RN/11/10
11 April 2011

Research
Note

Minimising Testing in Genetic Programming

W. B. Langdon

Electronic Mail: W,Langdon@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/staff/W.Langdon/

Abstract

The cost of optimisation can be reduced by evaluating candidate designs on only a fraction of all possible use cases. We show how genetic programming (GP) can avoid overfitting and evolve general solutions from fitness test suites as small as just one dynamic training case. Search effort can be greatly reduced.

Keywords

genetic algorithms, genetic programming, search, heuristic methods, artificial intelligence, software engineering, theory, over fitting, evolutionary learning, deceptive fitness landscapes, population convergence, correlations, GPU, GPGPU, 11-Mux, 20-mux, 37-multiplexor, bloat

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

1 Introduction

In many fields, particularly Software Engineering, it is often impossible to test all possible cases. So there is considerable interest in quantifying how much testing has been done and what level of confidence can be placed in the inferred quality of the item under test. In engineering many software metrics have been promoted to serve as proxies or estimates of the usefulness of test suites. In artificial intelligence, particularly Machine Learning, there are mathematically based, albeit weak, bounds on how far we can extrapolate from the results of tests [Sontag, 1998]. In machine learning, such as genetic programming [Poli *et al.*, 2008], the lack of generalisation is known as over fitting. With continuous symbolic regression [Koza, 1992] problems overfitting is characterised by excessively complex expressions which approximate the fixed training data well but may oscillate widely between them. This gives little confidence that the evolved expression captures the underlying pattern in the data. With symbolic regression benchmarks [Koza, 1992] the behaviour of the evolved model can usually be plotted but this is not necessarily the case with real problems [Bongard and Lipson, 2007].

Boolean problems are discrete and may have a large number of dimensions making visualisation tricky and often the only solution is to tabulate the whole function, yet as the number of dimensions increases this also rapidly becomes infeasible. Previously we evolved 37 dimensional solutions to the Boolean multiplexor problem [Langdon, 2010]. In [Langdon, 2010] we discussed the GP implementation but not how and why the evolved solutions work. A summary of our results can be found in [Langdon, 2011]. CUDA C code is available via FTP site `cs.ucl.ac.uk` directory `genetic/gp-code/gp32cuda.tar.gz`

The next section considers existing work on reducing the computational code of GP. Sections 3 and 4 describe the GP Multiplexor benchmarks whilst Section 5 gives our results. Section 6 tries to explain why GP continues to work when the information given to evolution is reduced. In Section 6.1 we consider how reducing the size of the training set changes who wins selection tournaments. Section 6.2 looks at when GP fails and is trapped at deceptive local optima. Section 6.3 looks at GP convergence and shows, except for extreme cases, convergence fitness case sub-sampling is similar to convergence with exhaustive testing. Section 7 considers practical aspects, e.g. when to stop a GP run.

2 Faster Optimisation by using Approximate Models

Models have long been an essential ingredient of the design process. Nowadays many of these models are computer based and there is considerable interest in using them to optimise designs. However full models can be computationally costly and optimisation processes (such as evolutionary algorithms) may require the model to be run many times with different parameters. This leads to pressure for less costly models or ways to approximate their answers.

In genetic programming (GP) the principle cost is usually the fitness function and often that scales directly with the number of fitness cases. So there is an incentive to reduce GP run time by using fewer fitness cases to make GP faster but still hopefully provide useful solutions. For example, [Banzhaf *et al.*, 1998, Sect. 10.1.5] advocates small different samples from the real world. Gathercole proposed a more controlled dynamic training subset selection (DSS) [Gathercole and Ross, 1994] in which ad hoc rules are used to select and change the test suite. DSS is used commercially [Foster, 2001] [Poli *et al.*, 2008, page 84]. Due to its simplicity, scaling and constant selection pressure, tournament selection is widely used in GP [Poli *et al.*, 2008]. We had earlier noted that it is not always necessary to run all fitness tests in order to tell who will win a tournament [Langdon, 1998]. [Teller and Andre, 1997] used statistical arguments to dynamically set the number of fitness cases in their RAT system. Although Ross [Ross, 2000a; Ross, 2000b] does not concentrate upon fitness sampling he also used a statistical approach (χ^2), Unlike RAT this is not used to control test suite sampling. (Fixed re-sampling rates are used.) [Ross, 2000a] reports mixed results with test suites of 50 and 1000 samples perhaps due to GP evolution prematurely converging due to a small population. [Giacobini *et al.*, 2002] is the first systematic study of training set size and learning.

Table 1: GP Parameters to Solve the 11-Multiplexor with less Fitness Testing

Terminals:	Boolean inputs A0 A1 A2, D0 D1 D2 D3 D4 D5 D6 D7
Functions:	AND, OR, NAND, NOR
Fitness:	Pseudo random samples of from 0 to 2048 fitness cases.
Fit 20-Mux:	Samples of 32 of 1 048 576 fitness cases.
Selection:	Generational, with 4 members tournaments. Tournaments run on same random sample. 1) Fixed sample, 2) new samples for each generation and 3) new samples for each tournament.
Population:	16 384 (20-Mux: 262 144)
Initial pop:	Ramped half-and-half 4:5
Parameters:	50% subtree crossover, 5% subtree 45% point mutation. Max depth 15, max size 511.
Termination:	first solution or 2 000 gens [extended runs 50 000 gens or pop is only leafs]

Coevolution [Hillis, 1992] can take a different approach. Instead of seeking to choose fitness tests in an unbiased way, it may deliberately seek tests that will drive evolution to defeat the current opposition. Thus it may overlook the need for tests to discriminate. One hope is that since the other populations are also evolving, the various biases will ultimately lead to better solutions.

3 GP Benchmarks

The 6 and 11 Boolean input Multiplexor problems have been extensively used in genetic programming research [Koza, 1992]. The 11-Mux is particularly suitable for our experiments since its complete test suite is large, however (particularly with GPU hardware speed up and the parallelisation tricks given in [Poli and Langdon, 1999]) it is not so large as to prevent exhaustive testing. Thus we can evolve our GP populations with a sample of the test suite but where necessary for our research we are still able to calculate the whole fitness. I.e. calculate performance on the whole test suite.

4 Non-Exhaustive Fitness Testing

There are 2^{11}, i.e. 2048, test patterns for the 11-Mux problem, Usually [Koza, 1992] the GP fitness function uses all 2048. As we shall see, this is unnecessary. In the next section we run a comprehensive series of experiments which show evolution is potentially able to extrapolate from the environment in which individuals are tested to solve the complete problem. Section 6 analyses these experiments.

The details of our GP are given in Table 1. There are no branches, so every test executes all the code (i.e. 100% test coverage). Notice the functions and terminals form a complete set which is not tailored to the 11-Mux benchmark. The members of each fitness test suite are chosen randomly from the full 2048 set. We run three sets of experiments: 1) test suite is fixed through out the run. 2) test suite is reselected every generation. 3) the test suite is reselected by every tournament.

5 Results

Table 2 shows sampling test cases rather than running them all can be highly effective. Even the simplest scheme, in which the test suite is kept constant throughout evolution, can reduce testing by a factor of eight and still reliably find solutions albeit at the cost of doubling the number of generations needed. With a limit of 2000 generations, both schemes where the test suite is randomly regenerated during the run can reliably solve the complete problem with as few as 8 test cases. Emboldened by this we re-ran all the runs with $\leq$4 dynamic tests with a limit of 50 000 generations (Table 2 top section). This showed the per tournament selection scheme can reliably solve the problem with a single test. However at the cost of increasing the number of generations about two hundred fold. (Figure 1 summarises these data.)

Table 2: Generation at which GP solved the 11-Mux given different numbers of randomly selected tests. First vertical group of 5 columns: same tests used for up to 2000 generations. 2^{nd} new random sample every generation ($\leq$50 000). 3^{rd} same sample used for tournaments of 4 individuals ($\leq$50 000 gens). Numbers are: 1^{st} quartile, median, and 3^{rd} quartile, number of runs finding a solution and total number of runs Typically we ran ten independent runs for each parameter setting. ("na" indicates no solution found).

