Crossover, Sampling, Bloat and the Harmful Effects of Size Limits

Stephen Dignum and Riccardo Poli

Department of Computing and Electronic Systems,
University of Essex,
Wivenhoe Park, Colchester, CO4 3SQ, UK
{sandig,rpoli}@essex.ac.uk

Abstract. Recent research [9,2] has enabled the accurate prediction of the limiting distribution of tree sizes for Genetic Programming with standard sub-tree swapping crossover when GP is applied to a flat fitness landscape. In that work, however, tree sizes are measured in terms of number of internal nodes. While the relationship between internal nodes and length is one-to-one for the case of a-ary trees, it is much more complex in the case of mixed arities. So, practically the *length* bias of subtree crossover remains unknown. This paper starts to fill this theoretical gap, by providing accurate estimates of the limiting distribution of lengths approached by tree-based GP with standard crossover in the absence of selection pressure. The resulting models confirm that short programs can be expected to be heavily resampled. Empirical validation shows that this is indeed the case. We also study empirically how the situation is modified by the application of program length limits. Surprisingly, the introduction of such limits further exacerbates the effect. However, this has more profound consequences than one might imagine at first. We analyse these consequences and predict that, in the presence of fitness, size limits may initially speed up bloat, almost completely defeating their original purpose (combating bloat). Indeed, experiments confirm that this is the case for the first 10 or 15 generations. This leads us to suggest a better way of using size limits. Finally, this paper proposes a novel technique to counteract bloat, sampling parsimony, the application of a penalty to resampling.

Keywords: Genetic Programming, Theory, Crossover, Search, Sampling, Bloat, Program Length.

1 Introduction

With the advent of a greater understanding of program search spaces—for example we now know that the functionality of programs reaches a limit as program length increases [6,4,5]—acquiring knowledge on how GP operators sample program length classes has become more and more urgent. Ideally, we would like to sample the length class where the smallest optimal programs can be found. Unfortunately, in general: a) one does not know where solutions (let alone most compact ones) are, and, b) genetic operators present specific length biases which are often unknown or only partially known and, therefore, are difficult to direct and control. In any case, a characterisation of operator bias is needed in understanding how GP will sample the search space in the first instance before technically sound problem specific modifications can be made.

M. O'Neill et al. (Eds.): EuroGP 2008, LNCS 4971, pp. 158–169, 2008.

GP, of course, applies a number of competing operators that like to sample the search space in different ways. In this paper we look at the application of standard crossover with uniform selection of crossover points, an operator for which recent research [9,2] has enabled the accurate prediction of the limiting distribution in the absence of selection, i.e., when GP is applied to a flat fitness landscape. In that work, however, tree sizes are measured in terms of number of internal nodes, which is not what GP users normally want and use. While the relationship between internal nodes and length is one-to-one for the case of a-ary trees, it is much more complex in the case of mixed arities. So, practically the *length* bias of subtree crossover remains unknown.

This paper starts filling this theoretical gap, by extending previous research [9,2] to include terminals as well as internal nodes in our program length distribution (Section 2). This shows that crossover will sample increasingly more smaller programs as the distribution converges. As smaller programs are less numerous than larger ones a large amount of resampling takes place. Empirical evidence gathered using two standard GP benchmark problems confirms this bias (Section 3). Although, selection is likely to initially work against the biases of crossover, as fitness converges, either during the later stages of a GP run or if an area of neutrality is reached, this bias will become more acute. In Section 4 we also study empirically how the situation is modified by the application of program length limits. Unexpectedly, the changes to the sampling biases introduced by such limits, further exacerbate bloat in early generations, thereby reducing the efficacy of size limits as mechanisms for bloat control.

In [2] a theory was put forward, the Crossover Bias bloat theory, which postulates that the main reason for bloat is precisely the oversampling of short programs produced by subtree crossover. Our findings confirm this tendency with and without length limits. So, in Section 5 we propose a novel technique to indirectly counteract bloat, sampling parsimony, which is effectively the application of a penalty for resampling. We study its behaviour with and without fitness. In both cases, we show that applying even slight resampling penalties, program growth can significantly be sped up or slowed down depending on the application of the penalty. Applying a 'resampled' penalty to programs confirms the crossover bias bloat theory of [2]. Applying a newly sampled program penalty, provides a natural way of acting on the very roots of bloat: the sampling and re-sampling of short programs.

