Genetic Programming and Evolvable Machines, 3, 81–91, 2002

Some Considerations on the Reason for Bloat

W. BANZHAF banzhaf@cs.uni-dortmund.de
Department of Computer Science, Dortmund University, Joseph-vonFraunhofer-Str. 20, 44227 Dortmund, Germany

W. B. LANGDON w.langdon@cs.ucl.ac.uk
Computer Science, University College, London, Gower Street, London, WC1E 6BT, UK

Received March 27, 2001; Revised November 21, 2001

Communicated by: Lee Spector

Abstract. A representation-less model for genetic programming is presented. The model is intended to examine the mechanisms that lead to bloat in genetic programming (GP). We discuss two hypotheses ("fitness causes bloat" and "neutral code is protective") and perform simulations to examine the predictions deduced from these hypotheses. Our observation is that predictions from both hypotheses are realized in the simulated model.

Keywords: genetic programming, linear genomes, effective fitness, neutral variations

1. Introduction

It has been well established over recent years, that in genetic programming (GP), as well as in other evolutionary algorithms with length-changing genotypes, a tendency for growing length of genotypes occurs [1, 4–6, 9, 20, 21, 25, 27]. This phenomenon has been termed "bloat" [30]. The phenomenon seems to be widespread in the absence of explicit countermeasures. Over a wide range of representations of programs and operators, even irrespective whether programs at all are evolved [22], similar behaviour occurs. Bloat is both distracting, since the reasons for it have not yet been clearly understood, and it obstructs the search, because it forces the GP algorithm into stagnation.

When analyzing the results of runs documented in Koza [9] it was remarked [4] that neutral code, that is code which does not affect the fitness of the genetic program under consideration, constitutes a major percentage of individuals. Non-effective code, as it is also called,[1] has been linked to the effects of operators on the genome [2, 6]. Specifically, it was observed that the percentage of neutral code in an individual is closely connected to the number of neutral actions operators perform on an individual's genome [21].

In recent years, a general consensus has emerged in the GP community that the two mentioned phenomena of code growth and neutral code appearance are strongly related. When comparing the ratio of effective to noneffective code in runs under standard conditions, we observed usually that the ratio drops to between 1:3

and 1:7 [7]. Thus there seems to be an inherent pressure in the algorithm to evolve neutral code. We think that the correlation between growth of neutral code and bloat is a causal relationship: growth of neutral code causes bloat.

Whereas various remedies against bloat have been put forward, parsimony pressure [8, 31], variable fitness [5, 14], coevolutionary approaches [3], homologous and other specifically acting operators [5, 16, 26], to name a few, the explanation for bloat and the growth of noneffective code is still disputed.

Here we shall concern ourselves with the two simplest explanations that we shall call hypothesis 1 (H1) and hypothesis 2 (H2). H1 has become known as the "fitness causes bloat" hypothesis [10, 12–15]. H2 is known as the "neutral code is protective" hypothesis [21]. Other hypotheses have been formulated and are being actively examined elsewhere (see [19]). We shall try to devise experiments which should be able to shed some new light on the two hypotheses. Possible outcomes of this examination are that either one of these hypotheses is valid and the other invalid, or that both are valid or both are invalid. We shall introduce a very simple model system with which to study length changes of individuals. This model can be extended in various directions as will be indicated in the last section of this article.

Before we discuss the model, H1 and H2 need to be articulated in more detail.

H1: Usually, there are multiple ways of representing any given behavior in a genotype if a variable length representation is allowed [11]. Search starts from short representations. It retains previously discovered high fitness representations and uses them as points of departure for future search. After a while there is little chance of finding a representation that does better than the representation(s) from which it was created. So the fitness selection bias favors representations which have the same fitness as those from which they were created. The easiest way to create one representation from another and retain the same fitness is for the new representation to represent identical behavior. Thus, the search may become a random search for new representations of the best solution found so far. There are many more long representations than short ones for the same solution, so such a random search (other things being equal) will find more long representations than short ones.

