Operator Equalisation and Bloat Free GP

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. Research has shown that beyond a certain minimum program length the distributions of program functionality and fitness converge to a limit. Before that limit, however, there may be program-length classes with a higher or lower average fitness than that achieved beyond the limit. Ideally, therefore, GP search should be limited to program lengths that are within the limit and that can achieve optimum fitness. This has the dual benefits of providing the simplest/smallest solutions and preventing GP bloat thus shortening run times. Here we introduce a novel and simple technique, which we call *Operator Equalisation*, to control how GP will sample certain length classes. This allows us to finely and freely bias the search towards shorter or longer programs and also to search specific length classes during a GP run. This gives the user total control on the program length distribution, thereby completely freeing GP from bloat. Results show that we can automatically identify potentially optimal solution length classes quickly using small samples and that, for particular classes of problems, simple length biases can significantly improve the best fitness found during a GP run.

Keywords: Genetic Programming, Search, Bloat, Program Length, Operator Equalisation.

1 Introduction

An intrinsic feature of traditional Genetic Programming (GP) is its variable length representation. Although, this can be considered one of the method's strengths, researchers have struggled with the phenomenon of bloat, the growth of program size during a GP run without a significant return in terms of program fitness, since GP's inception.

Numerous theories to explain bloat have been put forward including Replication Accuracy [1], Removal Bias [2], Nature of Program Search Spaces [3] and, more recently, Crossover Bias [4]. Numerous methods to control bloat have also been suggested [3,5,6], including, for example, size fair crossover or size fair mutation [7,8], Tarpeian bloat control [9], parsimony pressure [10,11,12], or using many runs each lasting only a few generations.

Research has shown that beyond a certain minimum program length the distributions of program functionality and fitness converge to a limit [13]. Before that limit, however, there may be program-length classes with a higher or lower average fitness than that achieved beyond the limit. Ideally, therefore, GP search should be limited to program lengths that are within the limit and that can achieve optimum fitness. We might want, for example, to restrict our search fixing program sizes at the point where our smallest optimal or near optimal solutions can be found thereby avoiding the need to search much larger spaces with the additional computational effort that entails. For most applications simpler solutions are also much more desirable than larger solutions.

In this paper we provide a method, *operator equalisation*, that can be applied easily to existing GP systems. This method forces GP to search specific length classes using pre-determined frequencies so that we can control the sampling rates of specific program lengths. As explained in Section 2, the method is very simple. This technique has several advantages. For example, whenever the length distribution and the corresponding sampling bias provided by standard operators is not suitable for a specific program space, we can change such a bias freely making it possible to sample or oversample certain length classes we believe can benefit our search. The user is given complete control over the program length distribution, and bloat can be entirely and naturally suppressed by simply asking operator equalisation to produce a static length distribution. We look at how different, static target length distributions can influence performance in Section 4. Furthermore, this method enables us to automatically sample and exploit the best fitness values associated with particular length classes as explained in Section 5.

2 Operator Equalisation

Investigations into the properties of program length have often used the tool of histogram representation in order to compare frequencies of programs sampled at particular lengths during a GP run [14,15,4]. Our operator equalisation method aims at controlling the shape of length histograms during a run. The method is loosely inspired by both the gray-level histogram equalisation method [16] used in image processing and digital photography to correct underexposed or overexposed pictures and the Tarpeian bloat control method [9] which, with a certain probability, by setting to zero the fitness of newly created programs of above average length effectively suppresses their insertion in the population. We have taken these ideas forward to see if by filtering which programs are allowed to be inserted in the population we can directly manipulate those frequencies in order to force GP to sample programs of particular lengths at pre-specified rates.

