EuroGP-2005, M. Keijzer, A. Tettamanzi, M. Tomassini, J. van Hemert and P. Collet editors, Lausanne, Switzerland, 30 March–1 April 2005, Springer-Verlag

Repeated Patterns in Tree Genetic Programming

W. B. Langdon[1] and W. Banzhaf[2]

[1] Computer Science, University of Essex, UK
[2] Computer Science, Memorial University of Newfoundland, Canada

Abstract. We extend our analysis of repetitive patterns found in genetic programming genomes to tree based GP.
As in linear GP, repetitive patterns are present in large numbers. Size fair crossover limits bloat in automatic programming, preventing the evolution of recurring motifs. We examine these complex properties in detail: e.g. using depth v. size Catalan binary tree shape plots, subgraph and subtree matching, information entropy, syntactic and semantic fitness correlations and diffuse introns. We relate this emergent phenomenon to considerations about building blocks in GP and how GP works.

1 Introduction

Repeated sequences are commonplace in natural genomes. Biologists have discovered a vast amount of repetition in the DNA of microbes, plants and animals [1]. In fact it is now known that less than 3% of a human genome consists of protein-coding genes whereas around 50% of it consist of repetitive sequences [2; 3]. Biologists have recently turned their attention toward these patterned sequences [4; 5; 6] because the huge percentage of it indicates that these sequences play a major role in hereditary biology. The question we are asking is whether this emergent phenomenon might also be present in artificial genomes used for genetic programming.

Our initial search turned up repetitive sequences in linear GP genomes [7]. Here we turn to tree GP genomes. We find there are indeed, small and large repeated patterns in large trees which have been evolved by genetic programming. It can be observed that evolved trees are incrementally constructed from high fitness subtrees which are, however, not classic GP building blocks. Instead diffuse introns ensure that most code is robust to change.

We suggest that observations of this type can shed some new light on the old question of building blocks in GP [8]. Do they exist? If so, how does GP use them? If they do not, how does genetic search succeed?

Our route in this paper is roundabout: we start by following up on our work which suggests repeated patterns are prevalent in linear genetic programming [7] but now look at tree based GP. We use our time series modelling and Bioinformatics classification test problems (described in Section 2 and [7]) to show that, despite high mutation rates, multiple large syntactic and semantic repeated

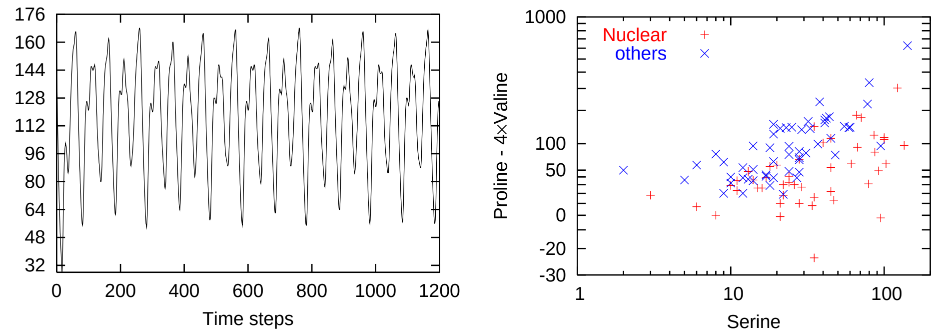


Fig. 1. Left: Mackey-Glass chaotic time series `http://neural.cs.nthu.edu.tw/jang/benchmark/`, $\tau = 17$. Right: Number of amino acids in nuclear and non-nuclear proteins. To reduce clutter only 5% of the proteins are plotted. The 3 (of 20) amino acids and function where suggested by sensitivity analysis of the smallest GP model.

patterns can occur in standard subtree crossover as well (Section 3). We deepen this analysis in Section 4: where we measure tree shape, entropy, sub-fitness and sensitivity within trees. This will lead us back to suggest (Section 5) at least in some simple modelling and prediction applications: 1) "introns" are somewhat diffused rather than discrete subtrees with a well defined root node that immediately nullifies their effect and 2) GP incrementally assembles solutions from large fit components, which are somewhat different from the classic "building block". Section 6 concludes.

2 Demonstration Problems

We have chosen two moderately difficult benchmark problems to represent typical modelling and prediction applications of genetic programming. Both were originally used as machine learning benchmarks. The Mackey-Glass chaotic time series has been used to demonstrate scientific, medical and financial modelling, e.g. [9]. The GP system is given historical data from which to predict a next value. We used the IEEE benchmark discretised into 8 bit unsigned integers, see Figure 1, left. All 1201 data points were used for training.

