Submitted to Late-Breaking Papers at the Genetic Programming 1997 Conference, J. R. Koza (ed)

Fitness Causes Bloat: Mutation

W. B. Langdon and R. Poli
School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK
{W.B.Langdon,R.Poli}@cs.bham.ac.uk http://www.cs.bham.ac.uk/~wbl, ~rmp
Tel: +44 (0) 121 414 4791, Fax: +44 (0) 121 414 4281

Technical Report: CSRP-97-16
21 May 1997

Abstract

The problem of evolving, using mutation, an artificial ant to follow the Santa Fe trail is used to study the well known genetic programming feature of growth in solution length. Known variously as "bloat", "fluff" and increasing "structural complexity", this is often described in terms of increasing "redundancy" in the code caused by "introns".

Comparison between runs with and without fitness selection pressure, backed by Price's Theorem, shows the tendency for solutions to grow in size is caused by fitness based selection. We argue that such growth is inherent in using a fixed evaluation function with a discrete but variable length representation. With simple static evaluation search converges to mainly finding trial solutions with the same fitness as existing trial solutions. In general variable length allows many more long representations of a given solution than short ones. Thus in search (without a length bias) we expect longer representations to occur more often and so representation length to tend to increase. I.e. fitness based selection leads to bloat.

1 Introduction

The tendency for programs in genetic programming (GP) populations to grow in length has been widely reported [Tackett, 1993; Tackett, 1994; Angeline, 1994; Tackett, 1995; Langdon, 1995; Nordin and Banzhaf, 1995; Soule *et al.*, 1996]. This tendency has gone under various names such as "bloat", "fluff" and increasing "structural complexity". The principal explanation advanced for bloat has been the growth of "introns" or "redundancy", i.e. code which has no effect on the operation of the program which contains it. ([Wu and Lindsay, 1996] contains a survey of recent research in biology on "introns".) Such introns are said to protect the program containing them from crossover [Blickle and Thiele, 1994; Blickle, 1996; Nordin *et al.*, 1995; Nordin *et al.*, 1996]. [McPhee and Miller, 1995] presents an analysis of some simple GP problems designed to investigate bloat. This shows that, with some function sets, longer programs can "replicate" more "accurately" when using crossover. I.e. offspring produced by crossover between longer programs are more likely to behave as their parents than children of shorter programs. [Rosca and Ballard, 1996a] provides a detailed analysis of bloat using tree schemata specifically for GP.

In this paper we advance a more general explanation which should apply generally to any discrete variable length representation and generally to any progressive search technique. That is bloat is not specific to genetic programming applied to trees using tree based crossover but should also be found with other genetic operators and non-population based stochastic search techniques such as simulated annealing and stochastic iterated hill climbing.

In the next section we repeat our argument that bloat is inherent in variable length representations such as GP [Langdon and Poli, 1997b]. In Sections 3 and 4 we expand our previous analysis of a typical GP demonstration problem to include solution by mutation in place of crossover, again showing that it suffers from bloat and also showing that bloat is not present in the absence of fitness based selection. Section 5 describes the results we have achieved and this is followed in Section 6 by a discussion of the potential advantages and disadvantages of bloat and possible responses to it. Finally Section 7 summarises our conclusions.

2 Bloat in Variable Length Representations

In general with variable length discrete representations there are multiple ways of representing a given behaviour. If the evaluation function is static and concerned only with the quality of trial solutions and not with their representations then all these representations have equal worth. If the search strategy were unbiased, each of these would be equally likely to be found. In general there are many more long ways to represent a specific behaviour than short representations of the same behaviour. Thus we would expect a predominance of long representations.

Practical search techniques are biased. There are two common forms of bias when using variable length representations. Firstly search techniques often commence with simple (i.e. short) representations, i.e. they have an in built bias in favour of short representations. Secondly they have a bias in favour of continuing the search from previously discovered high fitness representations and retaining them as points for future search. I.e. there is a bias in favour of representations that do at least as well as their initiating point(s).

On problems of interest, finding improved solutions is relatively easy initially but becomes increasingly more difficult. In these circumstances, especially with a discrete fitness function, there is little chance of finding a representation that does better than the representation(s) from which it was created (cf. "death of crossover" [Langdon, 1996a, page 222]). So the selection bias favours representations which have the same fitness as those from which they were created.

In general the easiest way to create one representation from another and retain the same fitness is for the new representation to represent identical behaviour. Thus, in the absence of improved solutions, the search may become a random search for new representations of the best solution found so far. As we said above, there are many more long representations than short ones for the same solution, so such a random search (other things being equal) will find more long representations than short ones. In GP this has become known as bloat.