The method requires users to specify the desired length distribution (which we will call `target`) that they wish the GP system to first achieve and then continue to use when sampling a program space. This allows one to specify both simple well known probability distributions (Section 4) and also coarser grained models (Section 5). During the initialisation of the GP system a `histogram` object is created. This needs only to be primed with the maximum size allowed, number of bins (the size of the bins being calculated from these) and of course the `target` distribution. Then the method requires wrapping the existing code for offspring generation with code that simply accepts or disallows the creation of a child based on its length. The wrapper is extremely simple:

```
repeat {
  <create a child using standard genetic operators>
} until( histogram.acceptLength( child.length ) )
```

Internally, the `histogram` object maintains a set of numbers, one for each length class, which act as acceptance probabilities. The `acceptLength` method simply generates a uniform random number between 0 and 1 and compares it against the acceptance probability associated with the length class associated to `child`, returning `true` if the random number is less than the acceptance probability, and `false` otherwise.

At the end of each generation the `histogram` object updates the acceptance probabilities for each class using the following formula:

```
newProbability = currentProbability + ( normalisedDiff * rate )
```

where `normalisedDiff` is the difference between the desired frequency specified in `target` and the current frequency divided by the desired frequency. Small discrepancies for large classes are, therefore, largely ignored. The user defined parameter `rate` determines how quickly the distributions should converge. After some experimentation the setting `rate=0.1` was found to work well and has been used in all experimentation presented in this paper.

As one can see the method can easily be applied to existing GP applications with minimal changes: users need to change only very few lines of code in their existing GP systems.

3 Test Problems

We have deliberately chosen two GP problems of differing natures, a parity problem and a symbolic regression problem, to show the benefits and limitations of this approach. As we will show the first requires a relatively large program size before fitness will significantly improve whilst the second is able to achieve relatively high, though far from optimal, fitness values with small program sizes.

The Even Parity problem attempts to build a function that evaluates to 1 if an even number of boolean inputs provided evaluate to 1, 0 otherwise. We have chosen a relatively large input set of size 10. However, it is possible to evaluate all possible fitness cases (1024) for each potential solution within a reasonable time given the short length limit imposed.

Our second problem is a 10-variate symbolic regression problem: $x_1x_2+x_3x_4+x_5x_6+x_1x_7x_9+x_3x_6x_{10}$ as described in [9][1], which we have called Poly-10. 500 test cases are used each comprising of a (uniform) randomly generated value for each variable ranging between -1 and 1 and the resulting value of the equation.

As with the Even-10 problem only functions with arity 2 are used: ADD, SUBTRACT, MULTIPLY and a protected division function called PDIV which returns the denominator if the resulting division is less than 0.001. No Ephemeral Random Constants (ERCs) were used.

[1] This problem can be simplified to $x_1(x_2+(x_7x_9))+x_3(x_4+(x_6x_{10}))+x_5x_6$ to give a smallest GP tree size of 19 nodes.

Both problems have been sourced from [9] with minor alterations[2] to enable comparison and analysis. Each problem is expected to bloat under non-constrained conditions the reasons for which are described in the original paper.

4 Equalising to Simple Program Length Distributions

All experiments were initialised using the GROW method [17] with depth 6. For simplicity subtree swapping crossover with uniform selection of crossover points was applied without mutation or replication. Elitism was not applied. We used tournament selection with tournament size of 2 in experimentation. The algorithm was generational. All experiments used a population of 10,000 and ran for 100 generations. Results were averaged over 100 runs. It should be noted that due to the wrapper-like implementation there is no reason why mutation, replication or other forms of crossover could not be applied in isolation or combination. In fact it is hard to imagine any form of standard GP experimental set-up which could not be used easily.

In order to satisfy our stated desire of bloat free GP we have chosen a strict, deliberately small, length limit of 100 nodes. This has the added benefit of allowing us to evaluate a large set of fitness cases for each potential solution within acceptable experimental run times.

4.1 Does Operator Equalisation Work?

Initial investigation using our parity and regression problems showed that using a fairly unforgiving initialisation method (GROW), i.e., that in no way matched to our desired length distributions, we could equalise program lengths within approximately 20 generations. This is shown in Figure 1 for the Poly-10 problem equalised for a uniform length distribution.