The second benchmark is a binary classification bioinformatics problem. Reinhardt and Hubbard [10] have shown that amino acids in a protein can be used to predict its location in the cell. They trained neural networks to distinguish between seven cellular locations in animals and microbes. We restrict ourselves to localising animal proteins (normally it is known if a protein is animal or bacterial) and a binary classification problem. To this end we evolve models which predict if an animal protein will be found in the cell nucleus or elsewhere. I.e. in the cell cytoplasm, in the mitochondria or outside the cell [10]. We used the same Swissprot data for 2427 proteins as used in [10]. There are 1097 nuclear (and 1330 non-nuclear) sequences of amino acids (see Figure 1, right). Data were split evenly into training and test sets.

Table 1. GP Parameters for Mackey-Glass time series prediction. (Parameters for protein localisation, where different, are given in brackets [proteins:]).

Function set:	MUL ADD DIV SUB operating on unsigned bytes [proteins: floats]
Terminal set:	Registers are initialised with historical values of time series. D128 128 time steps ago, D64 64, D32 32, D16 16, D8 8, D4 4, D2 2 and finally D1 with the previous value. Time points before the start of the series are set to zero. Constants 0..127. [Proteins: Number (integer) of each of the 20 amino acids in the protein. (Counts for code B were split evenly between aspartic acid D and asparagine N. Those for Z, between glutamic acid E and glutamine Q.) 100 unique constants randomly chosen from tangent distribution (50% between -10.0 and 10.0) [13]. (By chance none are integers.)]
Fitness:	RMS error [Proteins: $\frac{1}{2}$True Positive rate + $\frac{1}{2}$True Negative rate [14]]
Selection:	non elitist, tournament size 7. Pop Size 500 [proteins: 5000].
Initial pop:	ramped half-and-half (2:6) (50% of terminals are constants)
Parameters:	50% mutation (point 22.5%, constants 22.5%, shrink 2.5% subtree 2.5%). Max tree size 1000. Either 50% subtree crossover or 50% size fair crossover (90% on internal nodes), FXO fragments ≤ 30 [12]
Termination:	50 generations

3 Genetic Programming Configuration

Even though we expect crossover [11] to be responsible for repeated sequences, we follow recent GP practise and use a high mutation rate and a mixture of different mutation operators. In some runs, to avoid bloat, we also used size fair crossover (FXO) [12]. See Table 1.

Ten runs each with an initial population of 500, suggested this was too small for the protein localisation benchmark. There was a correlation (0.4 size fair and 0.2 two point (2XO) crossover) between the fitness of the best random tree and that of the best 50 generations later. So a population of 5000 and 50 generations was used. (The correlation co-efficient fell to 0.17 (FXO) and 0.12 (2XO) and mean holdout fitness rose 4% for both types of crossover.)

4 Results

4.1 Performance and Size of Mackey-Glass and Protein Programs

Table 2 summarises each of the ten runs with the two types of crossover on the Mackey-Glass modelling problem. As expected, size fair runs are both faster and evolve significantly smaller trees (Wilcoxon Two Sample Test p=0.007). Also as expected with standard GP, tree size increases up to the maximum size limit (1000) when evolution is continued to 500 generations. Figure 2 shows the fall in RMS error of the best individual in the population in each of the ten extended runs with standard crossover. It is the formation of repeated subtrees in these runs (and similar protein prediction runs) that we shall concentrate upon. While at first sight progress appears continuous, note that there are many generations

Table 2. Best Mackey-Glass prediction error after 50 generations of tree GP runs. Using size fair (FXO) and standard two point (2XO) crossover. Rows are RMS error and size of best of run tree and elapse time. Results after 500 generations (2XO only) show all runs improved fitness but trees increased enormously in size.

