EuroGP'98 First European workshop on Genetic Programming, W. Banzhaf, R. Poli, M. Schoenauer and T. C. Fogarty, editors, Paris, 14–15 April 1998, Springer-Verlag

Fitness Causes Bloat: Mutation

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. The problem of evolving, using mutation, an artificial ant to follow the Santa Fe trail is used to study the well known genetic programming feature of growth in solution length. Known variously as "bloat", "fluff" and increasing "structural complexity", this is often described in terms of increasing "redundancy" in the code caused by "introns". Comparison between runs with and without fitness selection pressure, backed by Price's Theorem, shows the tendency for solutions to grow in size is caused by fitness based selection. We argue that such growth is inherent in using a fixed evaluation function with a discrete but variable length representation. With simple static evaluation search converges to mainly finding trial solutions with the same fitness as existing trial solutions. In general variable length allows many more long representations of a given solution than short ones. Thus in search (without a length bias) we expect longer representations to occur more often and so representation length to tend to increase. I.e. fitness based selection leads to bloat.

1 Introduction

The tendency for programs in genetic programming (GP) populations to grow in length has been widely reported [Tac93, Tac94, Ang94, Tac95, Lan95, NB95, SFD96]. This tendency has gone under various names such as "bloat", "fluff" and increasing "structural complexity". The principal explanation advanced for bloat has been the growth of "introns" or "redundancy", i.e. code which has no effect on the operation of the program which contains it. ([WL96] contains a survey of recent research in biology on "introns"). Such introns are said to protect the program containing them from crossover [BT94, Bli96, NFB95, NFB96]. [MM95] presents an analysis of some simple GP problems designed to investigate bloat. This shows that, with some function sets, longer programs can "replicate" more "accurately" when using crossover. I.e. offspring produced by crossover between longer programs are more likely to behave as their parents than children of shorter programs. [PL98] argues the fraction of genetic material changed by crossover is smaller in longer programs. Such local changes may lead to GP populations becoming trapped at local peaks in the fitness landscapes. [RB96a] provides an analysis of bloat using tree schemata specifically for GP.

We advance a more general explanation which should apply generally to any discrete variable length representation and generally to any progressive search technique. That is bloat is not specific to genetic programming applied to trees using tree based crossover but should also be found with other genetic operators and non-population based stochastic search techniques such as simulated annealing and stochastic iterated hill climbing ([Lan98b] contains examples where such bloat does indeed occur and [LP98a] investigates bloat in GP with dynamic fitness functions).

The next section summarises our argument that bloat is inherent in variable length representations such as GP [LP97b]. In Sects. 3 and 4 we expand our previous analysis of a typical GP demonstration problem to include solution by mutation in place of crossover, again showing that it suffers from bloat and also showing that bloat is not present in the absence of fitness based selection. (This improves earlier experiments [LP97c] by removing the arbitrary limit on program size). Section 5 describes the results we have achieved and this is followed in Sect. 6 by a discussion of the potential advantages and disadvantages of bloat and possible responses to it. Finally Sect. 7 summarises our conclusions.

2 Bloat in Variable Length Representations

In general with variable length discrete representations there are multiple ways of representing a given behaviour. If the evaluation function is static and concerned only with the quality of each trial solution and not with its representation then all these representations have equal worth. If the search strategy were unbiased, each of these would be equally likely to be found. In general there are many more long ways to represent a specific behaviour than short representations of the same behaviour. Thus we would expect a predominance of long representations.

Practical search techniques are biased. There are two common forms of bias when using variable length representations. Firstly search techniques often commence with simple (i.e. short) representations, i.e. they have an in built bias in favour of short representations. Secondly they have a bias in favour of continuing the search from previously discovered high fitness representations and retaining them as points for future search. I.e. there is a bias in favour of representations that do at least as well as their initiating point(s).

On problems of interest, finding improved solutions is relatively easy initially but becomes increasingly more difficult. In these circumstances, especially with a discrete fitness function, there is little chance of finding a representation that does better than the representation(s) from which it was created. (Cf. "death of crossover" [Lan98a, page 206]. Section 5.6 shows this holds in this example). So the selection bias favours representations which have the same fitness as those from which they were created.

