Proceedings of 2022 International Conference on Embodied Intelligence, EI-2022 IOP Publishing
Mater. Sci. Eng. 1292 (2023) 012021 doi:10.1088/1757-899X/1292/1/012021

Open to Evolve Embodied Intelligence

W B Langdon

Department of Computer Science, University College London, Gower Street, WC1E 6BT, UK

E-mail: w.langdon@cs.ucl.ac.uk

Abstract. For the goal of automatically evolving Embodied Intelligence (EI), we investigate an open software architecture inspired by the high surface area to volume ratio of animal lungs, which aims to avoid information theoretic limits on long term evolution experiments (LTEE) encountered with monolithic genetic programming trees. Instead individuals are teams composed of 1023 trees whose inputs and outputs are linked by a low entropy loss branching data (air) pathway. Most trees are shallow and software engineering's failed disruption propagation (FDP) is observed in the small fraction of deep trees. After initial search, most improvements are at intermediate depths and performance is still rising even after 100 000 generations.

Despite the use of double precision for the bifurcating data interconnect, some information loss is seen, particularly in early generations. The static optimisation benchmark, appears to encourage early convergence, which locks the population into possibly sub-optimal phenotypes. Later thousands of small improvements, sometimes in large bloated ensemble members, appear to compensate for early overfitting.

Using tournament fitness selection and subtree crossover, we target pure nested side-effect free floating point functions, which are known to have low FDP, and high fidelity data paths, in the hope of generating code which is not too robust so as to prevent on going improvement. However, we again find genetic changes deep within trees are silent. For single precision, we find a maximum evolvability sweet spot with trees of depth 10 to 100. Accordingly, we suggest to evolve very large very complex programs needed for Embodied Intelligence, an open structure with a high surface area permitting most mutation sites to be within 10–100 levels of the organism's environment, and many better placed test oracles to monitor the impact of mutated code, will be needed.

1. Open Complex System

Efforts to evolve artificial intelligence are running into information theoretic limits on program depth. Instead can we evolve software systems which are like cell interiors (e.g. Figure 1). These could have processing concentrated in thin computing membranes in a permeating data interconnect environment. The high surface area membranes might be 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 high bandwidth communication with no, or little, processing ability and consequently little information loss. Figure 2 shows 1300 small programs arranged in an open structure.

Rich Lenski, in his long-term evolution experiment (LTEE) [3, 4] has demonstrated that Nature can continue to innovate in a static environment for more than 75 000 generations. We have shown genetic programming [5] can do similarly for at least 100 000 generations. However, when evolving deep structures, progress slows dramatically and therefore we feel monolithic

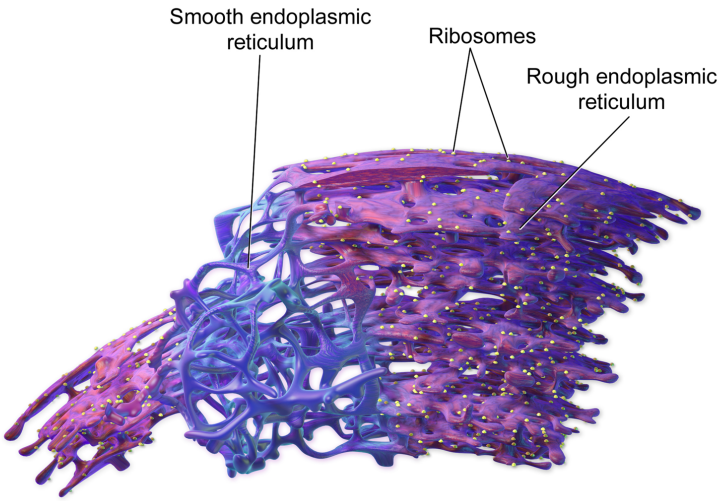

Figure 1. Small fraction of one of the active membranes within eukaryotic cells [1].