												Mean
FXO	error	4.42	4.38	4.85	4.89	4.01	4.92	3.84	4.65	3.66	4.80	4.44
	size	33	53	81	39	55	25	15	13	69	27	41
	secs	226	342	363	275	363	205	83	44	467	163	253
2XO	error	3.82	3.59	3.81	4.27	4.28	2.20	2.78	4.16	2.38	3.47	3.48
	size	59	45	143	117	47	87	91	43	123	145	90
	secs	617	384	610	416	412	503	543	269	967	645	537
2XO	error	3.74	1.51	1.18	3.66	3.41	1.09	2.78	3.78	1.08	1.85	2.41
500	size	793	705	669	957	963	883	847	923	957	467	816
gens	secs	13200	12200	11400	16100	11900	14500	11000	14300	22300	9500	13600

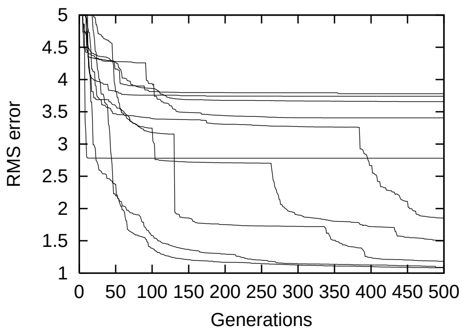


Fig. 2. Evolution of smallest RMS error in ten 2XO M-G runs. Despite size and shape changing from one generation to the next, for many successive generations the best fitness is identical to that in the previous generation. (Initial fitness, not shown, of the ten runs varied from 5.5 to 18.3.)

where the best fitness is identical to that in the previous generation even though the best individual in the population has been replaced (by crossover/mutation).

Table 3 summarises the ten runs on the protein prediction problem with both types of crossover. Again size fair crossover produces small trees more quickly than standard GP. As with Mackey-Glass both tree GP approaches produce models with a similar performance to linear GP [7]. That is GP is comparable to the best neural network approaches given in [10].

To confirm our previous results on the evolution of tree shapes [15; 16] also hold on the two benchmarks, Figure 3 plots the size (total number of nodes) and (maximum) depth of trees at every 10 (left) or 100 (right) generations during each of the 2×ten standard GP runs. The cross hairs give the population mean and standard deviation. As expected, the GP runs do not converge, instead the populations contain trees of different sizes and depths. Figure 3 is plotted on top of statistics relating not to GP but to the underlying distribution of binary trees (labelled “full”, “5%”, “peak”, “95%” and “minimal”) [16]. Cf. the Catalan distribution of subtree sizes [17, p241–242]. While initial populations contain only small trees, Figure 3 shows they evolve into populations of trees whose shape lies near that of the most popular trees in the underlying distribution. Note Figure 3 shows: in radically different problems, similar shaped trees evolve.

Table 3. Holdout set fitness on Bioinformatics benchmark. (Fitness is mean accuracy over nuclear and non-nuclear animal proteins.) 10 tree GP runs with size fair (FXO) and 10 with standard two point (2XO) crossover with a population of 5000 and 50 gens. As with Mackey-Glass, size fair runs are both faster and evolve smaller trees.

												Mean
FXO	percent	80	82	81	79	82	78	82	80	79	80	80
	size	57	77	43	47	69	77	85	59	53	41	61
	secs	1400	2300	1300	1200	2100	1700	1600	1700	1400	1400	1600
2XO	percent	81	82	80	82	83	82	83	83	82	81	82
	size	571	349	223	711	843	283	435	195	515	147	427
	secs	6100	5600	4200	6500	9600	4100	4500	4200	4800	3900	5400

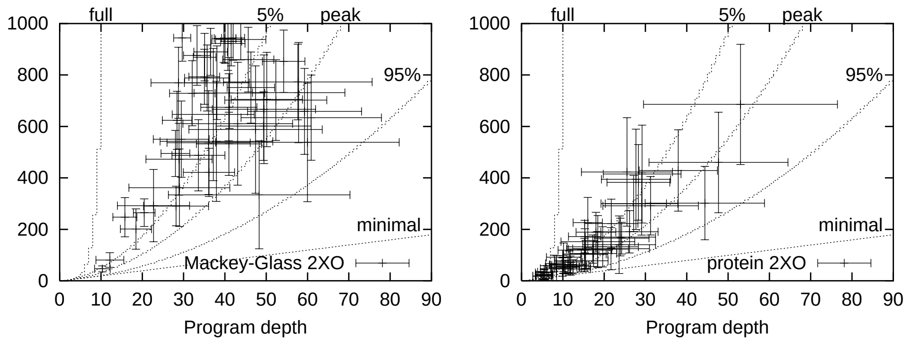


Fig. 3. Evolution of mean depth and size with mutation and standard crossover (2XO). 10 Mackey-Glass (left) and 10 protein runs (right). To reduce clutter standard deviations are only plotted every 100 generations (10 right). As expected [15], size increases until largest in population reach limit (1000) and much of the populations lie near the peak in the distribution of tree shapes.

