$2^{nd}$ On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2) 23–27 June 1997

# Fitness Causes Bloat

**W. B. Langdon, 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*

**Keywords:** genetic programming, structural complexity, introns, Occam's Razor, MDL, Price's Theorem

## Abstract

The problem of evolving 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. That is 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 [1, 2, 3, 4, 5, 6, 7]. 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. ([8] contains a survey of recent research in biology on "introns"). Such introns are said to protect the program containing them from crossover [9, 10, 11]. [12] 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. That is offspring produced by crossover between longer programs are more likely to behave as their parents than children of shorter programs. [13] 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 and 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 expand our argument that bloat is inherent in variable length representations such as GP. In Sections 3 and 4 we analyse a typical GP demonstration problem, showing that it suffers from bloat but 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 potential 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 a partial solution and not with its representation 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.

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: | 500 |
| Initial population: | Created using “ramped half-and-half” with a maximum depth of 6 |
| Parameters: | 90% one child crossover, no mutation. 90% of crossover points selected at functions, remaining 10% selected uniformly between all nodes. |
| Termination: | Maximum number of generations G = 50 |

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. That is 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” [14, 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 [15, 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 $32 \times 32$ 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 it eats it. The amount of food eaten is used as the fitness measure of the controlling program.

## 4 GP Parameters

Our GP system was set up to be the same as given in [15, pages 147–155] except the populations were allowed to continue to evolve even after an ant succeeded in traversing the whole trail, each crossover produces one child rather than two, tournament selection was used and the ants were allowed 600 operations (Move, Left, Right) to complete the trail. The details are given in Table 1, parameters not shown are as [16, page 655]. On each version of the problem 50 independent runs were conducted.

Since bloating is widespread in GP it is common to impose a limit on the size of programs. Commonly this is in

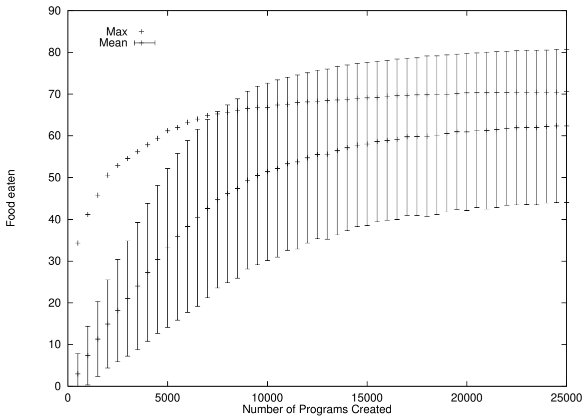


Figure 1: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs.

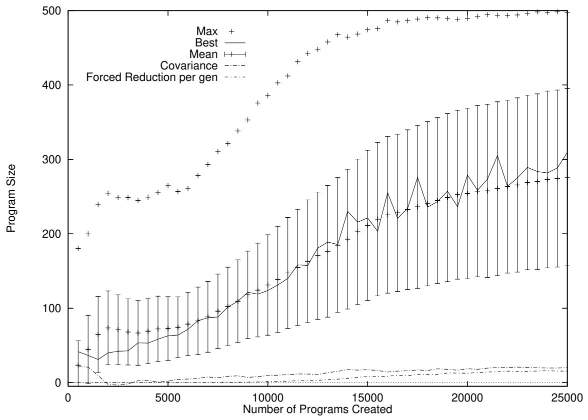


Figure 2: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Means of 50 runs.

the region of 50, although Koza imposed a maximum depth restriction of 17 on his evolved programs. As we are studying program size the relatively large limit of 500 was used. As we shall see, this allows three separate stages of evolution to be studied: the initial population, bloat and bloat constrained by an upper bound. Note a limit of 500 allows the evolved programs to be far bigger than required to solve the problem. For example the 100% correct solution given in [15, 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.

