Evolving Open Complexity

W. B. Langdon

24 October 2021

Abstract

Information theoretic analysis of large evolved programs produced by running genetic programming for up to a million generations has shown even functions as smooth and well behaved as floating point addition and multiplication loose entropy and consequently are robust and fail to propagate disruption to their outputs. This means, while dependent upon fitness tests, many genetic changes deep within trees are silent. For evolution to proceed at reasonable rate it must be possible to measure the impact of most code changes, yet in large trees most crossover sites are distant from the root node. We suggest to evolve very large very complex programs, it will be necessary to adopt an open architecture where most mutation site are within 10–100 levels of the organism's environment.

1 Background

Recently we have been investigating the long term evolution of genetic programming. Firstly, using Poli's submachine code GP [24, 25], evolving large binary Boolean trees [7] and more recently exploiting SIMD Intel AVX and multi-core parallelism to evolve floating point GP [9, 14, 12, 5]. Running for up to a million generations without size limits has generated, at more than a billion nodes, the biggest programs yet evolved and forced the development [10, 11] of, at the equivalent of more than a trillion GP operations per second, the fastest GP system. It has also prompted information theoretic analysis of programs [16]. (Of course information theory has long been used with evolutionary computing, e.g. [2].)

One immediately applicable result has been the realisation that in deep GP trees most changes have no impact on fitness and once this has been proved, for a given child, its fitness evaluation can be cut short and fitness simply copied from the parent. This can lead to enormous speed ups [13].

We have also considered traditional imperative (human written) programs and shown these too are much more robust than is often assumed [15, 6, 8]. Indeed we suggest that information theory provides a unified view of the difficulty of testing software [17, 1, 22].

The question of why fitness is so often exactly inherited [20, 21, 23] despite brutal genetic change is answered by the realisation that without side effect the disruption caused by the mutation must be propagated up the tree through a long chain of irreversible operations to the root node. Each function in the chain can loose entropy. In many cases deeply nested functions progressively loose information about the perturbation as the disruption fails to propagate to the program's output. Thus the mutation becomes invisible to fitness testing and its utility cannot be measured. Without fitness selective pressure, evolution degenerates into an undirected random walk.

In bloated structures information loss leads, from an evolutionary point of view, to extremely high resilience, robustness and so stasis. From the engineering point of view this is problematic, as then almost all genetic changes have no impact and evolutionary progress slows to a dawdle.

Since all digital computing is irreversible [18] it inherently losses information and so without short cuts, must lead to failed disruption propagation (FDP) [22]. We suggest in order to evolve large complex systems it must be possible to measure the impact of genetic changes, therefore we must control FDP and suggest in the next section that to evolve large systems, they be composed of many programs of limited nesting depth and structured to allow rapid communication of both inputs and outputs to the (fitness determining) environment.

