**Natural Computing (2008) 7(4) 589–613 doi:10.1007/s11047-007-9038-8 pre-print**

# Repeated Patterns in Genetic Programming

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

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

**Abstract.** Evolved genetic programming trees contain many repeated code fragments. Size fair crossover limits bloat in automatic programming, preventing the evolution of recurring motifs. We examine these complex properties in detail using depth v. size Catalan binary tree shape plots, subgraph and subtree matching, information entropy, sensitivity analysis, syntactic and semantic fitness correlations. Programs evolve in a self-similar fashion, akin to fractal random trees, with diffuse introns. Data mining frequent patterns reveals that as software is progressively improved a large proportion of it is exactly repeated subtrees as well as exactly repeated subgraphs. We relate this emergent phenomenon to building blocks in GP and suggest GP works by jumbling subtrees which already have high fitness on the whole problem to give incremental improvements and create complete solutions with multiple identical components of different importance.

**Keywords** Genetic alogorithms · ALU · SINE · Frequent subgraphs · Frequent subtrees · Macky-Glass · Poly-10 · Nuclear protein localisation · Tiny GP · GPquick · Evolution of program shape · Sensitivity analysis

## 1 Introduction

Genome Biology is full of surprising findings that need explanation. One of these was the discovery of repeated sequences of nucleotides in genomes which would sometimes stretch for millions of basepairs. Upon closer inspection it was found, that repeated sequences are commonplace in natural genomes. A vast amount of repetition in the DNA of microbes, plants and animals has been discovered [1]. For instance, 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. We ask whether this emergent phenomenon might also be present in artificial genomes used for genetic programming [7,8,9].

Initially, our search was conducted in genomes similar to natural genomes. We found multiple repetitive sequences in those linear GP genomes [10]. More recently, we have turned to tree GP genomes [11]. We find that there are indeed small and large repeated patterns in large trees once evolution has worked for a sufficiently long time. Evolved trees are incrementally constructed from high fitness subtrees. These subtrees 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 [12]. Do they really exist? If so, how does GP make use of them? If they do not, how does genetic search succeed? In a nutshell, the concept of building blocks suggests that solutions to a problem are assembled from highly fit smaller subsolutions.

We start by following up on our work which suggests repeated patterns are prevalent in linear genetic programming [10] by evolving solutions within two different tree based GP systems. We will use our time series modelling and Bioinformatics classification test problems and we will also use a recent benchmark (Poly-10 [13]).

The real world problems and benchmark are described in Section 2 and [10]. Analysing the evolved programs shows that, despite high mutation rates, multiple large repeated patterns can occur in standard subtree crossover as well as linear GP (Section 3). We deepen this analysis in Section 4 by measuring 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. Instead, as information passes up through the tree towards the root node (where it determines the program's output) it is progressively diluted. I.e. there is no single node in the tree which completely disables the code beneath it. 2) GP incrementally assembles solutions from large fit components. The components are self similar and to a large degree different from the classic "building block". Section 6 concludes.

## 2 Demonstration Problems

We have chosen three moderately difficult benchmark problems to represent typical modelling and prediction applications of genetic programming. The first two were originally used as machine learning benchmarks, whilst the third has been used by several authors in recent GP work. The Mackey-Glass chaotic time series has been used to demonstrate scientific, medical and financial modelling, e.g. [14]. 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. All 1201 sample data points were used for training.

The second benchmark is a binary classification bioinformatics problem. Reinhardt and Hubbard [15] 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

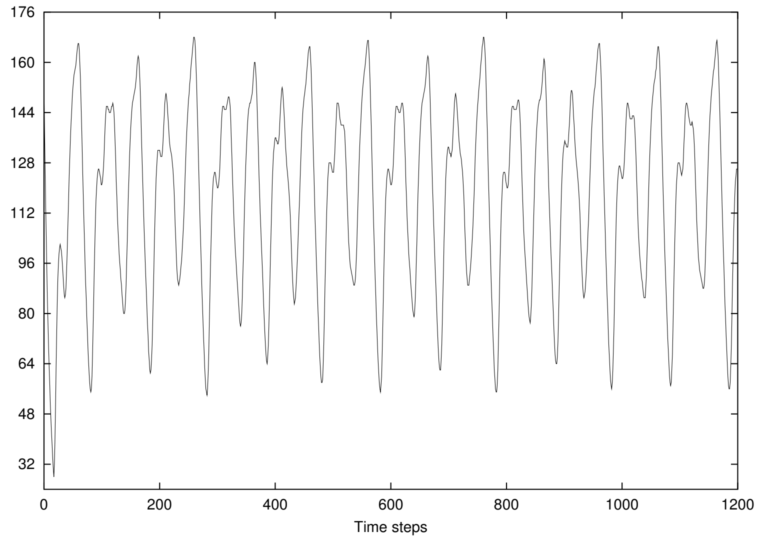

**Fig. 1.** Discrete Mackey-Glass chaotic time series `http://neural.cs.nthu.edu.tw/jang/benchmark/`, $\tau = 17$. 1201 data points, sampled every 0.1.

is animal or bacterial) and to 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 [15]. We used the same Swissprot data for 2427 proteins as used in [15]. There are 1097 nuclear (and 1330 non-nuclear) sequences of amino acids (see Figure 2). Data were split evenly into training and test sets.

The last benchmark is a symbolic regression of a multivariate cubic polynomial, Poly-10. Poly-10, is $f(x_1, \cdots, x_{10}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$. The 50 fitness cases are obtained by selecting uniformly at random each of the ten inputs from the range $[-1, +1]$.

## 3 Genetic Programming Configuration

Even though we expect crossover [7] to be responsible for repeated patterns, 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) [16]. See Tables 1 and 2. (Briefly size fair crossover is like normal subtree crossover except, after the crossover point in the first parent tree has been chosen randomly, the crossover point in the second parent is chosen so that the size of the exchanged subtree is more-or-less the same as the size of the subtree to be deleted.) To further demonstrate that repeated patterns may appear in a wide range of circumstances we also use a totally different tree GP system, tinyGP [17], on the Poly-10 benchmark.

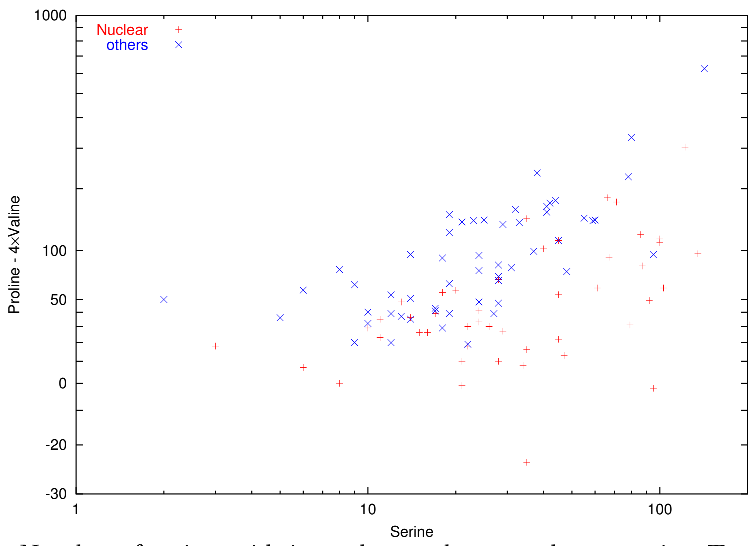

**Fig. 2.** Number of amino acids in nuclear and non-nuclear proteins. To reduce clutter only 5% of the proteins are plotted. The 3 (of 20) amino acids and function where suggested by sensitivity analysis of the smallest GP model.

TinyGP is a steady state GA [18], often used with a high point mutation rate (to combat bloat). Because we cannot expect to see large repeats in small (i.e. non bloated) programs, we run tinyGP for many more generations than are commonly used. This allows evolution more time to expand the trees, see Table 3.

Ten runs, each with an initial population of 500 individuals, 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. As a result, the correlation co-efficient fell to 0.17 (FXO) and 0.12 (2XO) and mean holdout fitness rose 4% for both types of crossover.

The best program in nine Poly-10 random initial populations is $x_1 \times x_2$. (This means all but one initial generation have identical fitness. Therefore the correlation across runs between initial fitness and final fitness, or anything else, is automatically near zero.) The lack of variation between runs of best initial populations, the poor performance and [21] all suggest that the already large population would have to be increased by one or more orders of magnitude to solve Poly-10. Nevertheless we can learn from unsuccessful runs.