H1 is a very general statement and is valid regardless of what type of search is conducted. In contrast, H2 references fitness of individuals explicitly:

H2: Neutral code promotes survivability of a genome because it provides locations in the genome where operators that would otherwise destroy the present fitness value can hit without a damaging effect. In fact the larger the fraction of neutral code to overall code is, the higher will be the probability for operators to strike in the neutral part, usually protecting the individual from deleterious changes. Thus the objective fitness of the algorithm is distorted in the presence of noneffective code, to the advantage of code with more neutral parts [2, 5, 6, 24].

There are a few general predictions characterizing these hypotheses:

P1 (supports H1): Beginning from short programs, the growth of code should be continuous because the number of better programs is always greater for larger size. This is true as long as there is no global size limit reached. It is notably true even if there is no fitness selection in effect.

P2 (supports H1 and H2): The advantage of neutral code is a marginal effect, becoming most visible when objective fitness cannot change any more or only with a very small probability.

P3 (supports H2): One should be able to measure a new observable, termed effective fitness, which can be associated with the distortion of fitness caused by the protective effect of neutral code.

Recently, effective fitness has been under discussion in the literature [5, 18, 21, 23, 28, 29]. Following [5] effective fitness of an individual j can be defined formally as

$$f_j^e = f_j\left[1 - \frac{C_j^e}{C_j^a} \cdot p \cdot p_j^d\right] \tag{1}$$

where f_j is the objective fitness, C_j^a is the absolute size of program j, C_j^e is the effective size of program j, and p, p_j^d are the application probability of the genetic operator under consideration and the probability of a destructive effect through application of it on program j, respectively.

2. The representation-less model

The model we are introducing here is representation-less. By that we mean that there are no features of the model that would link it to any sort of representation for programs. As a result, the system is stripped from all its domain-specific structures. If the observation is correct that qualitatively the same effects are visible in any sort of variable-length genomes (see Section 1), perhaps the phenomena in specific GP systems with representation are caused by these more general effects. Note that this model is really a "caricature" of a real GP system. As with all models, its focus is deliberately chosen—in this case to shed light on the code growth phenomenon regardless of the representation one is working in.

An individual in the representation-less model has only three features, an objective fitness, an effective length L_e and a noneffective length L_n. For simplicity, we shall assume all features to be positive integers ($\mathcal{N}$).

The population consists of M objects p^i, $i = 1, \ldots, M$ characterized by three features

$$p^i = (p_F^i, p_{L_e}^i, p_{L_n}^i)$$

with $p_F^i, p_{L_e}^i, p_{L_n}^i \in \mathcal{N}$, p_F^i being the fitness of the individual and $p_{L_e}^i + p_{L_n}^i = p_L^i$, the total length of the individual.

The GP algorithm is also modelled in a very simple fashion: First, operators work on single objects only (i.e., they resemble mutation more than crossover). Second, an operator changes the features of the individual to produce a variant: it either

changes its effective length and its fitness or it changes its noneffective length (in which case its fitness is unchanged by definition).

$$p^i \rightarrow p^{i\prime} = (p_F^{i\prime}, p_{L_e}^{i\prime}, p_{L_n}^{i}) \tag{2}$$

or

$$p^i \rightarrow p^{i\prime} = (p_F^{i}, p_{L_e}^{i}, p_{L_n}^{i\prime}) \tag{3}$$

Length changes, both effective and noneffective, happen in the smallest possible steps:

$$p_{L_e}^{i\prime} = p_{L_e}^{i} \pm 1 \tag{4}$$

or

$$p_{L_n}^{i\prime} = p_{L_n}^{i} \pm 1 \tag{5}$$

We shall assume that there is no intrinsic bias of the operator to prefer either length change direction.[2] With a probability of 50% there will be an increase in length and 50% of the time there will be a decrease. However the probability, P_e^i, of a length change in the effective code of an individual [cf. Equations (2) and (4)] will be dependent on the proportion of effective code in the individual:

$$P_e^i = \frac{p_{L_e}^{i}}{p_L^{i}} \tag{6}$$

The situation is similar to that of a normal GP individual (i.e., with a representation) where the probability of hitting the effective code is proportional to the ratio between effective and total code. It is clear from this that an individual has its own mechanism with which to regulate the changes in its effective code. Thus we can say that the individual has a means of self-adaptation.