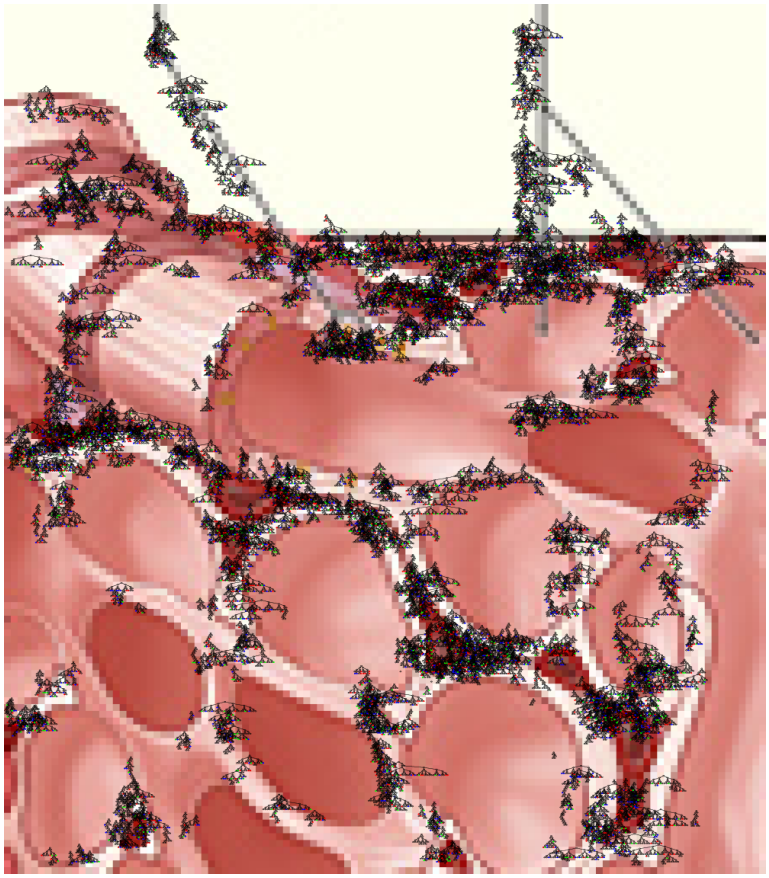

Figure 2. Lung like open complex evolving system composed of 1300 individual genetic programming functions (average height 9.22). These compute element are placed side-by-side to form an open structure. The gaps promote short-cut side effects between functions' input and outputs and the environment [2].

deep structures will not be sufficient to automatically evolve complex systems. Instead an open structure like Figure 2 may be needed.

Section 3 describes initial experiments using an architecture inspired by our lungs. Section 4 considers the discovery of the maximum evolvability sweet spot and how it may change over time or in other experiments. In Section 5 we consider ways to improve these experiments, and in Section 6 we conclude that although we do see continued fitness improvement, there is still a long way to go to demonstrate automatic Embodied Intelligence. But first, the next section describes recent work based on information theory, which lead to the view that failed disruption propagation is inevitable in both human designed and automatically evolved digital software, and therefore the need to control the depth of nesting in Embodied Intelligence.

2. Background: Evolution and Failed Disruption Propagation in Nested Software

Taking advantage of modern high performance parallel computers, we have been investigating the long term evolution of genetic programming (GP). Firstly, using Poli's submachine code GP [6, 7], to evolve large binary Boolean trees [8] and more recently exploiting SIMD Intel AVX and multi-core parallelism to evolve floating point GP [9, 10]. Running for up to a million generations without size limits has generated, at two billion nodes, the biggest programs yet evolved and forced the development [11, 12, 13] of, at the equivalent of more than a trillion GP operations per second, the fastest GP system [14, 15, 16]. It has also prompted detailed analysis of programs [17], including from an information theoretic [18] perspective [19, 20, 21]. (Of course information theory has long been used with evolutionary computing, e.g. [22].)

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 [23].

Without insight [24] into the construction of the test suite (i.e. in black box testing), and where fitness testing is limited by failed disruption propagation (FDP) [25], simply increasing the number of tests suffers diminishing returns and their joint effectiveness at *best* increases by a slow logarithmic $O(\log n)$ factor when tests are independent [21]. Instead FDP suggests, if possible, fitness tests should be augmented by checking values as close to the genetic change as possible. Alternatively to alleviate FDP, we could move the genetic changes closer to the existing test oracle [26] (at the tree's root). However this risks generating huge amounts of "dead" non-evolving deep code, which once created becomes fossilised and is never subsequently modified.

We have also considered traditional (human written) imperative programs and shown these to are much more robust than is often assumed [27, 28]. Indeed we suggest that information theory provides a unified view of the difficulty of testing software [29, 30, 25], particularly positioning test oracles [31, 32, 20].

The question of why fitness is so often exactly inherited [33, 34, 35], despite brutal genetic change, is answered by the realisation that without side effects, the disruption caused by mutation or crossover must be propagated up the tree through a long chain of irreversible operations to the root node. Being irreversible, 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 giving rise to a deep neutral network. Thus the mutations and crossovers become invisible to fitness testing and their utility cannot be measured. Without fitness selective pressure, evolution degenerates into an undirected random walk [8, 10].

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.

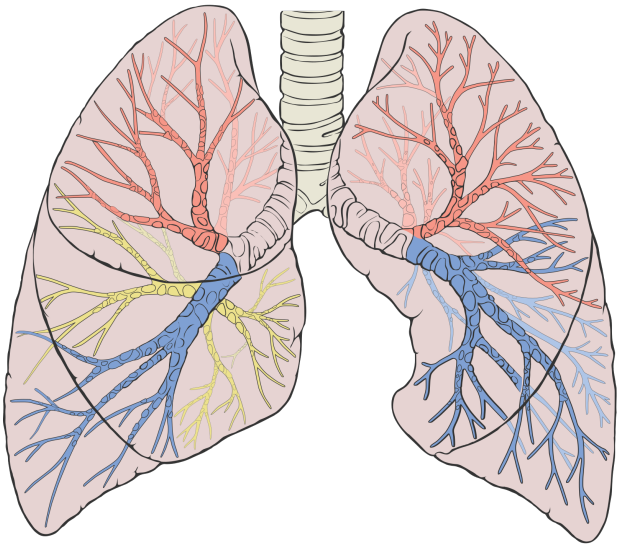

Figure 3. Human lungs [38].

Since all digital computing is irreversible [36], it inherently losses information and so without short cuts, must lead to failed disruption propagation (FDP). 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.

3. Experiments with a High Surface Area Lung Like Architecture

We seek a software architecture with a large interface between thin evolvable code and I/O data. We take inspiration from lungs (Figure 3), which hold a large surface area between the animal's body, particularly its blood vessels, and the air, in order to permit exchange of gases such as carbon dioxide and oxygen out of and into the animal. Although the lungs of different mammals vary enormously in size, the airways show a common bifurcating pattern with the diameter of the airways decreasing as $2^{-\frac{n}{3}}$ for the first 17 branches. Branching continues to n=24 but with a slower power law [37].