[1] In GPquick the protected division operator DIV is defined as DIV($x$,$y$) = $(y = 0)?\ 1 : x/y;$

[2] In tinyGP the protected division operator DIV is defined as DIV($x$,$y$) = $(|y| \leq 0.001)?\ x : x/y;$

**Table 1.** GPquick (C++) Parameters for Mackey-Glass time series prediction.

| | |
|---|---|
| Function set: | MUL ADD DIV[1] SUB operating on unsigned bytes |
| 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. |
| Fitness: | RMS error |
| Selection: | generational, non elitist, tournament size 7. Population size 500. |
| 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 $\leq 30$ [16] |
| Termination: | 50 (500) generations |

**Table 2.** GPquick Parameters for protein localisation.

| | |
|---|---|
| Function set: | MUL ADD DIV[1] SUB operating on floats |
| Terminal set: | Number (integer) of each of the 20 amino acids in the protein. 100 unique constants randomly chosen from tangent distribution (50% between -10.0 and 10.0) [19]. (By chance none are integers.) |
| Fitness: | $\frac{1}{2}$True Positive rate $+ \frac{1}{2}$True Negative rate [20] |
| Selection: | generational, non elitist, tournament size 7. Population size 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 $\leq 30$ [16] |
| Termination: | 50 generations |

**Table 3.** TinyGP (Java) Parameters for Poly-10

| | |
|---|---|
| Function set: | MUL ADD DIV[2] SUB operating on doubles. |
| Terminal set: | Ten inputs $x_1 \ldots x_{10}$. Training values selected at random from $-1 \ldots +1$. No constants. |
| Fitness: | Sum over 50 training examples of absolute difference between GP value and target value. |
| Selection: | Steady state (binary tournaments used both for selecting parents and for selecting who is replaced), non elitist, population size 10 000. |
| Initial pop: | created by random recursive growth (depth 2:6). |
| Parameters: | 10% subtree crossover (crossover points chosen uniformly in both parents to give a single child). 90% point mutation (rate 0.02 per node). No limit on tree size. |
| Termination: | 500 generations |

## 4 Results

### 4.1 Performance and Size of Mackey-Glass, Protein and Poly-10 Programs

Table 4 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 3 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 and Poly-10 runs that we shall concentrate upon. While at first sight progress appears continuous, note that there are many generations 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 4.** Best Mackey-Glass prediction error after 50 and 500 generations using size fair (FXO) and standard two point (2XO) crossover. Rows are RMS error and size of best of run tree and elapsed 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 |

Table 5 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 [10]. GP is comparable to the best neural network approaches given in [15]. Figures 4 and 5 show the evolution of fitness in ten 2XO runs (performance in the size fair runs evolves similarly).

For the Poly-10 symbolic regression problem, we again made ten independent runs. The accuracy and size of the best individual in the last generation of each run is reported in Table 6. Poly-10 is known to be a very hard problem

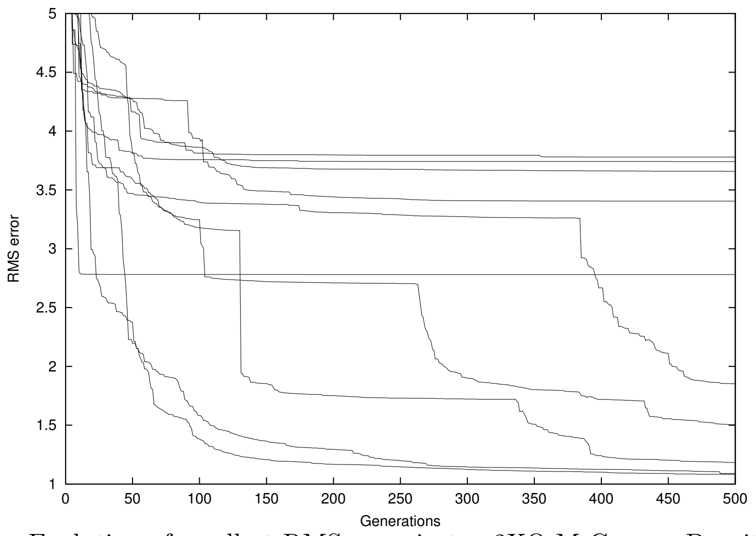

**Fig. 3.** Evolution of smallest RMS error in ten 2XO M-G runs. Despite size and shape changing from one generation to the next, for many successive generations the best fitness is identical to that in the previous generation. (Initial fitness, not shown, of the ten runs varied from an RMS error of 5.5 to 18.3.)

**Table 5.** Holdout set fitness on Bioinformatics benchmark. (Fitness is mean accuracy over nuclear and non-nuclear animal proteins.) 10 tree GP runs with size fair crossover (FXO) and 10 with standard two point crossover (2XO) using a population of 5000 and 50 generations. 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 |

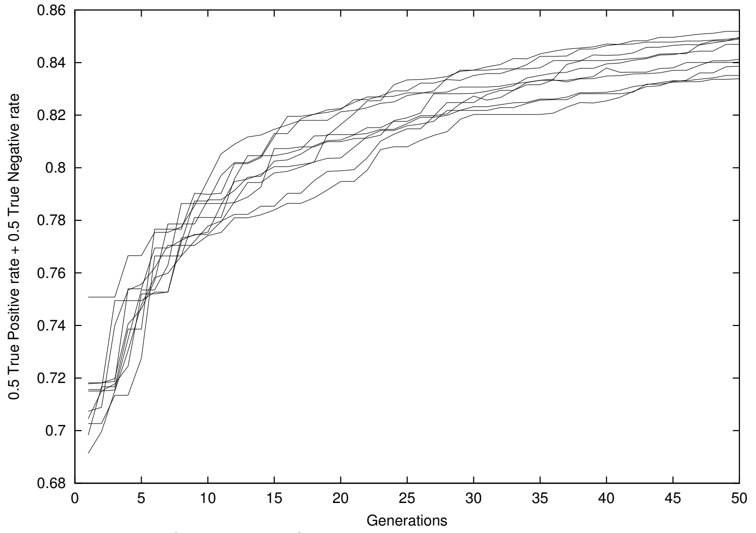

**Fig. 4.** Evolution of training fitness in ten Nuclear v. non-Nuclear protein classification runs with normal crossover (2XO). Note change in horizontal scale compared to Figure 3. Despite size and shape changing from one generation to the next, in 25% of generations the best training fitness is identical to that in the previous generation.

and so it is no surprise that a population of 10 000 is not sufficient. Even after 500 generations, no run solved the problem. With our selection of training data, $x_1 \times x_2$ is a strong attractor. In 9 of the 10 runs it is the best of the initial random programs. While in six of nine cases the population escapes from it within four generations, in the remaining three it remains the best until the end of the run.

**Table 6.** Best Poly-10 programs after 500 generations in ten tinyGP runs. Rows are absolute error summed over 50 training cases, size of best of run tree and elapsed time. Note runs 3, 5 and 10 collapse to the three node program $x_1 \times x_2$.

| | | | | | | | | | | | | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2XO | percent | 6.53 | 7.82 | 15.83 | 4.62 | 15.83 | 4.07 | 8.10 | 5.33 | 7.85 | 15.83 | 9.18 |
| | size | 155 | 6577 | 3 | 461 | 3 | 635 | 2419 | 1321 | 181 | 3 | 1176 |
| | secs | 1100 | 12100 | 200 | 1400 | 200 | 1100 | 10000 | 5900 | 1100 | 200 | 3327 |

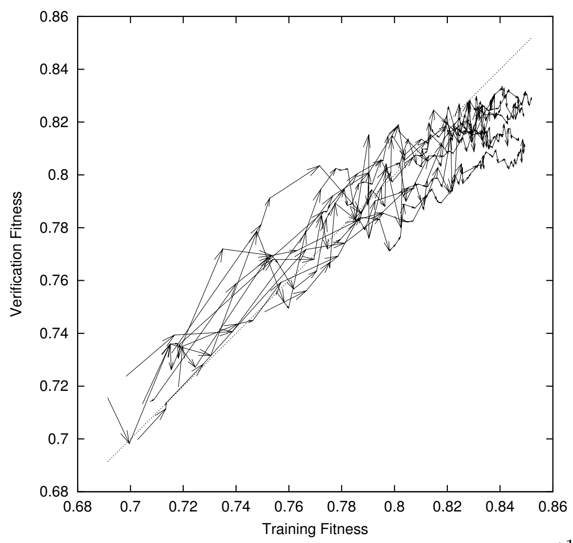

**Fig. 5.** Evolution of training v. generalisation fitness ($\frac{1}{2}$true positive rate+$\frac{1}{2}$true negative rate) in ten Nuclear v. non-Nuclear protein classification runs with normal crossover (2XO). (Same runs as Figure 4). Each arrow represents the change in performance of the best fitness individual in the population in one generation equivalent. Dotted diagonal line shows where training performance is identical to out of sample (generalisation) performance. Note performance at the end of the runs is below the diagonal, indicating overfitting. This is common in machine learning.