We draw our conclusions in Section 6.

2 Program Length Distributions in GP

2.1 Internal Node Distributions

In [9] strong theoretical and experimental evidence was provided that standard sub-tree swapping crossover with uniform selection of crossover points pushes a population of a-ary GP trees towards a limiting distribution of tree sizes of the form:

$$\Pr\{n\} = (1 - ap_a)\binom{an+1}{n}(1-p_a)^{(a-1)n+1}p_a^n \tag{1}$$

This is known as a Lagrange distribution of the second kind. $\Pr\{n\}$ is the probability of selecting a tree with n internal nodes and a is the arity of functions that can be used in

the creation an individual. The parameter p_a was shown to be related to μ_0, the mean program size at generation 0, and a according to the formula:

$$p_a = \frac{2\mu_0 + (a-1) - \sqrt{((1-a) - 2\mu_0)^2 + 4(1-\mu_0^2)}}{2a(1+\mu_0)} \tag{2}$$

Equation (1) was generalised using the Gamma function: $\Gamma(n+1) = n!$ in [2] to enable mixed arity tree internal node distributions to be predicted:

$$\Pr_g\{n\} = (1-\bar{a}p_{\bar{a}})\frac{\Gamma(\bar{a}n+2)}{\Gamma((\bar{a}-1)n+2)\Gamma(n+1)}(1-p_{\bar{a}})^{(\bar{a}-1)n+1}p_{\bar{a}}^n \tag{3}$$

$\bar{a}$ being an averaged arity of the primitive set. This can be calculated for mixed function arities from experimental initialisation parameters as follows:

$$\bar{a} = E(arity(F)) = \sum_f arity(f)P(F=f) \tag{4}$$

where f is a non-terminal to be used in the GP experiment, $arity(f)$ is a function returning the arity of the non-terminal f, and $P(F=f)$ is the probability that a particular non-terminal f will be selected for a non-terminal node by the tree initialisation procedure. For traditional FULL and GROW initialisation methods non-terminals are chosen with equal probability [7].

2.2 Program Length Distributions

Our first step towards extending Equation (3) to allow us to predict program length distributions with mixed arities, is to look at what we can say for certain regarding the relationship between number of internal nodes and program length. First, we know for a-ary trees where the arities of all nodes in the tree are the same, the length, ℓ, of a program can be expressed exactly in terms of the number of its internal nodes, n, using the following equation

$$\ell = a \times n + 1, \tag{5}$$

where a is the (fixed) arity of the internal nodes. Therefore, rearranging Equation (5) to obtain internal nodes in terms of length, i.e.,

$$n = \frac{\ell - 1}{a}, \tag{6}$$

and substituting this into Equation (1), we obtain that, for a-ary trees,

$$\Pr_l\{\ell\} = \begin{cases} \Pr\{\frac{\ell-1}{a}\} & \text{if } \ell \text{ is a valid length (i.e., } \frac{\ell-1}{a} \text{ is a non-negative integer),} \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $\Pr_l\{\ell\}$ is the limiting distribution of program lengths. This distribution applies, for example, for Boolean function induction problems where often all functions are binary and symbolic regression problems where often only the standard four arithmetic operations are used.

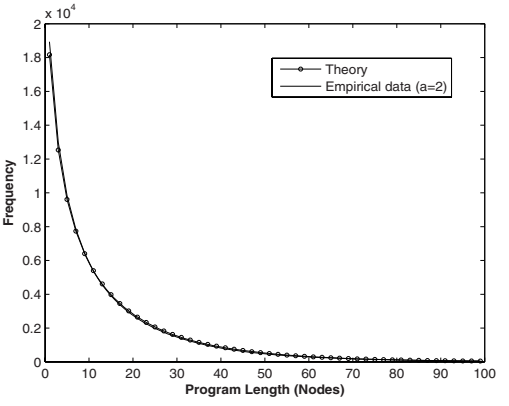

Fig. 1. Comparison between theoretical and empirical program length distributions for 2-ary trees initialised with FULL method (depth=3, initial mean size $\mu_0 = 15.0$, mean size after 500 generations $\mu_{500} = 14.49$). Invalid even lengths are ignored.