Tests	Start of run				Every generation				Every tournament			
	Generation			ok/runs	Generation			ok/runs	Generation			ok/runs
0	na	na	na	0/ 1	na	na	na	0/ 1	na	na	na	0/ 1
1	na	na	na	0/10	na	na	na	0/10	14000	16000	19000	10/10
2	na	na	na	0/10	na	na	na	2/10	2900	4100	5700	10/10
3	na	na	na	0/10	1400	2800	4100	8/10	1400	1900	2700	10/10
4	na	na	na	0/10	1000	1600	2000	10/10	1200	1500	2200	10/10
8	na	na	na	0/10	549	704	941	10/10	633	795	1047	10/10
16	na	na	na	0/10	378	541	766	10/10	389	449	588	10/10
32	na	na	na	0/10	186	342	369	10/10	271	296	401	10/10
64	na	na	na	0/10	145	189	246	10/10	221	326	352	10/10
128	na	na	na	2/10	145	164	254	10/10	161	181	215	10/10
256	123	177	368	10/10	99	106	138	10/10	161	184	200	10/10
512	79	81	88	10/10	98	107	171	10/10	141	167	181	10/10
1024	78	85	92	10/10	69	86	107	10/10	117	131	144	10/10
1536	79	82	89	10/10	75	89	117	10/10	105	117	150	10/10
1792	81	85	87	10/10	72	87	102	10/10	111	115	139	10/10
1920	75	83	95	10/10	72	80	83	10/10	115	122	137	10/10
1984	75	87	91	10/10	79	81	91	10/10	113	122	138	10/10
2016	72	79	84	10/10	68	74	77	10/10	100	108	121	10/10
2032	73	76	83	10/10	70	79	92	10/10	123	129	169	10/10
2040	74	88	103	10/10	77	78	88	10/10	105	132	147	10/10
2044	72	83	96	10/10	66	80	83	10/10	109	118	124	10/10
2045	77	86	92	10/10	70	72	76	10/10	117	134	181	10/10
2046	74	79	93	10/10	82	85	88	10/10	110	115	119	10/10
2047	74	80	97	10/10	65	74	80	10/10	112	120	147	10/10
2048	78	82	86	10/10	78	82	86	10/10	102	122	126	10/10
20-Mux 32 tests					1083	1179	1516	10/10				

The last line of Table 2 shows GP, albeit with a larger population, was able to solve the much bigger 20-multiplexor problem using only 32 of 1 048 576 tests (changed every generation).

As Figure 2 shows true fitness can increase when tests are randomly sampled like it does in a conventional GP (with 100% of the fitness cases being used) but needing more generations. Whilst the best observed fitness is not representative, of the population, the population's average observed fitness increases like its average true fitness.

All runs in Figure 2 bloat. With only 8 tests, increase in tree size (like increase in fitness) is slower than with all 2048 tests. However for the last 1016 generations trees are so bloated that their average size is more than 90% of the maximum permitted size. See also Figures 9 and 10.

With both true and observed fitness, the worst in population is essentially the opposite of the best in population, see Figure 3. A simple example shows how this might happen. Suppose the best parent in the previous generation scored x% and had OR as its root node and this was mutated into a NOR. The child's answer is exactly the opposite and so scores 100-x%. This is likely to be the worst score in the new generation, whereas the best is likely to be x%.

Koza's "Effort" [Koza, 1992] is given in Table 3. It suggests in these 11-Mux runs using as few as 4 tests out of 2048 can reduced fitness testing needed to be 99% sure of finding a solution from about 1.5 million fitness evaluations to the equivalent of about 80 000 (full) fitness evaluations. A nineteen fold saving. The estimates are noisy (note the magnitude of the error estimates, shown in brackets) and biased [Christensen and Oppacher, 2002]. Nevertheless, given the high proportion of successful runs, they give some idea of the

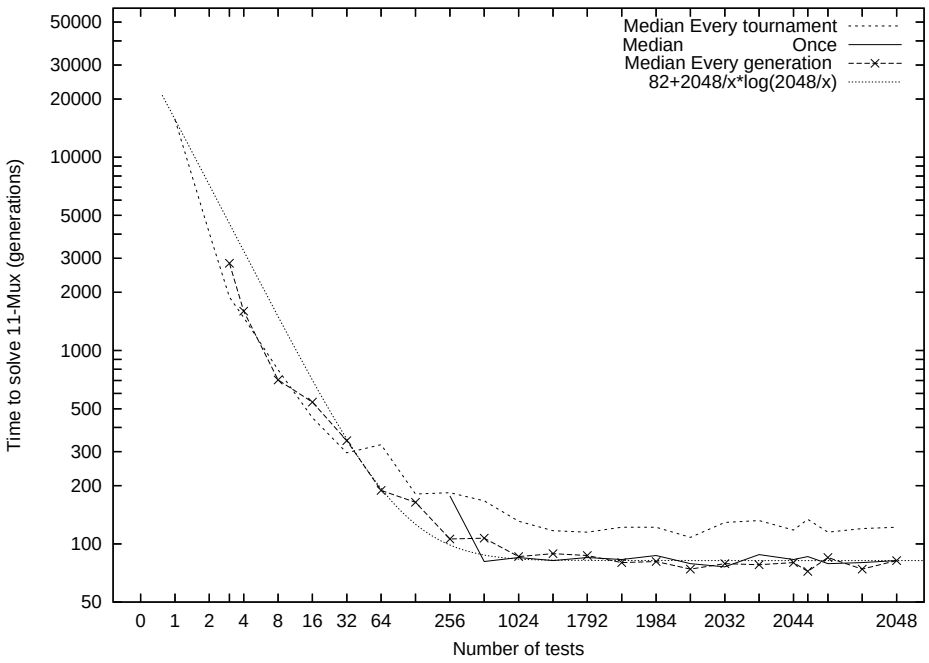

Figure 1: Average generation at which GP runs from Table 2 solved the 11-Mux problem. Note non-linear scales. Test suite created once (solid line) starts at 256. Every generation (line with $\times$) starts at 3. Whilst every group of 4 individuals (dashed line) starts at just one test. The fitted dotted line is the "building blocks" prediction, see page 14.

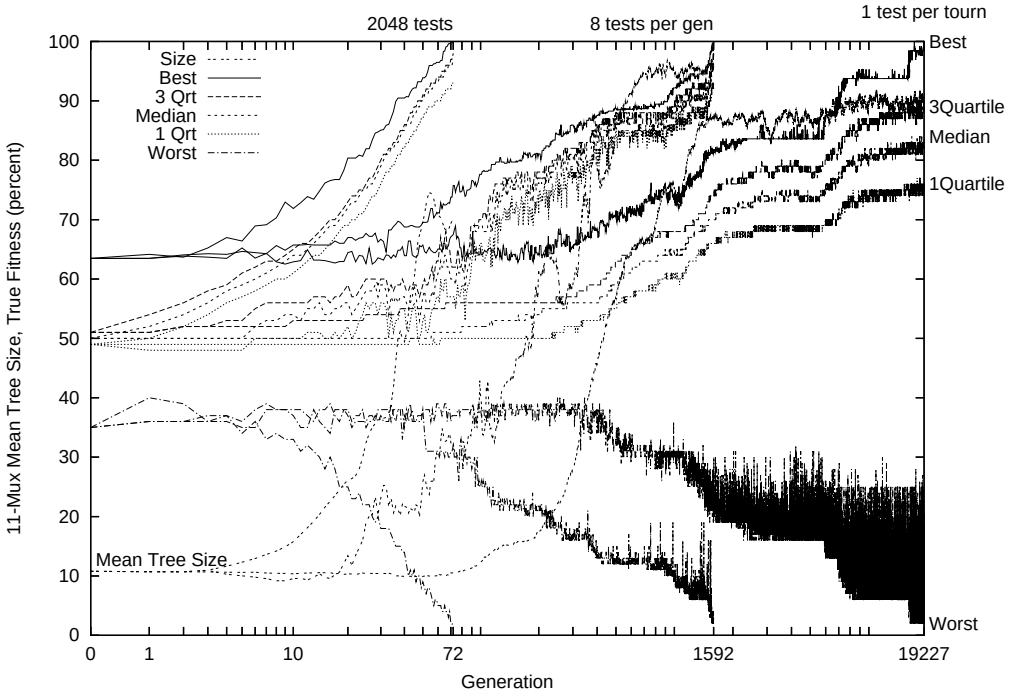

Figure 2: Three example 11-Mux runs: 1) first conventional run, 2) first run with 8 tests (of 2048) randomly chosen every generation. 3) first run with 1 test randomly chosen every tournament. Note non-linear scales. Average tree size (double dashed line) is plotted as a percentage of the maximum tree size (511).