### 4.2 Evolution of Program Shape

To confirm our previous results on the evolution of tree shapes [22,9] on the three problems, Figures 6–8 plot the size (total number of nodes) and (maximum) depth of trees during the standard GP runs. The cross hairs give the population mean and standard deviation. As expected (except as noted in the previous paragraph for three Poly-10 runs) the GP runs do not converge, instead the populations contain trees of different sizes and depths. Figures 6–8 are plotted on top of statistics relating not to GP but to the underlying distribution of binary trees (labelled "full", "5%", "peak", "95%" and "minimal") [9]. Cf. the Catalan distribution of subtree sizes [23, p241–242]. While initial populations contain only small trees, Figures 6–8 show they evolve into populations of trees whose shape lies near that of the most popular trees in the underlying distribution. Note that Figures 6–8 show similar shaped trees evolve in radically different problems.

### 4.3 Shape of Subtrees

The previous section has established that standard GP finds good models on both real world problems and programs' size and shape evolves as expected. Solutions were not expected to Poly-10 but, except for three runs which get trapped by $x_1 \times x_2$, Poly-10 programs also evolve to have shapes near to those of random trees. This section starts to consider what is happening inside the trees. Figures 9–11 use the same size-depth plots as Figures 6–8 to look at the evolved programs. Instead of taking an average of the whole of the population, Figures 9–11 plot a point for each node within each of the best trees. Lines of crosses are caused by chains of nodes in a tree where one argument is a small subtree and the other continues the chain. Subtrees tend to lie between the 5% and 95% lines. I.e. 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. This means that there is a strong tendency for trees to be composed of subtrees which are also approximately randomly shaped. This fractal self similarity would be expected of random trees.

For comparison Figure 12 plots the size v. depth distribution for seven randomly created binary trees whose sizes are the same as the highly evolved Poly-10 trees plotted in Figure 11. While obviously different in detail, it is clear the subtrees within evolved trees (as plotted in Figures 9–11) show similar behaviour to those in random binary trees.

### 4.4 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 13 shows the identical repeated patterns (allowing overlaps) for one evolved program. Between 56% and 91%

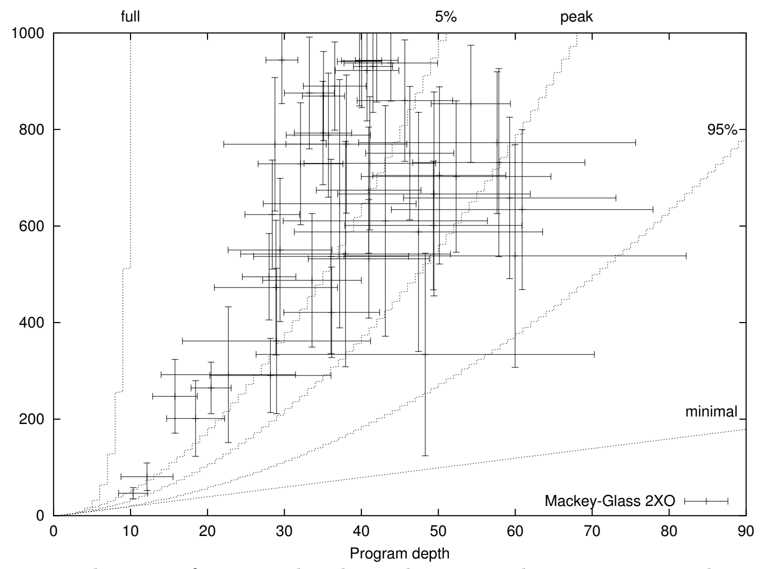

**Fig. 6.** Evolution of mean depth and size with mutation and standard crossover (2XO). 10 Mackey-Glass runs. To reduce clutter standard deviations are only plotted every 100 generations. As expected [22], size increases until largest in population reach limit (1000) and much of the populations lie near the peak in the distribution of tree shapes. Dotted lines indicate general features of binary trees. Moving left to right horizontally, i.e. constant tree size, "full" indicates shortest most balanced trees. The number of trees of a given size and depth increases up to the "peak". "5%" indicates 5% of trees, of the chosen size, lie between "full" and it. 95% lie to the left of the "95%" line. Finally the "minimal" line indicates trees without side branches, i.e. the deepest possible trees.

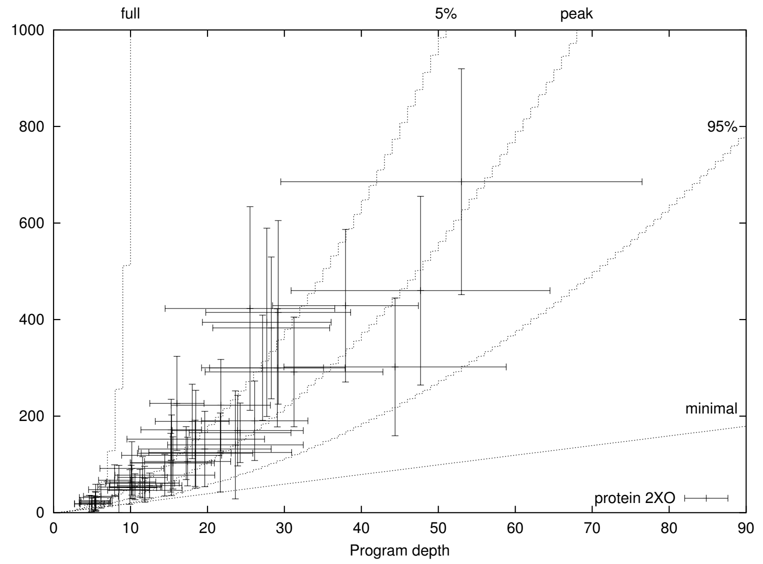

**Fig. 7.** Evolution of mean depth and size with mutation and standard crossover (2XO). 10 protein runs, standard deviations are only plotted every 10 generations. As with Figure 6, size increases until largest in population reach limit (1000) and much of the populations lie near the peak in the distribution of tree shapes.

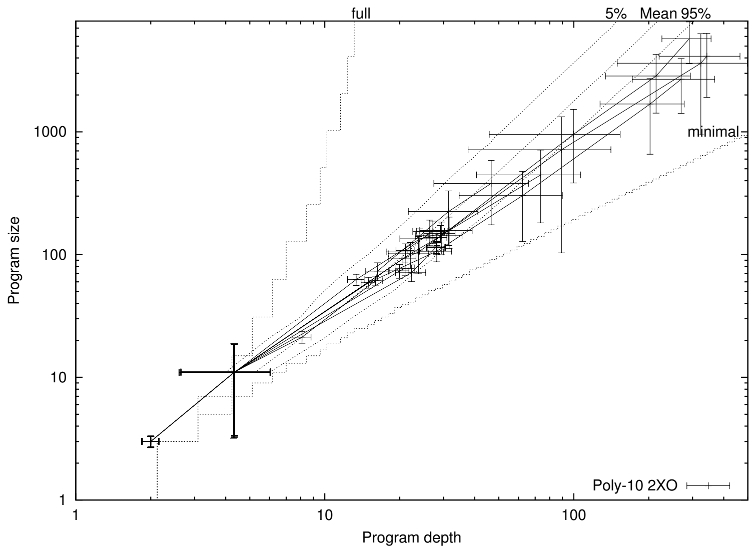

**Fig. 8.** Evolution of shape in ten Poly-10 populations. Standard deviations are plotted every 100 generations. Note, tinyGP does not impose size or depth limits, however as with Mackey-Glass and Protein prediction (cf. Figures 6 and 7) many individuals in the 7 bloated populations have shapes near those of random trees.