4.2 Shape of Subtrees

The previous section has established that standard GP finds good models on both problems and programs' size and shape evolves as expected. This section starts to consider what is happening inside the trees. Figure 4 uses the same size-depth plots as Figure 3 to look at the evolved programs. Instead of one point per tree, there is a point for each node in each of the best trees in the last population of each run. Mostly subtrees lie between the 5% and 95% lines. This indicates that subtrees within the best program at the end of the runs have distributions of size and shape similar to that of the whole trees in previous generations. I.e. there is a strong tendency for trees to be composed of subtrees which are also randomly shaped. This fractal self similarity would be expected of random trees.

4.3 Repeated Code Fragments

In all cases using standard crossover (2XO), GP evolved best of run trees containing large repeated patterns. As with linear GP, this happens despite a high level of mutation and a size limit. Figure 5 shows the identical repeated patterns

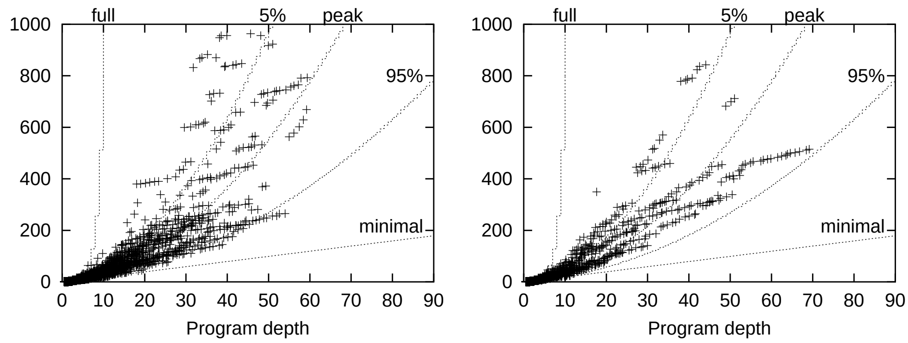


Fig. 4. Depth and size of every subtree in best of run trees (2XO). 10 Mackey-Glass (left) and 10 protein runs (right). Note the similarity with the shape of whole trees as they evolved, Figure 3. (Small amount of noise added to spread data that would otherwise be plotted directly on top of each other.)

(allowing overlaps) for one evolved program. Between 56% and 91% (mean 71%) of the ten best of run Mackey-Glass (2XO) models are part of repeated subgraphs which are too big to have formed by chance. The figures for the ten best of run protein prediction programs are: 33%–92%, mean 74%. See Figure 7. The replications in Figures 5 and 7 refer to any fragment of the whole tree, while the rest of Section 4 considers only whole non-overlapping subtrees.

4.4 Syntactically Repeated Subtrees

Figure 6 shows the location and size of exactly repeated subtrees in the largest of the protein prediction trees. Figure 8 refers to the same twenty best of run programs as Figure 7, however it considers only exactly repeated subtrees (rather than any fragments). The requirement to include all the leafs in a repeated fragment tends to reduce their size but we see a similar picture: in every run repeated subtrees (too large to be due to chance) are evolved.

4.5 Semantically Repeated Subtree Outputs

The previous sections have only considered repeated code at the syntax level. Now we consider the semantics of the evolved programs. Since there are no side-effects, repeated subtrees must return exactly the same values. Figure 9 shows on the training examples the fraction of the program where the semantic value of the subtrees are the same, and where they are highly correlated. It shows semantic repetition is even higher than when just considering program syntax. Part of the difference is due to constants (which are always correlated with each other). However this is not enough to explain all of the difference, suggesting non-trivial syntactically different subtrees have been evolved which produce correlated answers. Part of the explanation may be symmetries, such as $+$ and $\times$, whereby non-identical code calculates identical answers. Alternatively the monolithic fitness function may encourage redundant code.