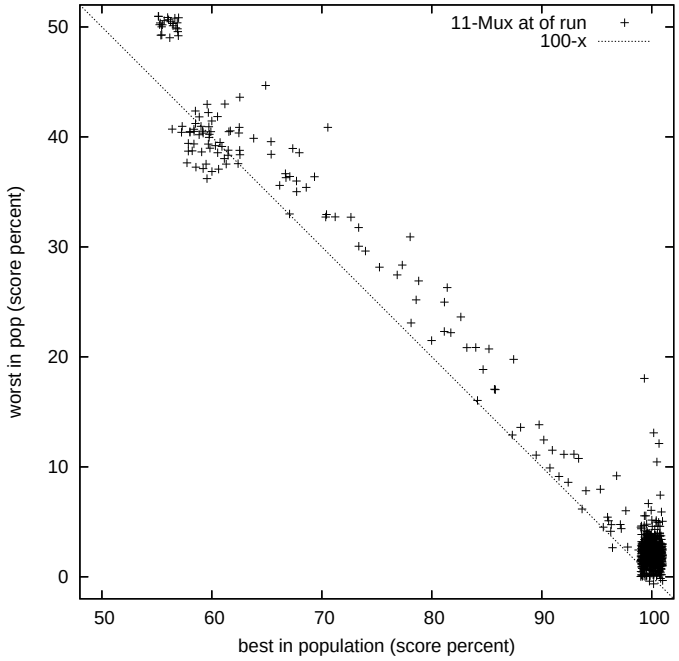

Figure 3: 11-Mux score of worst in population likely to be the opposite of the best. Last population of 722 runs. (1% noise added to spread data).

savings which can be achieved by using only a fraction of the test suite. This assumes the costs of genetic operations and generating the samples of the test suite are negligible. In large serious GP applications, these are indeed likely to be only a small component of the overall cost, which will remain dominated by fitness evaluation. The improvement ratio will be problem dependent and, as we previously showed [Langdon, 2010] it is likely to be even larger for larger problems. Indeed the 20-Mux and 37-Mux problems have only been solved by GP using test case sub-sampling.

6 Analysis

6.1 Changing Who Has Children

Figure 4 shows the impact of test suite size etc. on parent selection tournaments. As expected when fewer tests are used more individuals are chosen to be parents who would not have been selected using their true (rather than apparent) fitness. Figure 4 plots not only where the individuals differ but also that they had different (true) fitnesses.

The rest of this section relates wrong choice of parents to fitness convergence. In Figures 5, 6, 7, 8 and 13 we measure convergence by counting how many trees in the population lie in the same 1% bin as the most popular fitness value. The initial population is created at random and its fitness has a binomial distribution. Thus many trees score half marks. In the next generation (see Figure 5) crossover and mutation spread the population and phenotypic convergence falls from an average of 5950 in generation 0. Typically after generation 2, selection acts to reduce diversity and the fitness values are less spread out. With a large fixed test set eventually about a third of the population will have the same (true) fitness. This is also approximately the case for most runs where the test set is changed every generation (see Figure 6) and where each tournament has a different test set (see Figure 8, 64 tests, conv). We will discuss the interesting "all leafs" example in the next section. (See also Figure 7.) Moreover with tiny but rapidly changed test suits (Figure 8, 1 test) selection cannot cause the population to converge and instead towards the end of the run only 5% of the population have the most popular true fitness value. Note: true general solutions are evolved from these very diverse populations.

Table 3: "Effort" to solve the 11-Mux. First vertical group of 4 columns: same tests used for up to 2000 generations, 2^{nd} new random sample every generation ($\leq$50 000), 3^{rd} new sample every tournament ($\leq$50 000 gens). Effort 1^{st} column in each group. Numbers in brackets are estimated error. 3^{rd} column in each group is Effort rescaled by (tests used)/(total tests). "na" indicates no run found a solution. The biggest improvement comes when using only 4, 8 or 16 of 2048 tests.

Tests	Start of run ×1000				Every generation ×1000				Every tournament ×1000			
			×tests/2048				×tests/2048				×tests/2048	
0	na		na		na		na		na		na	
1	na		na		na		na		530000	(3800)	260	(1)
2	na		na		1400000	(1700000)	1400	(1700)	180000	(28000)	170	(27)
3	na		na		210000	(87000)	310	(130)	130000	(31000)	190	(45)
4	na		na		40000	(1400)	78	(2)	50000	(5300)	98	(10)
8	na		na		26000	(1300)	100	(5)	30000	(3600)	120	(14)
16	na		na		23000	(2900)	180	(22)	13000	(1500)	100	(12)
32	na		na		13000	(3800)	200	(60)	10000	(300)	160	(4)
64	na		na		8700	(4300)	270	(130)	7400	(490)	230	(15)
128	390000	(450000)	24000	(28000)	7700	(650)	480	(40)	7400	(2000)	460	(120)
256	12000	(2500)	1500	(310)	3000	(260)	380	(33)	7100	(2000)	890	(250)
512	2300	(360)	590	(90)	4700	(560)	1200	(140)	3500	(200)	870	(51)
1024	1600	(41)	810	(20)	2300	(74)	1100	(37)	3200	(400)	1600	(200)
1536	1800	(66)	1400	(49)	2700	(280)	2000	(210)	3600	(150)	2700	(110)
1792	1600	(33)	1400	(29)	2200	(82)	1900	(72)	4200	(670)	3700	(590)
1920	1800	(74)	1700	(69)	1500	(49)	1400	(46)	3000	(66)	2800	(61)
1984	1800	(98)	1700	(95)	3000	(970)	3000	(940)	4700	(890)	4500	(870)
2016	1600	(74)	1500	(73)	1500	(82)	1500	(81)	2500	(220)	2400	(220)
2032	1500	(57)	1500	(57)	4100	(550)	4100	(540)	6200	(1500)	6200	(1500)
2040	3200	(670)	3200	(670)	1700	(110)	1700	(110)	3100	(41)	3100	(41)
2044	4800	(2700)	4800	(2700)	1600	(0)	1600	(0)	2400	(25)	2400	(25)
2045	3700	(880)	3700	(880)	1800	(120)	1800	(120)	6800	(1500)	6800	(1500)
2046	4700	(1200)	4700	(1200)	1700	(98)	1700	(98)	2500	(180)	2500	(180)
2047	1700	(49)	1700	(49)	1500	(49)	1500	(49)	7300	(1900)	7300	(1900)
2048	1500	(25)	1500	(25)	1500	(25)	1500	(25)	2400	(160)	2400	(160)

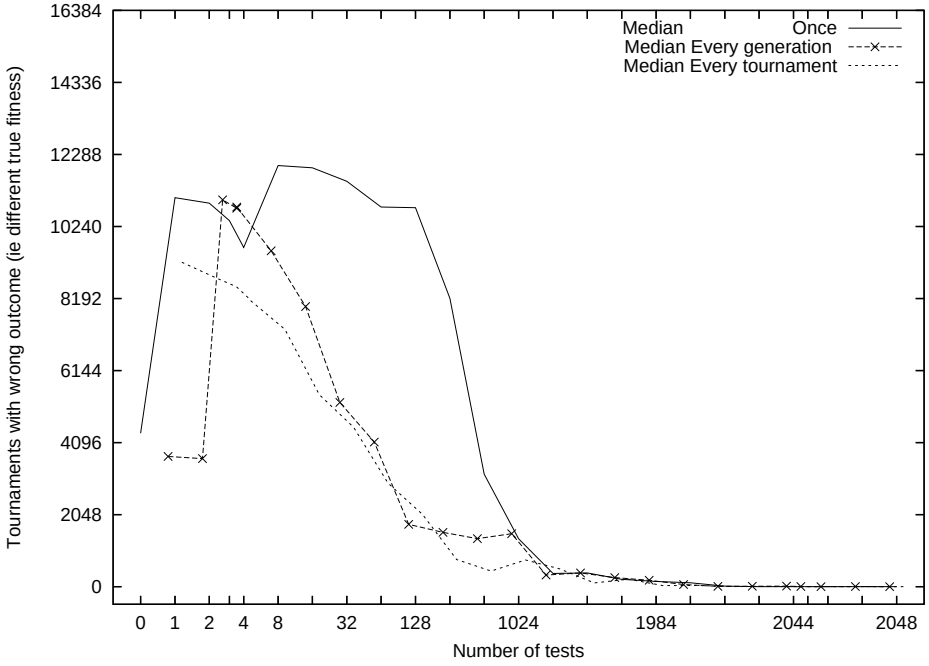

Figure 4: Impact of not using all tests on parent selection. Vertical axis is the median number of parents which should not have been chosen (averaged over last 20 generations of run). (NB. not only did the tournament choose a different individual but the two potential parents have different true fitnesses.) Typically when few tests are used, about three quarters of all tournaments are affected. Values for 0 and 1 test are explained in Section 6.2.