Assume a certain variation step has led to a change in effective length of the individual. Then, according to Equation (2), there will also be a change in the fitness of the individual. For the sake of simplicity, in our model fitness changes assume the simplest possible procedure. Specifically,

1. the fitness landscape is uncorrelated,
2. there is an equal fitness distribution at all lengths,
3. a very large population is considered.

If the effective length changes we assign a new fitness to $p^{i\prime}$ by drawing at random from the fitness distribution given in Figure 1. The new individual $p^{i\prime}$ is compared with another individual p^j drawn from the population at random. The fitter of the two will survive and stay in the population. This is equivalent to tournament selection in a steady-state GP.

If a variation step leads to a change in noneffective length, the fitness of the offspring $p^{i\prime}$ is identical to that of the parent p^i. Again, a tournament between the variant $p^{i\prime}$ and another (possibly worse) individual p^j decides which one will survive. Note that changes in noneffective length provide an easy route to neutral changes while effective length changes are seldom neutral if the individual has very high (or very low) fitness.

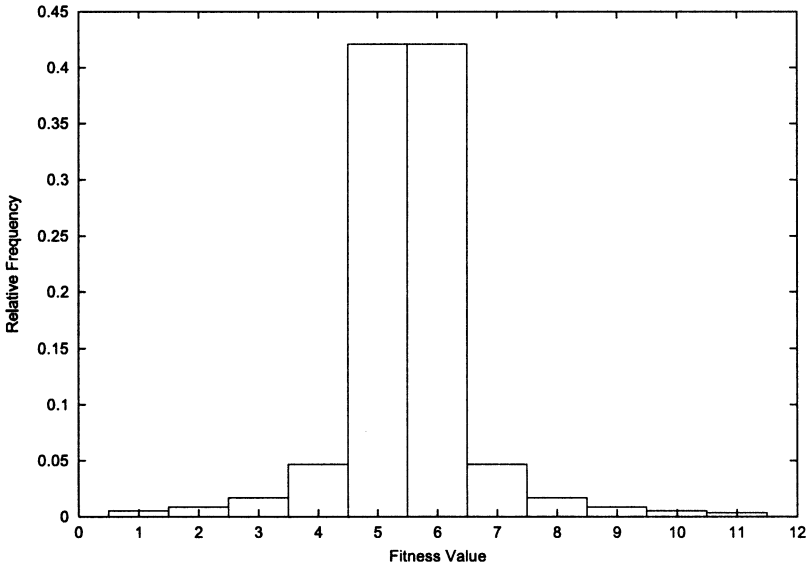

Figure 1. Fitness distribution for the experiments below. Fitness values are integers in the range $1 \ldots 11$. The best fitness of programs to be found is 11. Distribution is derived from $freq_i = (1/(i - (i_{\max}/2))^2) \times 1/Z$ with $i_{\max} = 11$ and Z the normalization factor.

3. Experiments

We performed a number of experiments with a fitness distribution according to Figure 1. This distribution is showing characteristic features of real fitness functions: A large number of average solutions, and a very small number of extremely high and low fitness values. The size of the population was 10000.

P1: Development with and without fitness selection

Figures 2 and 3 show a comparison of two runs with selection and without selection for different number of generations. With fitness selection the length of effective code shrinks and the length of noneffective code grows nearly linearly with time. Without selection there is still some increase in size. The linear growth and decay in the selective case is explained by Equations (4) and (5) which allow only a linear development. This is in agreement with [12].