With both problems there was a small dip for some of the early length classes. This is due to the fact that when the population has a uniform length distribution, crossover is less likely to produce very short programs than is ordinarily the case in the absence of equalisation. This is illustrated in Figure 2 where we look at the number of programs rejected by the wrapper at generation 100. As we can see the number of programs rejected for these length classes is extremely small. Our equaliser is, therefore, doing the best with what it has been presented by the underlying GP system.[3] The smallest class was always well populated. As the bias of subtree crossover towards sampling programs of a single node has been widely reported in the literature this is of little surprise.

[2] Our Even-10 problem has no NOT function. So all functions have an arity of 2. Also, Poly-10 here uses 500 fitness cases, where originally 50 were used.

[3] In other experiments (not reported) we found that the dip is slightly worsened by the use of larger tournament sizes since this increases the ability of selection to repeatedly present certain program sizes. The effect is, by contrast, reduced when using a steady state model, as GP can select newly created programs, i.e., those accepted by our equaliser, immediately without having to wait for a generation to be completed.

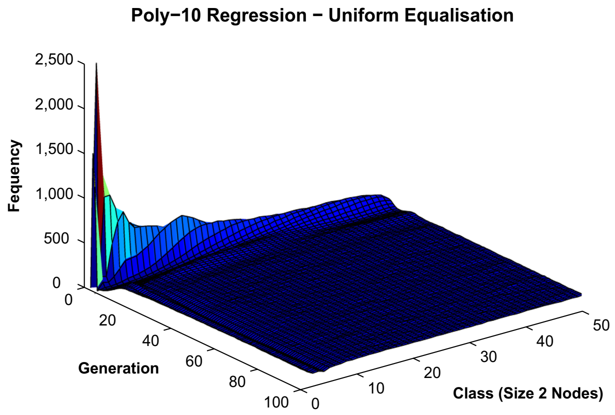

Fig. 1. Length histogram for Poly-10 regression problem with uniform equalisation of program length classes

Although it is possible to imagine extreme conditions where infinite loops could be encountered, for the experimentation detailed in the following sections, all runs were completed succesfully and no unusually large run-times were recorded. It is of course possible to add a simple retry limit to the wrapper code to escape such loops.

4.2 Efficiency of Different Length Distributions

Having established that our simple operator equalisation algorithm works for our test problem, we then applied this method to see how the use of elementary, easily recognisable, target probability distributions could affect our search. In this paper, we only consider static distributions, although operator equalisation works also with dynamic targets—a case that we will study in detail in future research.

In Figure 3 we see the final length distributions for the parity problem, i.e., at generation 100, for different target distributions. Each length class is 2 nodes in size. Given that all the functions in our function set (AND, OR, NOR, NAND, XOR and EQ) have an arity of 2, we have an individual class for every possible length up to our size limit.

We have chosen to look at a uniform distribution where each length is sampled with the same frequency, a triangular distribution which has a linearly increasing bias towards sampling larger programs, a 'reverse' triangle where smaller programs are sampled more often and a reverse exponential distribution where we sample larger programs exponentially more frequently than shorter ones. Note, the distribution for the length limit with no equalisation is also shown. In all cases the target distribution was reached very quickly. For example, after some initial fluctuations, as we can see in Figure 4, the average size for each of the experiments settled to a fixed value.