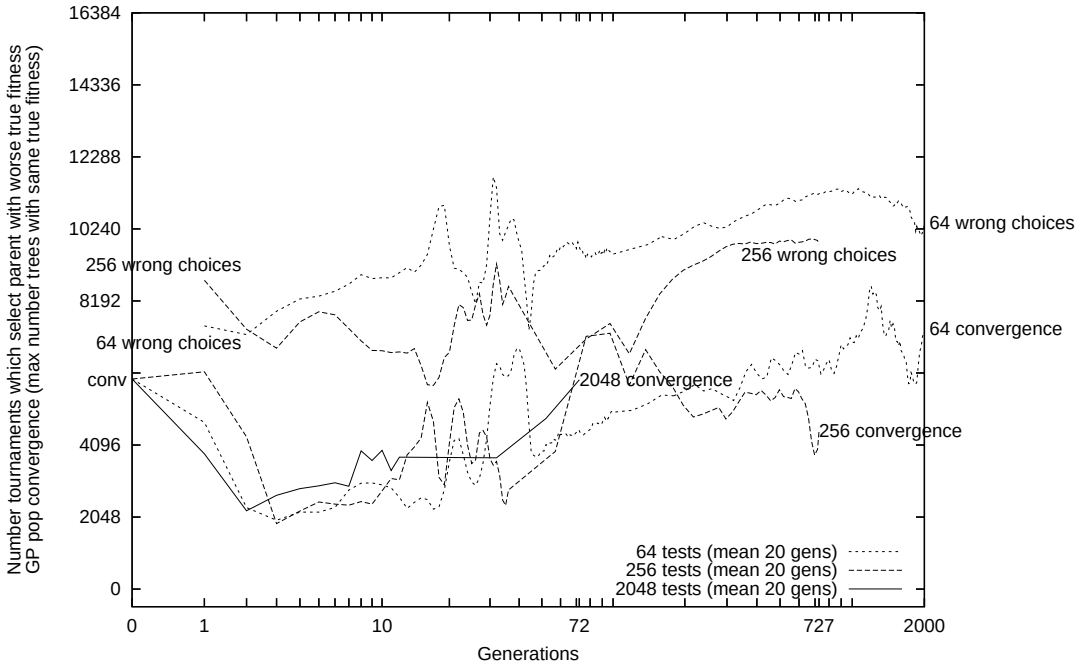

Figure 5: Evolution of selection errors and population convergence. Top two lines show the evolution of number of tournaments which chose wrongly with a fixed test subset. The lower 3 curves (conv) plot the number of individuals with most popular (true) fitness. The fraction of tournaments which choose wrongly in the unsuccessful runs with 64 tests (mean 36%) is on average only slightly higher than in successful runs with 256 tests (mean 32%). Plots smoothed by averaging over twenty generations. Obviously when all tests are used, no tournament chooses the wrong parent.

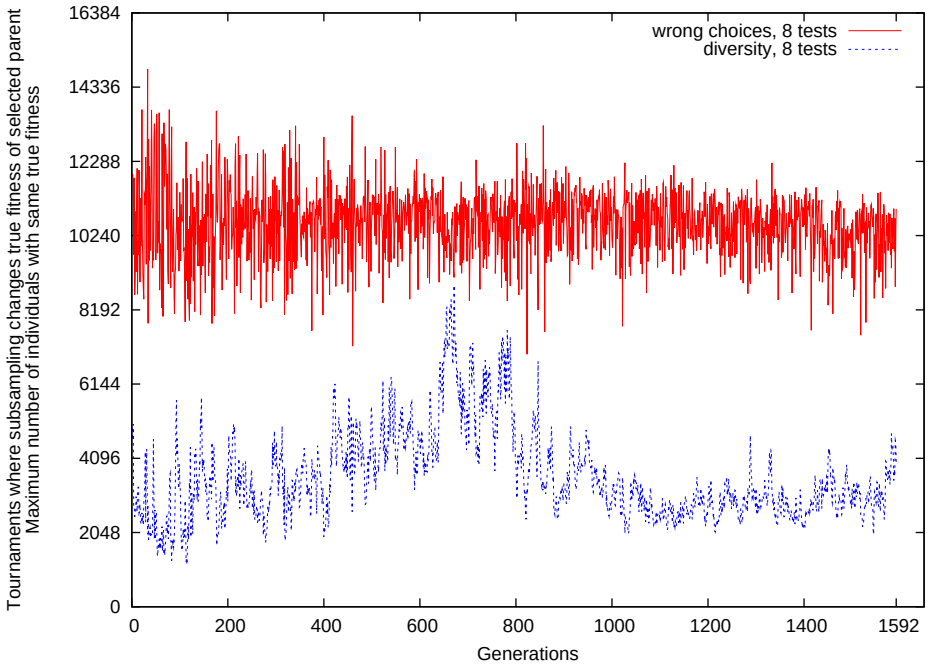

Figure 6: Selection errors (top) and convergence in the first run with 8 tests changed every generation. Notice GP is successful even though the fraction of tournaments choosing worse fitness parents is just below 3/4. (75% is the worst possible.) Typically about a quarter of the population converge on the most popular true fitness value (lower plot).

As expected, with a large number of tests most tournaments pick correctly a parent with the highest (true) fitness. As we substantially reduce the test suite more tournaments choose wrongly. For example, when using the same $1/8^{th}$ of the complete test suite, approximately half of tournaments choose wrongly. (Note Figure 5 plot 256 tests.) At first it seems surprising that GP can still work. Indeed if the number of tests is further reduced to 64 no run was successful. However there are four members in each tournament, with 256 tests, the best one is still being chosen half the time. The other 50% may choose the second best, which could be better than the remaining two trees in the tournament.

Assume the population remains reasonably diverse, so that the members of each tournament have different fitness values. Then the winner of each tournament has better fitness than the second best in the tournament. The fraction of tournaments where this is not true gives an indication of the noise in the observed fitness. If the noise is large and unbiased, the second best has a 50% chance of winning the tournament. (50% corresponds 8192 on Figure 4.) If the noise is even bigger, each of the four members of the tournament are equally likely to win. I.e., the maximum chance of choosing a wrong fitness value is 75%. In the cases where tests are changed during the run the average fraction of tournaments does indeed rise to about half when the test suite is reduced to 8. (Actually 45% and 58% on average.) When the tests used are fixed throughout the run then the half point is reached more quickly and with less than 256 fixed tests between about 1/2 and 3/4 of tournaments choose wrongly. Referring back to Table 2 we see test suites of 256 fixed tests are the smallest which reliably find solutions. Note, both dynamic schemes continue to find solutions as the number of tests is reduced (Figures 6, 7 and 8). However with only 1, 2 or 3 tests chosen each generation, it appears solutions might be possible except that selection is not strong enough to overcome the anti-bloat bias in subtree mutation to create smaller children or it may be deceived by leafs of above average fitness (next section, page 10). All the failing runs converged on populations with programs made only of leafs.