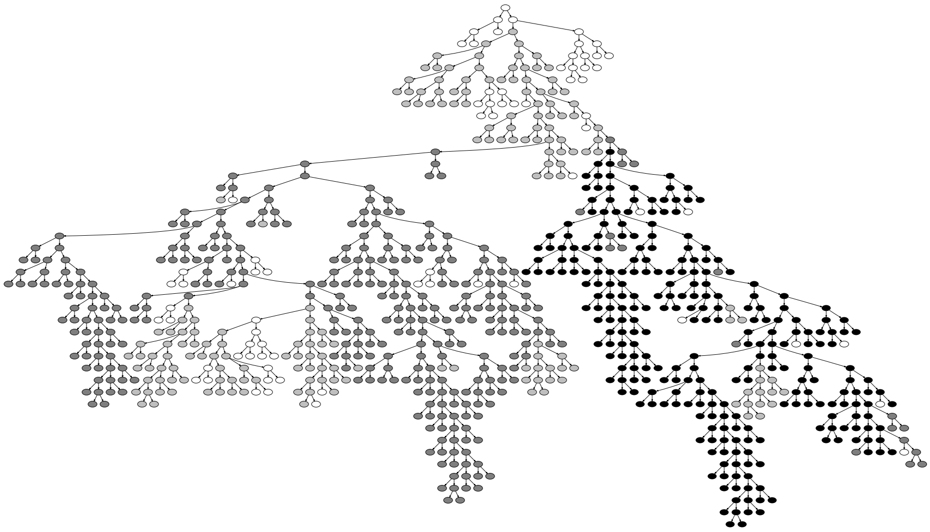


Fig. 5. Repeated patterns in the largest protein prediction program (2XO, 843 nodes). largest pattern (133 nodes) in black. Other nodes in repeated patterns are filled according to size of the repeated pattern (33–132 grey and 11–32 light grey). Unique nodes and nodes which are part of small patterns are not filled.

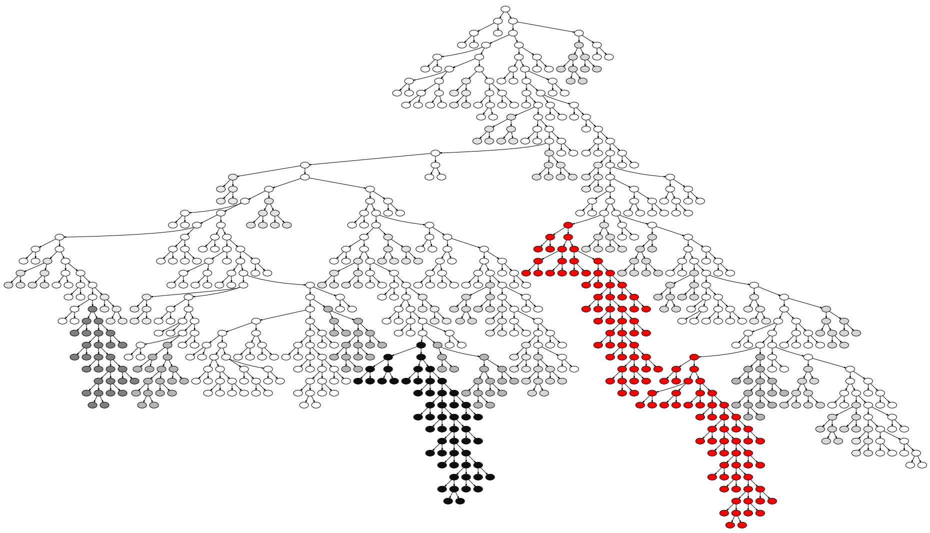


Fig. 6. Same program as in Figure 5. Here whole subtrees are exactly repeated. Nodes are filled according to size of the repeated subtree. Unique nodes and nodes which are part of small patterns (3 nodes or less) are not filled. Two largest (59 nodes, right hand side) coloured red. Note these are partially repeated elsewhere in the tree (e.g. 55 node subtree shaded black).

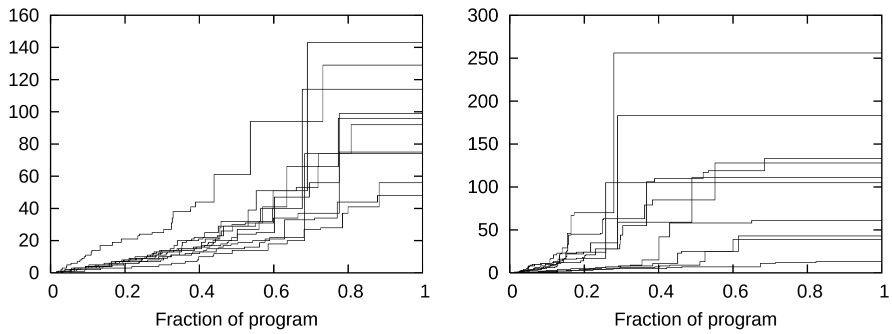


Fig. 7. Size of repeated pattern v. fraction of best of run trees (2XO). 10 Mackey-Glass (500 gens, left) and 10 protein runs (50 gens, right). In every run the largest repeated pattern is too big to arise by chance.