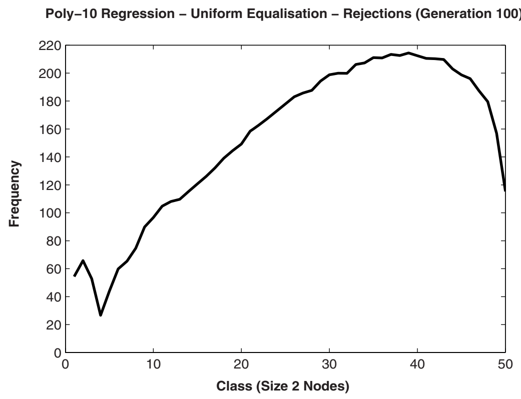

Fig. 2. Number of equaliser rejections at generation 100 for Poly-10 regression problem with uniform equalisation of program length classes

If we compare the best fitness values recorded for different target distributions (Figure 5), we can see that the push towards sampling larger programs has had a beneficial effect compared to using the simple length limit. The exponential distribution has a sharper upwards gradient for generations 20 to 60 than that of the triangular distribution although both eventually converge to the same value. The bias towards the sampling of smaller programs has had the most negative effect. Selection does, however, manage to improve fitness in all of our experiments. Perhaps surprisingly, all equalisation methods improve the best fitness value compared to the simple length limit during the early generations. The value of exploring certain length classes during early generations is discussed further below.

Unlike the Even-10 problem we can see in Figure 6 that the imposition of target length distributions has a negative effect on all forms of equalisation for best fitness compared to our simple length limit for the Poly-10 problem, any undersampling of smaller programs during the early generations having the most marked effect. It has long been known that in symbolic regression problems smaller programs can obtain relatively high fitness. In fact, the reverse triangle distribution performs as well as the simple length limit up to generation 15 and outperforms most other methods most of the time. This indicates that in this problem the dynamics of the length distribution is important, and GP benefits from exploring short programs for 10 or 15 generations and then progressively moves towards sampling longer programs, as GP with a simple length threshold does. So, this suggests that there could be benefits in using dynamic target distributions. As previously mentioned, we will explore this in future work.

Methods to detect this bias are discussed in the next section.

5 Length Class Sampling

As we have a method to directly influence the sampling of particular length classes, we can now look at two sampling techniques that can help us gain an

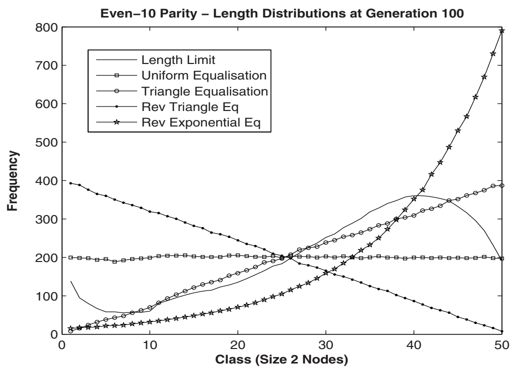

Fig. 3. Final length distributions for Even-10 parity problem using a strict length limit and different equalisation targets

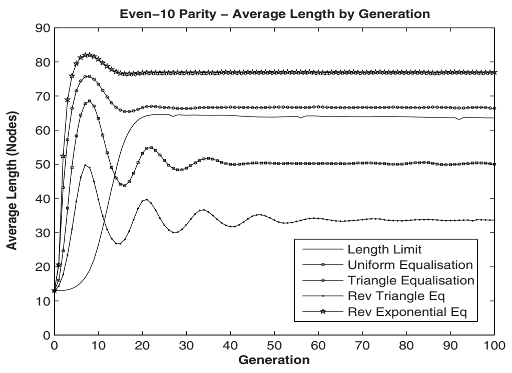

Fig. 4. Average length (nodes) for Even-10 parity problem using a strict length limit and different equalisation targets

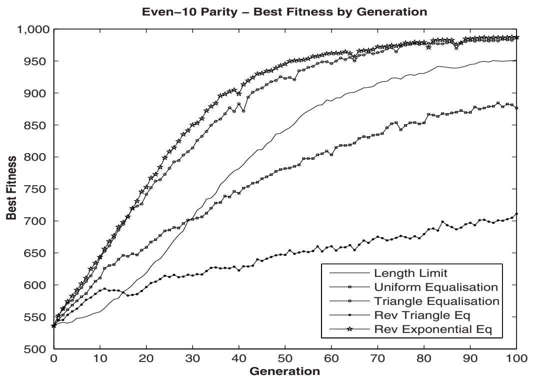

Fig. 5. Best fitness (number of test cases matched) for Even-10 parity problem using a strict length limit and different equalisation targets