# 5 Results

## 5.1 Standard Runs

In 50 independent runs 6 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 GP fitness behaving as expected with both the maximum and average fitness rising rapidly initially but then rising more slowly later in the runs. The GP 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 73.5. Program size remains fairly static from generation 4 to generation 11, when it begins to grow progressively towards the maximum allowed program size of 500.

Surprisingly the mean program length grows faster than the size of the best program within the population and between generations 2 and 12 it exceeds it (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. The size of the best program refers to that of a single individual. In generations with more than one individual program having the top score, one such program is chosen at random and labelled the "best". This random choice leads to random fluctuations in apparent program size which can be seen in Figure 2 after generation 27 despite averaging over 50 runs.

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

Price's Covariance and Selection Theorem [17] from population genetics relates the expected change in frequency of a gene $\Delta q$ 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 (see Equation 1). We have used it to help explain the evolution of the number of copies of functions and terminals in GP populations [14, 18].

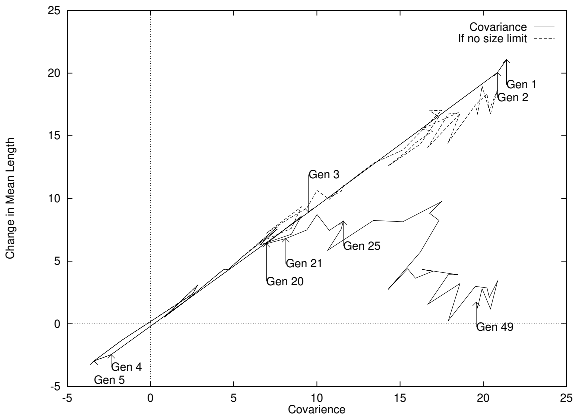


Figure 3: Covariance of program length and normalised rank based fitness v. change in mean length in next generation. Means of 50 runs.

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

In our GP the size of the population does not 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 [19, 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 until generation 20. After Generation 20 the rise in program length is smaller than predicted by Equation 1. The bulk of the discrepancy is due to the restriction on program length, as can be seen from the dotted line in Figure 3.

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. This provides some theoretical justification for the claim [2, page 112] that "average growth in size ... is proportional to selection pressure". Which is based upon experimental measurements with a small number of radically different selection and crossover operators on Tackett's "Royal Road" problem [2]. (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).

Between generations 4 and 20 the mean program size can be reasonably described by a parabola, i.e. growth is quadratic. This corresponds with the covariance rising linearly (slope 0.76) with number of generations.

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

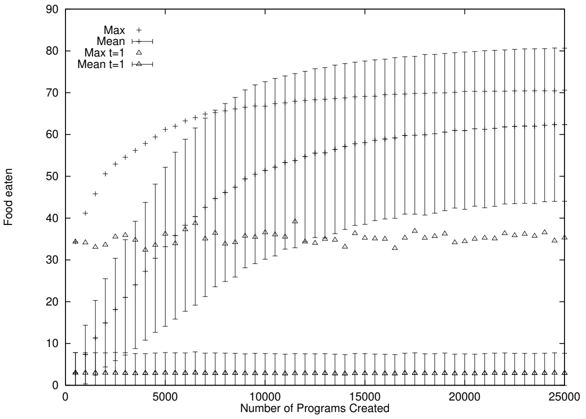


Figure 4: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs comparing tournament sizes of 7 and 1.

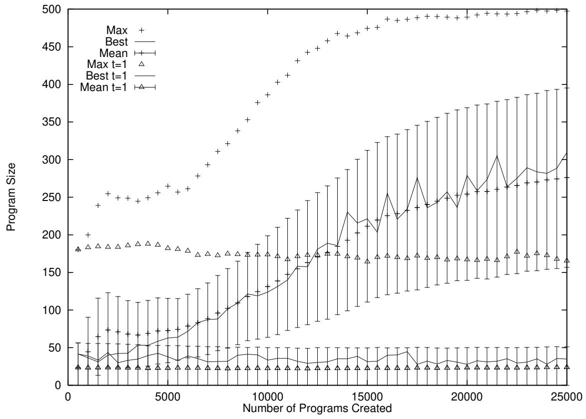


Figure 5: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Means of 50 runs.

