7th International Workshop on Intrinsically Motivated Open-ended Learning (IMOL 2025) Nicola Catenacci Volpi et al. *Eds.*, University of Hertfordshire. September 8-10 2025.

Open-Ended Evolution with Linear Genetic Programming

William B. Langdon
Department of Computer Science, University College London, Gower Street, London WC1E 6BT, UK

Abstract

Inspired by Richard Lenski's Long-Term Evolution Experiment, we use the quantised chaotic Mackey-Glass time series as a prolonged learning task for artificial intelligence in the form of steady state linear genetic programming using GPengine to reach up to 100 000 generations. Using two point crossover and point mutation we evolve programs of up to 4 million instructions. Typically finding hundreds of fitness improvements in the later stages of the runs.

Keywords: Autonomous open-ended learning in machines, LTEE, Voas PIE, information theory, failed disruption propagation, catalyst computing, skin depth, thin skinned software

Introduction

Richard Lenski's Long-Term Evolution Experiment Lenski et al. (2015) has shown, even in stable environments, bacteria can continue to evolve, even after 80 000 generations. (In contrast Homo Sapiens is some 9300 generations old.) Previously we have asked the question what happens if we allow artificial evolution, specifically genetic programming (GP) (Koza, 1992; Poli et al., 2008), to evolve for tens of thousands, even hundreds of thousands of generations Langdon and Banzhaf (2022). Whilst we found adaptation continued, in purely hierarchical tree GP using only crossover, we found the rate of innovation fell inversely in proportion to program size due to failed disruption propagation Petke et al. (2021); Langdon and Clark (2024, 2025) promoting population convergence Langdon (2022a). Information theory shows failed disruption propagation is inherent in digital computing and in deep programs can quickly lead to almost all changes (good or bad) being invisible, and so evolution simply drifting and learning stalling.

Instead we have tried to promote the idea that, to avoid failed disruption propagation stifling innovation, for open-ended learning we need to evolve thin walled software with a high surface area (such as inspired by human lungs Langdon (2022b)). We wish to ensure that semantic disruption in the bulk of the code (where most learning will occur) has only a short distance to travel to the surrounding environment and so is likely to be visible and so beneficial changes can be

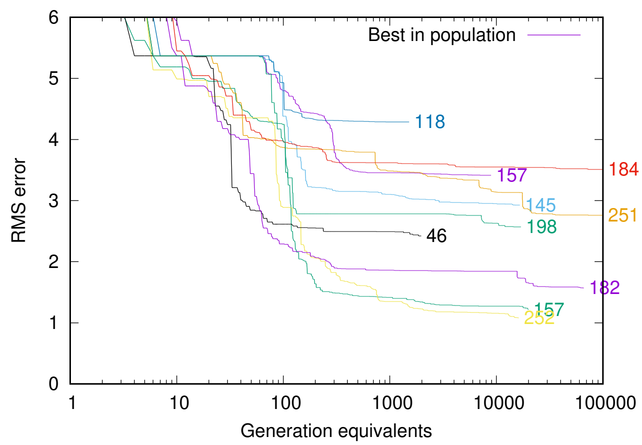

Figure 1: Improvement of best fitness in ten runs of discrete Mackey-Glass chaotic sequence prediction with population of 500. 8 runs cuts short by scheduled reboots. Number of fitness improvements given at the end of each run's trace.

seen and rewarded Langdon and Hulme (2024). Like chemical reactions occurring on a catalyst's surface, rather than a membrane or skin separating computing regions, computing occurs at the surface.

Our intention is investigate other evolving architectures. We start with linear genetic programming Banzhaf et al. (1998); Brameier and Banzhaf (2007) (Figure 1) but will in future investigate evolution of arrays or networks of such programs. We also swap from continuous (float) symbolic regression to predicting discrete (integer) time series, deliberately choosing a chaotic series, as it should prove hard enough to continually challenge learning. Indeed the Mackey-Glass series (Figure 2) can be extended should the predictor approach solving any finite part of it.

We do not want to impose arbitrary limits but it must be admitted that without size control we expect bloat Koza (1992); Tackett (1994); Langdon and Poli (1997); Altenberg (1994); Angeline (1994); Poli and McPhee (2013). Therefore we need a GP system not only able to run for perhaps a million generations but also able to cope with programs of well in excess of a million nodes.