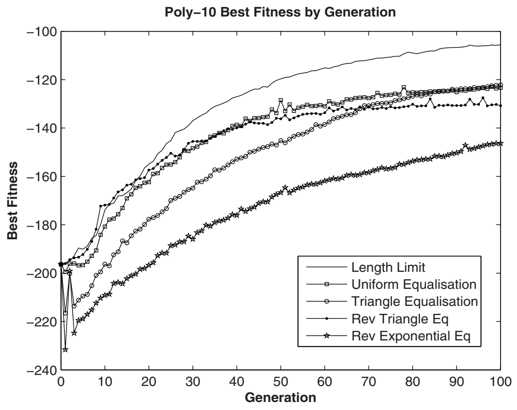

Fig. 6. Best fitness (Minus Mean Squared Error) for Poly-10 Symbolic Regression problem using a strict length limit and different equalisation methods

insight into the program space that we wish to search. We present these techniques in Sections 5.1 and 5.2. Both problems from the previous section were investigated using them.

Note that for the experiments described below we used the same GP system as in Section 4, but with two small, yet important, differences. Firstly, in order to remove any initial length bias the GROW initialisation method has been replaced with the RAND_TREE method described in [18]. Secondly, to show that useful insights into the program space of a problem can be achieved without undue computer resources, we have used both a smaller length limit of 80 nodes and a much reduced population size of 1,000.[4]

5.1 Single Length Classes

Using the RAND_TREE method we can sample without bias specific length classes. We can, therefore, look at the sampling of individual classes in isolation. For our experimentation the search space was divided into 20 equal length classes with each class sampling two distinct program lengths e.g. 1 and 3 for the first class, 5 and 7 for the second etc.[5] The objective was to find out which area (length class) of the search space would appear preferable to a GP system in the early generations of a run.

For the Even-10 problem (Figure 7) we can see quite clearly that there is small threshold where potential solutions cannot achieve anything better than 512 correct classifications, exactly half the total possible. However, as we move to larger program sizes we can see a distinct improvement in fitness. Selection will, therefore, quickly guide GP to larger programs in the early stages of a GP run.

[4] As we have used a smaller population size we cannot directly compare the best fitness results reported in this section with those reported in the previous section.

[5] Even sized programs are not possible for 2-ary trees.

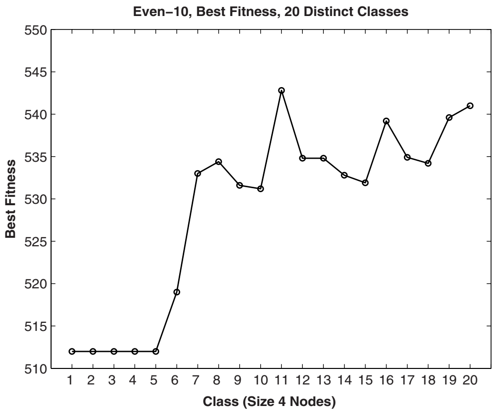

Fig. 7. Best fitness for Even-10 problem sampling 20 distinct size classes using the RAND_TREE method. Results are averages over 100 samples of 1,000 individuals each (1,000=GP population size).