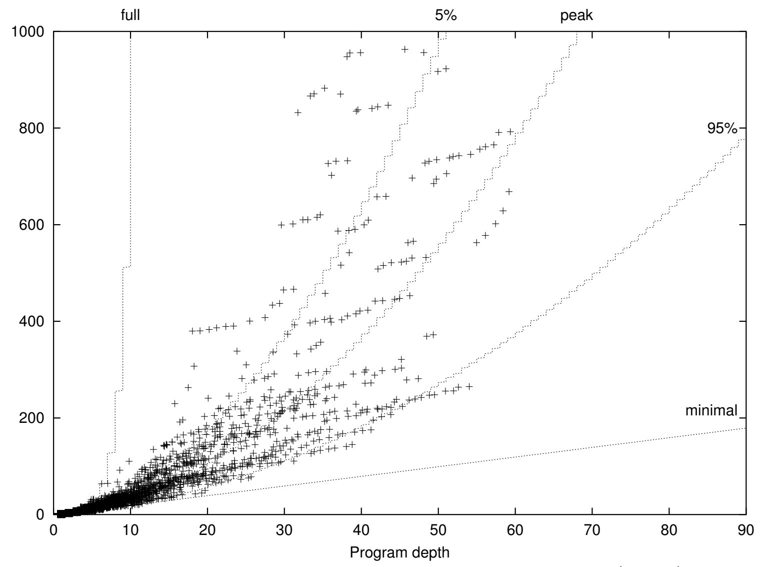

**Fig. 9.** Depth and size of every subtree in best of run trees (2XO). 10 Mackey-Glass runs. Note the similarity with the shape of whole trees as they evolved, see Figure 6. (Small amount of noise added to spread data that would otherwise be plotted directly on top of each other.)

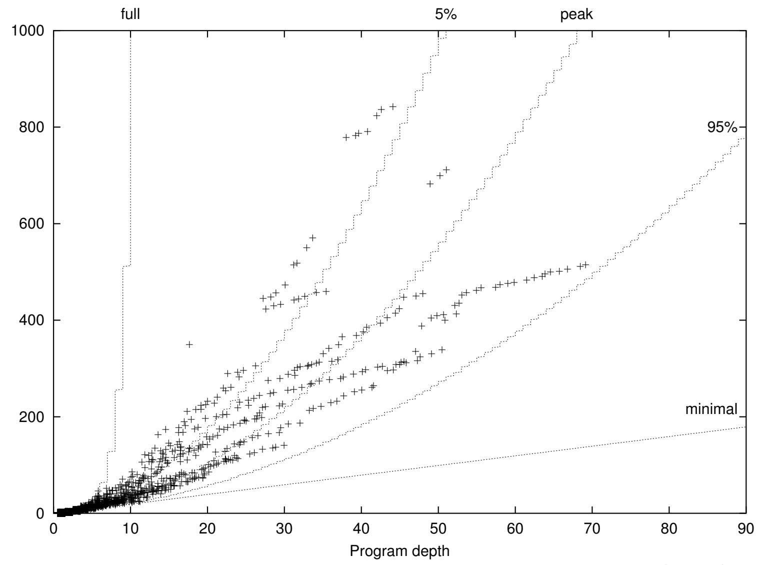

**Fig. 10.** Depth and size of every subtree in best of run trees (2XO). 10 protein runs. Note the similarity with the shape of whole trees as they evolved, Figure 7. (As Figure 9, small amount of noise added.)

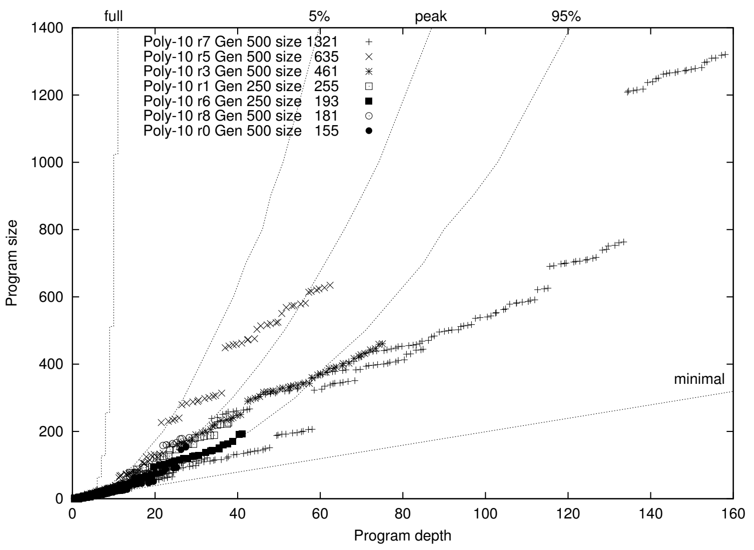

**Fig. 11.** Depth and size of every subtree in seven bloated Poly-10 trees. Due to extreme size of programs in runs 1 and 6 data for the best programs after 250 generations is presented for these two runs. Once again note the similarity with the shape of whole trees as they evolved, Figure 8. (Again a small amount of noise added to improve display.)

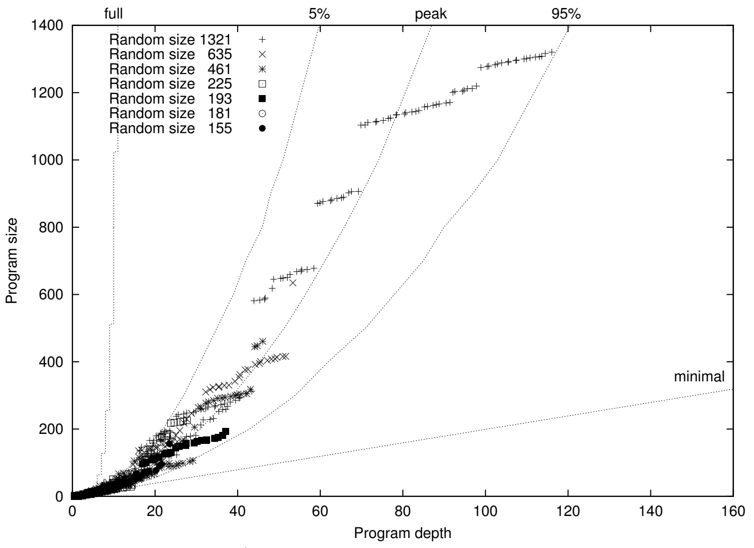

**Fig. 12.** Depth and size of every subtree in seven random binary trees. Size of trees chosen to be same as those in Figure 11. (Again a small amount of noise added to improve display.)

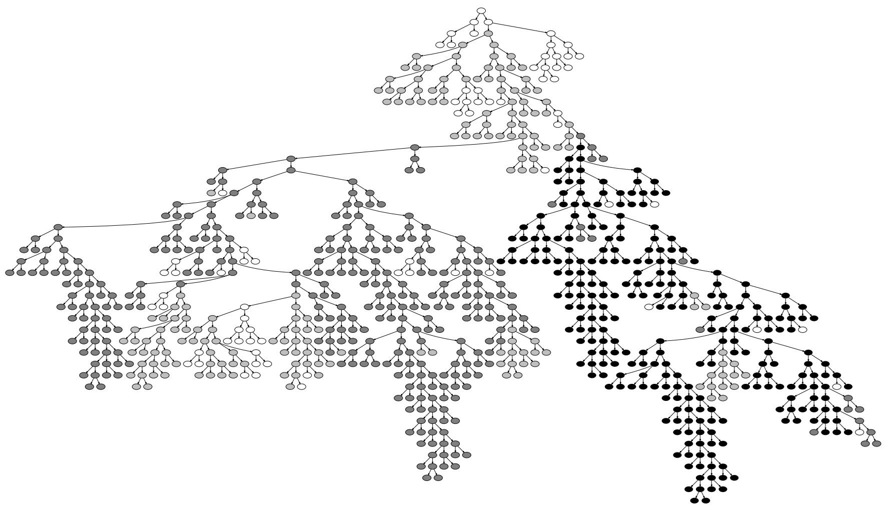

**Fig. 13.** Repeated patterns in the largest protein prediction program (2XO, 843 nodes). Largest pattern (133 nodes) in black. Other nodes in repeated patterns are filled according to size of the repeated pattern (33–132 grey and 11–32 light grey). Unique nodes and nodes which are part of small patterns are not filled.

(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. (Figure 14 shows in black an example of a repeated subgraph in a tree.) The values for the ten best of run protein prediction programs are: 33%–92%, mean 74%, and for seven symbolic regression runs: 65%–84% mean 72%, see Figure 15. The replications in Figures 13, 14 and 15 refer to any fragment of the whole tree, while the rest of Section 4 considers only whole non-overlapping subtrees.