With four individuals in a tournament, (on average) at most 3 out of 4 can be wrongly chosen. 3/4 corresponds to 12 288 on Figure 4. When the test suite is very reduced two effects conspire to push the fraction of wrong selections to this maximum: 1) with fewer tests, the estimate of true fitness is noisier. 2) fewer tests means there are fewer fitness levels and necessarily the population has less fitness diversity. Figure 4

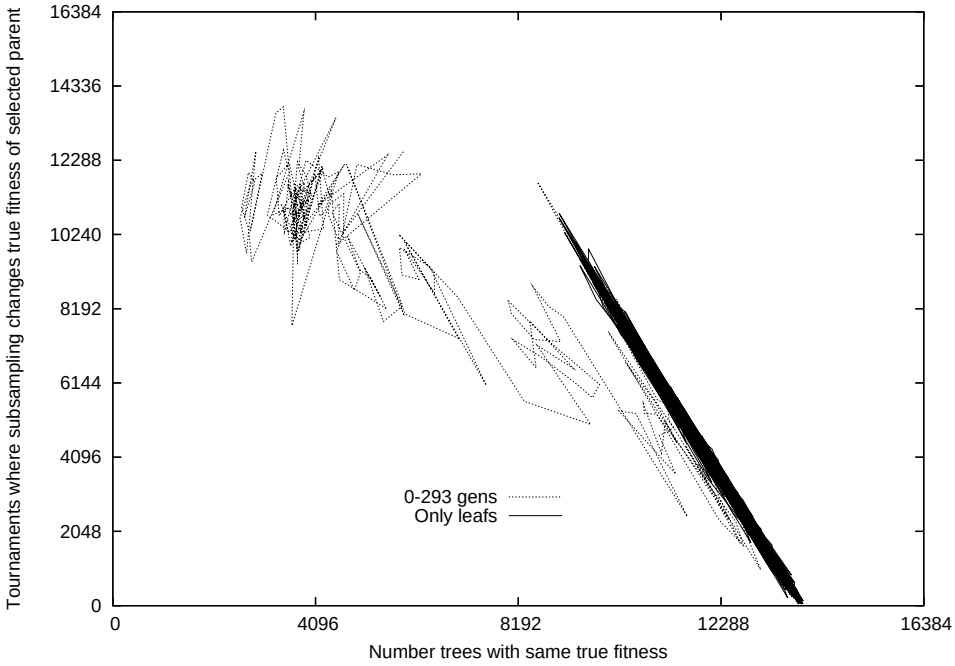

Figure 7: Evolution of fitness convergence (x-axis) v. wrong parent selection (vertical) in the first unsuccessful run with a single test changed every generation. The whole population evolves into tiny programs each consisting of a single leaf and fitness convergence increases. The correlation (-0.998) between convergence and selection after generation 293 is explained in Section 6.2 on page 10.

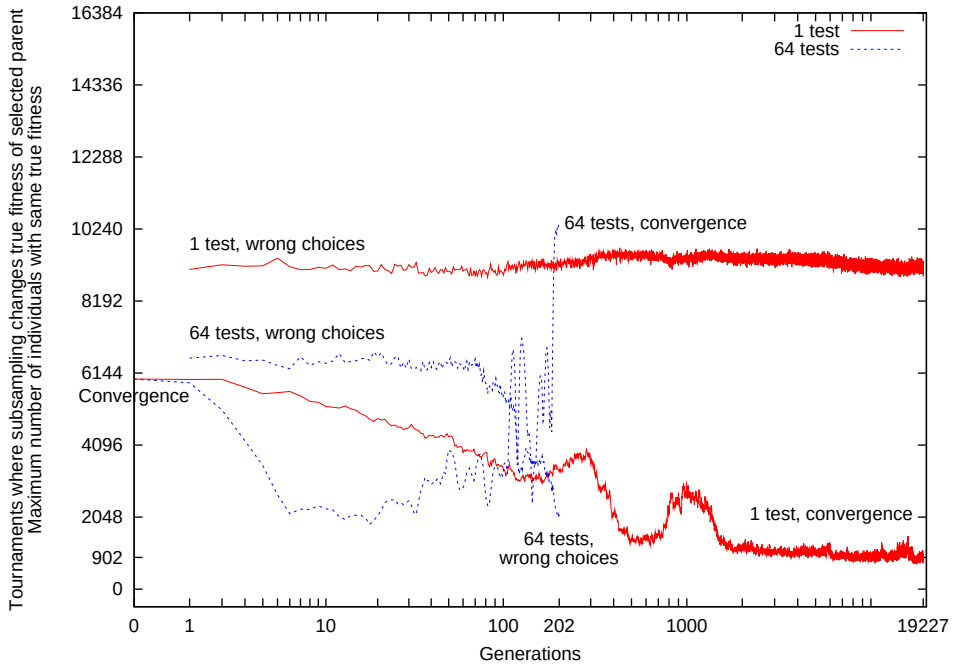

Figure 8: Evolution of number of tournament choosing wrongly and true fitness convergence for successful runs which change the test suite in each tournament. With 64 of 2048 tests, GP finds a 11-Mux solution in generation 202. As the run proceeds, the fraction of incorrect parent choices falls from 41% to an $1/8^{th}$. In contrast, the run with only one test per tournament takes 19 227 generations and the mean fraction of wrong tournaments is 56%.

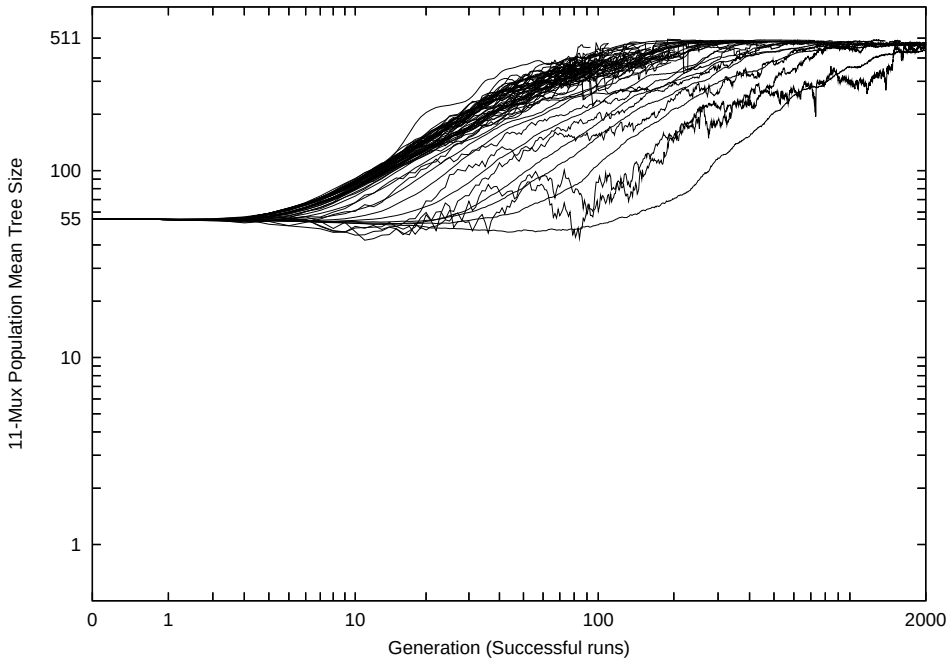

Figure 9: Mean tree size in 11-Mux populations. Medians of up to ten successful runs. Note non-linear scales.

shows all three test suite schemes approach 12 288. By the end of runs with eight fixed tests the population has converged so that almost all programs have the same observed fitness (8). Empirically 84% of tournaments are decided only by chance. With dynamically chosen tests suites it is smaller. E.g. for just one test it is 65%.

6.2 Size of Children

Figures 9 and 10 shows the evolution of the mean size of 11-Mux programs. In 700 of 723 runs the average tree size more than doubles. (Increase in solution size without consequent increase in fitness is known as bloat [Langdon *et al.*, 1999].) However some failing runs do not bloat but instead converge disastrously. At the end of 20 runs with tiny test suits ($\leq$3 tests) changed every generation the population collapses so that all 16 384 trees each consist only of a single leaf. This included all three runs where no tests where run, all of the runs where one test was randomly chosen every generation and nine of the 16 runs which failed where the 2 or 3 tests were selected every generation.

Without any fitness selection (tests=0), the populations do not bloat [Langdon and Poli, 1997] and also collapse to a single leaf. The nature of the 11-Mux problem dictates the address leafs A0–2 have fitness of 1024 and the remaining 8 data leafs D0–7 score 1152. The collapsed populations contain approximately equal numbers of each leaf. Since there is no (observed) fitness the first member of the each randomly chosen tournament will be chosen. The chance of A0–2 being selected as the first member of a tournament is 3/11. However for A0, A1 or A2 to be correctly chosen the tournament must contain only A0–2. If any of the three other members of the tournament are D0–7, then they should have won the tournament, but do not. I.e. the chance that a tournament chooses wrongly is $3/11 \times \left(1 - (3/11)^3\right)$ = 27%. (Corresponding to 4378 in Figure 4.) The actual values, averaged over the last twenty generations are 4364 (once or every generation) and 3806 (every tournament).