P2a: Accepted longer improvements vs. accepted shorter improvements

Figure 4 shows the relation between accepted longer and accepted shorter improved programs. There is no statistically significant difference between the two. One can see clearly that accepted fitness improvement events become more and more sparse as the run progresses extending out to generation 1000.

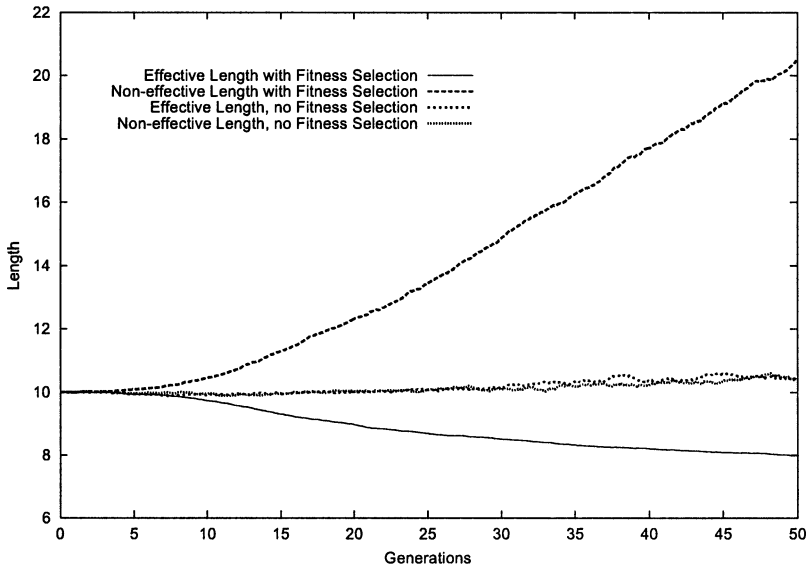

Figure 2. Comparative length development of a run with and a run without selection acting on the code. Short-term behavior for 50 generations. Behaviour starts to deviate visibly around generation 5.

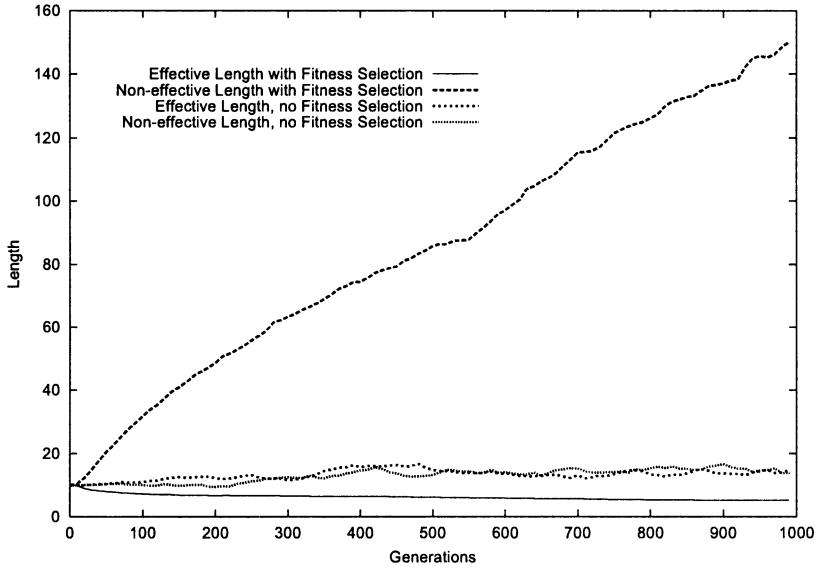

Figure 3. Comparative length development, same run as Figure 2. Long-term behavior for 1000 generations. Note that there is some growth of average length even in the absence of fitness-based selection: $\langle p_{L_n} \rangle = 14.7$, $\langle p_{L_n} \rangle = 13.8$, with $\langle\ \rangle$ signifying that the average is taken over the entire population.

