EuroGP'98 First European workshop on Genetic Programming, W. Banzhaf, R. Poli, M. Schoenauer and T. C. Fogarty, editors, Paris, 14–15 April, Springer-Verlag

Genetic Programming Bloat with Dynamic Fitness

W. B. Langdon and R. Poli

School of Computer Science, 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

Abstract. In artificial evolution individuals which perform as their parents are usually rewarded identically to their parents. We note that Nature is more dynamic and there may be a penalty to pay for doing the same thing as your parents. We report two sets of experiments where static fitness functions are firstly augmented by a penalty for unchanged offspring and secondly the static fitness case is replaced by randomly generated dynamic test cases. We conclude genetic programming, when evolving artificial ant control programs, is surprisingly little effected by large penalties and program growth is observed in all our experiments.

1 Introduction

The tendency for programs in genetic programming (GP) populations to grow in length has been widely reported [Tac93, Tac94, Ang94, Tac95, Lan95, NB95, SFD96]. In our previous work on this phenomenon (referred to as "bloat") [LP97, Lan97, Lan98b, LP98a] we have investigated the effect of commonly used fitness functions with a variety of genetic algorithm, population based and non-population based search techniques. These experiments have shown that bloat is not a unique phenomenon to genetic programming and we argue that it is inherent in discrete variable length representations using a simple scalar static fitness function. [NB95, Lan98b] suggest that non-performance effecting code (sometimes referred to as "introns") can contribute to bloat in population based search techniques, such as GP. Our explanations stress the role of simple static scalar fitness functions in selecting children which behave in the same way as their parents. In this paper we broaden our investigation to consider fitness functions which avoid selecting such children and dynamic fitness functions (where the test case changes every generation).

We continue to use the well known genetic programming bench mark problem of evolving control programs to guide an artificial ant along an intermittent trail of food pellets. The first experiments use the well known Santa Fe trail [Koz92], while this is replaced in the second set of experiments by randomly generated trails which are changed each generation (i.e. the population "sees" each trail only once). In these experiments the effects of changing the fitness function are studied.

In Sect. 2 we briefly describe the artificial ant problem and the genetic programming system used to solve it. In Sect. 3 we describe the fitness functions used in the two sets of experiments and introduce the penalty for copying the behaviour of ancestors. Our results are given in Sects. 4 and 5, which are followed by our conclusions in Sect. 6.

2 The Artificial Ant Problem

The artificial ant problem is described in [Koz92, pages 147–155]. It is a well studied problem and was chosen as it has a simple fitness function. [LP98b] shows it is a difficult problem for GP, simulated annealing and hill climbing but has many of the properties often ascribed to real world problems. 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 takes 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 evolutionary system we use is identical to [LP97] except the limit on the size of programs has been effectively removed by setting it to a very large value. The details are given in Table 1, parameters not shown are as [Koz94, page 655]. Note in these experiments we allow the evolved programs to be far bigger than required to solve the problem. (The smallest solutions comprise only 11 nodes [LP98b]).

Table 1. Ant Problem

Objective:	Find an ant that follows food trails
Terminal set:	Left, Right, Move
Functions set:	IfFoodAhead, Prog2, Prog3
Fitness cases:	The Santa Fe trail or randomly generated trails
Fitness:	Food eaten less "plagiarism" penalty
Selection:	Tournament group size of 7, non-elitist, generational
Wrapper:	Program repeatedly executed for 600 time steps.
Population Size:	500
Max program size:	no effective limit
Initial population:	Created using "ramped half-and-half" with a max depth of 6
Parameters:	90% crossover, 10% reproduction, no mutation
Termination:	Maximum number of generations G = 50

3 Fitness Functions

3.1 Santa Fe trail

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 within 600 time units is the score of the control program.

3.2 Random Trails

In the second set of experiments the fixed Santa Fe trail is replaced by 50 randomly generated trails each containing 80 food items. Each generation is tested on a different trail, however the order of the trails is the same in each run. (The test case is available via anonymous ftp node `ftp.cs.bham.ac.uk` directory `pub/authors/W.B.Langdon/gp-code` in file `dynamic.trl` revision 1.11). Each trail is created by appending (in randomly chosen orientations) 20 randomly chosen trail fragments each containing 4 food pellets. We use the 17 fragments shown in Fig. 1.

A uniform choice from these 17 fragments appeared to produce trails which were too difficult. Therefore, like the Santa Fe trail, the randomly produced trails were made easier at the start. This was implemented by increasing the chance of selecting the lower numbered fragments (as they have smaller gaps in the trail). In detail: 1) start at the origin facing along the x-axis with $n = 0$; 2) select a trail fragment uniformly from fragments numbered $1 \ldots n/2+1$ when n is less than 9 and uniformly from the whole set when it is bigger; 3) the chosen fragment is then rotated and/or reflected into a random orientation from those available (see Fig. 1). Make sure the transformation is compatible with the current direction; 4) the fragment is then appended to the trail, possibly changing the current direction and n is incremented; 5) unless there are already 20 fragments in the trail, go back to (2) and select another fragment. Once the trail is complete, it is checked to see it does not cross the start position and does not fold back over itself (i.e. food pellets are not closer than 2 grid squares, unless they are on the same part of the trail). If either check fails, the trail is discarded and a new one is created.