When the population is composed only of leafs there are just two fitness values in the population. Random fluctuations cause the proportion to change. When the population is dominated by either, tournaments are more likely to chose correctly. I.e., the chance of selecting wrongly falls as the fraction of individuals with the highest fitness rises (i.e. with increasing population convergence). This gives the near straight line in Figure 7 with slope -0.462. Although the relationship looks linear, it is actually a small segment of a curved relationship.

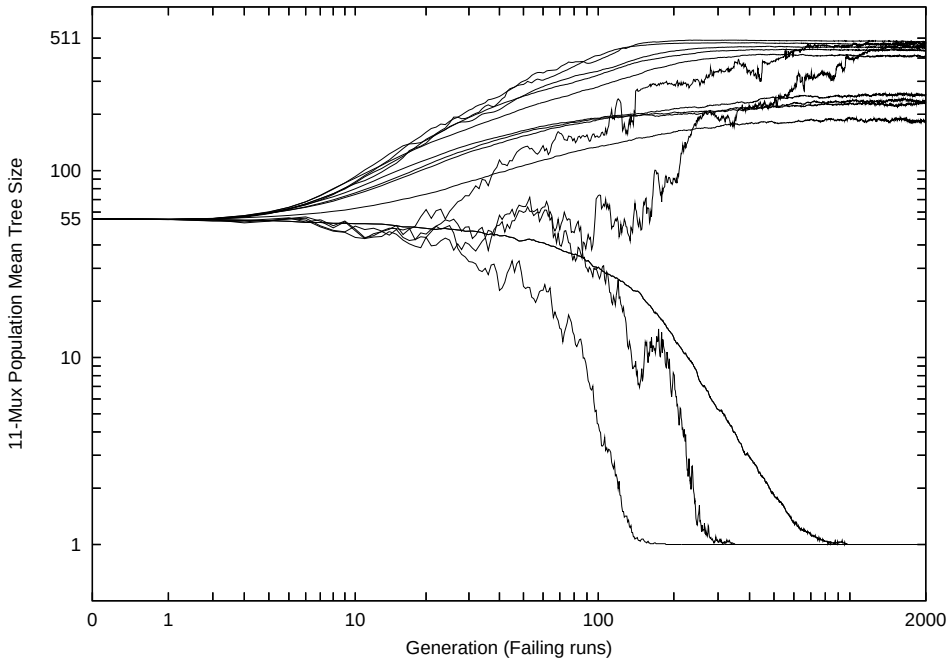

Figure 10: Mean tree size in 11-Mux populations. Medians of up to ten runs which terminated at 2000 generations without finding a solution. Note non-linear scales.

Notice that parts of the 11-Mux fitness landscape are deceptive. For example, whilst data leafs do not score highly, they do score 6% better than average and so can attract the population. Once GP has converged on such programs, neither crossover nor mutation can find higher fitness programs and the population is trapped on a false local peak in the fitness landscape.

6.3 Convergence of Fitness and Phenotypes with Small Test Suites

Although Figure 2 shows evolution with small dynamic test suites is similar to normal GP, it is more evolutionary. For example, on average, there are 70% more intermediate solutions with improving true fitness in the ten runs with one dynamic test. Of course this means the size of each step is reduced. E.g., their median improvement is only 8 compared to 14 in the runs which use all 2048 tests.

As expected, in most cases there is a very strong correlation between the fitness measured on the test suite and the true fitness (measured on all 2048 tests, see Figure 11). Only when the test suite is very small does the correlation fall towards zero. In runs with 4 or 8 fixed tests (which failed of course) the correlation actually becomes negative (suggesting over fitting, note points marked with $+$ or $\times$ in Figure 11). With more than 32 tests, the runs with fixed and variable test suites show little difference in correlation (even though their chances of finding solutions are different). In the absence of over fitting and genetic operations being trapped by tiny programs, GP can (given long enough) reliably evolve solutions with a correlation as low as 0.19.

The size of the breeding population, i.e. the number of individuals chosen to be parents of the next generation, is plotted in Figure 12. As expected in almost all runs it is close to the value predicted by Motoki [Motoki, 2002, p403] for tournaments of size four. Unsuccessful runs with a small fixed test set start with breeding populations near T4=44% but increase the size of their breeding populations as more members of the population have the same fitness. In the limit of random selection 63% of the population contribute to the next generation. Figure 12 shows the narrow change (max 44%) in the size of the breeding population between successful and unsuccessful runs. Whilst extreme convergence to a small number of fitness values does reduce selection and increase the breeding population, the change is modest and so will only make a modest change to the rate with which selection removes diversity from the population.

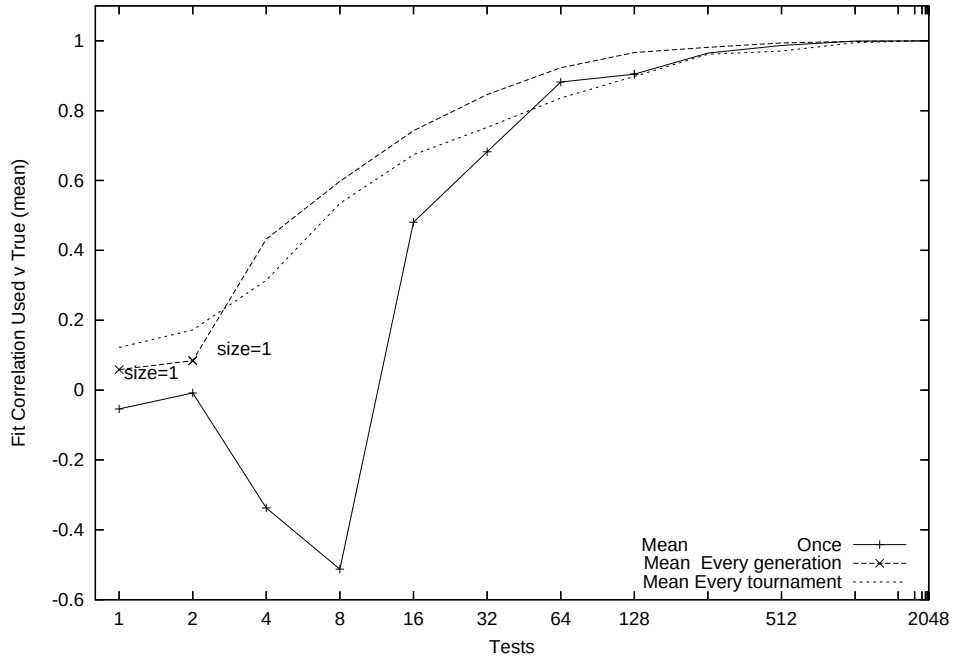

Figure 11: Mean correlation across whole 11-Mux runs between observed fitness (as used by tournament selection) and true fitness for differing test suite sizes. Runs which failed to find a solution marked with $+$ or $\times$.

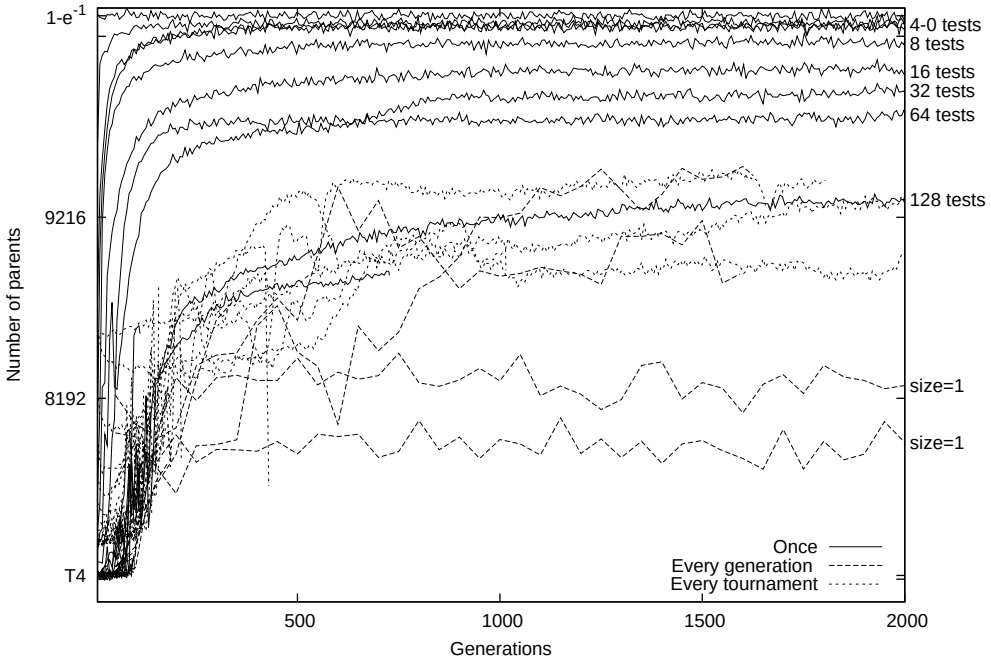

Figure 12: Size of breeding pool. *Most* runs near T4=7189 as expected. T4 label on right axis calculated according to Motoki's loss of diversity formula for tournaments of four [Motoki, 2002, p403]. Runs which fail have slightly larger breeding populations. The limit $1 - e^{-1}$ applies when there is no selection, just sampling noise. Means of 5 or 50 generations.