## 5.2 No Selection

A further 50 runs were conducted (using the same initial populations) and no fitness selection. Unsurprisingly no run found a solution and the maximum, mean and other fitness statistics fluctuate a little but are essentially unchanged (cf. Figure 4). Similarly program size statistics fluctuate but are essentially the same as those of the initial population (cf. Figure 5). This is in keeping with results with no selection reported in [2, page 112]. Given crossover produces random changes in length we might have expected the spread of lengths to gradually increase. This is not observed. The slow fall in maximum program size can be seen in Figure 5. There is also a small fall in mean standard deviation 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). Of the six runs which found ants that could eat all the food on the Santa Fe trail within 600 movements, four where found before generation 26 and of these three retained such programs until generation 50 while in one case the maximum fitness dropped below 89. In another run a solution was found in generation 41 but no other generations of this run contained solutions.

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.

After fitness selection is removed, if the crossover operator is unbiased, we expect the length of programs to fluctuate at random about the lengths of their parents. So the mean length should change little and the spread of lengths should increase. The length restriction means the crossover operator is biased because it must ensure the child it produces does not exceed the length limit. Where an unbiased choice of crossover points would cause the offspring to be too large, the roles of the two parents are reversed and a shorter (and legal) program is produced instead. This bias (which is more important as programs start to approach the length limit) explains the slow fall in program size seen in Figure 7. The spread of programs' lengths can also be seen. The average standard deviation rises from 92.3 in generation 26 to 107.6 at the end of the run.

## 5.4 Correlation of Fitness and Program Size

Figure 8 plots the correlation coefficient of program size and amount of food eaten by the ant it controls. (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 that in all cases there is a positive correlation (typically $+0.3$) between program performance and length of programs. Where selection is not used longer programs may be more fit simply because they are more likely to contain useful primitives (such as Move) than short

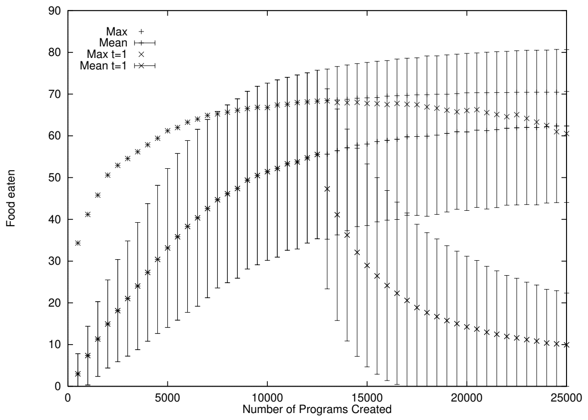


Figure 6: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs showing effect of removing fitness selection.

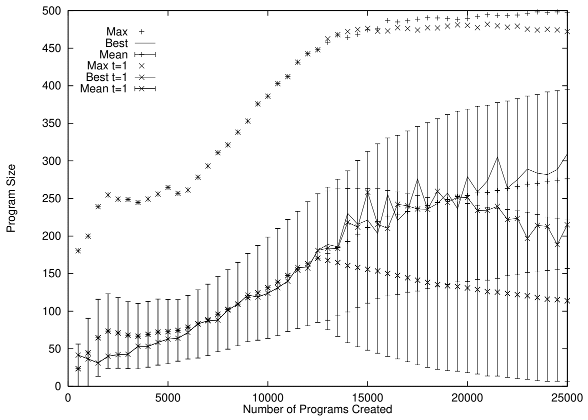


Figure 7: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Means of 50 runs.