To limit computational costs, in our initial experiment we limit our "lungs" to a bifurcation depth of n=10, giving 511 branch points (bronchioles) and 512 alveoli ends (total 1023, Figure 4). We anchor a traditional genetic programming (GP) tree [5] at each (giving 1023 evolvable trees per individual). We think of each tree as having the task of regulating the passage of data through the data channel beneath it. The output of each tree is discharged into the channel. In addition to the usual GP primitives (see Table 1), trees at branch points can monitor data upstream of them, data in the channel to their left and to their right. All the trees can read how deep they are in the "lung" using the special leaf D, which has a value 1 to 10. The 512 end point trees (D=10) in each individual are evaluated first. These are followed by the 256 trees closer to the "lung" "output" (D=9). Thus level 9 trees can access left, right and current values, which have been calculated from the outputs of the neighbouring level 10 trees. These are followed by the 128 level 8 trees, and so on until the level 1 tree.

To minimise entropy loss, double precision is used throughout the data channel, e.g. for calculating "current". However as usual, normal single precision floating point precision is used inside the trees and they generate single precision outputs. Double precision is also used when calculating the mean of the 1023 outputs during fitness calculation.

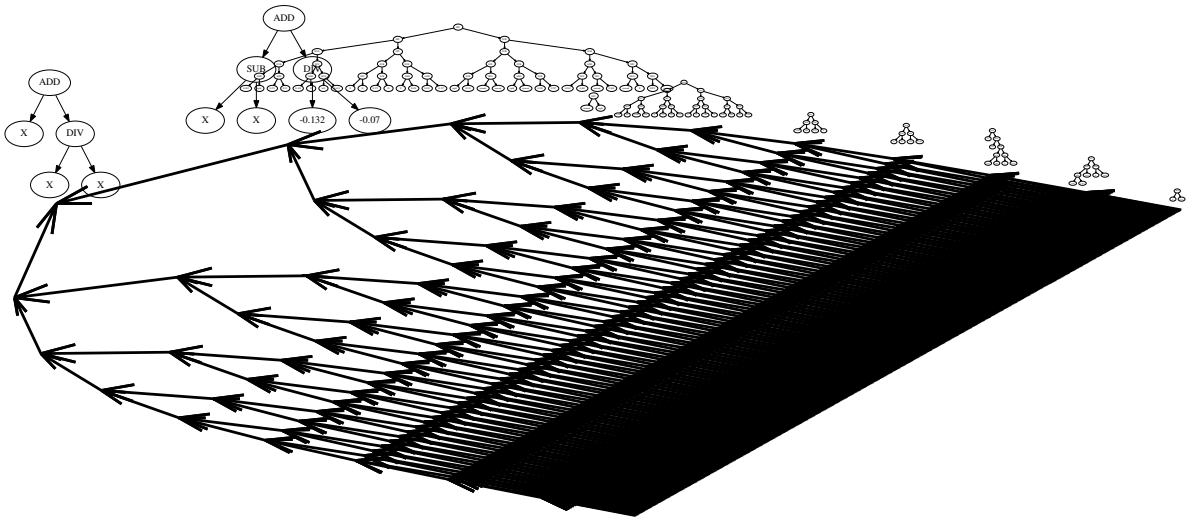

Figure 4. Lung like data structure. Data flows from 512 alveoli ends (right) collecting outputs of evolving trees and providing them with addition inputs. Ten of 1023 trees shown.

Table 1. Genetic programming lung experiment with Sextic polynomial symbolic regression

Tree leafs:	X D and 246 constants between -0.995 and 0.997 (bronchioles, D=1 to D=9) trees have additional leafs: left right current
Tree internal nodes:	MUL ADD DIV SUB
Fitness test cases:	48 fixed inputs -0.97789 to 0.979541 (randomly selected from -1.0 to +1.0) Target (Sextic polynomial) $y = xx(x-1)(x-1)(x+1)(x+1)$
Selection:	Tournament size 7 with fitness = $\frac{1}{48}\sum_{i=1}^{48} \lvert GP(x_i) - y_i \rvert$
Population:	500 teams each of 1023 trees. Population is fully mixed (panmictic), non-elitist, with separate, non-overlapping, generations.
GP parameters:	Initial trees are each created with ramped half and half [39], depth between 2 and 6. 100% unbiased subtree crossover. 100 000 generations. No size or depth limits.

DIV is protected division `(y!=0)? x/y : 1.0f`

For each test case, the GP's output is the mean of all 1023 trees. Fitness is the sum over the 48 test cases of absolute difference between the mean output and the target value. Occasionally an individual tree gives a very bad non-finite value. This invalidates the mean of all the trees, but such individuals have very poor fitness and are seldom selected to be parents and are excluded from the population without any special processing.

3.1. Initial Results

Figure 5 shows, as hoped, the new architecture is able to solve the problem in a reasonable time[1] and continues to show improvements even after many thousands of generations[2].

The runs show some aspects of GP convergence [10]. Although Figure 6 shows there is variation between runs, they all show large increase in size (bloat). Across the ten runs, 86% of trees in the best individual in the population do not change throughout the last 90% of the run. The distribution of these "converged" trees has a long tail but most are small and contain only

[1] On average the ten runs each took less than two days on a single Intel i7-4790 3.60 GHz core.

[2] On average 608 children that were better than their mums were created in the last 1000 generations.