### 4.5 Syntactically Repeated Subtrees

Figure 16 shows the location and size of exactly repeated subtrees in the largest of the protein prediction trees. Figure 17 gives the same data for the best program from the first Poly-10 symbolic regression run. Figure 18 refers to the same 27 best of run programs as Figure 15, 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, in which non-trivial subtrees evolved, repeated subtrees are evolved. Further the repeats are too large and/or numerous to be due to chance.

Obviously the bigger a subtree is the less likely it is to occur more than once in a random tree. Indeed the largest repeated subtrees observed in random programs contain three nodes and are repeated in only about one in 500

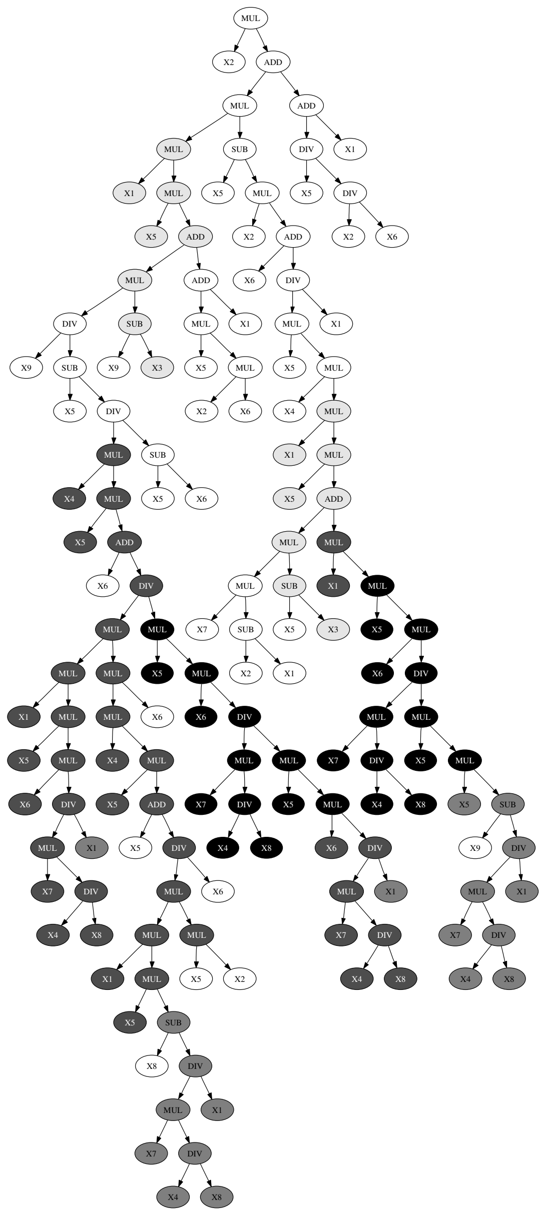

**Fig. 14.** Repeated patterns in best symbolic regression program from first Poly-10 run (155 nodes). Largest pattern (13 nodes) in black. Other nodes in repeated patterns are filled according to the size of the largest repeated pattern to which they belong (8–12 grey). Unique nodes and nodes which are part of small patterns which could arise by chance are not filled.

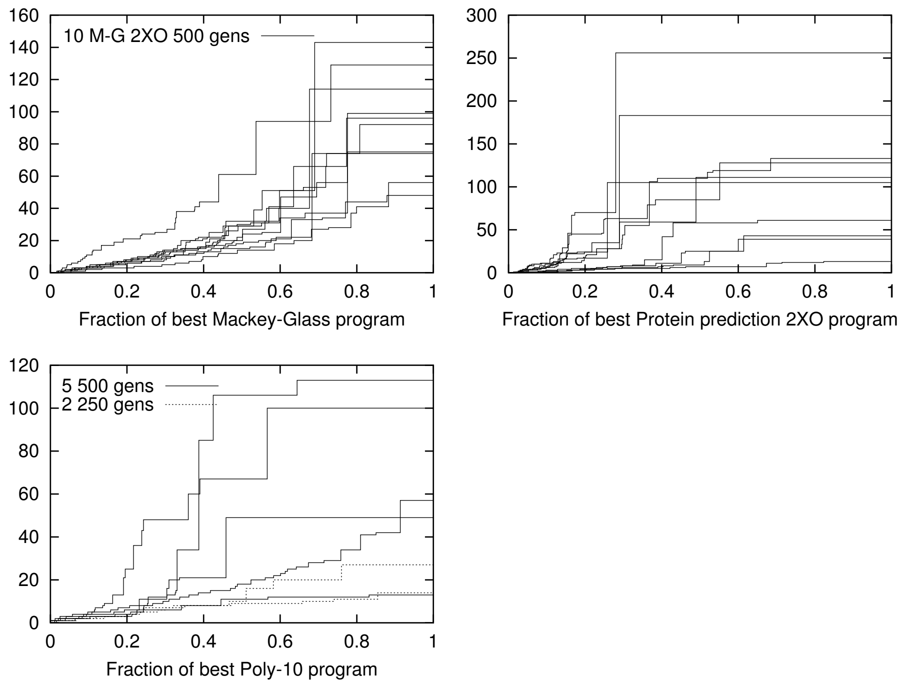

**Fig. 15.** Size of repeated pattern v. fraction of best of run trees (2XO). In every run the largest repeated pattern is too big to arise by chance. Depending on run 9–44% (Mackey-Glass), 8–67% (Protein) and 16–74% (Poly10) of these programs is not part of a repeated fragment of 11 or more nodes. I.e. at least 91–56% (Mackey-Glass), 92–33% (Protein) and 84–26% (Poly10) of the code is part of a non-random repeated structure. The three $x_1 \times x_2$ Poly-10 programs are not included. Also two Poly-10 programs at generation 500 are too large to be analysed exhaustively (2 419 and 6 577 nodes). These are replaced by the best program at generation 250 from the same runs (plotted with dotted lines, 225 and 193 nodes).

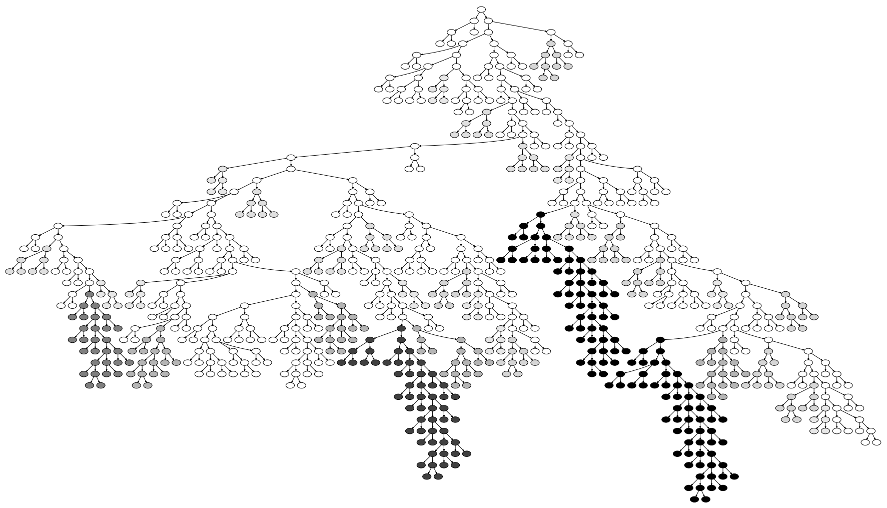
**Fig. 16.** Same program as in Figure 13. Here whole subtrees are exactly repeated. Nodes are filled according to size of the repeated subtree. Unique nodes and nodes which are part of small patterns (3 nodes or less) are not filled. Two largest (59 nodes, right hand side) filled with black. Note these are partially repeated elsewhere in the tree (e.g. 55 node subtree, centre of figure, shaded dark grey).

random programs, i.e. p-value 0.2%. Yet evolution produces repeated pattern far bigger than this. These large non-random repeats are highly significant. We have developed a model of random trees which estimates probabilities (p-values).

## 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. Figures 19 and 20 shows the variability within the largest protein location tree (2XO, 50 generations). We use information entropy [24] (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 such "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.