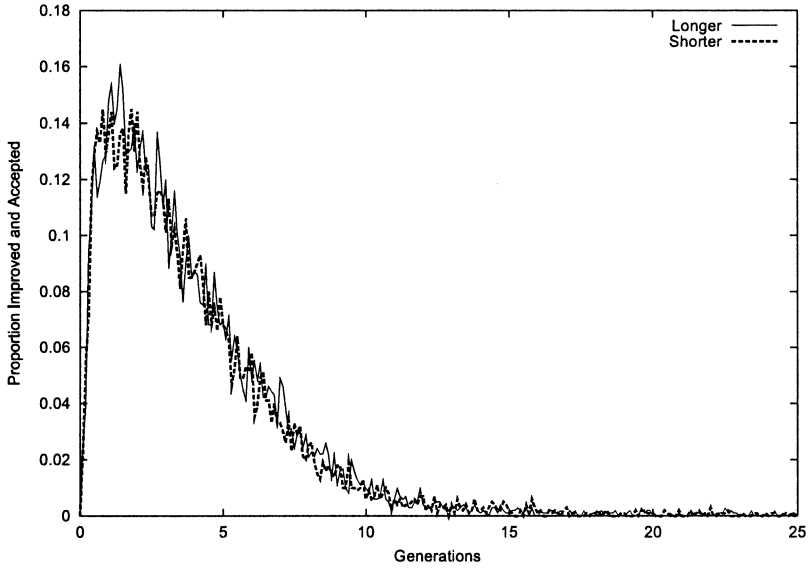

Figure 4. Development and comparison of accepted longer and shorter fitness improvements in a sample run. Successful improvements are most frequent in generation 2, before diminishing quickly and remaining low. No statistical significance can be given to the difference between the two lines.

P2b: Compare beginning and end of run in terms of accepted fitness neutral events

As noneffective length increases as a proportion of total length, the proportion of noneffective operations increases. Therefore, looking at the accepted neutral events, we would expect an increase in their total number. We can see from Figure 5 that there is only a small difference in favor of accepted elongation of programs compared to their shortening. This difference becomes more significant around generation 10, at a time when fitness improvements cease (cf. Figure 6).

P3: Compare effective vs. objective fitness and its predictive power for the survivability of a genome

Figure 6 shows the average fitness and the average effective fitness over the run. One can isolate a plateau of the effective fitness development around generation 5. Prior to that effective fitness was essentially equal to fitness. From generation 5 on, effective fitness shows distortion due to protection against the destructive effects of the operator. Note effective fitness starts to deviate from fitness earlier than length changes can be detected. Consequently it might be a very useful tool in predicting the further development of a run.