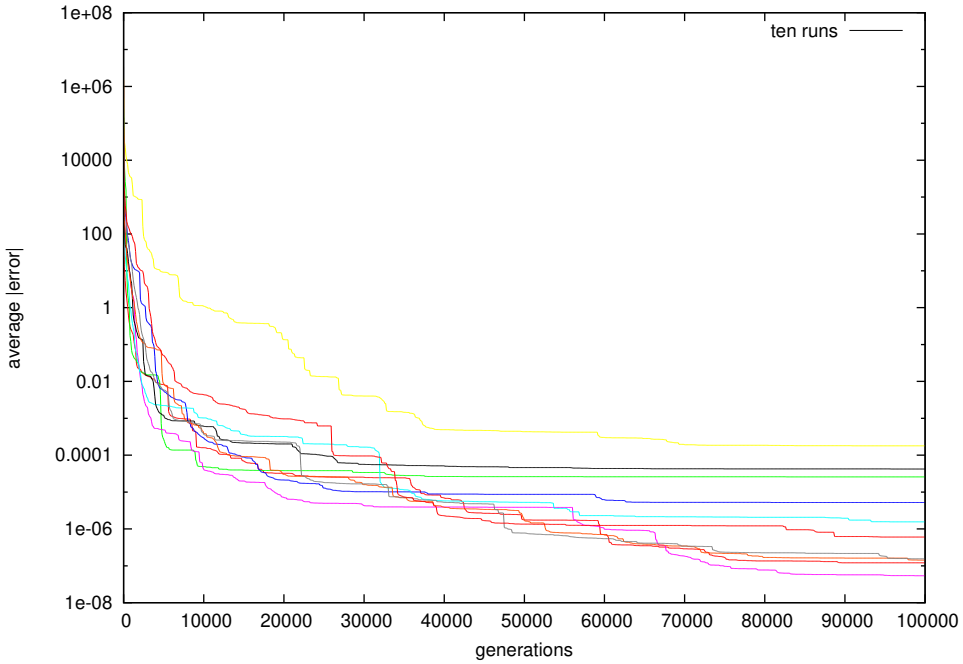

Figure 5. Evolution of mean absolute error in ten runs of Sextic polynomial [39] with population of 500, each team composed of 1023 trees. One crossover per team per generation, i.e. 1022 trees copied without modification from the 1st parent (mum).

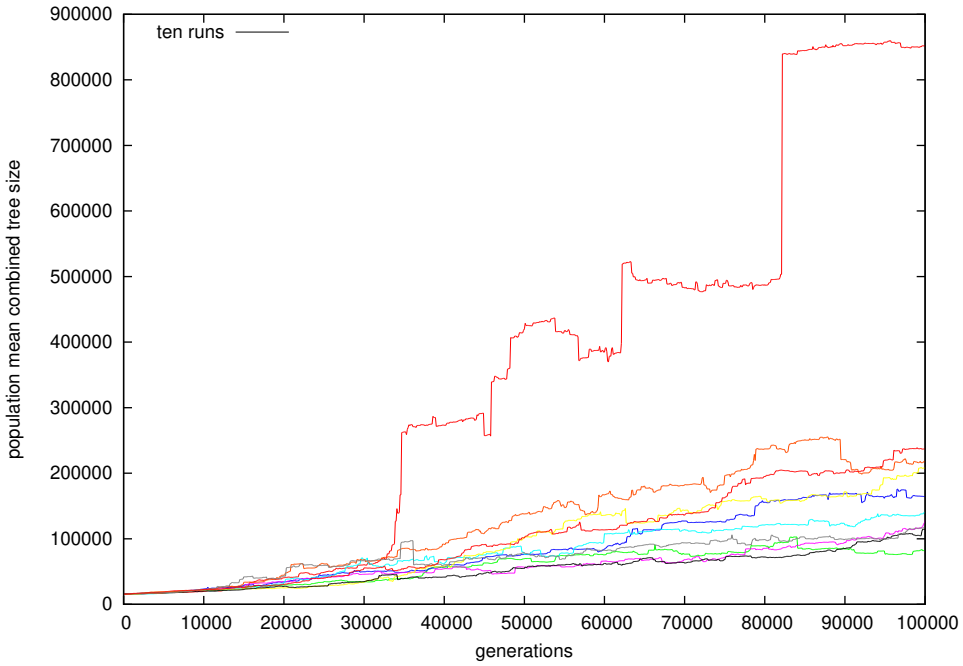

Figure 6. Evolution of total ensemble [40] size in ten runs. (Same colours as runs in Figure 5.)

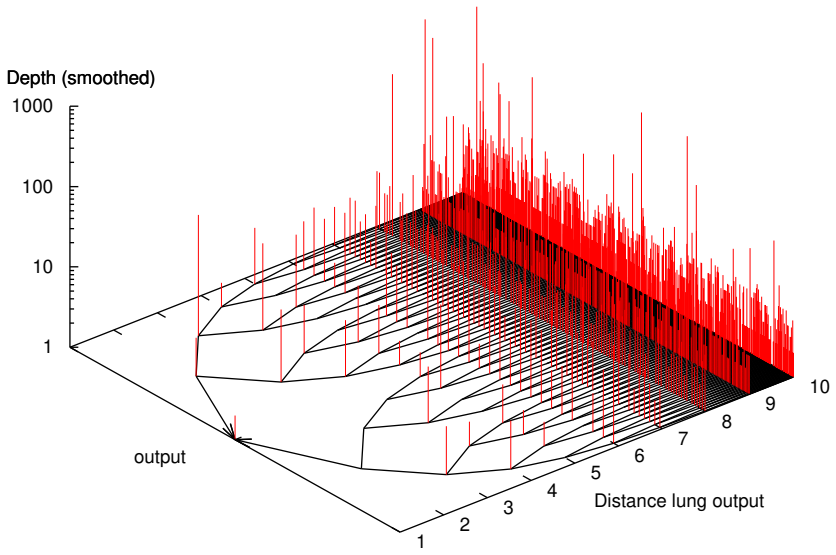

Figure 7. Depth of best in population GP trees after 100 000 generations in a typical run.