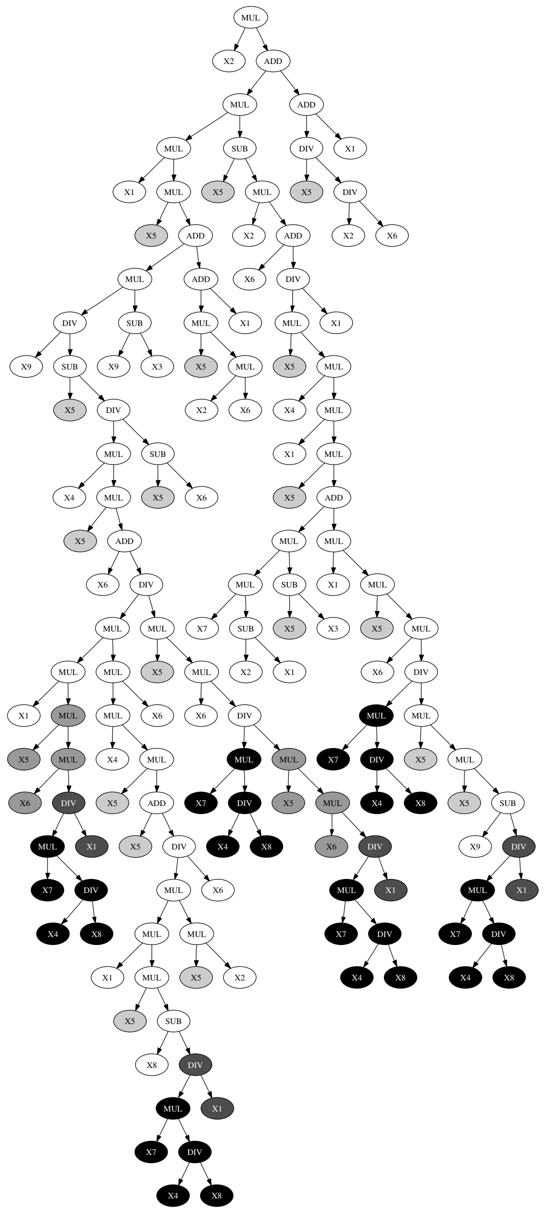

**Fig. 17.** Same program as in Figure 14. Here whole subtrees are exactly repeated. Nodes are filled according to how unlikely the repeat of the subtree is to be due to chance. Note six identical five node subtrees are coloured black (p-value $4\,10^{-20}$). They are part of four seven node subtrees. The two non-overlapping nodes are filled with dark grey. Two of these are in turn part of a pair of 11 node subtrees. Again non-overlapping nodes shaded, this time in grey. The 20 copies of input $x_5$ (p-value $10^{-5}$) are shaded in light grey.

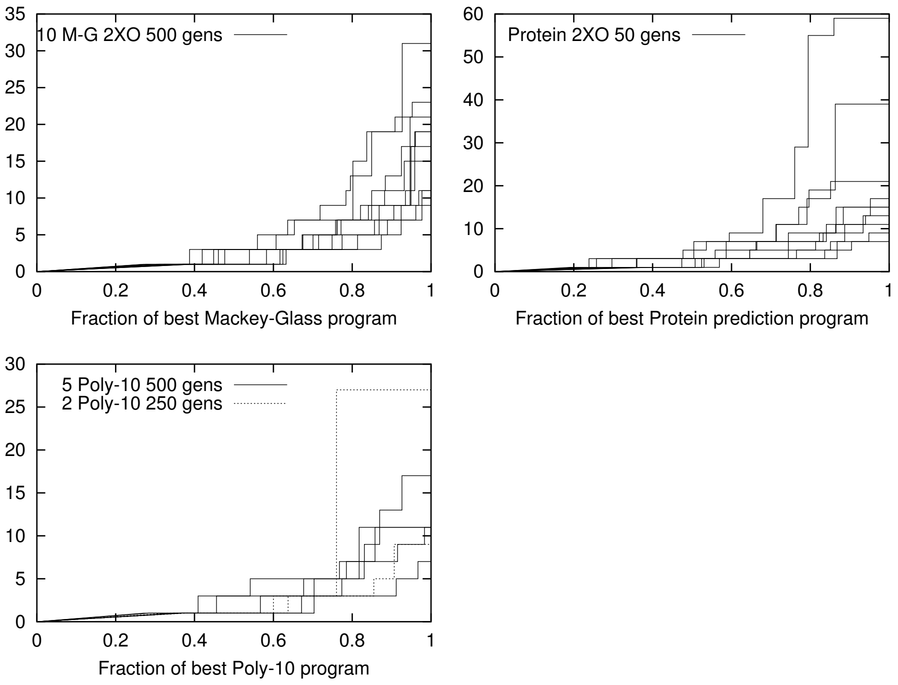

**Fig. 18.** Size of identical subtrees v. fraction of best of run trees (2XO). In every run the largest repeated subtree is too big to arise by chance. Indeed, depending on run, only 4–29% (Mackey-Glass), 4–30% (Protein) and 54–91% (Poly10) of these programs is not part of a repeated subtree of five or more nodes. I.e. at least 96–71% (Mackey-Glass), 96–70% (Protein) and 46–9% (Poly10) of the code is part of a non-random repeated subtree. The three $x_1 \times x_2$ Poly-10 programs are not plotted. Two large Poly-10 programs replaced by programs from generation 250 (plotted with dotted lines).

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.

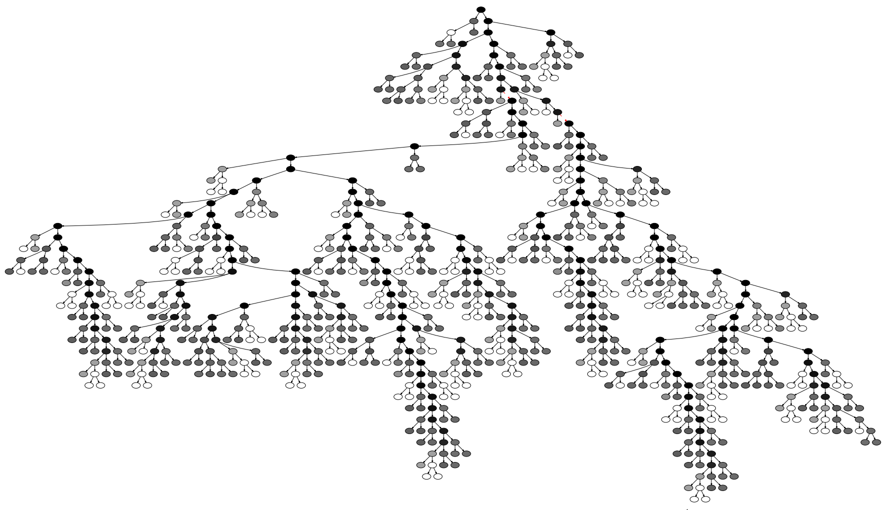

**Fig. 19.** Entropy of each node in largest protein program (cf. Figures 13, 16 and 22.) Darker grey indicates more variation across the training set. Note entropy tends to increase towards root. (At levels 7 and 9 there are two links, dotted red, where large subtrees pass through bottlenecks, i.e. entropy falls). C.f. Figure 20.

We can perform a similar entropy analysis on the symbolic regression problem but it is less informative. Remember that each of the training examples is randomly generated. This means none of the $50 \times 10$ input values are repeated. In at least some cases, the evolved program's subtrees never calculate the same value. So the plot corresponding to Figure 20 would be a flat line with every subtree having maximal entropy.

## 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. In protein predictions runs, between 15 and 78 (depending on the run) subtrees in each best of run program exceed

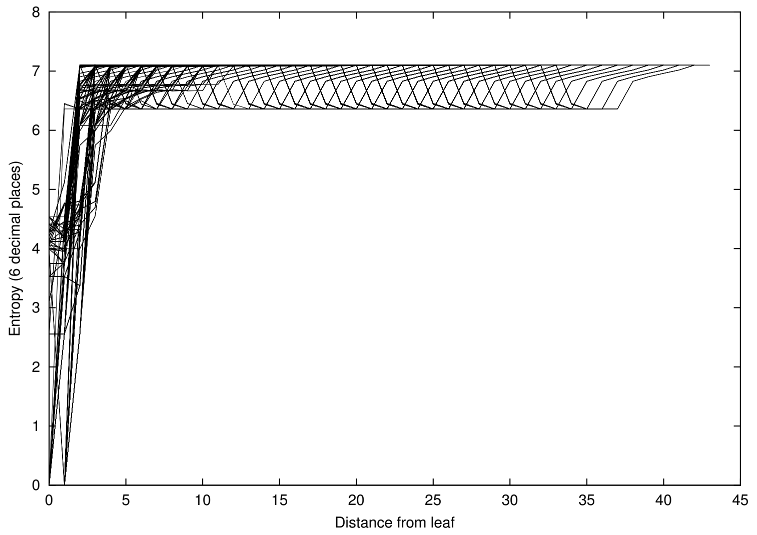

**Fig. 20.** Entropy on each of 422 paths from leaf to root in largest protein program (cf. Figures 13, 16 and 22). At levels 7 and 9 there are two links where large subtrees pass through bottlenecks. The bottlenecks near the root show up as repeated dips in the tail. This structure is an artifact caused by paths passing through similar routes near the root but having different lengths.

the performance of random search ($10^6$ ramped half-and-half trees), see Figure 21. Since fitness tends to fall away 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.