Table 4: Fraction (as grey scale) of all 2048 tests passed by 11-Mux pops. Black 0%, grey 50%, white 100% of tests passed on average by members of the population. Top row of each graphic: initial pop, generation 1, 2, .. 10, 20, .. 90, 100, 200, .. 1900 and last row first population where a solution was found. The 2048 tests are plotted horizontally. (Sorted so easiest tests are on the left. Same ordering in each plot.) Whilst time is plotted vertically (non linear scale, generation 0 at top of each plot).

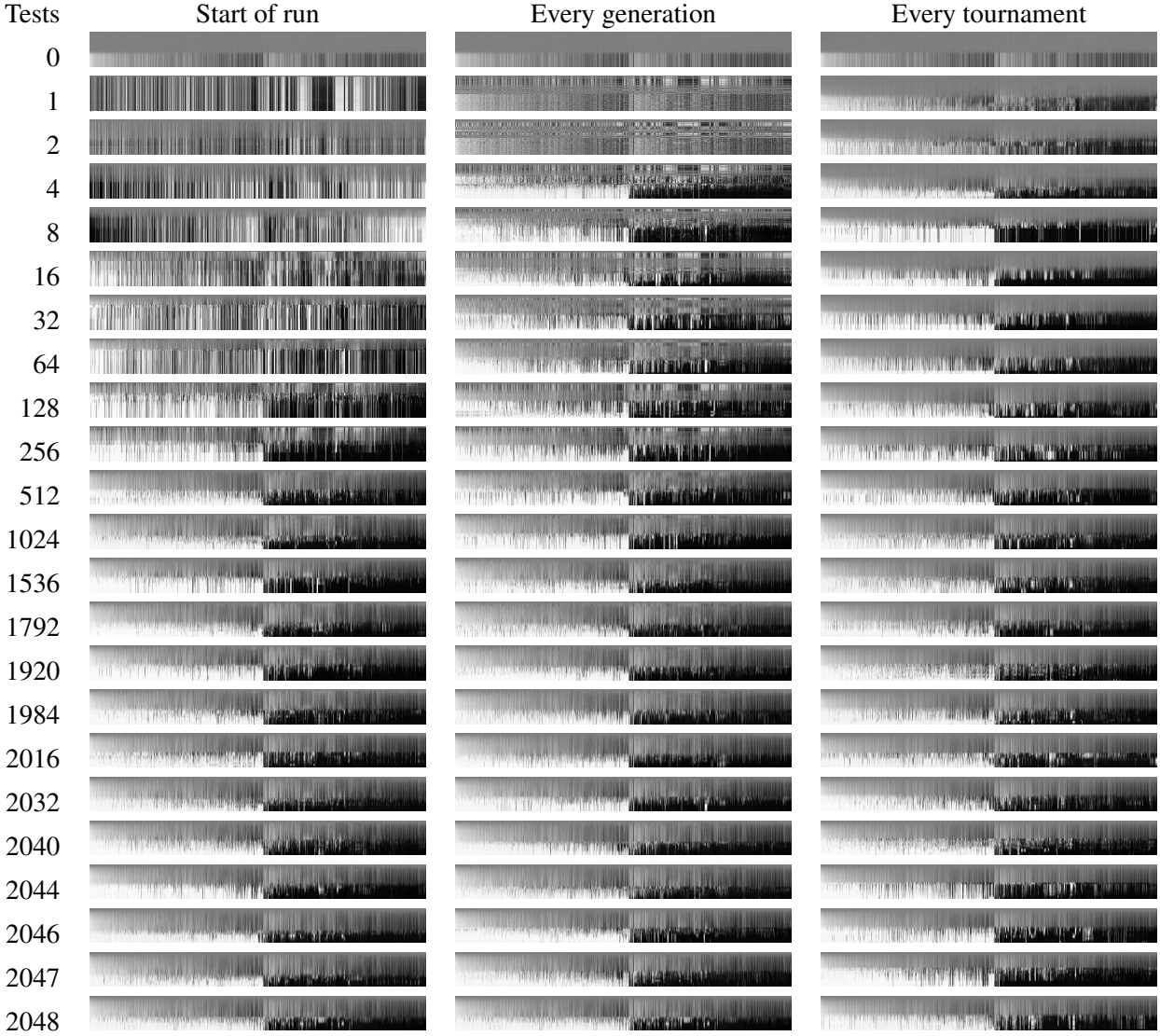

Table 4 shows convergence of phenotypes for the first run of each group of ten. Without any fitness selection (0 tests) the population remains uniform for hundreds of generations (uniform grey) but eventually the population converges to tiny programs which pass particular patterns of tests, meaning some tests are passed much more often than others. As the population converges, this gives rise to vertical black and white stripes.

With more tests, evolution is more complex but we still see many tests being failed by large fractions of the population for many generations. This may be due to "hitch hiking" (as seen in the Royal Road problems [Forrest and Mitchell, 1993; Smith *et al.*, 1993; van Nimwegen *et al.*, 1999]) in which a new better solution propagates through the population spreading both the new tests it passes and also its pattern of tests that it fails. (However, see page 14, only in a few cases does "hitch hiking" cause a population to unlearn a correct answer that has become entrenched in the population.)

Notice that runs converge with most of the population solving a large number (but not all) of the same tests. When a solution is found there are typically still about 46% of the tests which less than a $1/16^{th}$ of the pop-

ulation pass. However typically a parent of the solution will be amongst them. Suggesting whilst evolution is gradual, it proceeds by a series of exceptional individuals. That is, whilst the evolutionary trends in fitness and size show some waggles (see Figures 2, 9 and 10,) the trend is continuous gradual change. However fitness improvement does not come from the bulk population but from exceptional individuals.

Typically at the end of runs about 45% of tests are passed by at least 15/16th of the population. Thus by the final generation most tests are either passed by a small fraction of the population or almost all of it. Hence the last row of the images in Table 4 are mostly black or white, with little grey. In contrast to the first row, which is all grey.

7 Discussion

We have shown in some cases a single randomly chosen test, if it is changed sufficiently often, can be sufficient to reliably drive evolution to solutions. However typically we seek not only to evolve the population but to identify solutions. One test can not identify solutions. So we would be obliged to test part of the population more fully (perhaps every 100 generations). It is not clear when is the best time to check for solutions. If we test too early, there will be no solutions but we can abort tests earlier. Later in the run, the population may contain many solutions, so fully testing only a few individuals which pass the current test suite may be likely to find a solution. However if we decided to fully test everyone who passes the current test suite, this could be expensive. For example, in the run shown in Figure 2 in 39% of the generations more than a quarter of the population could be solutions as far as the 8 tests used in that generation are concerned. Thus for pragmatic reasons we might want to use more than eight tests, even so, particularly for larger problems, the fitness cases used could still be a tiny fraction of the whole.

Having a test suite for each tournament was introduced by [Langdon, 2010] to exploit random number generation on the GPU. When the test suite is different for every tournament, tournaments are deliberately restricted to choosing between individuals which are tested on the same test suite. Obviously the maximum number of fitness levels in a tournament is still four but since tournament members are chosen with reselection there is a likelihood of reselecting individuals and so on average tournaments have ≈2.5 (rather than 4) fitness levels to choose between. This reduced selection pressure (for ≥ 256 tests) may be the reason why per tournament runs need more generations (see dashed line in Figure 1). When there are 8 or fewer tests, tournament selection across the whole population also has fewer fitness levels to choose between. In fact, in both dynamic schemes, tournaments contain on average less than two fitness levels. This may be why (excluding the larger test suite and trapping by tiny programs) both schemes then have similar performance.