3 The Artificial Ant Problem

The artificial ant problem is described in [Koza, 1992, pages 147–155]. It is a well studied problem and was chosen as it has a simple fitness function. Briefly the problem is to devise a program which can successfully navigate an artificial ant along a twisting trail on a square 32×32 toroidal grid. The program can use three operations, Move, Right and Left, to move the ant forward one square, turn to the right or turn to the left. Each of these operations take one time unit. The sensing function IfFoodAhead looks into the square the ant is currently facing and then executes one of its two arguments depending upon whether that square contains food or is empty. Two other functions, Prog2 and Prog3, are provided. These take two and three arguments respectively which are executed in sequence.

The artificial ant must follow the "Santa Fe trail", which consists of 144 squares with 21 turns. There are 89 food units distributed non-uniformly along it. Each time the ant enters a square containing food the ant eats it. The amount of food eaten is used as the fitness measure of the control program.

Table 1: Ant Problem

Objective:	Find an ant that follows the "Santa Fe trail"
Terminal set:	Left, Right, Move
Functions set:	IfFoodAhead, Prog2, Prog3
Fitness cases:	The Santa Fe trail
Fitness:	Food eaten
Selection:	Tournament group size of 7, non-elitist, generational
Wrapper:	Program repeatedly executed for 600 time steps.
Population Size:	500
Max program size:	500
Initial population:	Created using "ramped half-and-half" with a maximum depth of 6
Parameters:	90% mutation, 10% reproduction
Termination:	Maximum number of generations G = 50

The evolutionary system we use is identical to [Langdon and Poli, 1997b] except the crossover operator is replaced by mutation. The details are given in Table 1, parameters not shown are as [Koza, 1994, page 655]. On each version of the problem 50 independent runs were conducted.

Note in these experiments we allow the evolved programs to be far bigger than required to solve the problem. For example the 100% correct solution given in [Koza, 1992, page 154] takes about 543 time steps to traverse the Santa Fe trail but has a length of only 18 nodes and this is not the most compact solution possible.

4 Tree Mutation

For our purposes it is necessary that the mutation operator be able to change the size of the chromosomes it operates. Ideally this should be unbiased in the sense of producing, of itself, no net change in size. In the general case, this is difficult to contrive, however the following operator, in all cases examined, produces offspring which are on average almost the same size as their parent (cf. Section 5.6). (NB this may not be the case with different ratios of node branching factors in either the terminal/function set or in the evolving populations.)

The mutation operator selects uniformly at random a node (which may be a function or terminal) and replaces it and the subtree descending from it with a randomly

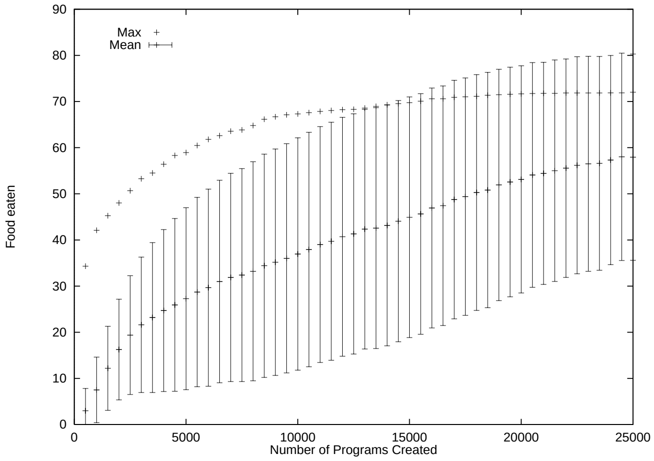

Figure 1: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs.

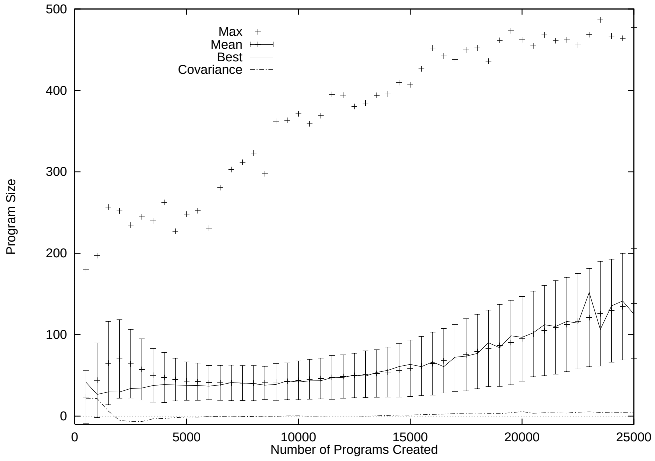

Figure 2: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Covariance of length and normalised rank based fitness shown dotted. Means of 50 runs.

created tree. The new tree is created using the same "half and half" method used to create the initial population [Koza, 1992, Page 92–93] however its maximum height is chosen at random from one to the height of the tree it is to replace. Finally a check is made that the new program does not exceed the maximum allowed size. If it does another new tree is randomly created but with its maximum allowed height reduced by one. If no acceptable size replacement has been found by the time the depth has been reduced to 2, a copy of the parent is used and no mutation is made. NB a terminal will always be replaced by a randomly selected terminal (i.e. there is no change in size) but a function can be replaced by a larger (but not deeper) or smaller tree, so that the program's size may change.