In practice it is difficult to create a contiguous winding trail of 80 food pellets in a 32×32 grid without it overlapping itself. Therefore a toroidal grid of 300×300 was used. The time available to the ant to transverse each trail is calculated as it is created. The time allowed is five plus the sum of the time allocated to each of the fragments it contains (see Fig. 1) plus a further five, to allow the ant to do some additional local searching, for each occasion when fragments at different orientations are used (i.e. where a new bend is introduced). In practice this makes the problem very difficult. In several runs the best programs evolved showed true trail following abilities but were marginally too inefficient to follow all the trails completely within the time limit.

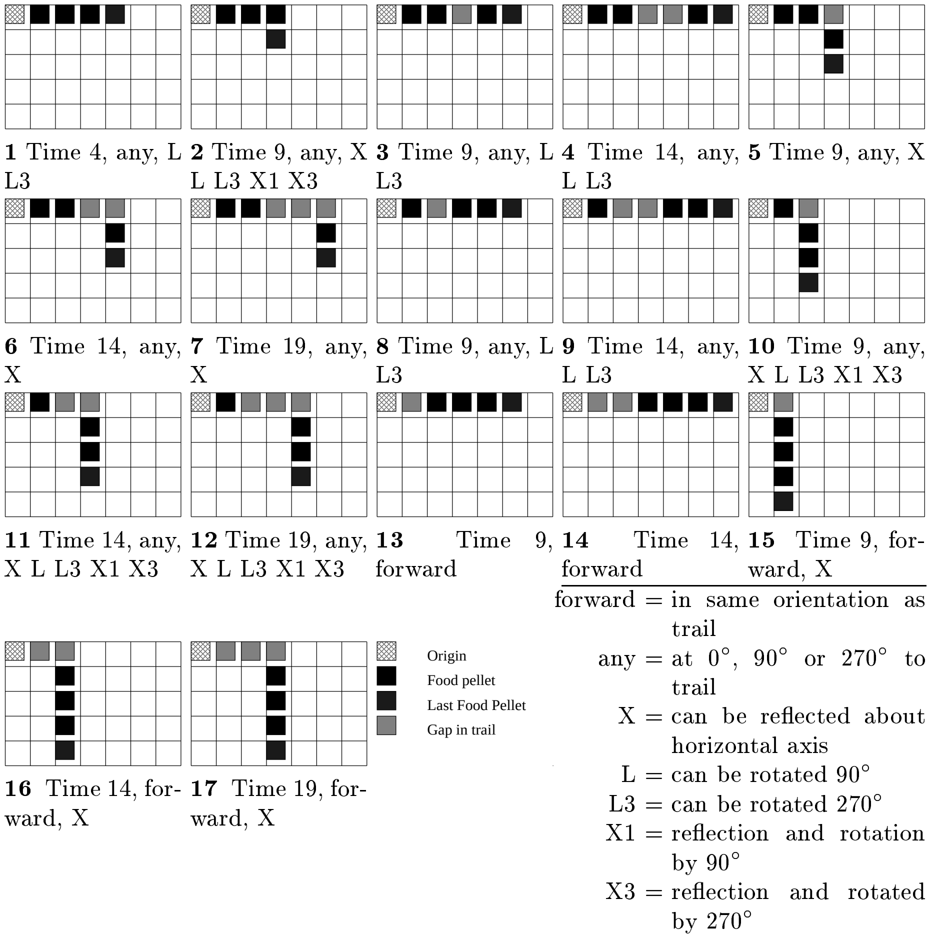

Fig. 1. Fragments of Santa Fe trail used to form random trails.

3.3 Plagiarism Penalty

In both sets of experiments (i.e. on the Santa Fe and on the random trails) runs were conducted with a range of fitness plagiarism penalties. The plagiarism penalty is applied to programs which, when run on the same test as their first parent, have the same score as that parent. (The first parent is defined to be the one from which they inherit their root node). E.g. when using the Santa Fe trail, fitness is reduced if a program causes the ant to eat the same number of food pellets as the program's first parent did. In the case of the dynamic test case, the program must be run both on its own test and the test for the previous generation, i.e. the test its parents where run on.

The smallest penalty (referred to as -0.5) causes the fitness to be reduced by half a food pellet. The other penalties reduce a program's fitness by a fixed fraction. The highest penalty (100%) sets the fitness to zero. As tournament selection is used, the effect the penalty has on the number of children given to each program is complicated, however when the penalty is large compared to the spread of scores within the population, even the best program has little chance of producing children for the next generation.