Figure 1 shows an observed plotted length distribution for 2-ary trees, with invalid (even) lengths removed, compared to that predicted by $\Pr_l$. The observed values are averages over twenty independent runs with populations of 100,000 individuals run for 500 generations.[1] As we can see there is a very close fit between the two curves.

Our next step is to extend the generalised formula for mixed-arity trees (Equation (3)) so as to predict length distributions rather than internal node distributions. We know that for a program length of 1, a single terminal, there will always be 0 internal nodes. Therefore, the predicting single node programs is a simple one-to-one mapping with the generalised formula for 0 internal nodes. However, other lengths can be obtained by different combinations of internal nodes of different arities. For example, one can obtain programs of length 3 by using one internal node of arity 2 or two internal nodes of arity 1.

As a first approximation, we will assume that we can still estimate the expected number of internal nodes in a tree of length ℓ by applying Equation (6), simply using $\bar{a}$ instead of a. We can then substitute the result into Equation (3) to obtain the distribution of lengths we are looking for. Naturally, between the variable ℓ and the variable n there is a difference in scale (the factor $\bar{a}$). So, we will need to normalise the values produced by Equation (3) to ensure the new distribution sums to 1.

Putting all of this together, we obtain an approximate model of the limiting distribution of program lengths in the case of primitive sets of mixed arities. Namely:

$$\Pr_v\{\ell\} = \begin{cases} \Pr_g\{0\} & \text{if } \ell = 1, \\ \frac{\Pr_g\{\frac{\ell-1}{\bar{a}}\}}{\bar{a}} & \text{if } \ell \text{ is a valid length greater than 1.} \end{cases} \tag{8}$$

Note, we do not require $\frac{\ell-1}{\bar{a}}$ to be an integer.

Since there were approximations in the original derivation of Equation (3) in [2], and we added further approximations in the derivation of Equation (8), one might wonder whether the model is sufficiently accurate to be of practical use. Figure 2 shows

[1] These and all other experimental parameters were chosen as in [2] for ease of comparison.

observed and theoretical values of the limiting length distribution experiment set up for internal nodes of arities of 1 and 2 where all lengths are valid, whilst Figure 3 compares the theoretical and empirical distribution obtained in a GP run with the primitive set of the Artificial Ant problem, which has internal nodes arities of 2, 2 and 3, for IF-FOOD-AHEAD, PROGN2 and PROGN3, respectively. Note, with this choice there is no way of generating programs of length $\ell = 2$. Finally, Figure 4 shows the results of using arities of 1, 2, 3 and 4. Note that in order to highlight the fit for larger and less common programs we used a log scale for frequency.

As one can see, the model in Equation (8) accurately models the distribution observed in real runs in all cases, with only minor deviations at the very short program lengths where some of the assumptions behind the model are less applicable.[2] However, generally both the theoretical model and the actual runs show that in almost all cases crossover will sample with high frequency small programs. The effects of this bias are investigated in the next section.

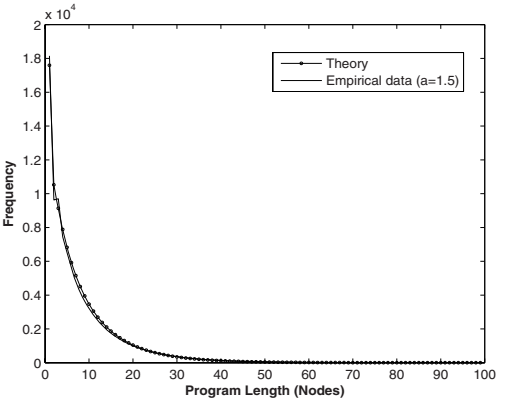

Fig. 2. Comparison between theoretical and empirical program length distributions for trees created with arity 1 and 2 functions initialised with FULL method (depth=3, initial mean size μ_0 = 8.13, mean size after 500 generations μ_{500} = 8.51). All lengths are valid.

3 Sampling and Resampling

Our first step is to see how standard crossover will sample the search space on a flat fitness landscape. Our primary purpose for doing this is simply to isolate the search bias for crossover. It should be noted, however, that, while in the presence of fitness gradients selection will counteract the crossover bias (this is analysed further in conjunction with selection in Section 5), there are situations where the crossover bias may become the prominent search bias. This may happen, for example when GP search reaches an area of neutrality, e.g., when GP operators, during an experimental run, are unable to escape areas of similar fitness.