1, 3 or 5 nodes (i.e. up to two functions and three leafs). Indeed, at the end of the runs, about half the best trees contain five or fewer nodes. In contrast, of those tree which do change their height in the last 90% of the run, 41% (5% of all trees) contain on average more than 10 levels by the end of their run. It is the largest trees amongst this 5% that give the great increase in size seen in Figure 6.

The location of the large trees is different in each run, and appears to be essentially random. Figure 7 shows for the best individual in the first run the depth and location of its constituent trees.

We expect our large evolved "lung" trees to behave as other large binary GP trees, so that on average the distance from crossover points to the trees' output is about half the depth of the tree [41, 42, 10]. For deep trees this means any disruption caused by crossover may fail to propagate to the root node and so the child's fitness will be equal to its parent's. Figure 8 reports failed disruption propagation (FDP) in the deep trees. The solid curve in Figure 8 is the empirical fit for FDP which suggests at least half of crossovers deeper than 63 levels will make no difference. (With other mixtures of functions, e.g. containing logic variables or conditional statements, we expect the depth needed to be much less [8, 20].)

In the case of shallow trees, crossovers must be near the root node, however Figure 9 shows that at the end of the runs, although almost all shallow crossovers do disrupt fitness only a tiny fraction improve fitness. Indeed by the end of the 10 runs the 2300 trees composed of a single leaf were sufficiently converged, that in the last 1000 generations all 1 122 850 crossovers gave back the same child. As expected with trees containing at least one function (i.e. size 3) most crossovers are disruptive (until trees are deep enough to suffer FDP). These small converged trees are the most resistant to improvement with only about 0.1% of children being better. However intermediate converged trees are more evolvable as deeper crossovers become possible and the fraction of improvements increases to about 0.5%. The deepest improvement is 335 levels from the root node (Figure 9). But only three of the 6 845 crossovers deeper than 200 improve fitness.

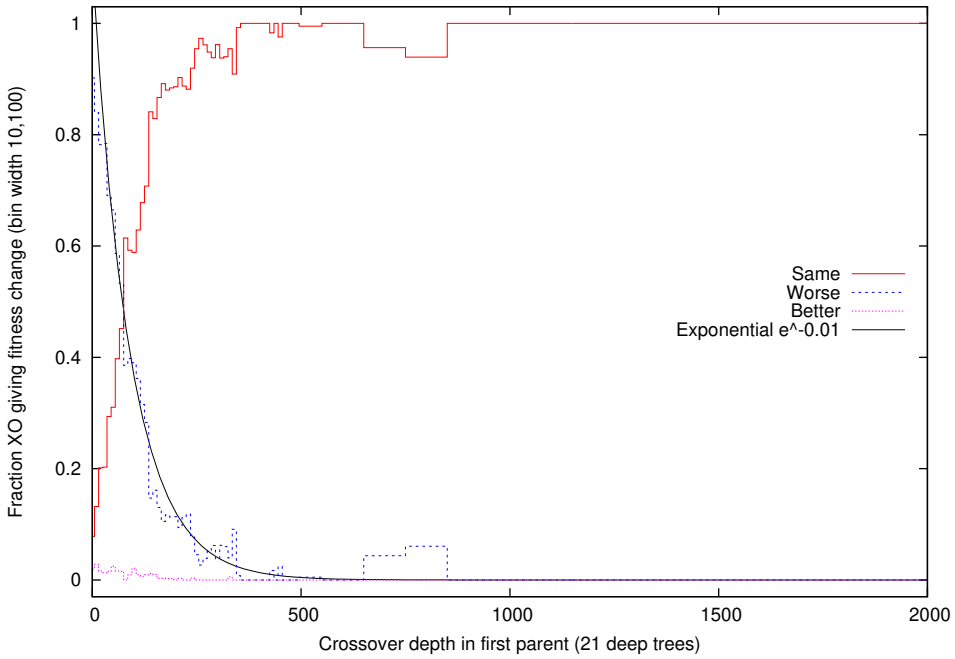

Figure 8. Solid red line shows the fraction of crossovers in 21 trees deeper than 300 levels that do not change fitness. Dashed blue make it worse. Dotted purple improve fitness. Grouped by distance from crossover site to the root node. Above 500, the bin size is increased from 10 to 100. The smooth curve (black) is the best RMS fit of an exponential decay to the fraction of crossovers which do change fitness. (Data gathered from last 1000 generations in all ten runs.)