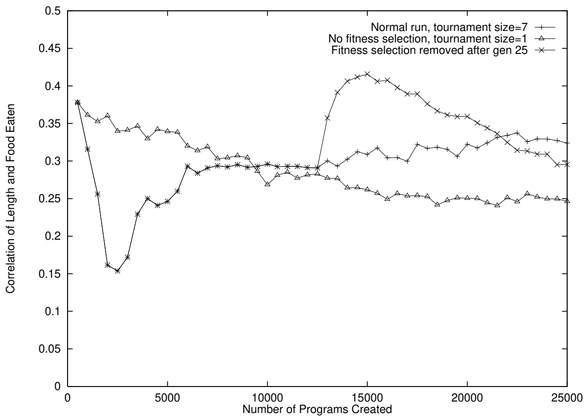


Figure 8: Correlation of program length and fitness, Normal runs, runs without selection and runs with selection removed halfway through. Means of 50 runs.

programs. When selection is used the correlation may be because when crossover reduces the length of a program it is more likely to disrupt the operation of the program and so reduce its fitness.

## 5.5 Effect of Crossover on Fitness

As expected in the initial generations crossover is highly disruptive (cf. Figure 9) with only 19.5% of crossovers producing a child with the same score as its first parent (i.e. the parent from which it inherits the root of its tree). However after generation 4 this proportion grows steady, by generation 18 more than half have the same fitness. At the end of the run this has risen to 69.8%.

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, no children are produced with a fitness greater than their parent. Similar behaviour has been reported on other problems [11] [20, page 183] [14, Chapter 7].

## 5.6 Non-Disruptive Crossover 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 crossover is

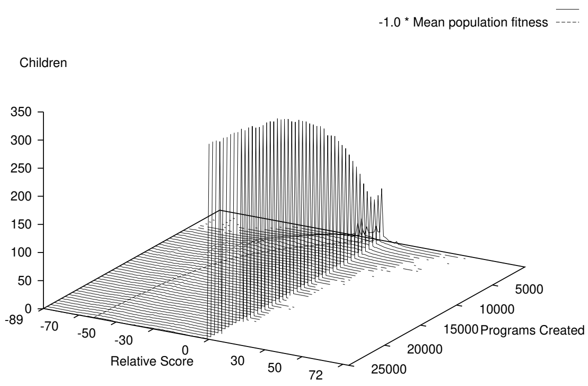


Figure 9: Fitness relative to First Parent, Normal runs, Means of 50 runs.

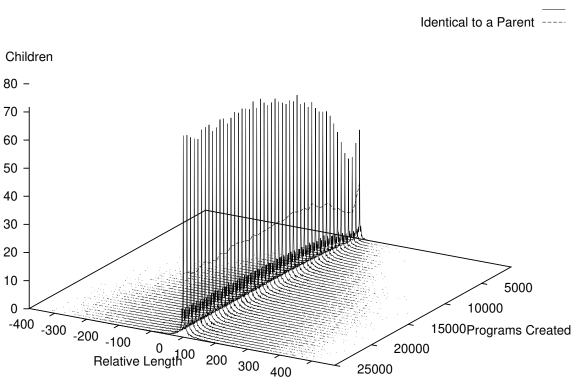


Figure 10: Change in length of offspring relative to first parent where score is the same. 50 Normal runs.

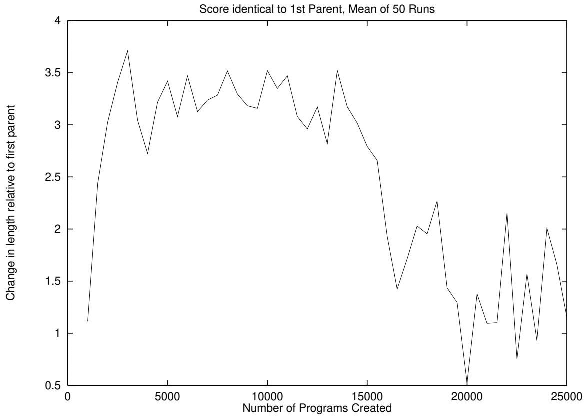


Figure 11: Mean change in length of offspring relative to first parent where score is the same. Normal runs.

