Lenski et al. 2015. Sustained fitness gains and variability in fitness trajectories in the long-term evolution experiment with Escherichia coli. *Proceedings of the Royal Society B* 282, 1821 (22 December 2015). http://dx.doi.org/10.1098/rspb.2015.2292

[16] Justyna Petke, David Clark, and William B. Langdon. 2021. Software Robustness: A Survey, a Theory, and Some Prospects. In *ESEC/FSE 2021, Ideas, Visions and Reflections*, Paris Avgeriou and Dongmei Zhang (Eds.). ACM, Athens, Greece, 1475–1478. http://dx.doi.org/10.1145/3468264.3473133

[17] Justyna Petke, Claire Le Goues, Stephanie Forrest, and William B. Langdon. 2018. Genetic Improvement of Software: Report from Dagstuhl Seminar 18052. *Dagstuhl Reports* 8, 1 (23 July 2018), 158–182. http://dx.doi.org/10.4230/DagRep.8.1.158