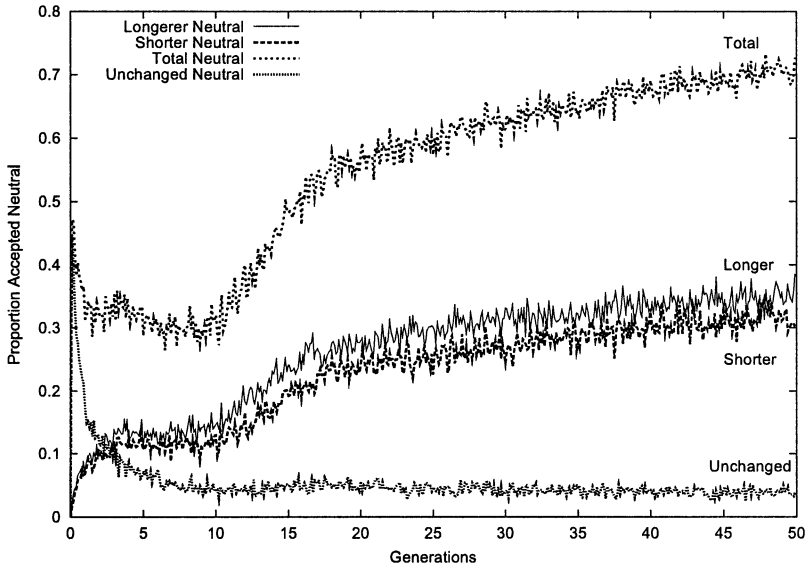

Figure 5. Evolution of acceptance of neutral code as a fraction of all accepted mutations. The total fraction of fitness neutral changes falls initially due to the initial decline in acceptance of neutral code of unchanged length. Total neutral events remain steady at about 30% but after generation 10 they start to grow steeply until generation 15 from whereon they grow slowly reaching about 70% of all neutral events around generation 50. There is only a small difference between accepted longer and shorter neutral variations.

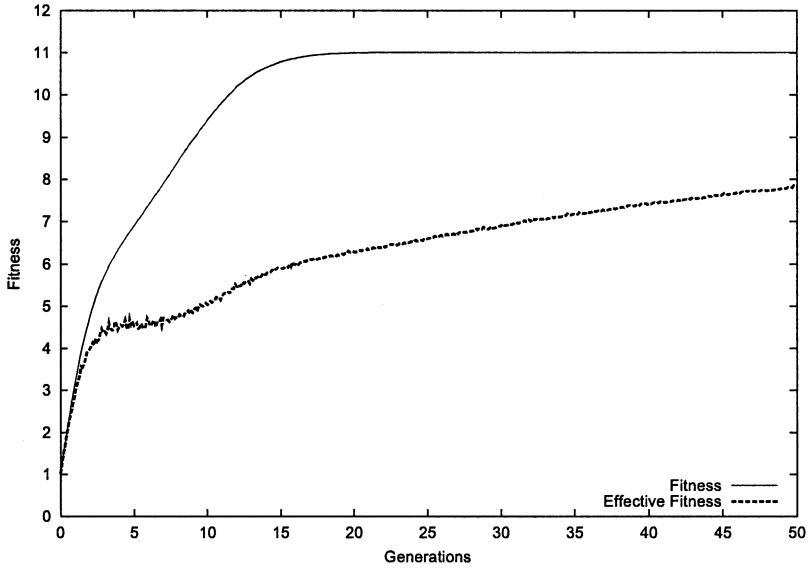

Figure 6. Development of fitness effective fitness [see Equation (1)] over a run. After 2 generations, effective fitness starts to deviate, clearly showing that destructive effects distort the objective fitness function. Effective fitness is only slowly approaching objective fitness, but is still on its way at generation 50.

4. Discussion

There are a number of ways the model introduced here can be made more realistic and complicated.

1. The fitness landscape can be made correlated [17]. For example, an offspring's fitness could be required to be in the "neighborhood" of the fitness of the parent. This could be achieved by drawing from a normal fitness distribution centered around the fitness of the parent.
2. The fitness distribution could be made to depend on the length of programs. For example, the distribution could be broader or sharper, depending on the effective length of programs. Specifically for very small length, one could choose an explicit fitness landscape, assigning each coordinate point a fitness.
3. Finite size effects of a small population size (relative to the search space) could be considered.
4. An equivalent to the crossover operator could be introduced, acting on two parents, and generating from one to n offspring which would inherit features from their parents in various ways.
5. The length-changing effects could be made nonlinear. Instead of changing length by a constant value [1 in Equations (4) and (5)] one could choose it to be proportional to the length.
6. Finally, an explicit representation for individuals and a fitness measure could be introduced, leading to the most realistic model to be considered.

5. Conclusion

In this article we have described a very simple model for variable-length genotypes. Using this model we discussed the two simplest explanations (fitness causes bloat" and "neutral code is protective") for code growth in Genetic Programming. The experiments have shown that both hypotheses have some explanatory power regarding the growth of code. At this point it seems too early to draw conclusions as to the relative strength of their effects. Thus we cannot say more than that there seems to be a superposition of effects at work in actual evolutionary systems.