In general the easiest way to create one representation from another and retain the same fitness is for the new representation to represent identical behaviour. Thus, in the absence of improved solutions, the search may become a random search for new representations of the best solution found so far. As we said above, there are many more long representations than short ones for the same solution, so such a random search (other things being equal) will find more long representations than short ones. In GP this has become known as bloat.

3 The Artificial Ant Problem

The artificial ant problem is described in [Koz92, pages 147–155]. It is a well studied problem and was chosen as it has a simple fitness function. [LP98b] shows its simpler solutions have characteristics often associated with real world programs but that GP and other search techniques find it difficult (possibly due to bloat). 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 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 is used as the fitness measure of the control program.

The evolutionary system we use is identical to [LP97b] except the crossover operator is replaced by mutation. The details are given in Table 1, parameters not shown are as [Koz94, page 655]. On each version of the problem 50 independent runs were conducted. 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 node [LP98b]).

Table 1. Ant Problem

Objective:	Find an ant that follows the “Santa Fe trail”
Terminal set:	Left, Right, Move
Functions set:	IfFoodAhead, Prog2, Prog3
Fitness cases:	The Santa Fe trail
Fitness:	Food eaten
Selection:	Tournament group size of 7, non-elitist, generational
Wrapper:	Program repeatedly executed for 600 time steps.
Population Size:	500
Max program size:	32,767
Initial population:	Created using “ramped half-and-half” with a max depth of 6
Parameters:	90% mutation, 10% reproduction
Termination:	Maximum number of generations G = 50

4 Tree Mutation

For our purposes it is necessary that the mutation operator be able to change the size of the chromosomes it operates. Ideally this should be unbiased in the sense of producing, of itself, no net change in size. The following operator, in all cases examined, produces offspring which are on average almost the same size as their parent (cf. Sect. 5.7). (Note this may not be the case with different ratios of node branching factors in either the terminal/function set or in the evolving populations). An alternative tree mutation operator which also tries to avoid size bias by creating random trees of a randomly chosen size is proposed in [Lan98b].

The mutation operator selects uniformly at random a node (which may be a function or terminal) and replaces it and the subtree descending from it with a randomly created tree. The new tree is created using the same "ramped half-and-half" method used to create the initial population [Koz92, Page 92–93] however its maximum height is chosen at random from one to the height of the tree it is to replace. Note a terminal will always be replaced by a randomly selected terminal (i.e. there is no change in size) but a function can be replaced by a larger (but not deeper) or smaller tree, so that the program's size may change.

5 Results

5.1 Standard Runs

In 50 independent runs 9 found "ants" that could eat all the food on the Santa Fe trail within 600 time steps. The evolution of maximum and mean fitness averaged across all 50 runs is given by the upper curves in Fig. 1. (In all cases the average minimum fitness is near zero). These curves show the fitness behaving as with crossover with both the maximum and average fitness rising rapidly initially but then rising more slowly later in the runs. The population converges in the sense that the average fitness approaches the maximum fitness. However the spread of fitness values of the children produced in each generation remains large and children which eat either no food or only one food unit are still produced even in the last generation.

Figure 2 shows the evolution of maximum and mean program size averaged across all 50 runs. Although on average mutation runs show somewhat different behaviour of program length compared to crossover they are similar in that after an initial period bloat starts and program lengths grow indefinitely.

5.2 No Selection

A further 50 runs were conducted using the same initial populations and no fitness selection. As with crossover no run found a solution and the maximum, mean and other fitness statistics fluctuate a little but are essentially unchanged In contrast to the case with crossover only, our mutation operator has a slight bias. In the first 50 generation this causes the population to gradually increase in size at the rate of $\frac{1}{3}$ of a node per generation