In many of the results given in the following sections the data falls into a low penalty group (20% or less) and a high penalty group (50% or more). For example in Fig. 3 the average sizes of programs in the population with penalties of 0%, -.5 and 20% lie close to each other, as do 50%, 80%, 95% and 100%, and the two groups are clearly separated. We can estimate the mean plagiarism penalty (when applied) as its size times the mean score in the population. The separation between runs where the penalty is effective and the others corresponds to when this estimate is bigger than the variation in program score across the population (as measured by its standard deviation). That is the plagiarism penalty seems to have little effect if it smaller than the existing variation in programs' scores before it is applied.

4 Santa Fe Trail Results

The results on the Santa Fe Trail are based on ten independent runs with each plagiarism penalty setting. The same ten initial populations are used with each plagiarism value.

From Fig. 2 we can see even the highest plagiarism penalty has only a little depressing effect on the maximum score in the population. Surprisingly the effect on the average program score is more obvious. Suppressing programs which copy their parents appears to considerably reduce the proportion of the best program to the rest of the population, thereby increasing the gap between the mean and maximum raw score. As expected, once the rate of finding higher scoring programs drops (about generation 20, 10,000 programs generated) the size of programs increases and the population bloats, cf. Fig. 3. The plagiarism penalty is unable to prevent this but appears to slow growth, in that when the penalty is 50% or more, by the end of the run programs are on average only half the size of those created in runs with lower penalties.

As expected near the end of runs with low penalty the maximum score in the population remains fixed for many generations. In contrast at the ends of runs with high penalties it varies rapidly. (It changes, either increases or decreases, on average every 2.3 generations in the last ten generations with a penalty of 100% and only increases once in the same period in the ten runs with no penalty. This difference is not obvious in Fig. 2 as it plots data averaged over all ten runs).

With low penalties the population converges in the sense that typically about 90% of programs in the final population have the same score. They are descended, via their first parents, from the same individual. The founding individual has the same score as them, as do its descendants in the genetic lines connecting it to them. In contrast runs with penalties of 20% or more don't appear to converge like this and typically there are only 1 or 2 programs with the highest score in the population.

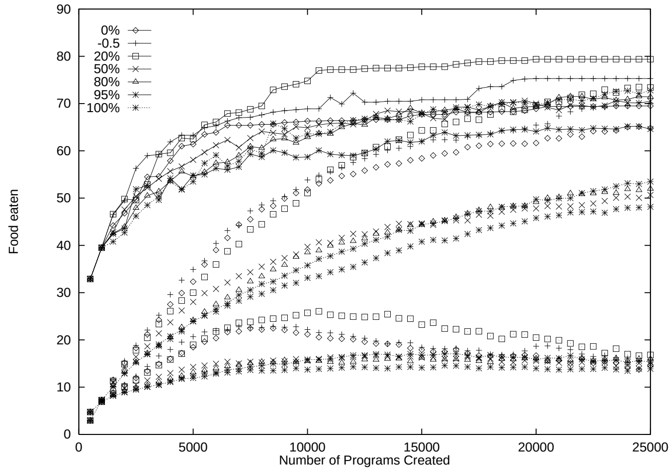

Fig. 2. Evolution of maximum, population mean and standard deviation of food eaten on Santa Fe trail as plagiarism penalty is increased. Means of 10 runs.

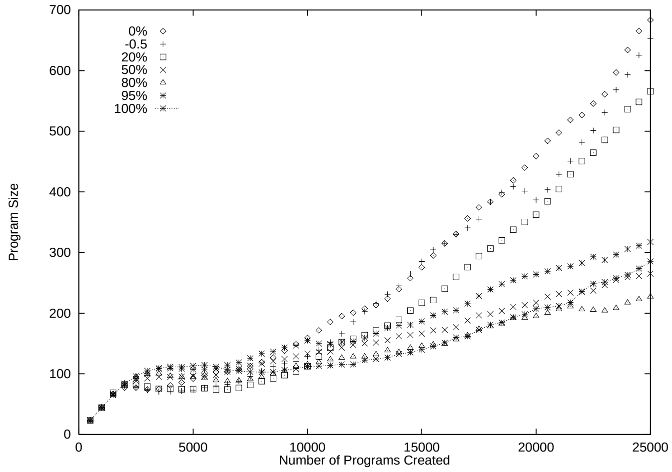

Fig. 3. Evolution of population mean program size on Santa Fe trail as plagiarism penalty is increased. Means of 10 runs.