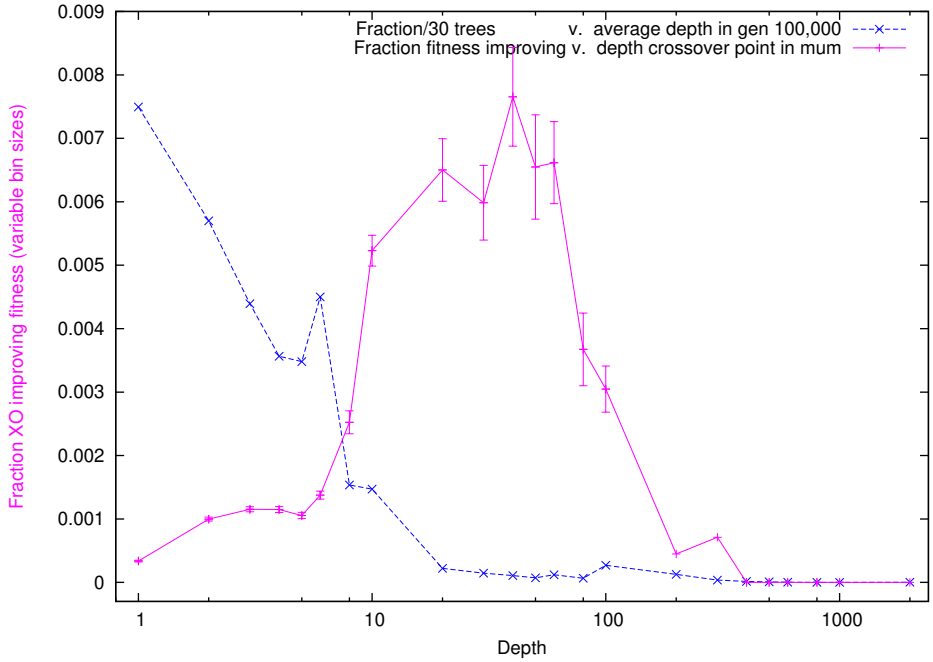

Figure 9. Solid purple line gives, for the ten runs in the last 1000 generations, the fraction of crossovers whose disruption not only propagates to the root node but also improves fitness (see also Figure 8). For comparison the dashed blue line gives the fraction of trees by their average depth at the end of the ten runs (rescaled by 1/30 to plot on same axis). Note log horizontal scale.

4. Evolution of Maximum Evolvability

In the last 1000 generations there is more chance of crossovers at least 7 levels deep being successful (see Figure 9). It appears that this is because this late in the run the population is now good and so only has scope for small fitness improvements. It seems that deeper changes may find small phenotype changes which are sometimes beneficial, whereas code changes close to the root node may make larger phenotypic changes, which could overshoot the target output. This trend seems to continue to about depth 20. Still deeper, failed disruption propagation (FDP) starts preventing many crossovers changing the phenotype at all.

We anticipate over considerably extended evolution, perhaps many millions of generations, continued improvement of the ensemble, will mean the fitness step size will continue to decrease, moving the "left cliff" of the evolvability sweet spot in Figure 9 to the right. However the "right FDP" edge will not move. Leading eventually to a narrowing of the maximum evolvability range. Possibly this will be accompanied by more team members moving into that narrowing range?

With a different mix of ingredients in our trees, especially if they include conditionals or Boolean operators, there may be considerably more entropy loss, so causing much more FDP. This will reduce the maximum depth limit on the location of maximum evolvability. (I.e. move the "FDP cliff" to the left.) To avoid pinching out our evolvability bubble, we now need to consider again its left hand edge. We need to ensure that the fitness function is not too discrete. Instead, the fitness calculation must be sufficiently graded, so that as we make progress and fitness improves, it is still possible for genetic changes to find small fitness improvements.

5. Discussion

In theoretical work on small digital circuits Wright and Laue [43] argue that digital evolution supports an "arrow of complexity" whereby maximum Kolmogorov complexity increases with time. Note that by using Kolmogorov complexity they measure not increase in program size (i.e. bloat) but changes in functionality. Here we see huge overfitting and so are seeing Kolmogorov complexity increase. It is doubtful that this is useful, but our goal is not to solve the benchmark problem, but to use it to explore ways and difficulties of extended evolution [44, 14].

Another interesting recent experiment by Kelly, Smith, Heywood and Banzhaf [45], considers extended genetic programming evolution (up to 89 days) using their "Tangled Program Graphs" representation. They evolve agents to play aspects of the video game ViZDoom.

In terms of our experiments, it seems we need more open ended benchmarks. We have stuck with just one genetic change per fitness evaluation [46]. However, with increasing numbers of team members this begins to look infeasible. Certainly in human genetic we see genomes being split into multiple diploid chromosomes with multiple crossover points and mutation sites and a more equal contribution of identifiable genes from the two parents. In terms of computational experiments it may be feasible to use multiple fitness measurements, possibly include cheap ways [47] to abort complete fitness evaluation early [48], to complement increased number of crossovers or mutations per individual per generation.

Although we should be wary of building too many preconception into any system which hopes to show extended evolution, having studied the smothering blanket of failed disruption propagation (FDP) in pure floating point code and also in integer [20] and logic [8] expressions, perhaps it is time to adopted more directed choices of genetic changes. Alternatively rather than cooking in our choices, perhaps such as system could learn from its own past and be self adaptive.

6. Conclusions

In all runs we see a few trees growing to enormous depth, which become resistant to phenotypic change due to failed disruption propagation. It may be that future evolving "thin membrane" systems will have to include mechanisms to prevent such growths or include mechanisms to

exorcise them. We also see shallow trees quickly discovered but then become less evolvable as the team as a whole improves. Nevertheless we have that demonstrated a new many tree architecture, inspired by the branching air pathways in lungs, is capable of supporting extended evolution. As with many evolutionary systems (and indeed life on earth) it shows early lock-in followed by many small evolutionary improvements. For our pure floating point expressions, typically about 70 trees in the ensembles evolve into the sweet spot of maximum evolvability and have a depth between 10 and 100 nested levels.