Figure 8 shows that, for the Poly-10 problem, when we initially sample the program space, we find that the smallest programs do indeed have relatively better fitness than their larger counterparts. This explains GP's concentration in this area during earlier generations in the experimentation reported in Section 4. Of course, these areas do not contain optimal solutions: we need at least 19 nodes to achieve that. However, to an initial random sampling these areas display a higher proportion of relatively fit programs than those of the larger program size search spaces sampled. This explains why without histogram equalisation GP first samples the short programs but then quickly moves towards the longer programs, where, upon sufficiently sampling, better solutions can be found. This also explains why equalisation with a reverse triangular distribution does well initially, but cannot compete with standard GP later on (see Figure 6). Finally, it also explains why equalisation with distributions that sample the longer programs more frequently, such as the reverse exponential distribution, produce much worse fitness that standard GP and reverse triangular equalisation, initially.[6]

5.2 Multiple Length Classes

Of course the picture may change significantly if we sample two or more classes, perhaps with differing proportions. Also, what may look like a good sampling histogram initially (upon the random sampling produced by initialisation) may later turn out to be suboptimal after many generations of GP exploration. So, in this section we look at how the picture changes when using multiple length classes in combination and when comparing the initial to the final generation of runs.

[6] The fitness plot for the reverse exponential distribution in Figure 6, however, remains parallel to the plot of standard GP, suggesting that given enough generations histogram equalisation with this distribution would eventually catch up.

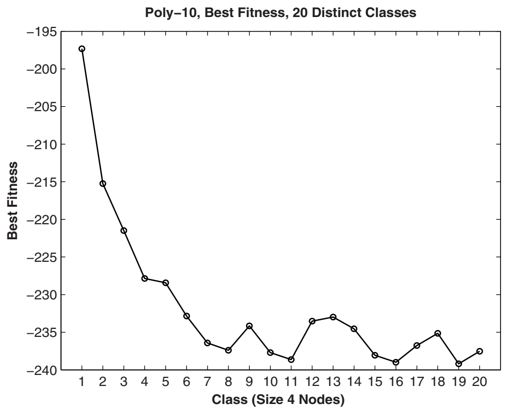

Fig. 8. Best fitness for Poly-10 problem in the same conditions as for Figure 7

To this end, we divided the length distribution into 4 bins of size 20 nodes and sampled each combination of bins using frequencies that were multiples of 20%. For example, bins 2 and 3 might have frequencies of 40% each, while bin 1 might have a frequency of 20% and bin 4 a frequency of 0%. Every combination (including those where some bins were empty) was sampled. There were 56 combinations in total. For each the resulting best fitness values at each generation were tabulated.

This produced a very large dataset, which we cannot report here due to space limitations. However, we report summaries of it in the form of the multiple linear regression formulas resulting from fitting the data at generations 0 and 100. The formulas will have the following form:

$$bestFitness = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 \tag{1}$$

β_0 is the constant term and β_i being the coefficient of each of the length classes X_i, X_1 being the smallest class. After the multiple linear regression was applied to the Even-10 problem the following formula was found for our initial generation:

$$bestFitness = 424.897 + 96.256X_1 + 103.899X_2 + 110.831X_3 + 113.911X_4 \tag{2}$$

As we can see there is a small improvement in best fitness as we search the larger classes. After applying GP search, at generation 100 the improvement is more distinct as shown by the regression formula:

$$bestFitness = 509.914 - 36.000X_1 - 12.000X_2 + 216.276X_3 + 342.748X_4 \tag{3}$$

For the Poly-10 problem for the first generation we obtain:

$$bestFitness = -179.595 - 15.509X_1 - 54.645X_2 - 56.678X_3 - 52.763X_4 \tag{4}$$

while after 100 generations the picture is somewhat different:

$$bestFitness = -147.146 - 33.712X_1 - 25.438X_2 - 46.835X_3 - 41.161X_4 \tag{5}$$

We can clearly see that for Poly-10 different parts of the search space yield different results for our initial generation and later stages of GP search. As one would expect from these results the best and worst combinations for our 100th generation showed a strong dislike for the third class. A 100% sampling of which, was indeed our worst result of -215.265, whilst more interestingly a broader sampling of the surrounding classes yielded the best results all of which were below -170.