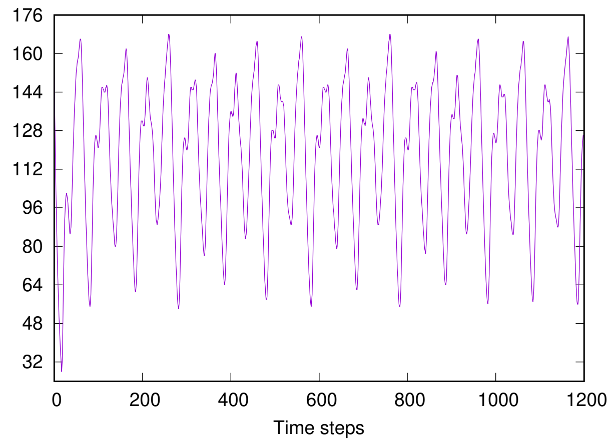

Figure 2: Discrete Mackey-Glass chaotic time series

Table 1: Mackey-Glass prediction with Linear GP

Terminal set:	Unsigned 8 bit integers. Variables R0, R1, R2, R3, R4, R5, R6, R7. Constants 0 to 127.
Function set:	$+ - \times$ DIV
Fitness cases:	1201 Mackey-Glass examples. Given 8 prior values (-1, -2, -4, ... -128 before) predict next y
Selection:	Tournament(2), fit = $\sum_{i=0}^{1201} \lvert \mathrm{GP}(\vec{x_i}) - y_i \rvert^2$
Population:	500, panmictic, steady state.
Parameters:	100 000 generations. Random initial population (500) size between 1 and 14 instructions. 90% two point 2 child crossover, 40% chance both XO children subjected to random point mutation 4 times. 10% reproduction.

DIV is protected division `(y!=0)? x/y : 0`

Experiments

Figure 1 shows typically even late into the run, linear GP continues to find ways to innovate. Also, not only do the programs increase in size, but so too does the number of instructions actually executed. We follow Peter Nordin's intron removal algorithm, i.e. remove instructions which do not impact the program's output R0. The fraction of remaining code is highly variable (1/3.3–1/700, median 1/14).

We anticipated power law Langdon (2000) or even exponential Nordin et al. (1995) growth in program size. However only one run of ten shows almost continual rapid increase in program length. The others show slower growth, sometimes followed by a rapid increase phase. Concentrating upon the two runs which completed 100 000 generations, Figure 3 shows, although innovation continues, the rate of fitness improvement appears to fall more-or-less linearly with increase in program size.

Even with relatively weak selection pressure Goldberg (1989) (steady state Syswerda (1990) populations with binary tournaments Blickle (1996); Langdon (1998)), Figure 3 shows the populations convergence in two senses, many individuals have the best fitness and, even stronger, many individuals return identical values across all the training cases.

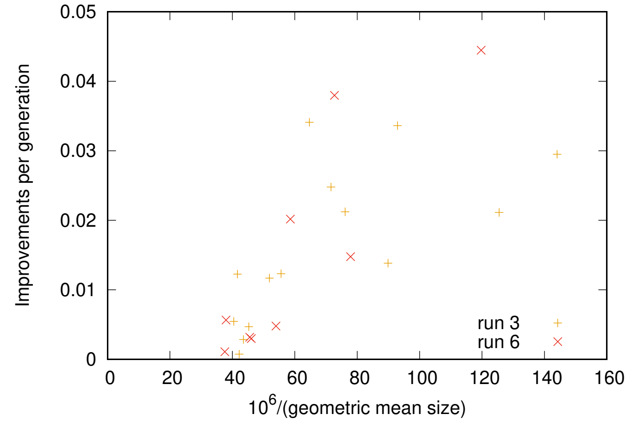

Figure 3: Rate of improvement after generation 1000 in two runs which completed 100 000 generations. To reduce noise, each $+\times$ point is average of ten improvements.

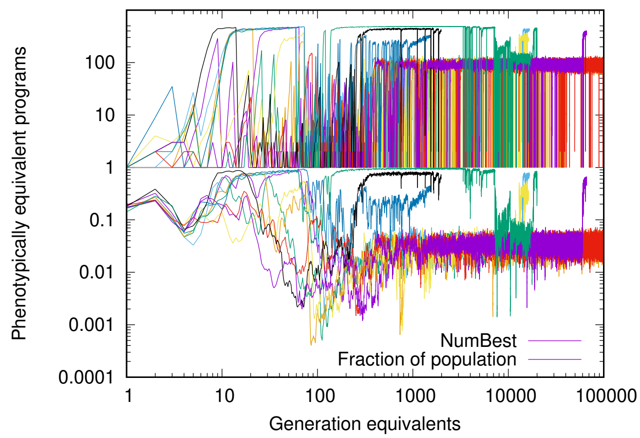

Figure 4: Mackey-Glass linear GP population convergence. Top: number of programs with the best fitness. Bottom: fraction of population which calculate identical answers across 1201 test cases.

Conclusions

We have shown that prolonged evolutionary learning is indeed possible with a simple linear genetic programming system. Our intention is to continue to enhance GPengine and use it as a framework to support analyse of open-ended coevolution of multiple data sharing learning programs.

Based on previous experience Langdon (2020, 2022c), we are confident that use of parallelisation e.g. multi-threading and AVX vector instructions, will greatly increase performance, allowing continual learning theory and experimentation on relatively modest hardware.

Acknowledgment

I would like to thank Sara Silva, John Woodward Woodward and Bai (2009), Eric Medvet, Wolfgang Banzhaf and Leonardo Trujillo.