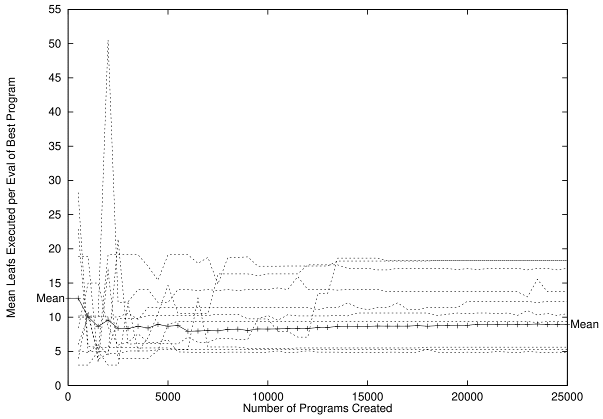


Figure 12: Energy used per call of "best" program. 10 normal runs.

random so we would expect to see it finding more long solutions than short ones. The change in program length of programs produced by crossover which have the same fitness as their first parent is plotted in Figure 10. At first sight Figure 10 is disappointingly symmetric with a large central spike of programs with the same length and rapidly falling tails either side of zero. Initially 8% of crossovers produce a child which has the same length and fitness as its first parent. This proportion falls to a minimum of 6% in generation 4 and then rises progressively to 15% at the end of the run. The dashed line on Figure 10 plots the number of children in each generation produced by crossovers that are identical to one or other of their parents. On the ant problem 5% of crossovers produce such clones.

Closer examination of the data in Figure 10 reveals that on average (cf. Figure 11) there is a bias towards longer programs. On average programs with the same fitness as their first parent are always longer than it. From generation 4 to 23 they are on average about 3 program nodes longer. Presumably due to the average program length approaching the upper limit, the bias falls to about 1 after generation 33 but remains positive.

Part of the reason for the central peak of Figure 10 is a result of another aspect of GP crossover: almost all the genetic material is inherited from the first parent. Crossover points are typically drawn from close to the leafs of both parents simply because for large bushy trees most parts of a tree are close to its leafs. While the expected size of crossover fragments depends in detail upon the trees selected as parents and the relative weighting applied to functions and terminals, cf. Table 1, typically both the inserted subtree and the subtree it replaces consist of a function and its leafs. Since these subtrees are short together they produce a small change in total size. It seems likely that small changes are less likely to effect fitness than big ones, hence the spike in Figure 10. Inheriting principally from one parent is in sharp contrast to more traditional GAs (and indeed sexual reproduction in "higher organisms" in nature) where offspring receive about the same amount of genetic material from each parent.

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 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, in the cases of those that have been analysed in detail, mostly code that has been used before is re-executed. That is large programs evolve which contain small functional parts.

# 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 [15, 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 [1, 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" [11, 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 [3, 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 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.

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 [21]. 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?

## 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 [15] or program length. ([22, 18] 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) [15, 23, 24]. [13] 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 [25].

The third method is to tailor the genetic operations. [26, page 469] uses several mutation operators but adjusts their frequencies so a "decrease in complexity is slightly more probable than an increase". [10] suggests targeting genetic operations at redundant code. This is seldom used, perhaps due to the complexity of identifying redundant code. [7] showed bloat continuing despite their targeted genetic operations. Possibly this was because of the difficulty of reliably detecting introns. There was a route whereby the GP could evolve junk code which masqueraded as being useful and thereby protected itself from removal. While [13] 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. That is fitness based selection leads to bloat.

In Sections 3, 4 and 5 we have taken a typical GP problem and demonstrated with fitness selection it suffers from bloat whereas without selection it does not. We have demonstrated that if fitness selection is removed, there is a slight bias in common GP crossover (caused by the practical requirement to limit bloat) which causes a slow reduction in program size. NB fitness causes bloat in spite of a small crossover bias in favour of parsimony. Detailed measurement of crossover confirms after an extended period of evolution, most crossovers are not disruptive (i.e. most children have the same fitness as their parents). It also shows children with the same fitness as their parents are on average consistently longer than them.