5 Results

5.1 Standard Runs

In 50 independent runs 10 found "ants" that could eat all the food on the Santa Fe trail within 600 time steps. The evolution of maximum and mean fitness averaged across all 50 runs is given in Figure 1. (In all cases the average minimum fitness is near zero). These curves show the fitness behaving as with crossover with both the maximum and average fitness rising rapidly initially but then rising more slowly later in the runs. However with mutation the rise in mean fitness is less rapid, e.g. it passes 50 at generation 36 compared to generation 17 with crossover. The population converges in the sense that the average fitness approaches the maximum fitness. However the spread of fitness values of the children produced in each generation remains large and children which eat either no food or only one food unit are still produced even in the last generation.

Figure 2 shows the evolution of maximum and mean program size averaged across all 50 runs. We see in the first four generations the average program length grows rapidly from an initial size of 23.5 to 70.2. Unlike with crossover the average program length then falls to about 41 but the maximum program size grows continually (excepting some noisy oscillation) until the maximum program size is approached. The mean program size remains fairly constant until generation 19 when bloat sets in and it begins to rise progressively towards the maximum.

As with crossover the mean program length grows faster than the size of the "best" program within the population. Between generations 2 and 12 the mean length exceeds that of the "best" program (cf. Section 5.1.1). In the later generations the mean program length and length of the best program on average lie close to each other.

5.1.1 Fitness is Necessary for Bloat – Price's Theorem Applied to Representation Size

Price's Covariance and Selection Theorem [Price, 1970] from population genetics relates the expected change in frequency of a gene Δq in a population from one generation to the next, to the covariance of the gene's frequency in the original population with the number of offspring z produced by individuals in that population:

$$\Delta q = \frac{\mathrm{Cov}(z, q)}{\overline{z}} \tag{1}$$

We have used it to help explain the evolution of the number of copies of functions and terminals in GP populations [Langdon, 1996a; Langdon and Poli, 1997a]. In our experiments the size of the population does not

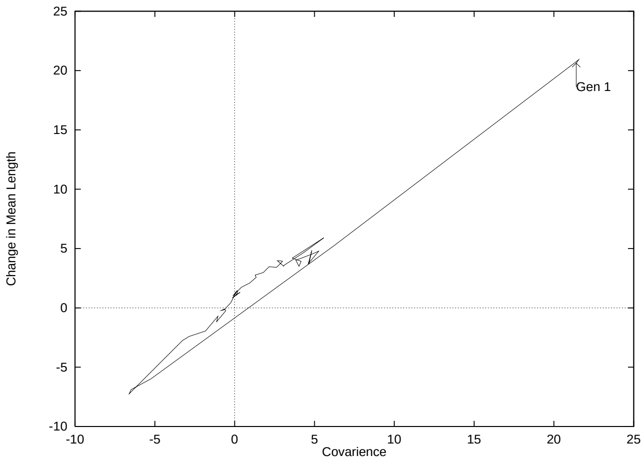

Figure 3: Covariance of program length and normalised rank based fitness v. change in mean length in next generation. Means of 50 runs.

change so $\overline{z} = 1$ and the expected number of children is given by the parent's rank so in large populations the expected change is approximately $\mathrm{Cov}(t(r/p)^{t-1}, q)$ as long as crossover is random. (t is the tournament size and r is each program's rank within the population of size p.)

Where representation length is inherited, such as in GP and other search techniques, Equation 1 should hold for representation length. More formally Price's theorem applies (provided length and genetic operators are uncorrelated) since representation length is a *measurement function* of the genotype [Altenberg, 1995, page 28]. If it held exactly a plot of covariance vs. change in mean length would be a straight line (assuming $\overline{z}$ is constant). The solid line plot in Figure 3 shows good agreement between theory and measurement. (The slight discrepancy is due to a small bias in the mutation operator, which is plotted in Figure 11.)

Essentially Equation 1 gives a quantitative measurement of the way genetic algorithms (GAs) search. If some aspect of the genetic material is positively correlated with fitness then, other things being equal, the next generation's population will on average contain more of it. If it is negative, then the GA will tend to reduce it in the next generation.