[2] Curing the slight mismatches for earlier lengths would require a more accurate estimation of number of internal nodes of each arity for small ℓ. We will investigate more precise models in future work.

In particular we are interested in finding out how much resampling goes on. This gives us an idea of the efficiency or otherwise of the search. To empirically analyse crossover sampling we took two out-of-the-box problems from the ECJ evolutionary toolkit [8]: 4 Bit Even Parity and the Artificial Ant. As the Parity problem uses Boolean operators only we know that, in the absence of selection, the limit program length distribution to be that of a 2-ary tree as shown in Figure 1, whilst, as previously discussed, the Artificial Ant will follow a distribution similar to the one in Figure 3.

Adjustments were made to ECJ to remove mutation, ensure uniform selection of crossover points, and to prevent a depth limit being applied. A population size of 1,000 individuals was used and the results for 200 generations were averaged over one hundred independent runs. All experiments were initialised using the RAMPED method [3] with a maximum depth of 6 and minimum depth of 2. A constant fitness value was returned in all cases.

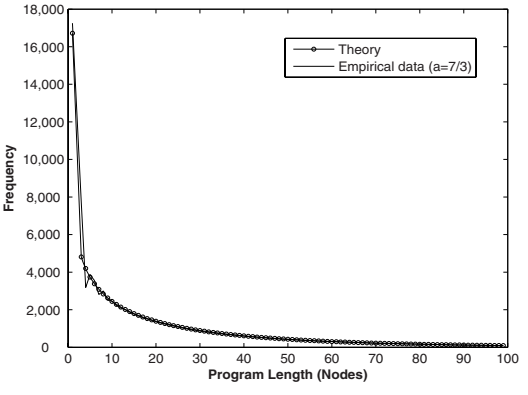

Fig. 3. As in Figure 2 but for arities 2, 2 and 3 (μ_0 = 32.12, μ_{500} = 33.22). Invalid length 2 is ignored.

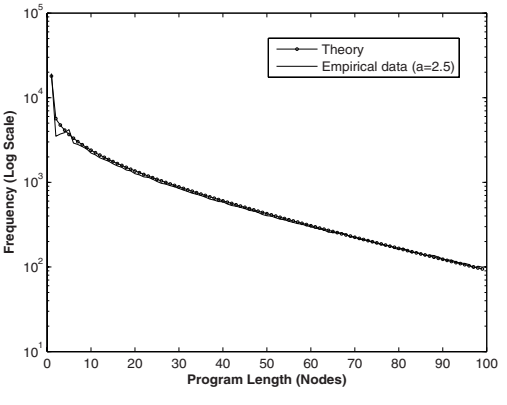

Fig. 4. As in Figure 2 but for arities 1, 2, 3 and 4 (μ_0 = 25.38, μ_{500} = 23.76). All lengths are valid.

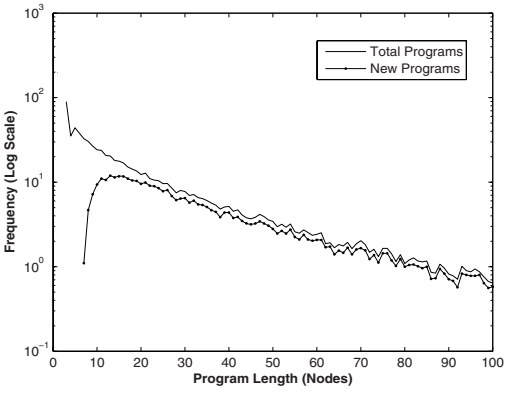

Fig. 5. Frequencies of new unique programs not sampled previously compared to all programs generated at generation 200, for the Artificial Ant Problem applied to a flat fitness landscape

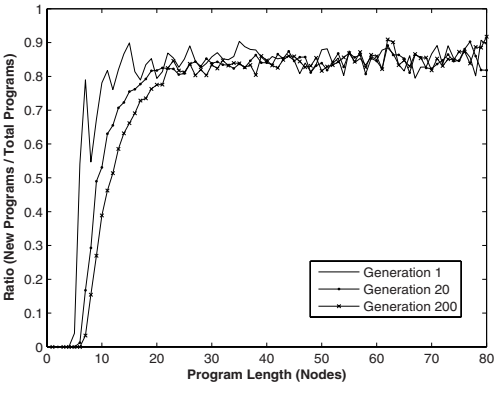

Fig. 6. Ratio of new unique programs not sampled previously compared to programs generated at generations 1, 20 and 200, for the Artificial Ant Problem applied to a flat fitness landscape