Acknowledgement
This work was initially inspired by conversations at Dagstuhl Seminar 18052 on Genetic Improvement of Software [49]. Supported by the Meta Oops project.

References

[1] Blaus B 2014 *WikiJournal of Medicine* **1** ISSN 2002-4436 URL http://dx.doi.org/10.15347/wjm/2014.010
[2] Langdon W B 2022 *SIGEVOlution newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation* **15** ISSN 1931-8499 URL http://dx.doi.org/10.1145/3532942.3532945
[3] Lenski R E *et al.* 2015 *Proceedings of the Royal Society B* **282** ISSN 0962-8452 URL http://dx.doi.org/10.1098/rspb.2015.2292
[4] Good B H *et al.* 2017 *Nature* **551** 45–50 URL http://dx.doi.org/10.1038/nature24287
[5] Poli R *et al.* 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) URL http://www.gp-field-guide.org.uk
[6] Poli R and Langdon W B 1999 *Advances in Genetic Programming 3* ed Spector L *et al.* (MIT Press) chap 13, pp 301–323 ISBN 0-262-19423-6 URL http://www.cs.ucl.ac.uk/staff/W.Langdon/aigp3/ch13.pdf
[7] Poli R and Page J 2000 *Genetic Programming and Evolvable Machines* **1** 37–56 ISSN 1389-2576 URL http://dx.doi.org/10.1023/A:1010068314282
[8] Langdon W B 2017 *GECCO '17 Companion* ed Wagner M *et al.* (Berlin: ACM) pp 235–236 URL http://dx.doi.org/10.1145/3067695.3075965
[9] Langdon W B 2019 *GECCO '19 Companion* ed Doerr C *et al.* (Prague, Czech Republic: ACM) pp 63–64 URL http://dx.doi.org/10.1145/3319619.3326770
[10] Langdon W B 2022 *Genetic Programming and Evolvable Machines* **23** 71–104 ISSN 1389-2576 URL http://dx.doi.org/10.1007/s10710-021-09405-9
[11] Langdon W B 2020 Fast generation of big random binary trees Tech. Rep. RN/20/01 Computer Science, University College, London Gower Street, London, UK URL https://arxiv.org/abs/2001.04505
[12] Langdon W B 2020 *SIGEVOLution newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation* **13** 2–4 ISSN 1931-8499 URL http://dx.doi.org/10.1145/3430913.3430914
[13] Langdon W B 2021 *Genetic Programming Theory and Practice XVIII* ed Banzhaf W *et al.* (East Lansing, MI, USA: Springer) pp 143–164 URL http://dx.doi.org/10.1007/978-981-16-8113-4_8
[14] Langdon W B and Banzhaf W 2022 *Artificial Life* **28** 173–204 ISSN 1064-5462 invited submission to Artificial Life Journal special issue of the ALIFE'19 conference URL http://dx.doi.org/10.1162/artl_a_00360
[15] Langdon W B *ACM Transactions on Evolutionary Learning and Optimization* ISSN 2688-299X just accepted URL http://dx.doi.org/10.1145/3539738
[16] Langdon W B 2022 A trillion genetic programming instructions per second ArXiv URL https://arxiv.org/abs/2205.03251
[17] Langdon W B 2022 *Complex Systems* **31** ISSN 0891-2513 forthcoming
[18] Renyi A 1987 *A Diary on Information Theory* Probability and Statistics, Applied Probability and Statistics Section (Chichester: John Wiley and Sons) ISBN 0 471 9971 8
[19] Langdon W B *et al.* 2021 *5th Workshop on Landscape-Aware Heuristic Search* GECCO '21 Companion ed Veerapen N *et al.* (Internet: ACM) pp 1683–1691 URL http://dx.doi.org/10.1145/3449726.3463147
[20] Langdon W B 2022 *GECCO '22 Companion* ed Trautmann H *et al.* (Boston, USA: ACM) URL http://dx.doi.org/10.1145/3520304.3528878
[21] Langdon W B *et al.* 2022 *GECCO '22* ed Rahat A *et al.* (Boston, USA: ACM) pp 964–972 URL http://dx.doi.org/10.1145/3512290.3528738
[22] Deschaine L M *et al.* 2011 *Journal of Mathematical Machines and Systems* 50–61 URL http://www.immsp.kiev.ua/publications/eng/2011_2/
[23] Langdon W B 2021 *EuroGP 2021: Proceedings of the 24th European Conference on Genetic Programming*

(*LNCS* vol 12691) ed Ting Hu *et al.* (Virtual Event: Springer Verlag) pp 229–246 URL http://dx.doi.org/10.1007/978-3-030-72812-0_15

[24] Petke J *et al.* 2015 *IEEE Transactions on Software Engineering* **41** 901–924 URL http://dx.doi.org/10.1109/TSE.2015.2421279

[25] Petke J *et al.* 2021 *ESEC/FSE 2021, Ideas, Visions and Reflections* ed Avgeriou P and Zhang D (Athens, Greece: ACM) pp 1475–1478 URL http://dx.doi.org/10.1145/3468264.3473133