For a given distribution of representation lengths Equation 1 says the change in mean representation length will be linearly related to the selection pressure, z. This provides some theoretical justification for the claim [Tackett, 1994, page 112] that "average growth in size ... is proportional to selection pressure". His claim is based upon experimental measurements with a small number of radically different selection and crossover operators on Tackett's "Royal Road" problem. (Solutions to the Royal Road problem are required to have prespecified syntactic properties which mean "that the optimal solution has a fixed size and does not admit extraneous code selections". Such solutions are not executable programs.)

Referring back to Figure 2 and considering the length of the best solution and the covariance in the first four generations, we see the covariance correctly predicts the mean length of programs will increase. In contrast if we assumed the GA would converge towards the individual with the best fitness in the population, the fact that the mean length is higher than the mean length of the best individual would lead us to predict a fall in mean program length. That is Price's Theorem is the better predictor. This is because it considers the whole population rather than just one (albeit the best in the population).

Where fitness selection is not used (as in the following sections), each individual in the population has an equal chance of producing children and so the covariance is always zero. Therefore Price's Theorem predicts on average there will be no change in length.

5.2 No Selection

A further 50 runs were conducted using the same initial populations and no fitness selection. As with crossover no run found a solution and the maximum, mean and other fitness statistics fluctuate a little but are essentially unchanged (cf. Figure 4). In contrast to the case with crossover only, our mutation operator has a slight bias. In the first 50 generation this causes the population to gradually increase in size at the rate of $\frac{1}{3}$ of a node per generation (cf. Figure 5). Given mutation produces random changes in length we expect the spread of lengths to gradually increase. Unlike with crossover where the mean maximum program size fell slowly, the average maximum size grows towards the upper limit and the mean standard deviation grows steadily from 32.8 to 67.1 (In contrast with crossover the mean standard deviation fell from 32.8 in the initial population to 27.3 at the end of the runs).

5.3 Removing Selection

A final 50 runs were conducted in which fitness selection was removed after generation 25 (i.e. these runs are identical to those in Section 5.1 up to generation 25). The evolution of maximum and mean fitness averaged across all 50 runs is given in Figure 6. As expected Figure 6 shows in the absence of fitness selection the fitness of the population quickly falls.

Of the ten runs which found ants that could eat all the food on the Santa Fe trail within 600 movements, two first found solutions after generation 26 and so failed to find solutions when fitness selection was removed, however two other runs did find solutions when it was removed. In all cases mutation eliminated solutions from the population after fitness selection was removed, typically within 13 generations.

After fitness selection is removed, the length of pro-

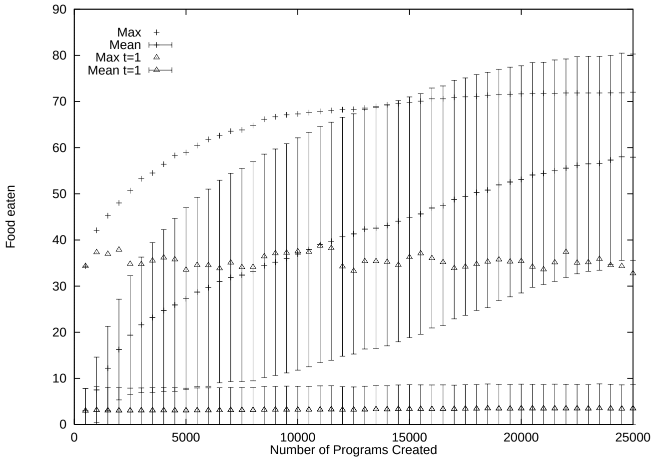

Figure 4: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs comparing tournament sizes of 7 and 1.

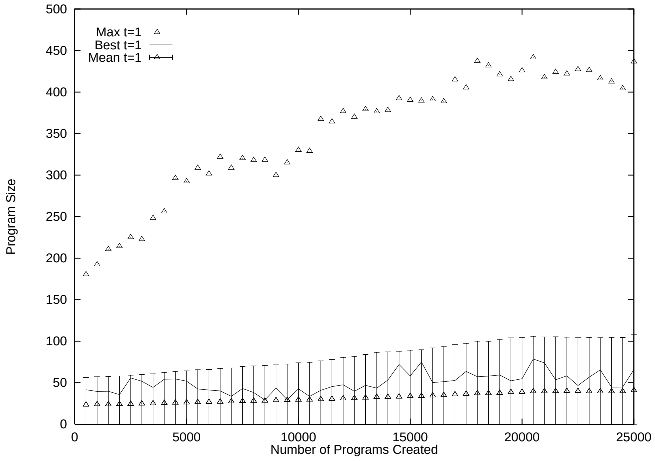

Figure 5: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Means of 50 runs with random selection.

grams behaves much as when there is no selection from the start of the run. I.e. bloat is replaced by the slow growth associated with the mutation operator and the spread of program lengths gradually increases (cf. Figure 7)