Further evidence for the lack of convergence is contained in Fig. 4 which indicates where the penalty is low the population evolves so that it is quickly dominated by programs which have the same score as their first parent. While with a high penalty this fraction of the programs increases only slightly and remains near its value in the first few generations. Looking back to earlier generations we see the same picture. With no penalty, typically about 90% of the population has the same score as its first grandparent (i.e. the one it inherited its root from). While with higher penalties it is in the region of 3%. Note this convergence is not the same as convergence in linear genetic algorithms, the population variety is high. Without a plagiarism penalty the fraction of different programs in the population rises to lie near 99% by the end of the run. While with a plagiarism penalty of 50% or more it is still high and reaches about 95% by generation 50. (The slight difference between these figures may be simply due to the larger size of programs when the penalty is low, cf. Fig. 3). Studying the successful crossovers, i.e. those that produced offspring which cause more food to be eaten than their (first) parent, shows runs without a penalty converge in that crossover seldom makes improvement after generation 10 (5,000 programs created) (in the ten run there were only 40 improvements after generation 25, compared to a total of 3,451). Whereas with a 100% penalty the population remains "on the boil" with parents of lower score being selected and crossover continuing to make improvements on them until the end of the run. (E.g. In the first run with a 100% penalty there were 3,098 such crossovers, about 1,730 after generation 25).

Another aspect of the convergence of runs with low penalties is individuals with the same score are selected to be parents. E.g. at the end of all but one run with no penalty, all parents had identical scores. With high penalties there is more variation (on average 29.7 different scores acting as first parents to children in the final population of runs with a 100% penalty).

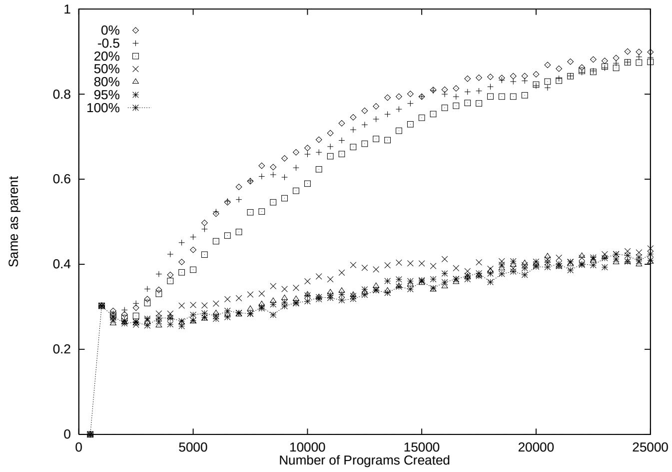

Fig. 4. Evolution of proportion of population with same score as first parent on Santa Fe trail as plagiarism penalty is increased. Means of 10 runs.

4.1 Correlation of Fitness and Program Size

In GP program size is inherited and so we can apply Price's Covariance and Selection Theorem [Pri70, LP98a] to it. Provided our genetic operators are unbiased with respect to length, the expected change in mean program length from one generation to the next is given by the covariance between program length and normalised fitness in the previous generation. (Normal GP crossover is unbiased provided size or depth restrictions don't effect the population. I.e. children produced by crossover are on average the same size as their parents [LP97]). Figure 5 shows in the first few generations there is a strong covariance between length and fitness. We suggest this is because in the initial random populations long programs tend to do better simply because they are more likely to contain useful primitives such as Move. In the next few generations strong selection acts to remove useless programs and consequently the covariance falls. The plagiarism penalty appears to dilute this effect of selection so the covariance in high penalty runs takes more generations to fall. There is then a period of about eight generations during which crossover finds many improved solutions and covariance remains small after which the normal increase is seen in low penalty runs and the populations bloat. Figure 5 shows strong plagiarism penalties appear to prevent the covariance increasing towards the end of the runs so reducing the bloat seen with low penalties (cf. Fig. 3). However the covariance remains positive and some increase in programs size is seen even with the highest penalty.

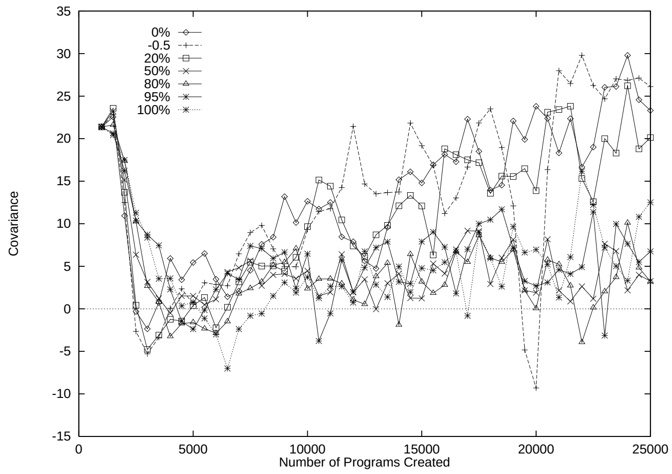

Fig. 5. Evolution of covariance of program length and normalised rank based fitness on Santa Fe trail as plagiarism penalty is increased. Means of 10 runs.

Figure 6 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 and exclude the plagiarism penalty). Figure 6 shows in all cases there is a positive correlation between program score (rather than fitness) and length of programs. This correlation does not vary strongly with the plagiarism penalty.