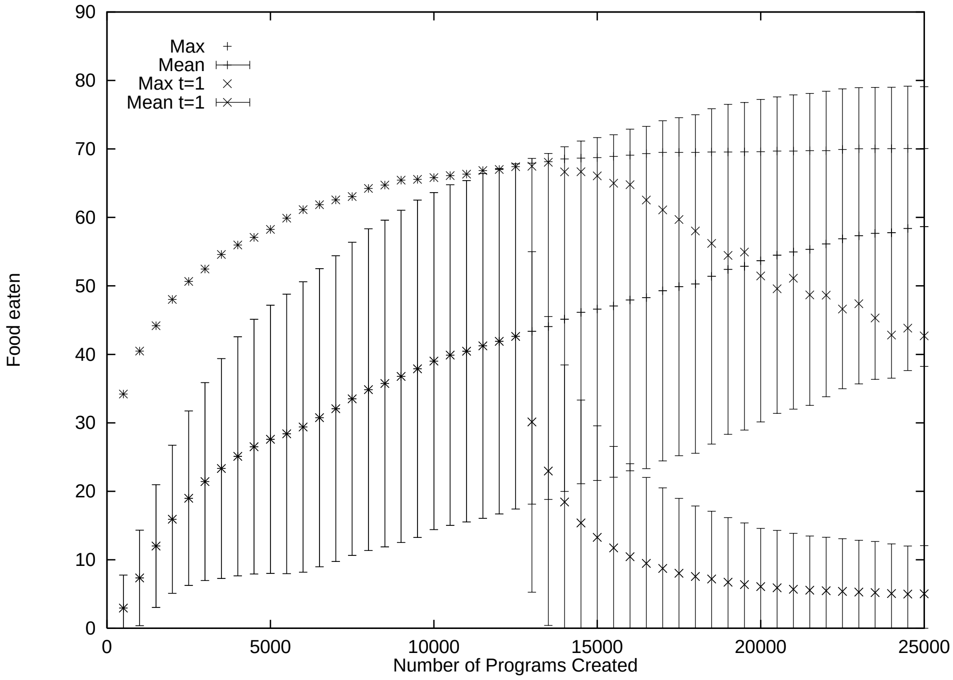

Fig. 1. Evolution of maximum and population mean of food eaten. t=1 curves show effect of removing fitness selection after generation 25. Error bars indicate one standard deviation. Means of 50 runs.

5.3 Removing Selection

A final 50 runs were conducted in which fitness selection was removed after generation 25 (i.e. these runs are identical to those in Sect. 5.1 up to generation 25, 12,500 programs created). The evolution of maximum and mean fitness averaged across all 50 runs is given in Fig. 1. As expected Fig. 1 shows in the absence of fitness selection the fitness of the population quickly falls.

After fitness selection is removed, the length of programs behaves much as when there is no selection from the start of the run. I.e. bloat is replaced by the slow growth associated with the mutation operator and the spread of program lengths gradually increases (cf. Fig. 2)

5.4 Fitness is Necessary for Bloat – Price's Theorem Applied to Representation Size.

Price's Covariance and Selection Theorem [Pri70] from population genetics relates the expected change in frequency of a gene Δq in a population from one generation to the next, to the covariance of the gene's frequency in the original population with the number of offspring z produced by individuals in that population:

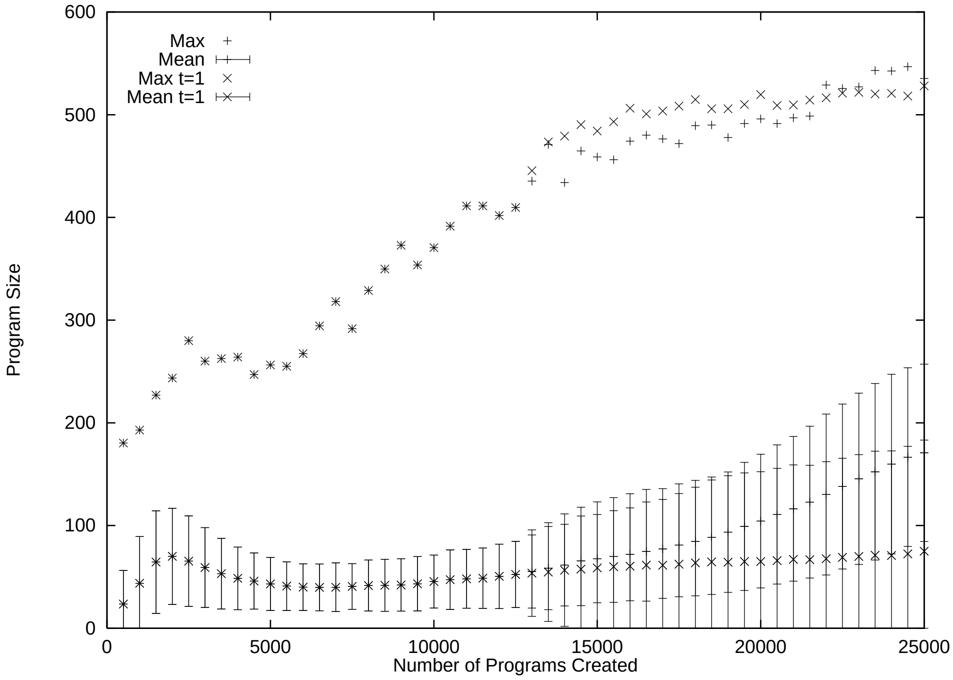

Fig. 2. Evolution of maximum and population mean program length. t=1 curves show effect of removing fitness selection after generation 25. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population. Means of 50 runs.

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

We have used it to help explain the evolution of the number of copies of functions and terminals in GP populations [LP97a, Lan98a]. In our experiments the size of the population does not change so $\overline{z} = 1$ and the expected number of children is given by the parent's rank so in large populations the expected change is approximately $\mathrm{Cov}(t(r/p)^{t-1}, q)$ as long as crossover is random. (t is the tournament size and r is each program's rank within the population of size p).

Where representation length is inherited, such as in GP and other search techniques, (1) should hold for representation length. More formally Price's theorem applies (provided length and genetic operators are uncorrelated) since representation length is a *measurement function* of the genotype [Alt95, page 28].

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

5.5 Correlation of Fitness and Program Size

In all cases there is a positive correlation between program score and length of programs. In the initial population the correlation is .37 on average, indicating that long random programs do better than short ones. This may be because they are more likely to contain useful primitives (such as Move) than short programs. Also short ones make fewer moves before they are reinitialised and re-executed. This may increase the chance of them falling into unproductive short cyclic behaviour that longer, more random, programs can avoid. Selection quickly drives the population towards these better individuals, so reducing the correlation to .13 In the absence of selection (or after selection has been removed halfway through a run) mutation tends to randomise the population so that the correlation remains (or tends towards) that in the initial population.

5.6 Effect of Mutation on Fitness

In the initial generations mutation is disruptive with 64.6% producing a child with a different score from its parent. However mutation evolves, like crossover, to become less disruptive. By the end of the run only 25.4% of mutants have a different score from their parent.

The range of change of fitness is highly asymmetric; many more children are produced which are worse than their parent than those that are better. By the end of the run, only 0.02% of the population are fitter than their parent. Similar behaviour has been reported using crossover on other problems [NFB96] [RB96b, page 183] [Lan98a, Chapter 8].

5.7 Mutation and Program Length

We see from Fig. 3 initially on average mutation makes little change to the length of programs, however, after bloat becomes established, our mutation operator produces children which are slightly shorter than their parents on average. I.e. once the change in the fitness of the population slows, program size bloats despite length changes introduced by mutation. The positive covariance of fitness and length shows this bloat is driven by fitness selection.

6 Discussion

6.1 Do we Want to Prevent Bloat?

From a practical point of view the machine resources consumed by any system which suffers from bloat will prevent extended operation of that system. However in practice we may not wish to operate the system continually. For example it may quickly find a satisfactory solution or better performance may be achieved by cutting short its operation and running it repeatedly with different starting configurations [Koz92, page 758].