5.4 Correlation of Fitness and Program Size

Figure 8 plots the correlation coefficient of program size and amount of food eaten. (Correlation coefficients are equal to the covariance after it has been normalised to lie in the range $-1 \ldots +1$. By considering food eaten we avoid the intermediate stage of converting program score to expected number of children required when applying Price's Theorem.) Figure 8 shows in all cases there is a positive correlation between program score (rather than fitness) and length of programs. In the initial population, long random programs do better than short ones. This may be because they are more likely to contain useful primitives (such as Move) than short programs. Also short ones make fewer moves before they are reinitialised and re-executed. This may increase the chance of them falling into unproductive short cyclic behaviour that longer, more random, programs can avoid. Selection quickly drives the population towards these better individuals, so reducing the correlation. With our mutation operator it falls to a much lower level than with crossover (when it was typically in the region of $+0.3$). This might indicate that mutation is less disruptive than crossover. In the absence of selection (or after selection has been removed halfway through a run) mutation tends to randomise the population so that the correlation remains (or tends towards) that in the initial population.

5.5 Effect of Mutation on Fitness

In the initial generations mutation is disruptive (cf. Figure 9) with 64.6% producing a child with a different score from its parent. However mutation evolves, like crossover, to become less disruptive. By the end of the run 69.1% of mutants have the same score as their parent.

The range of change of fitness is highly asymmetric; many more children are produced which are worse than their parent than those that are better. By the end of the run, only 0.02% of the population are fitter than their parent. Similar behaviour has been reported using crossover on other problems [Nordin *et al.*, 1996] [Rosca and Ballard, 1996b, page 183] [Langdon, 1996a, Chapter 7].

5.6 Non-Disruptive Mutation and Program Length

In Section 2 we argued that there are more long programs with a given performance than short ones and so a random search for programs with a given level of performance is more likely to find long programs. We would expect generally (and it has been reported on a number of problems) that crossover produces progressively fewer improved solutions as evolution proceeds and instead in many problems selection drives it to concentrate upon finding solutions with the same fitness. Apart from the minimum and maximum size restrictions mutation is nearly random so we would expect to see it finding more long solutions than short ones. The distribution of changes in program size produced by our mutation operator has a large central spike of programs with the same length and rapidly non-montonic falling tails either

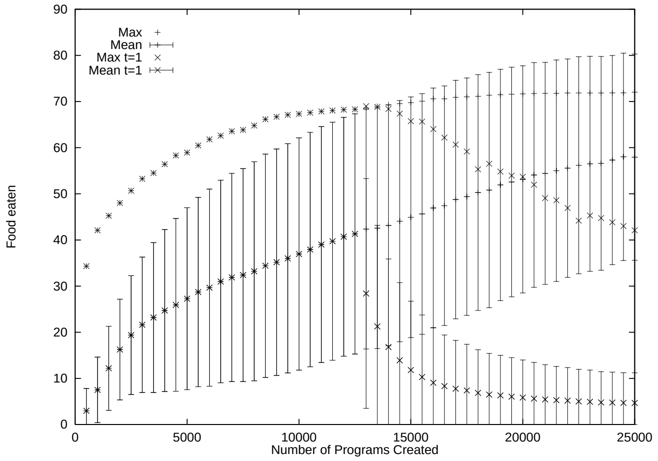

Figure 6: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs showing effect of removing fitness selection.

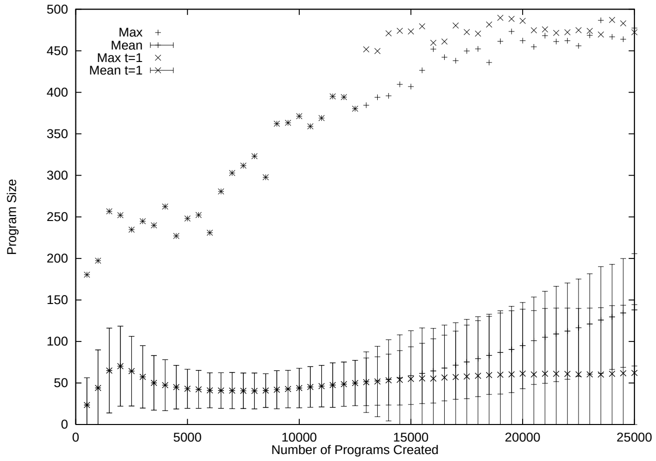

Figure 7: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Means of 50 runs showing effect of removing fitness selection.

side of zero. Figure 10 plots the change in length of only those programs produced which have worse fitness than their parent. Like when considering all mutations Figure 10 has a large central spike of programs with the same length and rapidly falling tails either side of zero. (There is an asymmetry as most programs selected for mutation are nearer to the minimum program size than to the maximum.) The proportion of children produced by mutation which are identical to their parent remains on average fairly constant at about 13% throughout the runs.