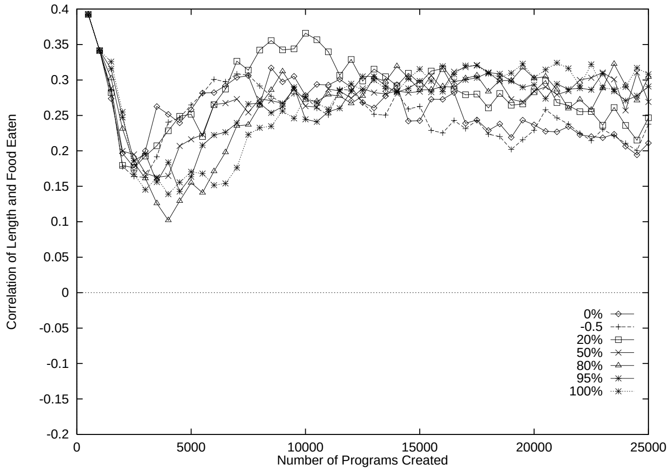

Fig. 6. Correlation of program length and food eaten on Santa Fe trail as plagiarism penalty is increased. Means of 10 runs.

4.2 Fraction of Bloated Programs Used

Every program terminal uses one time unit each time it is executed. This gives us a convenient, if crude, way to estimate the amount of code being used within each program. Figure 7 plots the average number of terminals executed each time a program is run. Initially on average 6.8 terminals are used per program execution, this then rises rapidly to 19.1 before falling back to 9.3 (no penalty). Runs with high penalties are similar except they rise to slightly higher peaks, 26.3, and take longer to fall back to similar values (10.7, 100% penalty). The initial rise in the average may simply be due to selection acting to remove those with lower values, however it appears crossover finds better solutions which execute only a small fraction on their terminals each time the programs are run.

Assume that there are two frequently occurring cases, either the program starts with the ant facing food or facing an empty square. Using this we can estimate the number of terminals contained in the two parts of the program most often used as no more than twice the average number executed each time it is run. I.e. less than 20 by the end of the run. This is clearly a very small fraction of the whole (on average programs exceed 300 nodes, and so must contain at least 150 terminals, by the end of the run, cf. Fig. 3). It appears evolution promotes the creation of small islands of useful code in large programs since such programs are less liable to disruption by crossover. Figure 7 implies the essential nature of the evolved (partial) solutions is unchanged by the plagiarism penalty. Which suggests the penalty reduces the rate at which introns grow but they are still present, even with the highest penalties. This is surprising because the stability of such programs in the face of crossover means they will suffer the penalty which should prevent them creating children in the next generation. We now turn to the dynamic fitness function results, and will see in this respect GP evolves significantly differently.

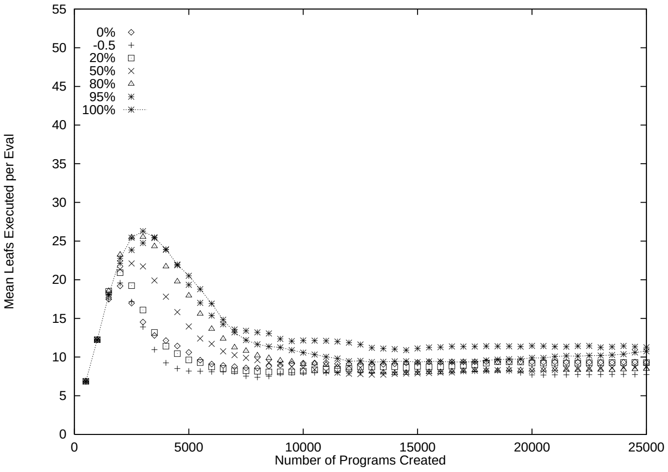

Fig. 7. Time used per call of each program in population on Santa Fe trail as plagiarism penalty is increased. Mean of 10 runs.

5 Random Trails

This section reports results from 50 independent runs with each plagiarism penalty setting. As in Sect. 4 the same initial populations are used with each plagiarism value. Some of the differences between this section and Sect. 4 are due to the larger number or runs made, which reduces the influence of stochastic effects.

To gain a measure of the ability of GP to generalise from the example trials, the populations were also tested on the complete set of 50 trails and statistics for the complete set were gathered. This information is used only for reporting purposes and only the current (and previous) trails influence the course of evolution. (As running all programs on all 50 trails is CPU expensive, this data was only collected for the first ten of the 50 runs in each experiment).

Figure 8 shows the evolution of scores using the new dynamic fitness function. Much of the variation between one generation and the next can be ascribed to the different difficulty of the trials. If we consider Fig. 10 we see rather more monotonic rises in program score. Referring to Fig. 8 again, with 50 program runs, we can see a clear separation into runs with low penalty performing approximately 6–9 food pellets better than those with high penalties. This difference is amplified if instead of looking at average performance, we look at the number of runs where at least one solution to the current trail was found, cf. Fig. 11.

The random trails seem to be a harder problem than the Santa Fe trail, only three runs evolved programs which could follow all 50 trails. Another aspect of this increased difficulty is the evolved solution given in [Koz92, page 154] to the Santa Fe trail scores less than half marks on the random trails. In contrast each of the first programs found which could pass all 50 random trails can also follow the Santa Fe trail.