[26] Bruce B R *et al.* 2019 *IEEE Transactions on Software Engineering* **45** 1150–1169 ISSN 0098-5589 URL http://dx.doi.org/10.1109/TSE.2018.2827066

[27] Langdon W B and Petke J 2015 *Complex Systems Digital Campus E-conference, CS-DC'15* Proceedings in Complexity ed Parrend P *et al.* (Springer) pp 203–211 invited talk URL http://dx.doi.org/10.1007/978-3-319-45901-1_24

[28] Langdon W B 2016 *Software is not fragile* vol 5 (Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik) chap 4.7, pp 100–101 URL http://dx.doi.org/10.4230/DagRep.5.10.89

[29] Langdon W B *et al.* 2021 Information loss leads to robustness IEEE Software Blog URL http://blog.ieeesoftware.org/2021/09/information-loss-leads-to-robustness-w.html

[30] Clark D *et al.* 2020 Software robustness: A survey, a theory, and some prospects Presented at Facebook Testing and Verification Symposium 2020 URL https://fbresearchevents.bevylabs.com/events/details/facebook-tav-symposium-division-facebook-testing-and-verification-symposium-presents-dress-rehearsal-facebook-tav-symposium-2020/

[31] Jahangirova G *et al.* 2016 *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA'16)* (Saarbruecken, Germany: ACM) pp 247–258 URL http://dx.doi.org/10.1145/2931037.2931062

[32] Terragni V *et al.* 2020 *ESEC/FSE 2020* ed Cohen M B and Zimmermann T (Sacramento, California, USA) pp 1178–1189 URL http://dx.doi.org/10.1145/3368089.3409758

[33] Petke J *et al.* 2019 *7th edition of GI @ GECCO 2019* ed Alexander B *et al.* (Prague, Czech Republic: ACM) pp 1715–1721 URL http://dx.doi.org/10.1145/3319619.3326870

[34] Petke J and Blot A 2020 *The First International Workshop on Automated Program Repair (APR@ICSE 2020)* ed Shin Hwei Tan *et al.* (internet: ACM) pp 13–14 URL http://dx.doi.org/10.1145/3387940.3392180

[35] Pimenta C G *et al.* 2020 *EvoCOP 2020* (*LNCS* vol 12102) ed Paquete L and Zarges C (Seville, Spain: Springer) pp 114–130 URL http://dx.doi.org/10.1007/978-3-030-43680-3_8

[36] Langdon W B and Poli R 2008 *Natural Computing* **7**(1) 21–43 invited contribution to special issue on Unconventional computing URL http://dx.doi.org/10.1007/s11047-007-9044-x

[37] Iber D 2021 *Cellular Networks in Development* (*Current Topics in Developmental Biology* vol 143) ed Affolter M (Academic Press) chap 6, pp 205–237 URL http://dx.doi.org/10.1016/bs.ctdb.2021.02.002

[38] Lynch P J and Jaffe C C 2006 Lungs diagram with internal details Yale University School of Medicine, Center for Advanced Instructional Media URL https://en.wikipedia.org/wiki/Lung

[39] Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press) ISBN 0-262-11170-5 URL http://mitpress.mit.edu/books/genetic-programming

[40] Langdon W B *et al.* 2003 *Applications of Evolutionary Computing, EvoWorkshops2003* (*LNCS* vol 2611) ed Raidl G R *et al.* (Essex, UK: Springer) pp 87–98 URL http://dx.doi.org/10.1007/3-540-36605-9_9

[41] Sedgewick R and Flajolet P 1996 *An Introduction to the Analysis of Algorithms* (Addison-Wesley) ISBN 0-201-40009-X

[42] Langdon W B 2000 *GECCO-2000* ed Whitley L D *et al.* (Las Vegas, Nevada, USA: Morgan Kaufmann) pp 451–458 ISBN 1-55860-708-0 URL http://gpbib.cs.ucl.ac.uk/gecco2000/GA069.pdf

[43] Wright A H and Laue C L 2021 *GECCO '21* ed Chicano F *et al.* (internet: ACM) pp 840–848 URL http://dx.doi.org/10.1145/3449639.3459393

[44] Altenberg L 1994 *Advances in Genetic Programming* ed Kinnear, Jr K E (MIT Press) chap 3, pp 47–74 URL http://dynamics.org/~altenber/PAPERS/EEGP/

[45] Kelly S *et al.* 2021 *ACM Transactions on Evolutionary Learning and Optimization* **1** ISSN 2688-299X URL http://dx.doi.org/10.1145/3468857

[46] Langdon W B 1998 *Genetic Programming and Data Structures* (Kluwer) ISBN 0-7923-8135-1 URL http://dx.doi.org/10.1007/978-1-4615-5731-9

[47] Maxwell III S R 1994 *IEEE World Congress on Computational Intelligence* vol 1 (Orlando, Florida, USA) pp 413–417a ISBN 0-7803-1899-4 URL http://dx.doi.org/10.1109/ICEC.1994.349915

[48] Tackett W A 1994 *Foundations of Genetic Algorithms 3* ed Whitley L D and Vose M D (Estes Park, Colorado, USA: Morgan Kaufmann) pp 271–297 ISBN 1-55860-356-5 published 1995 URL http://dx.doi.org/10.1016/B978-1-55860-356-1.50017-0

[49] Petke J *et al.* 2018 *Dagstuhl Reports* **8** 158–182 ISSN 2192-5283 URL http://dx.doi.org/10.4230/DagRep.8.1.158