This is our main point here: A length-unbiased operator that is applied according to a probability as in Equation (6) will inevitably lead to code growth. The model demonstrates that this is independent of the fitness landscape, the type of operator used and the representation of programs. In other words, if and when an operator is applied to the effective and noneffective parts of the code controlled by Equation (6), we predict code-growth.

Acknowledgment

We would like to thank the Deutsche Forschungsgemeinschaft (DFG) for support under Sonderforschungsbereich (SFB) 531.

Notes

1. Following Angeline [4], the term "introns" has also been used in recent years. For most GP systems, however, this is not appropriate. A notable exception is [7].
2. This might provide an additional source of bloat [27], but will be excluded here for the sake of simplicity.

References

1. L. Altenberg, "Emergent phenomena in genetic programming," in Proc. 3rd Annual Conf. Evolutionary Programming, A. Sebald and D. Fogel (eds.), World Scientific: Singapore, 1994, pp. 233–241.
2. L. Altenberg, "The evolution of evolvability in genetic programming," in Advances in Genetic Programming, K. Kinnear Jr. (ed.), MIT Press: Cambridge, MA, 1994, pp. 47–74.
3. P. J. Angeline and J. Pollack, "Competitive environments evolve better solutions for complex tasks," in Proc. Fifth Int. Conf. Genetic Algorithms (ICGA-93), Champaign-Urbana, IL. S. Forrest (ed.), Morgan Kaufmann: San Francisco, CA, 1993, pp. 264–270.
4. P. J. Angeline, "Genetic programming and emergent intelligence," in Advances in Genetic Programming, K. Kinnear Jr. (ed.), MIT Press: Cambridge, MA, 1994, pp. 75–98.
5. W. Banzhaf, P. Nordin, R. Keller, and F. Francone, Genetic Programming—An Introduction, Morgan Kaufmann: San Francisco, CA, 1998.
6. T. Blickle and L. Thiele, "Genetic programming and redundancy," in Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken), Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, Max-Planck-Institut für Informatik Technical Report Nr. MPI-I-94-241, 1994, pp. 33–38.
7. M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," IEEE Trans. Evolutionary Computation, vol. 5, pp. 17–26, 2001.
8. H. Iba, H. de Garis, and T. Sato, "Genetic programming using a minimum description length principle," in Advances in Genetic Programming, K. E. Kinnear Jr. (ed.), MIT Press: Cambridge, MA, 1994, pp. 265–284.
9. J. R. Koza, Genetic Programming, MIT Press: Cambridge, MA, 1992.
10. W. B. Langdon, "The evolution of size in variable length representations," in 1998 IEEE Int. Conf. Evolutionary Computation, Anchorage, Alaska, USA, 5–9 May 1998, IEEE Press: Piscataway, NJ, 1998, pp. 633–638.
11. W. B. Langdon, "Scaling of program tree fitness spaces," Evolutionary Computation, vol. 7, no. 4, pp. 399–428, 1999.
12. W. B. Langdon and R. Poli, "Fitness causes bloat," in Soft Computing in Engineering Design and Manufacturing. P. K. Chawdhry, R. Roy, and R. K. Pant (eds.), Springer-Verlag: London, 1997, pp. 13–22.
13. W. B. Langdon and R. Poli, "Fitness causes bloat: Mutation," in Proc. First European Workshop on Genetic Programming, W. Banzhaf, R. Poli, M. Schoenauer and T. Fogarty (eds.), vol. 1391 of LNCS, Paris, 14–15 April 1998. Springer-Verlag: Berlin, 1998, pp. 37–48.
14. W. B. Langdon and R. Poli, "Genetic programming bloat with dynamic fitness," in Proc. First European Workshop on Genetic Programming, W. Banzhaf, R. Poli, M. Schoenauer, and T. Fogarty (eds.), vol. 1391 of LNCS, Paris, 14–15 April 1998. Springer-Verlag: Berlin, 1998, pp. 96–112.
15. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster, "The evolution of size and shape," in Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U.-M. O'Reilly and P. J. Angeline (eds.), MIT Press: Cambridge, MA, 1999, pp. 163–190.
16. W. B. Langdon, "Size fair and homologous tree crossover for tree GP," Genetic Programming and Evolvable Machines, vol. 1, pp. 95–120, 2000.
17. W. B. Langdon and W. Banzhaf, "Genetic programming on difficult search spaces," in Proc. Parallel Problem Solving from Nature, Paris 2000, M. Schoenauer et. al. (eds.), LNCS 1917, Springer-Verlag: Berlin, 2000, pp. 201–210.
18. W. B. Langdon and R. Poli, Foundations of Genetic Programming, Springer-Verlag: Berlin, 2002.