There is some evidence that the random trails are themselves enough to avoid convergence of the GP population. Without a penalty there are typically only a

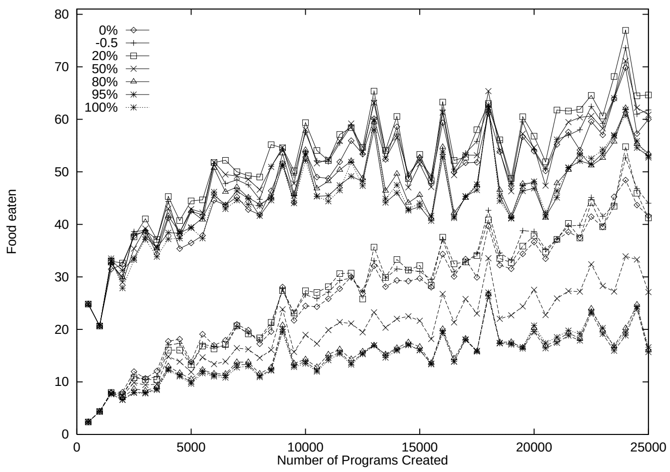

Fig. 8. Evolution of maximum and population mean of food eaten on random trails as plagiarism penalty is increased. Means of 50 runs.

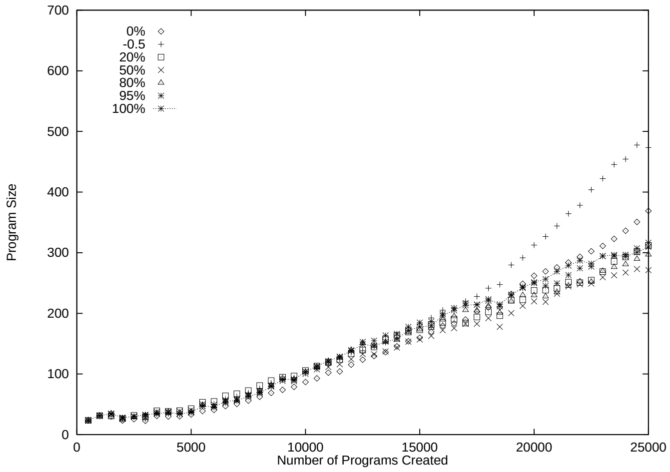

Fig. 9. Evolution of population mean program size on random trails as plagiarism penalty is increased. Means of 50 runs.

handful of programs with the best score by the end of the run. This falls to one or two with large penalties. However lack of convergence would suggest that the fraction of children behaving as their parents would be small, whereas it behaves similarly to the Santa Fe runs and with low penalties (20% or less) the fraction rises to about 80% by the end of the run (cf. Fig. 12). With higher penalties the fraction also behaves like the Santa Fe runs and remains near its initial value until the end of the runs. Again variety is near unity and does not show this convergence at all. (With no penalty then variety reaches 97% on average at generation 50 and with a penalty it is about 95% on average again).

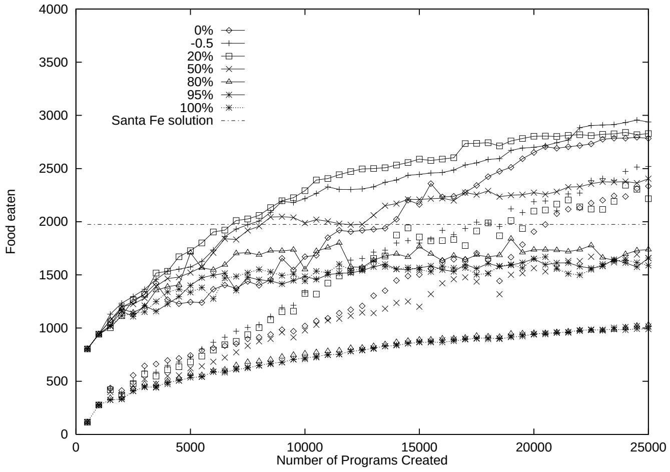

Fig. 10. Evolution of maximum and population mean of food eaten on all 50 trails as plagiarism penalty is increased. Means of 10 runs.

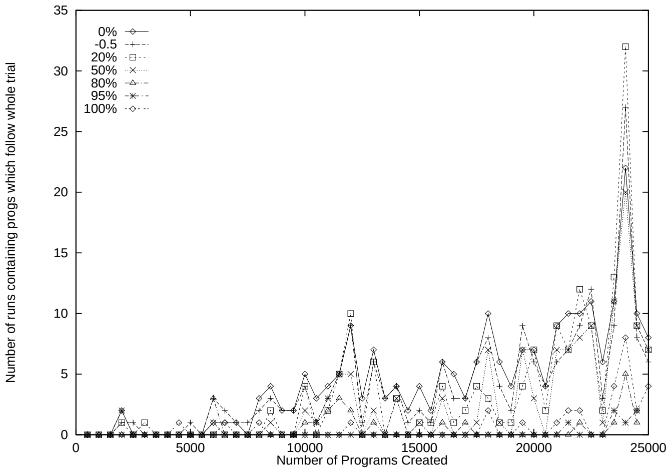

Fig. 11. Number of populations containing a solution to current trail on random trails as plagiarism penalty is increased. 50 runs.