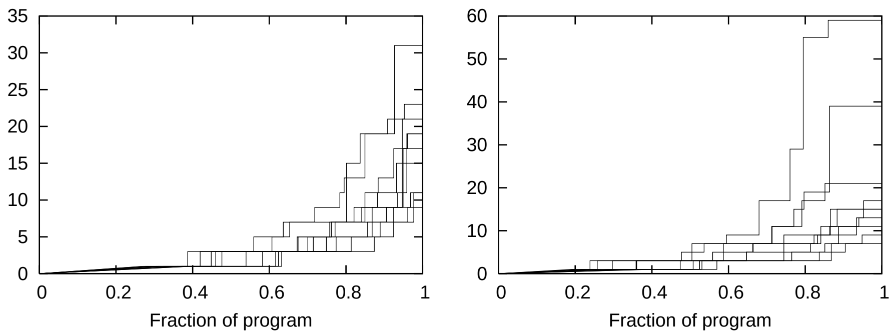
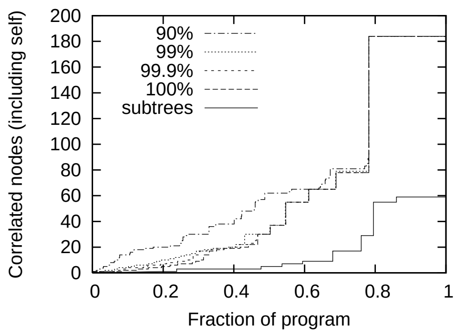


Fig. 8. Size of identical subtrees v. fraction of best of run trees (2XO). 10 Mackey-Glass (500 gens, left) and 10 protein runs (50 gens, right). In every run the largest repeated subtree is too big to arise by chance.

Fig. 9. Upper curves show number of highly correlated subtrees v. fraction of the best largest protein prediction tree (2XO run 4, cf. Figures 5, 6 and 10–12). For comparison, the lower (solid) curve refers to syntactic repeats (rather than semantic). It is the top curve from Figure 8 (right). Many subtrees (22%) produce a constant. This gives rise to the sudden jump at 0.78 but only explains part of the difference between syntactic and semantic repeats.



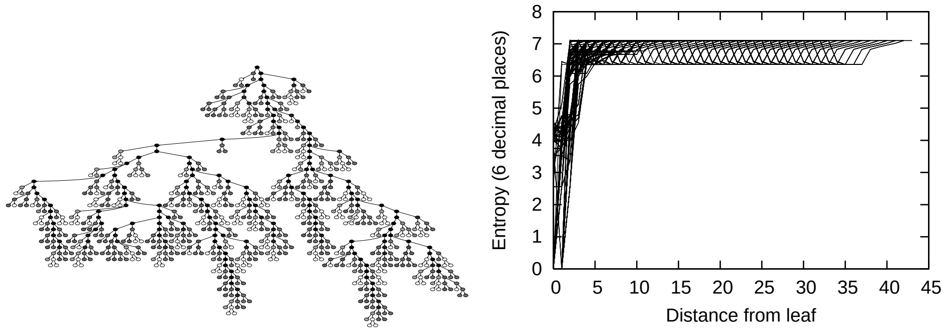


Fig. 10. Left: Entropy of each node in largest protein program (cf. Figures 5, 6 and 9–12.) Darker grey indicates more variation across the training set. At levels 7 and 9 there are two links where large subtrees pass through bottle necks. **Right:** Entropy on each of 422 paths from leaf to root. The bottle necks near the root show up as repeated dips in the tail. This structure is an artifact caused by paths passing through similar routes near the root but having different lengths.

4.6 Entropy of Subtrees

As might be expected, variation in values calculated by subtrees across the training set has a strong tendency to increase from the leafs to the root. This is also true of random programs. Figure 10 shows the variability within the largest protein location tree (2XO, 50 generations). We use information entropy [18] (calculated using signal value to 6 decimal places) as our measure of variation.