6 Conclusions

In this paper we have introduced operator equalisation, a programatically simple method that can be easily applied to current experimental environments that allows us to finely bias GP search to specific program lengths. In particular, when method can force GP to sample the search space using static (and arbitrary) length distributions. This completely and naturally suppresses bloat.

We have applied this method to first see how simple bias can influence the results of two different but potentially bloating problems. The Even-10 parity problem was shown to have a simple positive bias towards longer programs within the 'experimentally-friendly' 100 node limit we have specified, whilst the Poly-10 regression problem was shown to have a positive bias towards the sampling of shorter programs during early generations.

Using simple statistical techniques we have then shown how we can use the method to quickly gain information about the search space and the best way to sample it with GP (with and without equalisation).

The primary aim of bloat free GP is to sample program spaces in such a way that we allow GP to discover optimal or acceptable near-optimal solutions without wasting resourses searching ever larger spaces with little return with regard to fitness. Here we have made some strong steps in this direction. An automatic method of defining the appropriate search space for a GP problem may not be so far off. For example, there is no reason why the method introduced in this paper cannot be applied to the initial setting of size limits (either maximum or minimum), or even to define a dynamic schedule for biasing the sampling of programs to certain sizes over the entire run or during different stages of a GP run.

References

1. McPhee, N.F., Miller, J.D.: Accurate replication in genetic programming. In: Eshelman, L. (ed.) Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA 1995), 15-19 July, pp. 303–309. Morgan Kaufmann, San Francisco (1995)
2. Soule, T., Foster, J.A.: Removal bias: a new cause of code growth in tree based evolutionary programming. In: 1998 IEEE International Conference on Evolutionary Computation, 5-9 May, pp. 781–786. IEEE Press, Los Alamitos (1998)
3. Langdon, W.B., Soule, T., Poli, R., Foster, J.A.: The evolution of size and shape. In: Spector, L., Langdon, W.B., O'Reilly, U.-M., Angeline, P.J. (eds.) Advances in Genetic Programming 3, vol. 8, pp. 163–190. MIT Press, Cambridge (1999)

4. 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, 7-11 July, vol. 2, pp. 1588–1595. ACM Press, New York (2007)
5. Soule, T., Foster, J.A.: Effects of code growth and parsimony pressure on populations in genetic programming. Evolutionary Computation 6(4), 293–309 (1998)
6. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. Evolutionary Computation 14(3), 309–344 (2006)
7. Langdon, W.B.: Size fair and homologous tree genetic programming crossovers. Genetic Programming and Evolvable Machines 1(1/2), 95–119 (2000)
8. Crawford-Marks, R., Spector, L.: Size control via size fair genetic operators in the PushGP genetic programming system. In: Langdon, W.B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N. (eds.) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, 9-13 July, pp. 733–739. Morgan Kaufmann Publishers, San Francisco (2002)
9. Poli, R.: A simple but theoretically-motivated method to control bloat in genetic programming. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 204–217. Springer, Heidelberg (2003)
10. Zhang, B.-T., Mühlenbein, H.: Evolving optimal neural networks using genetic algorithms with Occam's razor. Complex Systems 7, 199–220 (1993)
11. Zhang, B.-T., Mühlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evolutionary Computation 3(1), 17–38 (1995)
12. Zhang, B.-T., Ohm, P., Mühlenbein, H.: Evolutionary induction of sparse neural trees. Evolutionary Computation 5(2), 213–236 (1997)
13. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Heidelberg (2002)
14. Poli, R., McPhee, N.F.: Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tetamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, Springer, Heidelberg (2001)
15. 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, Springer, Heidelberg (2007)
16. Rosenfeld, A., Kak, A.C.: Digital Picture Processing, vol. 1,2. Academic Press, London (1982)
17. Koza, J.R. (ed.): Genetic Programming: On the Programming of Computers by Means of Natural Selection, USA, MIT Press, Cambridge (1992)
18. Iba, H.: Random tree generation for genetic programming. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 144–153. Springer, Heidelberg (1996)