As with crossover, mutation becomes increasingly less disruptive (after 200 generations on average over ten runs only 10% of mutations produce a change in fitness). We see from Figure 11 on average mutation makes little change to the length of programs, however, after bloat becomes established, mutations which don't change fitness on average reduce program length less than other mutations. I.e. once the change in the fitness of the population slows, program size bloats despite length changes introduced by mutation.

5.7 *Fraction of Bloated Programs Used*

Every program terminal uses one energy unit each time it is executed. Thus the amount of energy used when executing a program gives the actual number of terminals it has used. From Figure 12 we see (like with crossover) typically only 5–10 terminals are used per execution (a move and four turns allows the ant to move forward whilst searching either side of it). While different parts of the program could be executed each time it is called, we expect the large programs evolved contain only small functional parts. In one bloated solution only 18 of 462 nodes in the program could be executed.

6 Discussion

6.1 *Do we Want to Prevent Bloat?*

From a practical point of view the machine resources consumed by any system which suffers from bloat will prevent extended operation of that system. However in practice we may not wish to operate the system continually. For example it may quickly find a satisfactory solution or better performance may be achieved by cutting short its operation and running it repeatedly with different starting configurations [Koza, 1992, page 758].

In some data fitting problems growth in solution size may be indicative of "over fitting", i.e. better matching on the test data but at the expense of general performance. For example [Tackett, 1993, page 309] suggests "parsimony may be an important factor not for 'aesthetic' reasons or ease of analysis, but because of a more direct relationship to fitness: there is a bound on the 'appropriate size' of solution tree for a given problem".

By providing a "defence against crossover" [Nordin *et al.*, 1996, page 118] bloat causes the production of many programs of identical performance. These can consume the bulk of the available machine resources and by "clogging up" the population may prevent GP from effectively searching for better programs.

On the other hand [Angeline, 1994, page 84] quotes results from fixed length GAs in favour of representations which include introns, to argue we should "not ... impede this emergent property [i.e. introns] as it may be crucial to the successful development of genetic programs". Introns may be important as "hiding places" where genetic material can be protected from the current effects of selection and so retained in the population. This may be

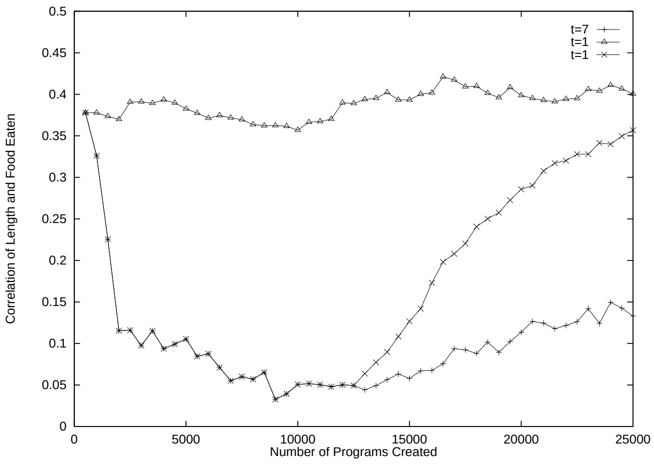

Figure 8: Correlation of program length and fitness: normal runs, runs without selection and runs with selection removed halfway through. Means of 50 runs.

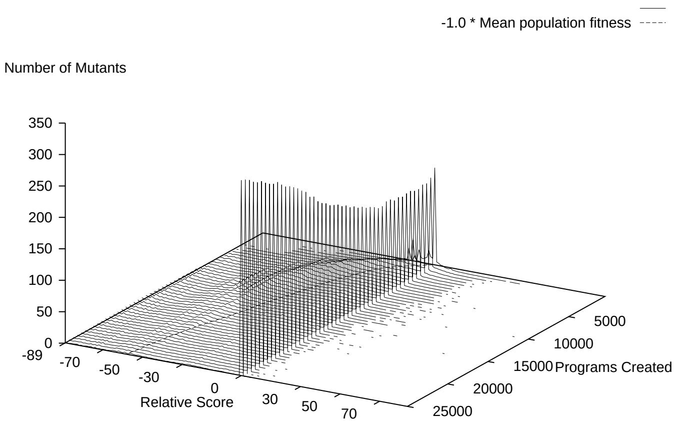

Figure 9: Fitness relative to Parent: normal runs. Means of 50 runs.

especially important where fitness criteria are dynamic. A change in circumstance may make it advantageous to execute genetic material which had previously been hidden in an intron. [Haynes, 1996] shows an example where a difficult GP representation is improved by deliberately inserting duplicates of evolved code.