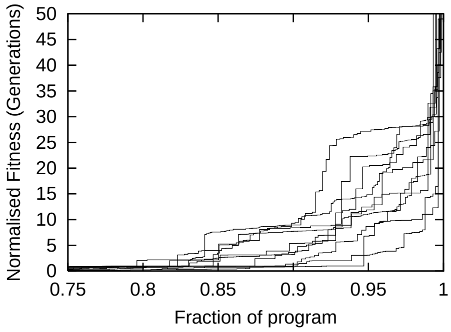
The protein location programs do not contain "classic" intron nodes. I.e. there are few places deep in the tree where information passes only from one input of a function to its output, totally ignoring the other input. The entropy, if any, of "classic" intron nodes would come from just one input. Thus the entropy of an "all or nothing" intron would be the same as that of its active argument.

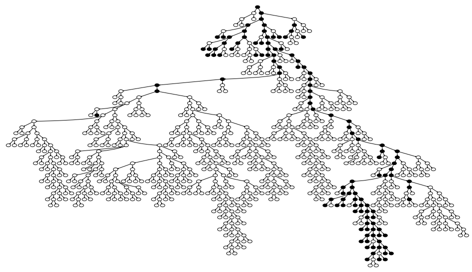
Sometimes entropy (i.e. variability) falls from the leaf towards the root are caused by a SUB subtree with both arguments referring to the same amino acid. This has no variation since it always yields zero, so the subtree has less entropy than either of its leafs. (Random programs also contain bottleneck nodes of low entropy.) Most cases where entropy falls are very close to a leaf. However a few of the largest protein location (2XO) programs do possess bottlenecks where entropy falls on the output of a large subtree. This means the subtree has less effect on the whole program.

4.7 Fitness of Subtrees

As might be expected, correlation or anti-correlation with training data tends to rise from the leafs to the root. Between 15 and 78 (depending on the run) subtrees in each best of run program exceed the performance of random search (10^6 ramped half-and-half trees). See Figure 11. Since fitness tends to fall away

Fig. 11. High fitness (or anti-fitness) subtrees as a fraction of the 10 best protein trees (2XO). Note range of horizontal axis. Since fitness is a very non-linear function, we define a normalised fitness as being, for each run, the generation in which a program of the corresponding fitness was first found. All runs exceeded the best fitness found in a million random trees programs by generation 8.





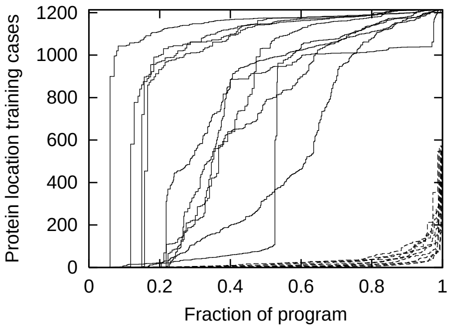


Fig. 12. Importance of nodes within protein prediction trees. **Left:** Largest protein prediction tree. The 125 (15%) subtrees which change more than 10 training cases are highlighted in black. (Same example as in Figures 5, 6–10.) Note several large repeated subtrees do not contribute to fitness. **Right:** Number of training cases which subtrees influences as a fraction of the 10 2XO best of run programs. Solid curves plot where impact is more than 0.005%. Dashed lines: node causes prediction to change.

from the root, there are more lower fitness subtrees. Secondly, despite being non-elitist, fitness increases monotonically. Therefore the fitness distribution within the best subtrees can also be explained by saying: the longer evolution has had to work since a fitness level was reached the larger the number of subtrees exceeding that fitness there will be.

4.8 Importance of Subtrees (Sensitivity analysis)

While the trees do not contain “classic introns”, where one argument of a function has no impact on its output, some nodes do have much more impact than others. To see this, we replaced each subtree in turn by its median value and counted the number of training cases where this changed the output. The upper solid curves on the right of Figure 12 plot the number of fitness cases where the output was changed by more than 0.005%. While the lower dashed curves show the number of cases where subtrees contribute to fitness, i.e. the number of training cases where replacing it changed the program’s prediction. Between 5%

and 23% of nodes in protein prediction programs have less than 0.005% impact on all training cases. If we consider just fitness (lower dashed curves) this rises to between 7% and 57% of the program. I.e. on average 30% of subtrees can be replaced without changing any of the program's predictions.

5 Discussion

Sections 4.1 and 4.2 confirm (cf. [15]) trees have evolved to the same fractal shape as random trees but Sections 4.3–4.5 show repeated syntactic and semantic patterns which are far from random. Sections 4.6 and 4.7 suggest GP programs (with non-Boolean function sets without side effects) are composed of high fitness subtrees which mostly pass information upwards towards the root. That is, they are not dominated by classic "introns" (which ignore data from one or more subtrees). However the sensitivity analysis (Section 4.8) shows large parts of the tree, including repeated parts, can be replaced by a constant and have no or little effect on fitness.