The total number of programs for each length was recorded at each generation along with the number of programs for each length that had been sampled in a previous generation. Taking the artificial ant problem, as we can see in Figure 5, at generation 200 the number of new unique programs is extremely small compared to that of the total for that generation. The majority of all programs sampled under these conditions are of course in the smaller length classes.

As a ratio, new programs divided by total programs, plotted in Figure 6, it is clear that newly sampled programs are being generated at the larger length classes and that crossover is progressively resampling more and more programs.

4 Effects of Size Limits

The standard technique to control bloat, namely the application of a depth or length limit, is known to have significant effects on GP dynamics (see, for example, [1]). Unfortunately, we don't have a mathematical model for the limit distribution of sizes (neither in terms of internal nodes nor in terms of lengths) in the presence of length limits. However, we can conduct experimentation to study their effects on such a distribution. Figure 7 shows the affect of applying length limits of 25, 50 and 100 to the Artifical Ant problem. The effect of the length limit is that programs become more frequent in the smaller length classes. This over-sampling exacerbates the wasteful resampling of programs of smaller lengths.

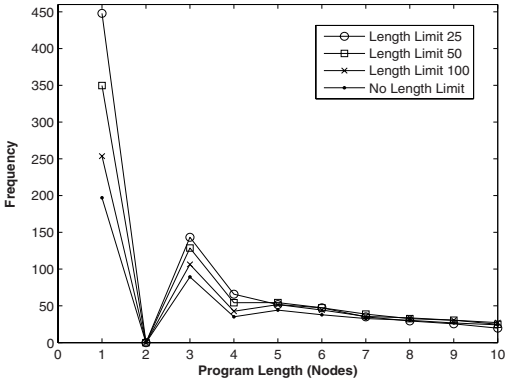

Fig. 7. Comparison of sampling frequencies associated with length limits for the Artificial Ant Problem applied to a flat fitness landscape

In the presence of fitness, this effect can be counteracted but not cancelled by selection. So, one should expect more sampling and resampling of short programs. However, following the line of reasoning of the crossover bias bloat theory [2], we know that for most problems these programs cannot be solutions, and in fact are typically very unfit, and, so, longer programs will be preferentially selected, leading to bloat. Thus, *size limits effectively increase the tendency to bloat since they induce more sampling of short programs, and, so, in the presence of non-flat fitness landscapes, GP populations rush towards the limit even more quickly than in the absence of the size limit!* This effect is particularly clear if one looks at the mode (the peak) of the program length distribution with and without length limits. Figure 8 shows how the mode (averaged over 100 independent runs) changes generation by generation for different limits in the case of the Parity problem (with selection). We can see that smaller size limits encourage GP to sample larger programs in the early generations before the size limit is reached. We found this effect because we looked into how the crossover sampling bias interacts with size limits. The effect has never been noticed before, probably because it becomes apparent only if one uses the right statistical tools: the mode of the size distribution (which is almost never used in reporting GP results).

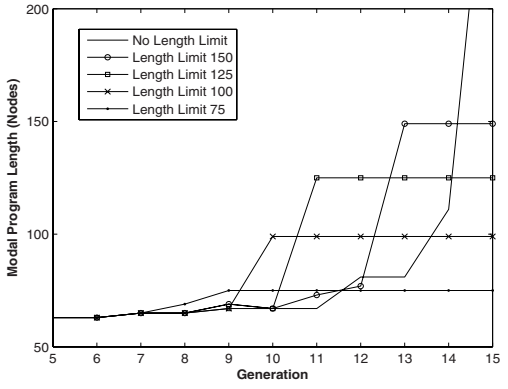

Fig. 8. Comparison of modal (peak) classes associated with length limits for the 4 Bit Even Parity Problem with selection