In Section 5.1.1 we apply Price's Theorem for the first time to program lengths within GP populations. We confirm experimentally that it fits GP populations 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 and the British Council. We would like to thank Wolfgang Banzhaf, Terence Soule and Tom Haynes for helpful suggestions and critisim.

## References

[1] Walter Alden Tackett, 1993, Genetic programming for feature discovery and image discrimination, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, University of Illinois at Urbana-Champaign, July, pp. 303–309.

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

[3] Peter John Angeline, 1994, Genetic programming and emergent intelligence, *Advances in Genetic Programming*, chapter 4, pp. 75–98, MIT Press, Cambridge, MA, USA.

[4] Walter Alden Tackett, 1995, Greedy recombination and genetic search on the space of computer programs, *Foundations of Genetic Algorithms 3*, Estes Park, Colorado, USA, 31 July–2 August, pp. 271–297.

[5] W. B. Langdon, 1995, Evolving data structures using genetic programming, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Pittsburgh, PA, USA, July, pp. 295–302.

[6] Peter Nordin and Wolfgang Banzhaf, 1995, Complexity compression and evolution, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Pittsburgh, PA, USA, July, pp. 310–317.

[7] Terence Soule, James A. Foster, and John Dickinson, 1996, Code growth in genetic programming, *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, July, pp. 215–223.

[8] Annie S. Wu and Robert K. Lindsay, 1996, A survey of intron research in genetics, *Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation*, Berlin, Germany, September, pp. 101–110.

[9] Tobias Blickle and Lothar Thiele, 1994, Genetic programming and redundancy, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, pp. 33–38, Max-Planck-Institut für Informatik (MPI-I-94-241).

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

[11] Peter Nordin, Frank Francone, and Wolfgang Banzhaf, 1996, Explicitly defined introns and destructive crossover in genetic programming, *Advances in Genetic Programming 2*, chapter 6, pp. 111–134, MIT Press, Cambridge, MA, USA.

[12] Nicholas Freitag McPhee and Justin Darwin Miller, 1995, Accurate replication in genetic programming, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Pittsburgh, PA, USA, July, pp. 303–309.

[13] Justinian P. Rosca and Dana H. Ballard, 1996, Complexity drift in evolutionary computation with tree representations, Technical Report NRL5, University of Rochester, Computer Science Department, Rochester, NY, USA.

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

[15] John R. Koza, 1992, *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press, Cambridge, MA, USA.

[16] John R. Koza, 1994, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge MA, USA.

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

[18] W. B. Langdon and R. Poli, 1997, An analysis of the MAX problem in genetic programming, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA, July.

[19] Lee Altenberg, 1995, The Schema Theorem and Price's Theorem, *Foundations of Genetic Algorithms 3*, Estes Park, Colorado, USA, 31 July–2 August, pp. 23–49.

[20] Justinian P. Rosca and Dana H. Ballard, 1996, Discovery of subroutines in genetic programming, *Advances in Genetic Programming 2*, chapter 9, pp. 177–202, MIT Press, Cambridge, MA, USA.

[21] Chris Gathercole and Peter Ross, 1994, Dynamic training subset selection for supervised learning in genetic programming, *Parallel Problem Solving from Nature III*, Jerusalem, October, pp. 312–321.

[22] Chris Gathercole and Peter Ross, 1996, An adverse interaction between crossover and restricted tree depth in genetic programming, *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, July, pp. 291–296.

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

[24] Hitoshi Iba, Hugo de Garis, and Taisuke Sato, 1994, Genetic programming using a minimum description length principle, *Advances in Genetic Programming*, chapter 12, pp. 265–284, MIT Press, Cambridge, MA, USA.

[25] William B. Langdon, 1996, Data structures and genetic programming, *Advances in Genetic Programming 2*, chapter 20, pp. 395–414, MIT Press, Cambridge, MA, USA.

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