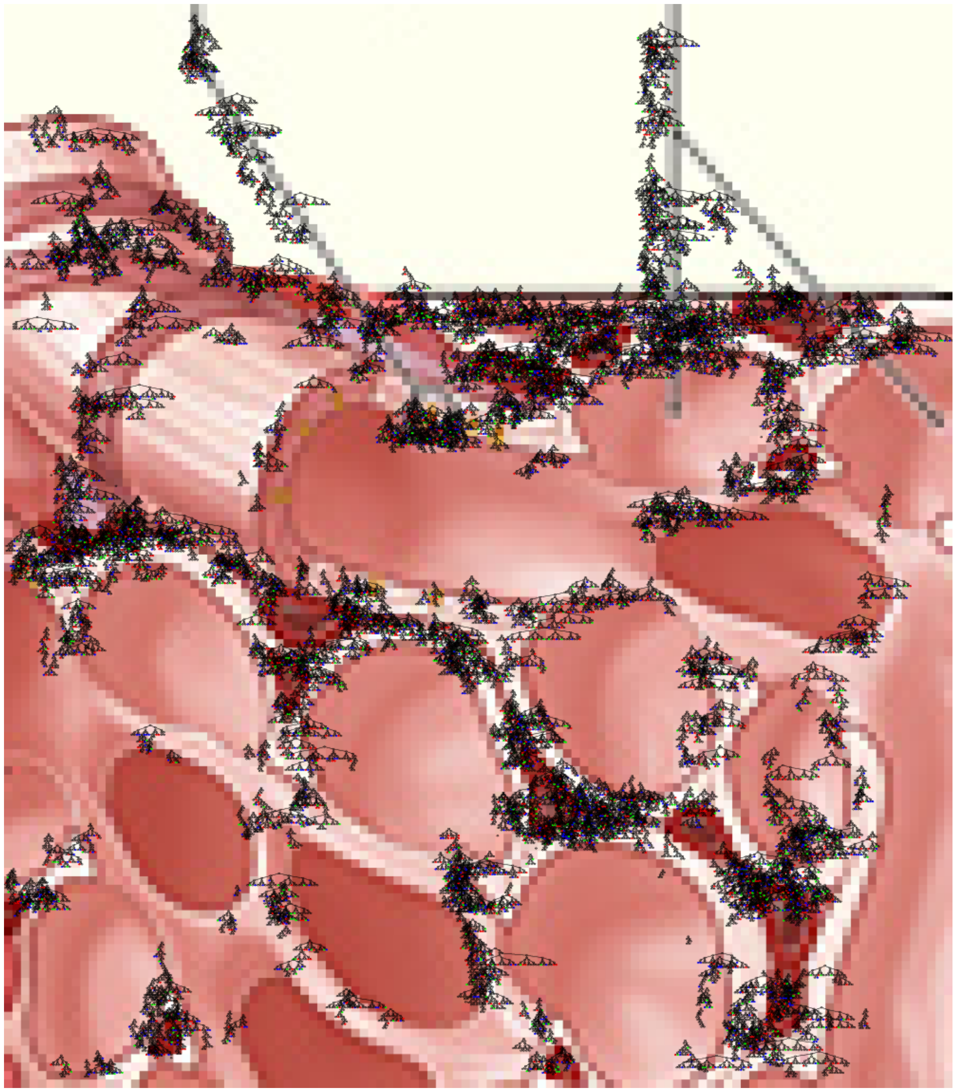

Figure 1: Lung like open complex evolving system composed of 1300 individual GP programs or functions. These compute element are placed side-by-side to form an open structure. The gaps promote short cut side effects between functions' input and outputs and the environment.

2 Open Complex System

Can we evolve systems more like cell interiors, with high surface area membranes composed of very many small adjacent programs each of limited depth placed side by side. The membranes forming an open structure with many gaps between them. The gaps themselves supporting rapid communication (which might be implemented using global memory or enhanced communication links or buses) with no, or little, processing ability and consequently little information loss.

Figure 1 shows 1300 programs arranged in an open structure (based on fit Sextic polynomial [4] trees with average height 9.22, quartile range 7–11).

Rich Lenski, in his long-term evolution experiment (LTEE) [19, 3] has demonstrated that Nature can continue to innovate in a static environment for more than 75 000 generations. We have shown GP can do similarly for at least 100 000 generations. However when evolving deep structures, progress slows dramatically and therefore we feel monolithic deep structures will not be sufficient to automatically evolve complex systems. Instead an open structure like Figure 1 may be needed.

References

[1] D. Clark, W. B. Langdon, and J. Petke. Software robustness: A survey, a theory, and some prospects. Presented at Facebook Testing and Verification Symposium 2020, 1-3 December 2020.

[2] L. M. Deschaine, P. Nordin, and J. D. Pinter. A computational geometric/information theoretic method to invert physics-based MEC models attributes for MEC discrimination. *Journal of Mathematical Machines and Systems*, (2):50–61, 2011.

[3] B. H. Good, M. J. McDonald, J. E. Barrick, R. E. Lenski, and M. M. Desai. The dynamics of molecular evolution over 60,000 generations. *Nature*, 551:45–50, 18 October 2017.

[4] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[5] W. B. Langdon. Genetic programming convergence. *Genetic Programming and Evolvable Machines*.

[6] W. B. Langdon. *Software is not fragile*, volume 5, chapter 4.7, pages 100–101. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2016.

[7] W. B. Langdon. Long-term evolution of genetic programming populations. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 235–236, Berlin, 15-19 July 2017. ACM.

[8] W. B. Langdon. Long-term stability of genetic programming landscapes. In N. Veerapen, F. Daolio, A. Liefooghe, S. Verel, and G. Ochoa, editors, *GECCO 2017 Workshop on Landscape-Aware Heuristic Search*, Berlin, 2017. No paper, presentation only.