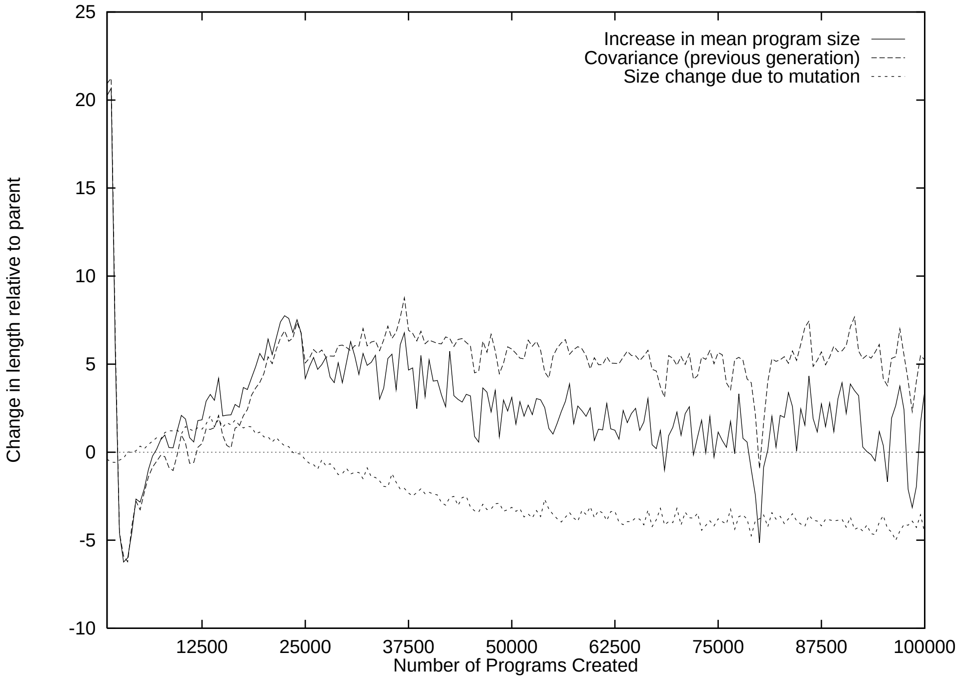

Fig. 3. Mean change in length of offspring relative to parent: normal runs. Means of 50 runs.

In some data fitting problems growth in solution size may be indicative of "over fitting", i.e. better matching on the test data but at the expense of general performance. For example [Tac93, page 309] suggests "parsimony may be an important factor not for 'aesthetic' reasons or ease of analysis, but because of a more direct relationship to fitness: there is a bound on the 'appropriate size' of solution tree for a given problem".

By providing a "defence against crossover" [NFB96, page 118] bloat causes the production of many programs of identical performance. These can consume the bulk of the available machine resources and by "clogging up" the population may prevent GP from effectively searching for better programs.

On the other hand [Ang94, page 84] quotes results from fixed length GAs in favour of representations which include introns, to argue we should "not ... impede this emergent property [i.e. introns] as it may be crucial to the successful development of genetic programs". Introns may be important as "hiding places" where genetic material can be protected from the current effects of selection and so retained in the population. This may be especially important where fitness criteria are dynamic. A change in circumstance may make it advantageous to execute genetic material which had previously been hidden in an intron. [Hay96] shows an example where a difficult GP representation is improved by deliberately inserting duplicates of evolved code.

In complex problems it may not be possible to test every solution on every aspect of the problem and some form of dynamic selection of test cases may be required [GR94]. For example in some cases co-evolution has been claimed to be beneficial to GP. If the fitness function is sufficiently dynamic, will there still be an advantage for a child in performing identically to its parents? If not, will we still see such explosive bloat?

6.2 Three Ways to Control Bloat

Three methods of controlling bloat have been suggested. Firstly, and most widely used is to place a universal upper bound either on tree depth [Koz92] or program length. ([GR96, LP97a] discuss unexpected problems with this approach).

The second (also commonly used) is to incorporate program size directly into the fitness measure (often called parsimony pressure) [Koz92, ZM93, IdS94]. [RB96a] gives an analysis of the effect of parsimony pressure which varies linearly with program length. Multi-objective fitness measures where one objective is compact or fast programs have also been used [Lan96].

The third method is to tailor the genetic operations. [Sim93, page 469] uses several mutation operators but adjusts their frequencies so a "decrease in complexity is slightly more probable than an increase". [Bli96] suggests targeting genetic operations at redundant code. This is seldom used, perhaps due to the complexity of identifying redundant code. [SFD96] showed bloat continuing despite their targeted genetic operations. Possibly this was because of the difficulty of reliably detecting introns. I.e. there was a route whereby the GP could evolve junk code which masqueraded as being useful and thereby protected itself from removal. While [RB96a] propose a method where the likelihood of potentially disruptive genetic operations increases with parent size.