In Poly-10 runs, subtrees in evolved solutions also tend to increase in fitness toward the root. Typically only the root node has a fitness (or correlation with the target) that exceeds that of $x_1 \times x_2$. This may be due to the difficulty of the problem, exacerbated by the random test data.

## 4.8 Importance of Subtrees (Sensitivity analysis)

While the trees do not contain large numbers of "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. (Those which changed the prediction in more than ten fitness cases are highlighted in Figure 22.) The upper solid curves in Figure 23 plot the number of fitness cases where the output was changed by more than 0.005%. While

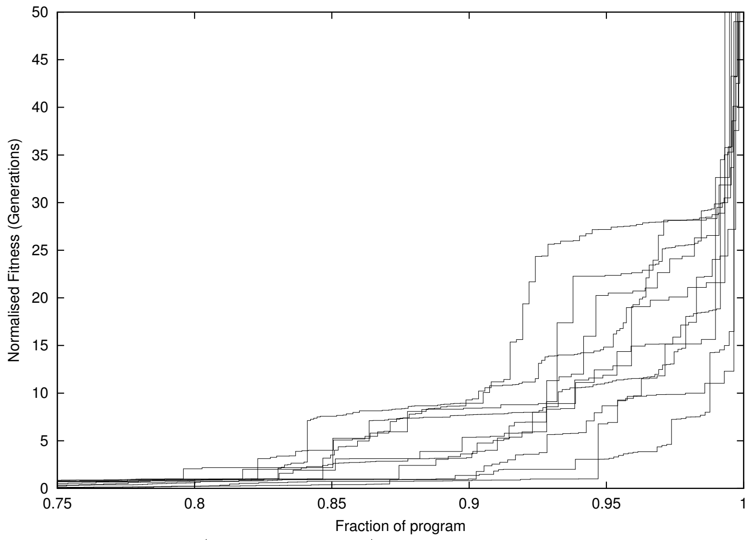

**Fig. 21.** High fitness (or anti-fitness) subtrees as a fraction of the 10 best protein trees (2XO). Note range of horizontal axis. Since fitness is a very non-linear function, we define a normalised fitness as being, for each run, the generation in which a program of the corresponding fitness was first found. All runs exceeded the best fitness found in a million random trees programs by generation 8.

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.

For the Poly-10 problem deciding how to quantify the importance of subtrees proved more problematic. Since the test cases are chosen with (approximately) equal positive and negative values, the median values are usually near zero. This tends to make DIV operations appear important. To avoid this problem, the importance of subtrees was measured by replacing them with leafs and calculating the mean change in fitness. Figure 24 shows the impact of replacing each subtree with each input $x_1 \dots x_{10}$ in the first symbolic run. While Figure 25 plots the sensitivity for all seven non-trivial Poly-10 best of run programs. Figure 25 makes clear that the performance of evolved trees on the training data depends little on a large proportion of the tree. We can also see that bigger evolved trees are less sensitive.

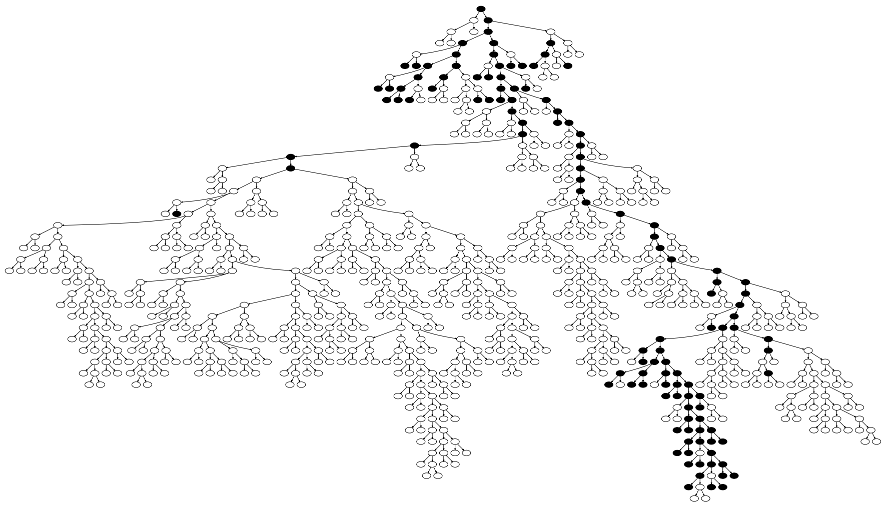

**Fig. 22.** Importance of nodes within protein prediction trees. Largest protein prediction tree. The 125 (15%) subtrees which change more than 10 training cases are highlighted in black. (Same example as in Figures 13, 16 and 19.) Of the remaining 725, 277 have no impact on fitness at all, while a further 151 affect only one (of 1213) training case. Note several large repeated subtrees (which must produce the same values) make little contribution to fitness.

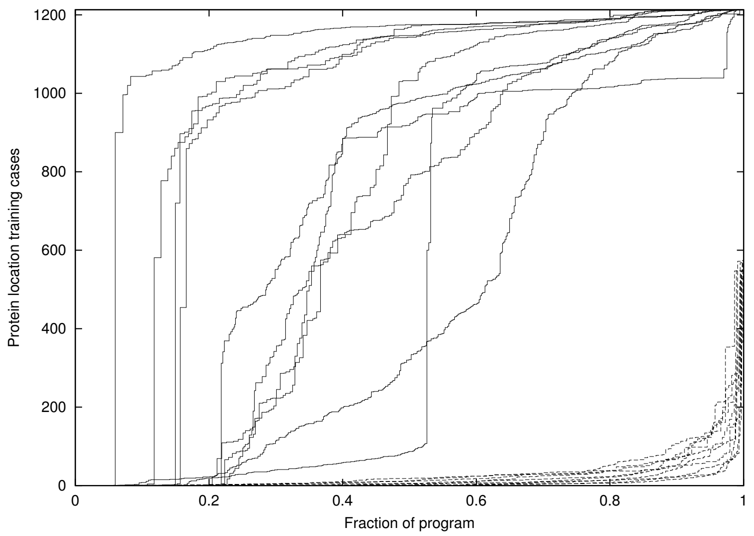

**Fig. 23.** Number of training cases which subtrees influences as a fraction of the 10 2XO best of run programs. Solid curves plot where impact is more than 0.005%. Dashed lines: node causes prediction to change.

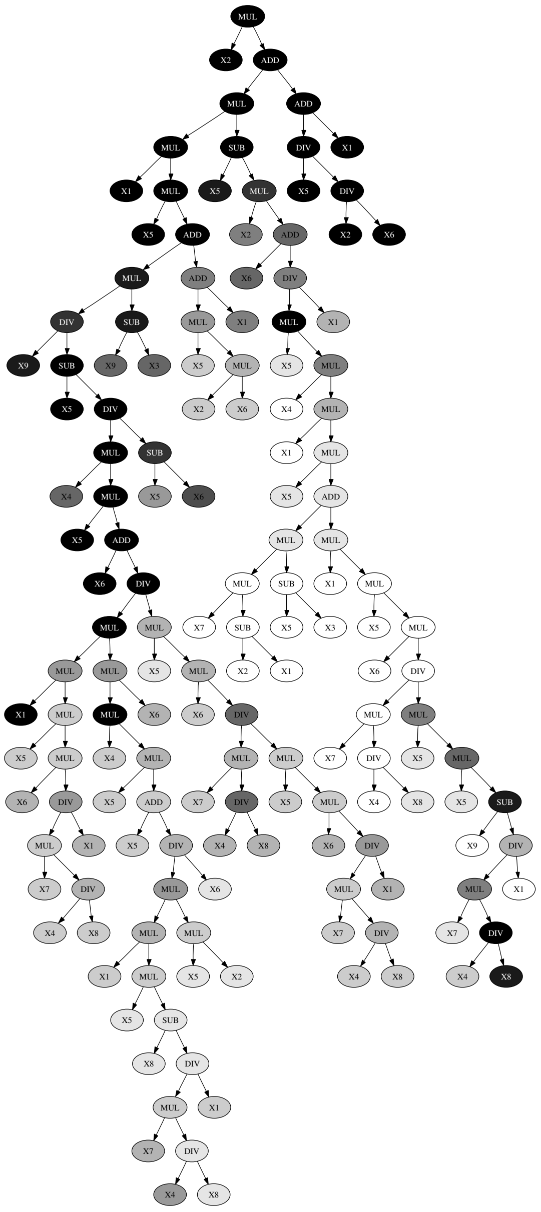

**Fig. 24.** Importance of nodes within symbolic regression trees (best program in first Poly-10 run, cf. Figures 14 and 17). Subtrees which when replaced by a leaf would produce on average more than a 10% change in fitness are shaded. More than 100% change are highlighted in black. Note identical subtrees make different contributions to fitness.