[9] W. B. Langdon. Parallel GPQUICK. In C. Doerr, editor, *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 63–64, Prague, Czech Republic, July 13-17 2019. ACM.

[10] W. B. Langdon. Fast generation of big random binary trees. Technical Report RN/20/01, Computer Science, University College, London, Gower Street, London, UK, 13 Jan. 2020.

[11] W. B. Langdon. Multi-threaded memory efficient crossover in C++ for generational genetic programming. *SIGEVOLution newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation*, 13(3):2–4, Oct. 2020.

[12] W. B. Langdon. Fitness first. In W. Banzhaf, L. Trujillo, S. Winkler, and B. Worzel, editors, *Genetic Programming Theory and Practice XVIII*, East Lansing, MI, USA, 19-21 May 2021. Springer. Forthcoming.

[13] W. B. Langdon. Incremental evaluation in genetic programming. In T. Hu, N. Lourenco, and E. Medvet, editors, *EuroGP 2021: Proceedings of the 24th European Conference on Genetic Programming*, volume 12691 of *LNCS*, pages 229–246, Virtual Event, 7-9 Apr. 2021. Springer Verlag.

[14] W. B. Langdon and W. Banzhaf. Continuous long-term evolution of genetic programming. In R. Fuechslin, editor, *Conference on Artificial Life (ALIFE 2019)*, pages 388–395, Newcastle, 29 July - 2 Aug. 2019. MIT Press.

[15] W. B. Langdon and J. Petke. Software is not fragile. In P. Parrend, P. Bourgine, and P. Collet, editors, *Complex Systems Digital Campus E-conference, CS-DC'15*, Proceedings in Complexity, pages 203–211. Springer, Sept. 30-Oct. 1 2015. Invited talk.

[16] W. B. Langdon, J. Petke, and D. Clark. Dissipative polynomials. In N. Veerapen, K. Malan, A. Liefooghe, S. Verel, and G. Ochoa, editors, *5th Workshop on Landscape-Aware Heuristic Search*, GECCO 2021 Companion, pages 1683–1691, Internet, 10-14 July 2021. ACM.

[17] W. B. Langdon, J. Petke, and D. Clark. Information loss leads to robustness. IEEE Software Blog, 12 September 2021.

[18] W. B. Langdon and R. Poli. Mapping non-conventional extensions of genetic programming. *Natural Computing*, 7:21–43, Mar. 2008. Invited contribution to special issue on Unconventional computing.

[19] R. E. Lenski et al. 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.

[20] J. Petke, B. Alexander, E. T. Barr, A. E. I. Brownlee, M. Wagner, and D. R. White. A survey of genetic improvement search spaces. In B. Alexander, S. O. Haraldsson, M. Wagner, and J. R. Woodward, editors, *7th edition of GI @ GECCO 2019*, pages 1715–1721, Prague, Czech Republic, July 13-17 2019. ACM.

[21] J. Petke and A. Blot. Refining fitness functions in test-based program repair. In S. H. Tan, S. Mechtaev, M. Monperrus, and M. Prasad, editors, *The First International Workshop on Automated Program Repair (APR@ICSE 2020)*, pages 13–14, internet, 2020. Association for Computing Machinery.

[22] J. Petke, D. Clark, and W. B. Langdon. Software robustness: A survey, a theory, and some prospects. In P. Avgeriou and D. Zhang, editors, *ESEC/FSE 2021, Ideas, Visions and Reflections*, pages 1475–1478, Athens, Greece, 23-28 Aug. 2021. ACM.

[23] C. G. Pimenta, A. G. C. de Sa, G. Ochoa, and G. L. Pappa. Fitness landscape analysis of automated machine learning search spaces. In L. Paquete and C. Zarges, editors, *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2020)*, volume 12102 of *Lecture Notes in Computer Science*, pages 114–130, Seville, Spain, 15-17 Apr. 2020. Springer.

[24] R. Poli and W. B. Langdon. Sub-machine-code genetic programming. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 13, pages 301–323. MIT Press, Cambridge, MA, USA, June 1999.

[25] R. Poli and J. Page. Solving high-order Boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. *Genetic Programming and Evolvable Machines*, 1(1/2):37–56, Apr. 2000.