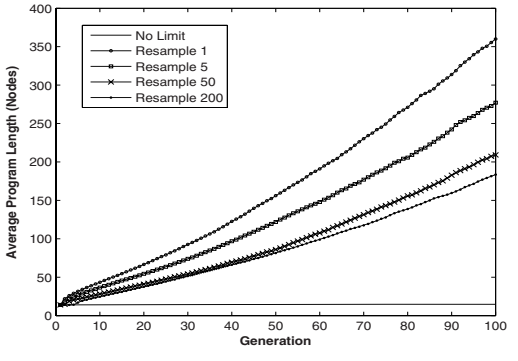

Fig. 9. Comparison of average program size applying resampling limits to the 4 Bit Even Parity Problem with a flat fitness landscape

These results suggest that, if size limits are imposed to combat bloat, then these should not be applied from generation 1, but much later and on demand, for example, if the average program size exceeds some pre-fixed threshold. This would avoid speeding up program growth in the early generations of a run.

Naturally, virtually all methods to combat bloat give more selective preference to shorter program than to longer ones. If in so doing they cause an oversampling of the short programs w.r.t. the base case (i.e., in the absence of the anti-bloat method)—which many do—then we should expect this phenomenon to still take place also with other bloat-control mechanisms, although perhaps with a lesser degree. We will explore this issue in future research.

5 Bloat and Sampling Parsimony

In section 3 we looked at how crossover likes to progressively sample smaller programs and the resulting resampling of programs, hence re-evaluations, that result from this. In this section we look at the prevention of resampling and its effect on program length.

To understand the effect of resampling and to control it, we have employed a novel technique which we have called *Sampling Parsimony*. This has two parameters, a resampling penalty to be applied, which is implemented as a percentage reduction of fitness, and a count of the number of times that a unique program can be sampled before that penalty is applied or removed.

Our first application is to look at how average program length will be affected by the application of a super penalty ensuring that a resampled program will not be reselected in the next generation. Using our standard ECJ problems with parameters as described in section 3 from Figure 9 we can see that, as we progressively prevent resampling by lowering our resampling limit, we increase the average size of the programs in our population. We have in effect created an effective fitness landscape [6] where the ability for a child to exist in the next generation is solely determined by whether that program has previously been sampled.

From our earlier analysis it, is unsurprising that we see that by depressing the fitness of resamples we will increase the sampling of larger programs, thereby increasing the average program size as we are in effect penalising smaller programs. What is more interesting is that we have managed to isolate the Crossover Bias bloating effect as described in [2]. Our method only penalises children and prevents them from being parents rather than preventing their creation. GP, therefore, uses larger programs as parents (see Figure 6), hence, increasing the average size of children and thereby increasing the average program size in the next generation. As smaller children are still created by crossover but have no chance of being chosen by selection, this process will continue. Even a relatively large resampling allowance of 200 on our flat landscape will greatly increase program size.

We apply our resampling penalty method to the Ant Problem with selection in Figure 10. We can see that our penalty, increases program growth within 100 generations. This is because we have, effectively, accelerated the Crossover Bias effect (crossover creating small programs that selection then ignores) already present in the 'No Limit' distribution. Practically, we can see that this acceleration only happens beyond a problem specific value of the number of resamples allowed, suggesting that experimental resampling restrictions may not attract significant additional program growth once an acceptable limit has been determined.[3]

Finally we reverse our method to apply a penalty to all programs from the beginning. We only remove the penalty after a specific number of resamples have been achieved, thereby allowing a program to be selected as a parent only after it has been sampled a number of times. From Figure 11 we can see that program growth is significantly reduced by applying a single sample penalty, whilst progressively increasing the sampling threshold before normal fitness is applied will reduce program growth towards a limit of approximately 50 samples.[4]

[3] Experimentation showed that no changes are observed beyond 5 resamples for the 4 Bit Even Parity problem, and approximately 15+ for the Artificial Ant.

[4] Approximately 5 for the Parity Problem, again the threshold is problem dependant.