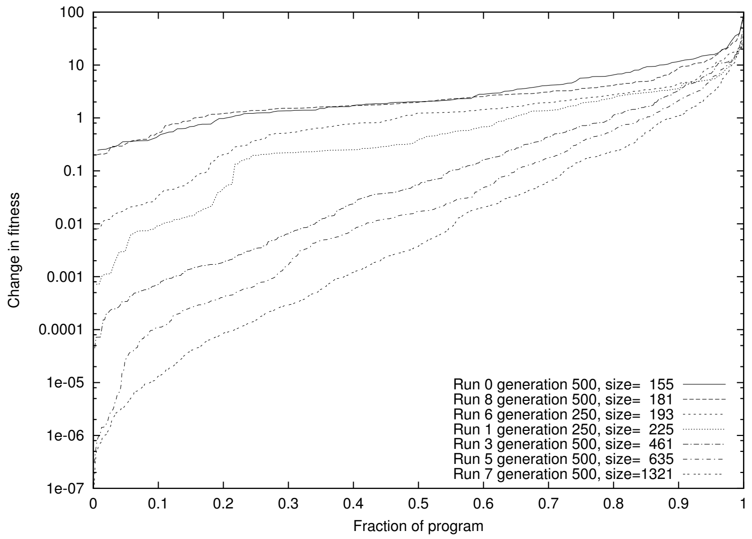

**Fig. 25.** Mean change in fitness when each part of program is replaced by a leaf. Showing only a small part of large evolved programs is very important within symbolic regression trees. Best programs from seven Poly-10 runs. Three runs converged to $x_1 \times x_2$ are not plotted. The huge programs evolved by generation 500 in runs 1 and 6 are replaced by the best of generation 250.

Depending upon run, between 15% and 89% of Poly-10 trees can be replaced with leafs without changing their error by more than 1.0, averaged across all ten inputs $x_1 \dots x_{10}$. (In Figure 24 a fitness difference of 1.0 is shaded with the lightest grey, e.g. the leaf X8 furthest from the root.) Another way of looking at the data presented in Figure 25 is to consider which subtrees are important I.e. which subtrees – when replaced – will on average change fitness by at least 1.0. In these seven Poly-10 runs, there are only 94–153 important nodes in the evolved programs. (Note by this point in the runs, half the population was within 1.6–3.1 of the best in the population. Since we are using binary tournament selection, a change in fitness of 1.0 will reduce their expected rate of producing children by a factor of less than 2 [25, page 71].)

## 5 Discussion

Sections 4.2 and 4.3 confirm (cf. [22]) trees have evolved to the same fractal shape as random trees but Sections 4.4 and 4.5 show repeated patterns which are far from random. Sections 4.6 and 4.7 suggest GP programs (composed of 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 that large parts of the tree, including repeated parts, can be replaced (e.g. by a constant or input) and this will 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" [12]: 1) They are not small; 2) they have high fitness on the whole problem, rather than sub-components of it. It appears evolution is 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 than others.

In simple genetic programming problems, the preferred subtrees are not classic building blocks, since they tend to have high fitness on the whole problem rather than on components of the problem. Also as GP jumbles together copies of subtrees to create complete solutions, similar, or even identical, components (which may in themselves have similar, or indeed, identical problem solving abilities) tend to have very different importance in the whole program. So over time, evolved programs accumulate exact or nearly exact copies of useful code but most copies have only a marginal impact.

## 6 Conclusions

Correlation between performance of initial and evolved populations suggests lack lustre initial random programs can have an impact on the final outcome. While results from different problems are mixed, such correlation might yet prove to be a useful population size analysis tool or aid to finding a restart heuristic.

As expected, size fair crossover (FXO) [16] and a range of mutation operators controlled bloat [22]. In these experiments, the compact models performed 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. The situation seems to resemble that found in genomes, where certain segments of genes have much more impact on the final organism than others.

While it is always difficult to generalise from a limited number of examples, we have investigated a variety of representations, genetic operations and generational strategies, implemented in two different languages, using three diverse non-trivial problems (two successfully solved and one less so). In every case, where program size allows, we have seen the spontaneous emergence of repeated patterns in both linear [10] and tree based GP. 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 while WBL was at University College, London and Essex University. WB thanks NSERC for grant RGPIN 283304-04.

**Source Code**

A modified version of Andy Singleton's GPquick (GProc) can be obtained via anonymous ftp site `cs.ucl.ac.uk` directory `genetic/gp-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. R. J. Britten and D. E. Kohnen. Repeated sequences in DNA. *Science*, 161:529–540, 1968.
2. A. F. A. Smit. The origin of interspersed repeats in the human genome. *Current Opinions in Genetics and Development*, 6:743–748, 1996.
3. C. Patience, D. A. Wilkinson, and R. A. Weiss. Our retroviral heritage. *Trends in Genetics*, 13:116–120, 1997.
4. J. R. Lupski and G. M. Weinstock. Short, interspersed repetitive DNA sequences in prokaryotic genomes. *Journal of Bacteriology*, 174:4525–4529, 1992.
5. G. Toth, Z. Gaspari, and J. Jurka. Microsatellites in different eukaryotic genomes: Survey and analysis. *Genome Research*, 10:967–981, 2000.
6. G. Achaz, E. P. C. Rocha, P. Netter, and E. Coissac. Origin and fate of repeats in bacteria. *Nucleic Acids Research*, 30:2987–2994, 2002.
7. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.
8. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction.* Morgan Kaufmann, 1998.
9. W. B. Langdon and R. Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002.
10. W. B. Langdon and W. Banzhaf. Repeated sequences in linear genetic programming genomes. *Complex Systems*, 15(4):285–306, 2005.
11. W. B. Langdon and W. Banzhaf. Repeated patterns in tree genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 190–202, Lausanne, Switzerland, 30 March - 1 April 2005. Springer.
12. U-M. O'Reilly and F. Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994 (published 1995). Morgan Kaufmann.

13. R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217, Essex, UK, 14-16 April 2003. Springer-Verlag.
14. H. Oakley. Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 17, pages 369–389. MIT Press, 1994.
15. A. Reinhardt and T. Hubbard. Using neural networks for prediction of the subcellular location of proteins. *Nucleic Acids Research*, 26(9):2230–2236, 1 May 1998.
16. W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119, April 2000.
17. R. Poli. TinyGP. See TinyGP GECCO 2004 competition at `http://cswww.essex.ac.uk/staff/sml/gecco/TinyGP.html`, 2004.
18. G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the third international conference on Genetic Algorithms*, pages 2–9, George Mason University, 4-7 June 1989. Morgan Kaufmann.
19. W. B. Langdon. *Genetic Programming and Data Structures* Kluwer, Boston, 1998.
20. W. B. Langdon and S. J. Barrett. Genetic programming in data mining for drug discovery. In A. Ghosh and L. C. Jain, editors, *Evolutionary Computing in Data Mining*, volume 163 of *Studies in Fuzziness and Soft Computing*, chapter 10, pages 211–235. Springer, 2004.
21. R. Poli. Personal communication, 2005.
22. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, 1999.
23. R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 1996.
24. C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* The University of Illinois Press, Urbana, 1964.
25. T. Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis.* PhD thesis, Swiss Federal Institute of Technology, Zurich, November 1996.