In complex problems it may not be possible to test every solution on every aspect of the problem and some form of dynamic selection of test cases may be required [Gathercole and Ross, 1994]. For example in some cases co-evolution has been claimed to be beneficial to GP. If the fitness function is sufficiently dynamic, will there still be an advantage for a child in performing identically to its parents? If not, will we still see such explosive bloat?

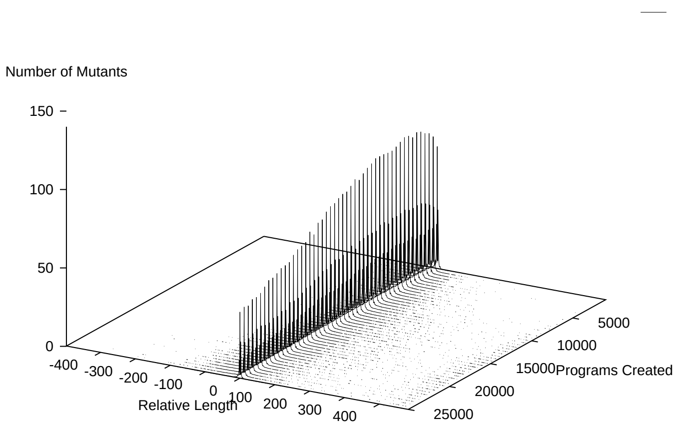

Figure 10: Change in length of offspring relative to parent where score is worse: normal runs. Means of 50 runs.

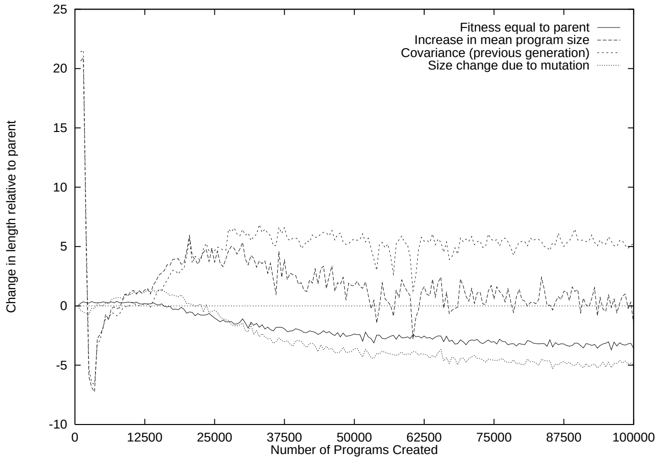

Figure 11: Mean change in length of offspring relative to parent: normal runs. Means of 50 runs.

6.2 *Three Ways to Control Bloat*

Three methods of controlling bloat have been suggested. Firstly, and most widely used (e.g. in these experiments), is to place a universal upper bound either on tree depth [Koza, 1992] or program length. ([Gathercole and Ross, 1996; Langdon and Poli, 1997a] discuss unexpected problems with this approach).

The second (also commonly used) is to incorporate program size directly into the fitness measure (often called parsimony pressure) [Koza, 1992; Zhang and Mühlenbein, 1993; Iba *et al.*, 1994]. [Rosca and Ballard, 1996a] gives an analysis of the effect of parsimony pressure which varies linearly with program length. Multi-objective fitness measures where one objective is compact or fast programs have also been used [Langdon, 1996b].

The third method is to tailor the genetic operations. [Sims, 1993, page 469] uses several mutation operators

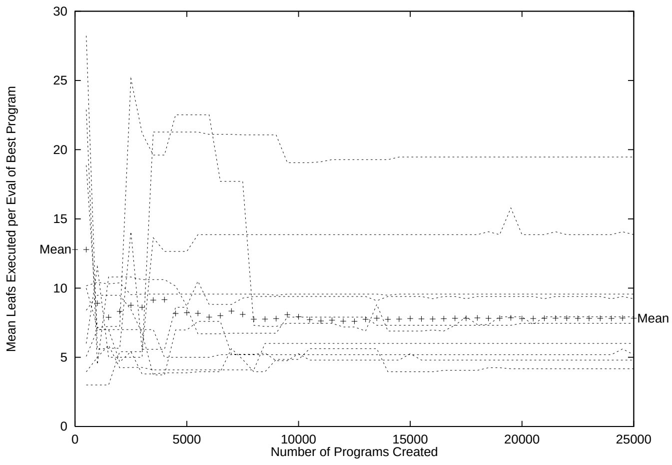

Figure 12: Energy used per call of "best" program. 10 of 50 normal runs.

but adjusts their frequencies so a "decrease in complexity is slightly more probable than an increase". [Blickle, 1996] suggests targeting genetic operations at redundant code. This is seldom used, perhaps due to the complexity of identifying redundant code. [Soule *et al.*, 1996] showed bloat continuing despite their targeted genetic operations. Possibly this was because of the difficulty of reliably detecting introns. I.e. there was a route whereby the GP could evolve junk code which masqueraded as being useful and thereby protected itself from removal. While [Rosca and Ballard, 1996a] propose a method where the likelihood of potentially disruptive genetic operations increases with parent size.