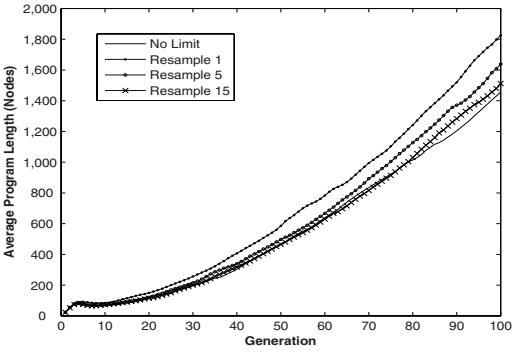

Fig. 10. Comparison of average program size applying resampling limits to the Artificial Ant Problem with selection

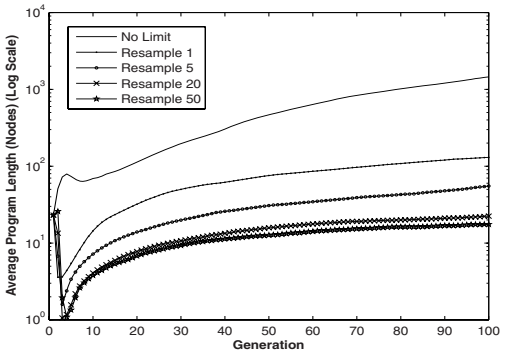

Fig. 11. Comparison of average program size applying sampling penalties to the Artificial Ant Problem with selection

Although its effect on bloat is self evident, it remains to be seen whether the sampling parsimony method can be successfully applied to improving overall program fitness over an entire run. We leave this for future work. The current 'blanket' method is of course very unsophisticated in that we prevent entire search spaces from being investigated without regard to program fitness. However, we believe that this remains an interesting technique that is worth exploring in greater depth and which might find application is a variety of areas, including, for example, escaping experimental stagnation under various conditions.

6 Conclusions

In this paper we have presented a limiting length distribution for GP with standard crossover with uniform selection of crossover points. This distribution now includes external nodes along with internal nodes, thereby extending previous research. Empirical validation confirms the accuracy of our model. Both theory and experiments show that the application of this form of crossover will quickly enable a population to converge to

a distribution that will exponentially sample smaller programs. As there are exponentially fewer unique smaller programs than larger ones, the sampling of new programs becomes less likely during a GP run if only crossover is applied. The effect becomes more prevalent as fitness values converge. This bias also becomes more acute with the application of a length limit, where, in addition to wasting more resources in resamples, it has further important consequences. In particular, we find that size limits initially speed up bloat, almost completely defeating their original purpose of combating bloat.

Although the application of selection before any fitness convergence will work against the crossover bias, smaller programs will always be created by crossover. As it is unlikely that these programs will be able to obtain a reasonable fitness, particularly during later stages of a GP run, they will be ignored by selection for the next generation and only larger parents will be selected. The continuing application of selection and crossover, therfore, causes the mean program size to increase, thereby creating bloat.

To explore what happens if one directly addresses this sampling-related cause for bloat, we have introduced a novel technique called Sampling Parsimony to tackle bloat. Curiously, this can be used accelerate growth as well as to reduce its effect. We have not, however, directly verified if Selection Parsimony is competitive with other anti-bloat techniques. We will address this question in future work.

References

1. Crane, E.F., McPhee, N.F.: The effects of size and depth limits on tree based genetic programming. In: Yu, T., Riolo, R.L., Worzel, B. (eds.) Genetic Programming Theory and Practice III, Ann Arbor, May 12-14. Genetic Programming, ch. 9, pp. 223–240. Springer, Heidelberg (2005)
2. Dignum, S., Poli, R.: Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In: Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.) GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, London, July 7-11, vol. 2, pp. 1588–1595. ACM Press, New York (2007)
3. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
4. Langdon, W.B.: How many good programs are there? How long are they? In: De Jong, K.A., Poli, R., Rowe, J.E. (eds.) Foundations of Genetic Algorithms VII, Torremolinos, Spain, Sepember 4-6 2002, pp. 183–202. Morgan Kaufmann, San Francisco (published, 2003)
5. Langdon, W.B.: Convergence of program fitness landscapes. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1702–1714. Springer, Heidelberg (2003)
6. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Heidelberg (2002)
7. Luke, S.: Two fast tree-creation algorithms for genetic programming. IEEE Transactions on Evolutionary Computation 4(3), 274–283 (2000)
8. Luke, S.: ECJ 13: A Java-based Evolutionary Computation Research System (2005), `http://cs.gmu.edu/~eclab/projects/ecj/`
9. Poli, R., Langdon, W.B., Dignum, S.: On the limiting distribution of program sizes in tree-based genetic programming. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 193–204. Springer, Heidelberg (2007)