7 Conclusions

We have generalised existing explanations for the widely observed growth in GP program size with successive generations (*bloat*) to give a simple statistical argument which should be generally applicable both to GP and other systems using discrete variable length representations and static evaluation functions. Briefly, in general simple static evaluation functions quickly drive search to converge, in the sense of concentrating the search on trial solutions with the same fitness as previously found trial solutions. In general variable length allows many more long representations of a given solution than short ones of the same solution. Thus (in the absence of a parsimony bias) we expect longer representations to occur more often and so representation length to tend to increase. I.e. current simple fitness based selection techniques lead to bloat.

In earlier work [LP97b] we took a typical GP problem and demonstrated with fitness selection it suffers from bloat whereas without selection it does not. In Sects. 3, 4 and 5 we repeated these experiments replacing crossover with mutation and showed fitness selection can still cause bloat. ([Lan98b] shows bloat can also occur with simulated annealing and hill climbing). We have demonstrated that if fitness selection is removed, bloat is stopped and program size changes little on average. As expected in the absence of selection, mutation is free to change program size at random and the range of sizes increases, as does the mean size (albeit very slowly). Detailed measurement of mutation confirms after an extended period of evolution, most are not disruptive (i.e. most children have the same fitness as their parents).

In Sect. 5.4 we applied Price's Theorem to program lengths within evolving populations. We confirmed experimentally that it fits unless bias in the genetic operators have significant impact. We used Price's Theorem to argue fitness selection is required for a change in average representation length. In Sect. 6 we discussed the circumstances in which we need to control bloat and current mechanisms which do control it but suggest a way forward may be to consider more complex dynamic fitness functions. Preliminary investigations in this direction are reported in [LP98a].

Acknowledgements

This research was funded by the Defence Research Agency in Malvern.

References

[Alt95] Lee Altenberg. The Schema Theorem and Price's Theorem. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[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.

[Bli96] Tobias Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*. PhD thesis, Swiss Federal Institute of Technology, Zurich, November 1996.

[BT94] Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, 1994. Max-Planck-Institut für Informatik (MPI-I-94-241).

[GR94] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature III*, pages 312–321, Jerusalem, 9-14 October 1994. Springer-Verlag.

[GR96] Chris Gathercole and Peter Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 291–296, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Hay96] Thomas Haynes. Duplication of coding segments in genetic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 344–349, Portland, OR, August 1996.

[IdS94] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 12, pages 265–284. 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, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[Lan96] William B. Langdon. Data structures and genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 20, pages 395–414. MIT Press, Cambridge, MA, USA, 1996.

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

[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. Forthcoming.

[LP97a] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 222–230, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[LP97b] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry, R. Roy, and R. K. Pan, editors, *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag London, 23-27 June 1997.

[LP97c] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In John Koza, editor, *Late Breaking Papers at the GP-97 Conference*, pages 132–140, Stanford, CA, USA, 13-16 July 1997. Stanford Bookstore.

[LP98a] W. B. Langdon and R. Poli. Genetic programming bloat with dynamic fitness. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, LNCS, Paris, 14-15 April 1998. Springer-Verlag. Forthcoming.

[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. submitted to GP-98.

[MM95] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[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, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[NFB95] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA, 9 July 1995.

[NFB96] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 6, pages 111–134. MIT Press, Cambridge, MA, USA, 1996.

[PL98] Riccardo Poli and William B Langdon. On the ability to search the space of programs of standard, one-point and uniform crossover in genetic programming. Technical Report CSRP-98-7, University of Birmingham, School of Computer Science, January 1998. submitted to GP-98.

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

[RB96a] Justinian P. Rosca and Dana H. Ballard. Complexity drift in evolutionary computation with tree representations. Technical Report NRL5, University of Rochester, Computer Science Department, Rochester, NY, USA, December 1996.

[RB96b] Justinian P. Rosca and Dana H. Ballard. Discovery of subroutines in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 9, pages 177–202. MIT Press, Cambridge, MA, USA, 1996.

[SFD96] Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Sim93] K. Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9:466–476, 1993.

[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. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 271–297, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[WL96] Annie S. Wu and Robert K. Lindsay. A survey of intron research in genetics. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 101–110, Berlin, Germany, 22-26 September 1996. Springer-Verlag.

[ZM93] Byoung-Tak Zhang and Heinz Mühlenbein. Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems*, 7:199–220, 1993.