Like the Santa Fe trail, runs with low penalties show convergence in that typically the scores of the parents of the vast majority of the children in the last generation are identical (or have one of two values). With runs with high penalty there is more variation (with on average 29.1 different scores acting as first parents to children in the final population).

5.1 Correlation of Fitness and Program Size

Comparing Fig. 13 with Fig. 5 we see there is no longer a clear separation between runs based upon strength of plagiarism penalty. After generation 15

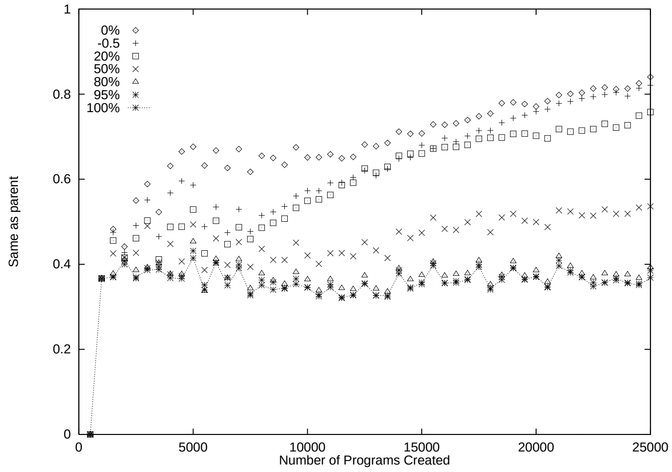

Fig. 12. Evolution of proportion of population with same score as first parent on random trails as plagiarism penalty is increased. Means of 50 runs.

(7,500 programs created) low penalty runs tend to have smaller covariances than with the Santa Fe trail. This is reflected in the generally lower increase in program size. The lower covariance may be simply due to increased randomness in the fitness function. In contrast when we look at the correlation between score and size (i.e. excluding penalty, cf. Fig. 14) the high penalty runs have an obviously lower correlation. This shows that the penalty has changed the nature of the evolved programs.

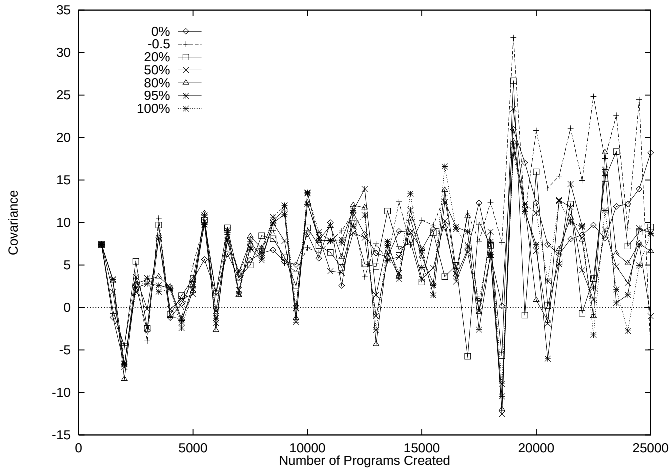

Fig. 13. Evolution of covariance of program length and normalised rank based fitness on random trails as plagiarism penalty is increased. Means of 50 runs.

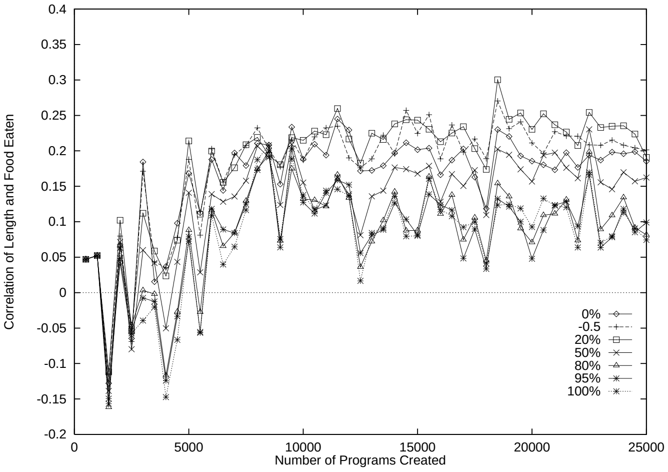

Fig. 14. Correlation of program length and food eaten on random trails as plagiarism penalty is increased. Means of 50 runs.