We suggest the repeated patterns seen in GP used for modelling and prediction are not like classic GA "building blocks" [8]. They are not small. They have high fitness on the whole problem, rather than sub-components of it. It appears evolution is gradually, haphazardly, assembling a complete program by repeatedly reusing subtrees it has already discovered in ways allowing it to squeeze out marginal incremental improvements. In the process some components become of lesser importance in the final program.

6 Conclusions

Correlation between performance of initial and evolved populations suggests lack lustre initial random programs can have an impact on the final outcome. Correlation might be a useful population size analysis tool.

As expected, size fair crossover (FXO) [12] and a range of mutation operators controlled bloat [15]. In these experiments, the compact models were slightly worse than the much larger ones evolved with standard crossover and mutation.

Entropy and subtree fitness analysis suggest genetic programming (GP) succeeds in finding ways to put together moderately sized fit subtrees to yield larger trees containing few highly sensitive components with higher performance.

While it is always difficult to generalise from a limited number of examples, we have seen for two diverse non-trivial problems the spontaneous emergence of repeated patterns in both linear and tree based GP and with a variety genetic operations. This leads use to tentatively suggest on problems, without tight limits on tree size, depth, etc., where bloat is possible, GP will generally evolve programs containing copious repeated patterns. Although this work is far from complete, we suggest future analysis may: discover further spontaneous effects which arise from evolution rather than the programmer, cast light on the workings of GP and may lead to new automatic programming techniques.

Acknowledgements

This work was carried out at University College, London. WB acknowledges support from an NSERC discovery grant under RGPIN 283304-04.

Source Code

Code to generate Graphviz format dot files from GP programs can be found at `http://www.cs.ucl.ac.uk/staff/W.Langdon/lisp2dot.html`.

References

1. Britten, R.J., Kohnen, D.E.: Repeated sequences in DNA. Science **161** (1968) 529–540
2. Smit, A.F.A.: The origin of interspersed repeats in the human genome. Current Opinions in Genetics and Development **6** (1996) 743–748
3. Patience, C., Wilkinson, D.A., Weiss, R.A.: Our retroviral heritage. Trends in Genetics **13** (1997) 116–120
4. Lupski, J.R., Weinstock, G.M.: Short, interspersed repetitive DNA sequences in procaryotic genomes. Journal of Bacteriology **174** (1992) 4525–4529
5. Toth, G., Gaspari, Z., Jurka, J.: Microsatellites in different eukaryotic genomes: Survey and analysis. Genome Research **10** (2000) 967–981
6. Achaz, G., Rocha, E.P.C., Netter, P., Coissac, E.: Origin and fate of repeats in bacteria. Nucleic Acids Research **30** (2002) 2987–2994
7. Langdon, W.B., Banzhaf, W.: Repeated sequences in linear genetic programming genomes. Complex Systems (2005). In press.
8. O'Reilly, U.M., Oppacher, F.: The troubling aspects of a building block hypothesis for genetic programming. In Whitley, L.D., Vose, M.D., eds.: Foundations of Genetic Algorithms 3, Morgan Kaufmann (1995) 73–88
9. Oakley, H.: Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. In Kinnear, Jr., K.E., ed.: Advances in Genetic Programming. MIT Press (1994) 369–389
10. Reinhardt, A., Hubbard, T.: Using neural networks for prediction of the subcellular location of proteins. Nucleic Acids Research **26**(9) (1998) 2230–2236
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
12. Langdon, W.B.: Size fair and homologous tree genetic programming crossovers. Genetic Programming and Evolvable Machines **1**(1/2) (2000) 95–119
13. Langdon, W.B.: Genetic Programming and Data Structures. Kluwer (1998)
14. Langdon, W.B., Barrett, S.J.: Genetic programming in data mining for drug discovery. In Ghosh, A., Jain, L.C., eds.: Evolutionary Computing in Data Mining. Volume 163 of Studies in Fuzziness and Soft Computing. Springer (2004) 211–235
15. Langdon, W.B., Soule, T., Poli, R., Foster, J.A.: The evolution of size and shape. In Spector, L. *et at.*, eds.: Advances in GP 3. MIT Press (1999) 163–190
16. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer (2002)
17. Sedgewick, R., Flajolet, P.: An Introduction to the Analysis of Algorithms. Addison-Wesley (1996)
18. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. The University of Illinois Press, Urbana (1964)