Appendix

GPengine

GPengine is based on code provided by Peter Nordin. In Langdon and Nordin (2001) we evolved functions with four outputs while in Langdon and Banzhaf (2005) this was reduced to one for the Mackey-Glass prediction problem. GPengine is available via `https://github.com/wblangdon/GPengine`.

Mackey-Glass

We used the IEEE benchmark Mackey-Glass chaotic time series `http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/mackey_glass.tar.gz` $\tau = 17$, 1201 data points, sampled every 0.1, see Figure 2. Mackey-Glass is a continuous problem. The benchmark converts it to discrete time and digitises the continuous data to give byte sized integers (by multiplying by 128 and rounding to the nearest integer) Langdon and Banzhaf (2005).

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press.

Angeline, P. J. (1994). Genetic programming and emergent intelligence. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA.

Blickle, T. (1996). *Theory of Evolutionary Algorithms and Application to System Synthesis*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland.

Brameier, M. and Banzhaf, W. (2007). *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer.

Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley.

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

Langdon, W. B. (1998). *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston.

Langdon, W. B. (2000). Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119.

Langdon, W. B. (2020). Genetic improvement of genetic programming. In Brownlee, A. S., Haraldsson, S. O., Petke, J., and Woodward, J. R., editors, *GI @ CEC 2020 Special Session*, page paper id24061, internet. IEEE Computational Intelligence Society, IEEE Press.

Langdon, W. B. (2022a). Genetic programming convergence. *Genetic Programming and Evolvable Machines*, 23(1):71–104.

Langdon, W. B. (2022b). Open to evolve embodied intelligence. In Iida, F., Hughes, J., Abdulali, A., and Hashem, R., editors, *Proceedings of 2022 International Conference on Embodied Intelligence, EI-2022*, volume 1292 of *IOP Conference Series: Materials Science and Engineering*, page 012021, Internet, Cambridge. IOP Publishing.

Langdon, W. B. (2022c). A trillion genetic programming instructions per second. ArXiv:2205.03251.

Langdon, W. B. and Banzhaf, W. (2005). Repeated sequences in linear genetic programming genomes. *Complex Systems*, 15(4):285–306.

Langdon, W. B. and Banzhaf, W. (2022). Long-term evolution experiment with genetic programming. *Artificial Life*, 28(2):173–204. Invited submission to Artificial Life Journal special issue of the ALIFE'19 conference.

Langdon, W. B. and Clark, D. (2024). Deep mutations have little impact. In An, G., Blot, A., Nowack, V., Krauss, O., and Petke, J., editors, *13th International Workshop on Genetic Improvement @ICSE 2024*, pages 1–8, Lisbon. ACM. Best paper.

Langdon, W. B. and Clark, D. (2025). Deep imperative mutations have less impact. *Automated Software Engineering*, 32:article number 6.

Langdon, W. B. and Hulme, D. (2024). Sustaining evolution for shallow embodied intelligence. In Abdulali, A., Hughes, J., and Iida, F., editors, *Proceedings of 2023 International Conference on Embodied Intelligence, EI-2023*, volume 1321 of *IOP Conf. Series*, page 012007, Internet, Cambridge, UK. IOP Publishing. In EI 2024 but for production reasons published in EI 2023 proceedings.

Langdon, W. B. and Nordin, P. (2001). Evolving hand-eye coordination for a humanoid robot with machine code genetic programming. In Miller, J. F., Tomassini, M., Lanzi, P. L., Ryan, C., Tettamanzi, A. G. B., and Langdon, W. B., editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 313–324, Lake Como, Italy. Springer-Verlag.

Langdon, W. B. and Poli, R. (1997). Fitness causes bloat. In Chawdhry, P. K., Roy, R., and Pant, R. K., editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London.

Lenski, R. E. 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).

Nordin, P., Francone, F., and Banzhaf, W. (1995). Explicitly defined introns and destructive crossover in genetic programming. In Rosca, J. P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA.

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

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Poli, R. and McPhee, N. F. (2013). Parsimony pressure made easy: Solving the problem of bloat in GP. In Borenstein, Y. and Moraglio, A., editors, *Theory and Principled Methods for the Design of Metaheuristics*, Natural Computing Series, pages 181–204. Springer.

Syswerda, G. (1990). A study of reproduction in generational and steady state genetic algorithms. In Rawlings, G. J. E., editor, *Foundations of genetic algorithms*, pages 94–101. Morgan Kaufmann, Indiana University. Published 1991.

Tackett, W. A. (1994). Greedy recombination and genetic search on the space of computer programs. In Whitley, L. D. and Vose, M. D., editors, *Foundations of Genetic Algorithms 3*, pages 271–297, Estes Park, Colorado, USA. Morgan Kaufmann. Published 1995.

Woodward, J. R. and Bai, R. (2009). Canonical representation genetic programming. In Xu, L., Goodman, E. D., Chen, G., Whitley, D., and Ding, Y., editors, *GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 585–592, Shanghai, China. ACM.