5.2 Fraction of Bloated Programs Used

Figure 15 confirms in the random trails high plagiarism penalties change the nature of the evolved programs. We see programs with very different behaviour evolve than is the case with the Santa Fe trail or with low penalties. The large number of ant operations per program execution (cf. Fig. 15) might indicate that non-general, or trail specific, programs have been evolved. This is confirmed to some extent if we compare the best scores in the last generation on the last trail with that on all 50 trails. We see performance on the training case (cf. Fig. 8) is proportionately higher than on all 50 trails (cf. Fig. 10). Concentrating on 100% penalty, the mean scores are 52 out of 80, i.e. 66% (averaged over 50 runs) compared to 1590 out of 4,000, i.e. 40% (averaged over 10 runs). That is populations evolved with high plagiarism penalties do considerably better on the immediate training case than they do on the more general one. The difference is not so marked in runs with low penalties.

6 Conclusions

In our earlier work on the evolution of representation size we stressed the importance of individuals with the same fitness as their parents', showing increase in average size in the later stages of evolution could in many cases be ascribed to them dominating the population. In Sect. 4 we introduce a fitness based penalty on programs which don't innovate. Even very large penalties produce only slight reductions in the best of run performance and, in these experiments, cut bloat by about a half.

In the experiments in Sect. 5 we have broaden research into bloat to consider non-static fitness functions. In these experiments a dynamic fitness function also cuts bloat by about a half. We also report combining our dynamic fitness function

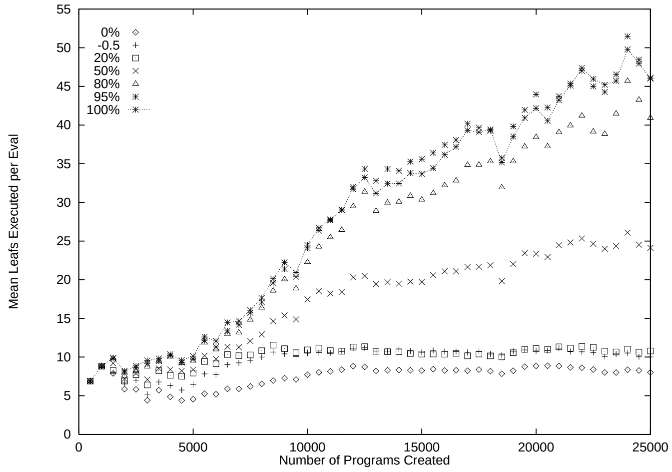

Fig. 15. Time used per call of each program in population on random trails as plagiarism penalty is increased. Mean of 50 runs.

with the plagiarism penalty and note this can also produce a small reduction in the best of run performance but can change the nature of the programs evolved and reduce their ability to generalise.

It is clear that suppressing the large numbers of programs produced in the later stages of conventional GP runs which all have the same performance by using a plagiarism penalty has not prevented bloat completely. In both sets of experiments there is bloating we suspect that this is due to shorter programs in the population being more effected by crossover than longer ones, i.e. their children follow the trails less well.

Our second set of experiments tend to confirm some of the benefits claimed for dynamic fitness measures. E.g. every dynamic fitness run (without a plagiarism penalty) produced programs which performed better on the 50 random trails than the example program evolved on just the Santa Fe trail.

We have deliberately chosen to study ant problems since they are difficult for GP and have properties often ascribed to real world programs (such as rugged landscapes, multiple solutions, competing conventions, poor feedback from partial solutions). Nevertheless it would be interesting to analyse bloat in other GP domains. Further work is needed to understand how GP populations are able to maintain their peak performance even when the selection function appears to prevent direct copying from the best of one generation to the next. This would appear to require constant innovation on the part of the population. Current GP can "run out of steam" so that GP populations stop producing improved solutions [Lan98a, pages 206]. Therefore techniques which encourage constant innovation are potentially very interesting.

Acknowledgements

This research was funded by the Defence Research Agency in Malvern. We would like to thank the anonymous referees for their helpful comments.

References

[Ang94] Peter John Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press, 1994.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[Koz94] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[Lan95] W. B. Langdon. Evolving data structures using genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 295–302, 1995. Morgan Kaufmann.

[Lan97] W. B. Langdon. Fitness causes bloat in variable size representations. Technical Report CSRP-97-14, University of Birmingham, School of Computer Science, 14 May 1997. Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97.

[Lan98a] W. B. Langdon. *Data Structures and Genetic Programming*. Kulwer, 1998.

[Lan98b] W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, USA, 5-9 May 1998. Forthcomming.

[LP97] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry *et al*, editors, *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag London, 23-27 June 1997.

[LP98a] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. This volume.

[LP98b] W. B. Langdon and R. Poli. Why ants are hard. Technical Report CSRP-98-4, University of Birmingham, School of Computer Science, January 1998.

[NB95] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, 1995. Morgan Kaufmann.

[Pri70] George R. Price. Selection and covariance. *Nature*, 227, August 1:520–521, 1970.

[SFD96] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, *et al*, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, 1996. MIT Press.

[Tac93] 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, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[Tac94] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems, 1994.

[Tac95] Walter Alden Tackett. Greedy recombination and genetic search on the space of computer programs. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 271–297, 1995. Morgan Kaufmann.