With fixed testing, correlation between true and observed fitness is only a partial indicator of success. (But negative correlation suggests a problem.) For example, with 64 tests the correlation between true and observed fitness is much the same regardless of which of the three sampling schemes is used (Figure 11). However all the runs with a static choice of test suite of size 64 fail. At the end of these runs almost the whole population passes all of the 64 tests but the populations are seriously overfitting and typical true scores are near 70%. Whereas at the end of runs where 64 tests are repeatedly changed, observed fitness is mostly less than 2% bigger than true fitness (Figure 13), indicating changing the test suite leads to less bias.

Data in Table 4 support the view that GP populations collect building blocks. To be specific, only in 74 cases (on average) does a population forget a test once it has learnt it. I.e. in an average run there are about 74 tests where $>$7/8th of the population pass it at some point but later the fraction falls below 1/8th. Perhaps because evolution is slower, in 29 of the 40 extended runs with $\leq$4 tests no test was forgotten once a population had learnt it. Whereas on average in the last generation of successful runs 1026 tests are passed by more than 7/8th of the population. Figure 2 shows a logarithmic rise in performance. This might be because GP population solve problems by randomly assembling building blocks (like a coupon collector). The classic coupon collector which takes O($n \log(n)$). The number of generations required approximately follows the dotted line in Figure 1. This suggests the number of coupons n scales as the ratio of the total

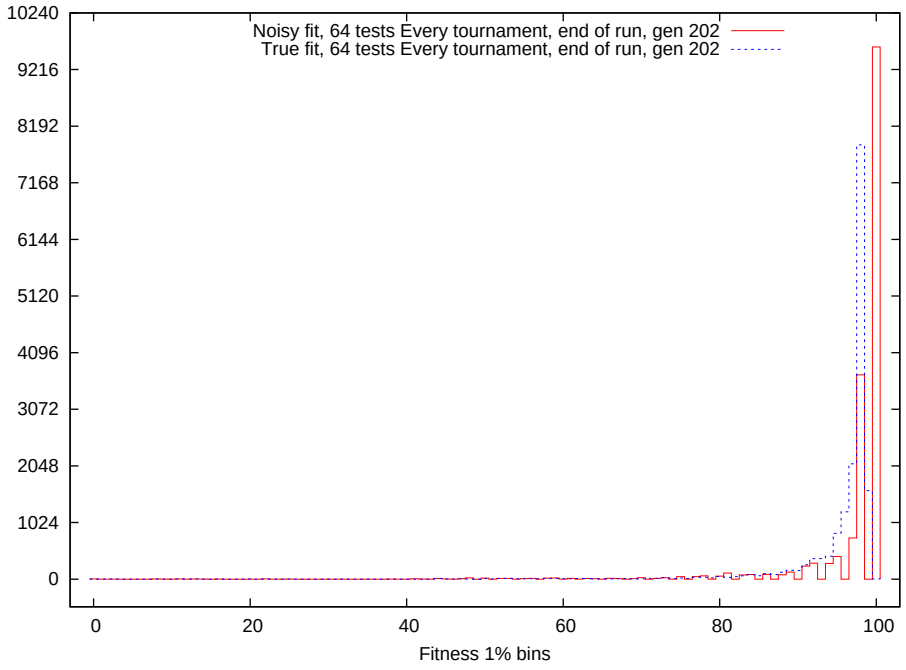

Figure 13: At end of first run with 64 tests changed every tournament more than half the population are measured to be within 1% of the solution and slightly less than half have a true fitness of ≈98%

number of tests divided by the number used. The similarity with the coupon collector suggests perhaps GP assembles solutions from building blocks which are equally difficult to find.

8 Conclusions

Previous ad-hoc work has shown that in many cases genetic programming can evolve general solutions using a fraction of the test suite [Poli *et al.*, 2008]. In the 37-multiplexor, less than a millionth of the whole test suite was needed [Langdon, 2010].

We have investigated GP's ability to generalise. With modest fixed subsets, general solutions can be evolved. However if the subset is further reduced, even a randomly chosen fixed subset can lead to GP overfitting and failing to evolve general solutions.

When the test subset is randomly changed every generation, due to random re-sampling, some tests may be selected several times before all the others have been selected once. With T tests in each generation on average it will take about $16\,800/T$ generations [Feller, 1957] for the whole GP system to be exposed to the whole of the 11-Mux test suite. Yet, as Table 2 shows for ≤ 64 tests per generation, GP can evolve general solutions when not only has no individual seen the whole test suite, but none of its ancestors has. Nor, indeed have all of its ancestors combined, been exposed to the entire test suite. (With 32 tests per generation, it would take on average 473 000 generations to sample the complete 20-Mux test suite, yet on average GP evolved solutions in only 1179 generations.)

Random re-sampling may permit GP to evolve general solutions, even in deceptive problems, from far fewer fitness tests potentially leading to large speed ups.

Acknowledgments

I am grateful for the assistance of Gernot Ziegler of nVidia.

Funded by EPSRC grant EP/G060525/2.

References

[Banzhaf *et al.*, 1998] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, January 1998.

[Bongard and Lipson, 2007] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *PNAS, Proceedings of the National Academy of Sciences of the United States of America*, 104(24):9943–9948, 12 June 2007.

[Christensen and Oppacher, 2002] Steffen Christensen and Franz Oppacher. An analysis of Koza's computational effort statistic for genetic programming. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 182–191, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.

[Feller, 1957] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley and Sons, New York, 2 edition, 1957.

[Forrest and Mitchell, 1993] Forrest and Mitchell. Relative building block fitness and the building block hypothesis. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126, Vail, Colorado, USA, 26-29 July 1992 1993. Morgan Kaufmann.

[Foster, 2001] James A. Foster. Review: Discipulus: A commercial genetic programming system. *Genetic Programming and Evolvable Machines*, 2(2):201–203, June 2001.

[Gathercole and Ross, 1994] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321, Jerusalem, 9-14 October 1994. Springer-Verlag.

[Giacobini *et al.*, 2002] Mario Giacobini, Marco Tomassini, and Leonardo Vanneschi. How statistics can help in limiting the number of fitness cases in genetic programming. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 889, New York, 9-13 July 2002.

[Hillis, 1992] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, volume X of *Sante Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, 1992.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT press, 1992.

[Langdon and Poli, 1997] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London, 23-27 June 1997.

[Langdon *et al.*, 1999] William B. Langdon, Terry Soule, Riccardo Poli, and James A. Foster. The evolution of size and shape. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, 1999.

[Langdon, 1998] William B. Langdon. *Genetic Programming and Data Structures*. Kluwer, Boston, 1998.

[Langdon, 2010] W. B. Langdon. A many threaded CUDA interpreter for genetic programming. In Anna Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 146–158, Istanbul, 7-9 April 2010. Springer.

[Langdon, 2011] W. B. Langdon. Generalisation in genetic programming. In *GECCO 2011*. ACM, 2011.

[Motoki, 2002] Tatsuya Motoki. Calculating the expected loss of diversity of selection schemes. *Evolutionary Computation*, 10(4):397–422, 2002.

[Poli and Langdon, 1999] Riccardo Poli and William B. Langdon. Sub-machine-code genetic programming. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 13, pages 301–323. MIT Press, 1999.

[Poli *et al.*, 2008] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[Ross, 2000a] Brian J. Ross. The effects of randomly sampled training data on program evolution. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 443–450, Las Vegas, Nevada, USA, 10-12 July 2000.

[Ross, 2000b] Brian J. Ross. Probabilistic pattern matching and the evolution of stochastic regular expressions. *Applied Intelligence*, 13:285–300, 2000.

[Smith *et al.*, 1993] Robert E. Smith, Stephanie Forrest, and Alan S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(2):127–149, 1993.

[Sontag, 1998] Eduardo D. Sontag. VC dimension of neural networks. In *Neural Networks and Machine Learning*, pages 69–95. Springer, 1998.

[Teller and Andre, 1997] Astro Teller and David Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[van Nimwegen *et al.*, 1999] Erik van Nimwegen, James P. Crutchfield, and Melanie Mitchell. Statistical dynamics of the royal road genetic algorithm. *Theoretical Computer Science*, 229(1-2):41–102, 1999.