19. S. Luke, "Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Bloat." PhD thesis, University of Maryland, College Park, MD, 2000.
20. N. McPhee and J. Miller, "Accurate replication in genetic programming," in Genetic Algorithms: Proc. Sixth Int. Conf. (ICGA95), Pittsburgh, PA, L. Eshelman (ed.), Morgan Kaufmann: San Francisco, CA, 1995, pp. 303–309.
21. P. Nordin and W. Banzhaf, "Complexity compression and evolution," in Proc. Sixth Int. Conf. Genetic Algorithms (ICGA95), Pittsburgh, PA, L. Eshelman (ed.), Morgan Kaufmann: San Francisco, CA, 1995, pp. 310–317.
22. P. Nordin, W. Banzhaf, and F. Francone, "Introns in nature and in simulated structure evolution," in Biocomputing and Emergent Computation, Proc. BCEC97 Skovde, Sweden, September 1–2, 1997, D. Lundh, B. Olsson, and A. Narayanan (eds.), World Scientific: Singapore, 1997, pp. 22–35.
23. R. Poli, "Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover," Genetic Programming and Evolvable Machines, vol. 2, pp. 123–163, 2001.
24. A. Singleton, Electronic Communication, 1994.
25. T. Soule, J. A. Foster, and J. Dickinson, "Code growth in genetic programming," in Genetic Programming 1996: Proc. First Annual Conf., Stanford, CA, J. Koza, D. Goldberg, D. Fogel, and R. Riolo (eds.), MIT Press: Cambridge, MA, 1996, pp. 215–223.
26. T. Soule and J. A. Foster, "Code size and depth flows in genetic programming," in Genetic Programming 1997: Proc. Second Annual Conf., Stanford, CA, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (eds.), Morgan Kaufmann: San Francisco, CA, 1997, pp. 313–320.
27. T. Soule, Code Growth in Genetic Programming. PhD thesis, University of Idaho, Moscow, ID, 1998.
28. C. Stephens and J. Mora Vargas, "Effective fitness as an alternative paradigm for evolutionary computation I: General formalism," Genetic Programming and Evolvable Machines, vol. 1, pp. 363–378, 2000.
29. C. Stephens and J. Mora Vargas, "Effective fitness as an alternative paradigm for evolutionary computation II: Examples and applications," Genetic Programming and Evolvable Machines, vol. 2, pp. 7–32, 2001.
30. W. A. Tackett, "Recombination, selection and the genetic construction of computer programs," Dissertation: Department of Electrical Engineering Systems, University of Southern California, Los Angeles, CA, 1994.
31. B.-T. Zhang and H. Mühlenbein, "Genetic programming of minimal neural nets using Occam's razor," in Proc. Fifth Int. Conf. Genetic Algorithms (ICGA-93), Champaign-Urbana, IL, S. Forrest (ed.), Morgan Kaufmann: San Francisco, CA, 1993, pp. 342–349.