7 Conclusions

We have generalised existing explanations for the widely observed growth in GP program size with successive generations (*bloat*) to give a simple statistical argument which should be generally applicable both to GP and other systems using discrete variable length representations and static evaluation functions. Briefly, in general simple static evaluation functions quickly drive search to converge, in the sense of concentrating the search on trial solutions with the same fitness as previously found trial solutions. In general variable length allows many more long representations of a given solution than short ones of the same solution. Thus (in the absence of a parsimony bias) we expect longer representations to occur more often and so representation length to tend to increase. I.e. fitness based selection leads to bloat.

In earlier work [Langdon and Poli, 1997b] we took a typical GP problem and demonstrated with fitness selection it suffers from bloat whereas without selection it does not. In Sections 3, 4 and 5 we repeated these experiments replacing crossover with mutation and showed fitness selection can still cause bloat. We have demonstrated that if fitness selection is removed, bloat is stopped and program size changes little on average. As expected in the absence of selection, mutation is free to change program size at random and the range of sizes increases, as does the mean size (albeit very slowly). In contrast crossover of small programs typical of the initial population does not seem so free and random growth was not observed in our earlier experiments. Detailed measurement of mutation confirms after an extended period of evolution, most are not disruptive (i.e. most children have the same fitness as their parents).

In Section 5.1.1 we apply Price's Theorem to program lengths within evolving populations. We confirm experimentally that it fits unless restrictions on program size have significant impact. We used Price's Theorem to prove fitness selection is required for a change in average representation length. In Section 6 we discussed the circumstances in which we need to control bloat and current mechanisms which do control it but suggest a way forward may be to consider more complex dynamic fitness functions.

Acknowledgements

This research was funded by the Defence Research Agency in Malvern.

References

[Altenberg, 1995] Lee Altenberg. The Schema Theorem and Price's Theorem. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, 1995. Morgan Kaufmann.

[Angeline, 1994] Peter John Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*.

[Blickle and Thiele, 1994] Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, 1994. Max-Planck-Institut für Informatik (MPI-I-94-241).

[Blickle, 1996] Tobias Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*. PhD thesis, Swiss Federal Institute of Technology, 1996.

[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*, 1994.

[Gathercole and Ross, 1996] Chris Gathercole and Peter Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In John R. Koza, etal, editors, *Genetic Programming 1996*.

[Haynes, 1996] Thomas Haynes. Duplication of coding segments in genetic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 344–349, Portland, OR, August 1996.

[Iba *et al.*, 1994] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*. 1994.

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

[Koza, 1994] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. 1994.

[Langdon and Poli, 1997a] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza, etal, editors, *Genetic Programming 1997* Morgan Kaufmann.

[Langdon and Poli, 1997b] W. B. Langdon and R. Poli. Fitness causes bloat. Technical Report CSRP-97-09, University of Birmingham, 26 March 1997. Submitted to WSC2.

[Langdon, 1995] W. B. Langdon. Evolving data structures using genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*,

[Langdon, 1996a] W. B. Langdon. *Data Structures and Genetic Programming*. PhD thesis, University College, London, 27 September 1996.

[Langdon, 1996b] William B. Langdon. Data structures and genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 20.MIT Press, 1996.

[McPhee and Miller, 1995] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Morgan Kaufmann.

[Nordin and Banzhaf, 1995] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*.

[Nordin *et al.*, 1995] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, 9 July 1995.

[Nordin *et al.*, 1996] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 6,.1996.

[Price, 1970] George R. Price. Selection and covariance. *Nature*, 227, August 1:520–521, 1970.

[Rosca and Ballard, 1996a] Justinian P. Rosca and Dana H. Ballard. Complexity drift in evolutionary computation with tree representations. Technical Report NRL5, University of Rochester, 1996.

[Rosca and Ballard, 1996b] Justinian P. Rosca and Dana H. Ballard. Discovery of subroutines in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*.

[Sims, 1993] K. Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9:466–476, 1993.

[Soule *et al.*, 1996] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, etal, editors, *Genetic Programming 1996*, pages 215–223, 1996. MIT Press.

[Tackett, 1993] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, Morgan Kaufmann.

[Tackett, 1994] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, 1994.

[Tackett, 1995] Walter Alden Tackett. Greedy recombination and genetic search on the space of computer programs. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*.

[Wu and Lindsay, 1996] Annie S. Wu and Robert K. Lindsay. A survey of intron research in genetics. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature IV*. Springer-Verlag.

[Zhang and Mühlenbein, 1993] Byoung-Tak Zhang and Heinz Mühlenbein. Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, 7:199–220, 1993.