GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, 8-12 July 2000

Quadratic Bloat in Genetic Programming

W. B. Langdon
Centrum voor Wiskunde en Informatica,
Kruislaan 413, NL-1098 SJ Amsterdam
W.B.Langdon@cwi.nl http://www.cwi.nl/~bill/
Tel: +31 20 592 4093, Fax: +31 20 592 4199

Abstract

In earlier work we predicted program size would grow in the limit at a quadratic rate and up to fifty generations we measured bloat O(generations$^{1.2-1.5}$). On two simple benchmarks we test the prediction of bloat O(generations$^{2.0}$) up to generation 600. In continuous problems the limit of quadratic growth is reached but convergence in the discrete case limits growth in size. Measurements indicate subtree crossover ceases to be disruptive with large programs (1,000,000) and the population effectively converges (even though variety is near unity). Depending upon implementation, we predict run time O(no. generations$^{2.0-3.0}$) and memory O(no. generations$^{1.0-2.0}$).

1 INTRODUCTION

It has been known for some time that programs within GP populations tend to rapidly increase in size as the population evolves [Koza, 1992, Altenberg, 1994, Tackett, 1994, Blickle and Thiele, 1994, Nordin and Banzhaf, 1995, Nordin, 1997, McPhee and Miller, 1995, Langdon, 1998b, Soule *et al.*, 1996, Langdon *et al.*, 1999]. If unchecked this consumes excessive machine resources and so is usually addressed either by enforcing a size or depth limit on the programs or by an explicit size component in the GP fitness [Koza, 1992, Iba *et al.*, 1994, Zhang and Mühlenbein, 1995, Rosca, 1997, Rosca and Ballard, 1996] although other techniques have been proposed [Ryan, 1994, Blickle, 1996, Nordin *et al.*, 1996, Soule and Foster, 1997, Soule, 1998, Hooper *et al.*, 1997, Angeline, 1998, Langdon, 2000]. However there is still interest in exploring why such bloat happens, its speed and the limits (if any) on bloat.

We extend [Langdon *et al.*, 1999, Langdon, 2000] to consider unbounded artificial evolution for hundreds of generations. Naturally bloat in such circumstances is extremely resource intensive and so we have been restricted to considering simple benchmark problems, one discrete and one continuous. A binary tree version of the Boolean 6 multiplexor problem [Koza, 1992] and a symbolic regression of a fourth order polynomial. At up to a million elements, these may be amongst the largest programs deliberately evolved so far.

In Section 2 we briefly restate our argument that bloat occurs on average at a sub-quadratic rate. Section 3 describes two experiments which look for the proposed quadratic limit, their results are given in Section 4. While Section 5 shows measurements of GP specific convergence, which explains why, in one case, the quadratic limit is not reached. Section 6 gives results on the potentially beneficial impact of commonly used depth and size limits. Section 7 discusses our theory and results in comparison with other theories of bloat. General conclusions for GP are drawn in Section 8.

2 THEORY

2.1 DISTRIBUTION OF FITNESSES

Provided programs are bigger than some problem and fitness level dependent threshold, the distribution of their fitness values in the GP search space does not change with length [Langdon, 1999b]. Figure 1 shows an example distribution of fitness against size. Note as size increases the lines tend to lie parallel to the y-axis, indicating little dependence upon size. Thus the number of programs with a given fitness is distributed like the total number of programs. The number of programs rises approximately exponentially with program length. Also most programs have a maximum depth near $2\sqrt{\pi(\text{internal nodes})}$ (ignoring terms O($N^{1/4}$) [Flajolet and Oldyzko, 1982]. See dashed parabola line on Figure 2.

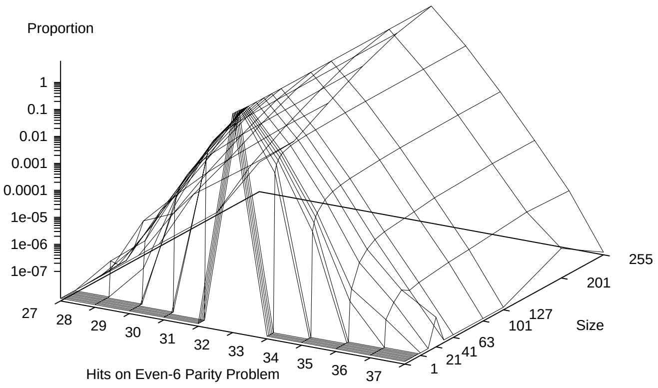

Figure 1: Proportion of trees which yield each fitness level on the even-6 parity problem. Trees of fitness levels 0..26 and 38..64 are very rare. Note log z scale.

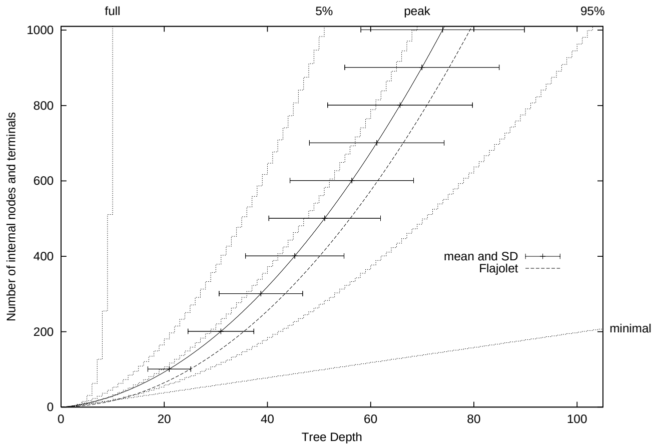

Figure 2: Distribution of binary trees by size and maximum depth. Solid line and error bars indicate the mean and standard deviation of the depth for trees of a give size. The dashed line is the large tree limit for the mean, i.e. $2\sqrt{\pi(\text{internal nodes})}$. The full tree and minimal tree limits are shown with dotted lines, as are the most likely shape (peak) and the 5% and 95% limits (which enclose 90% of all programs of a given size).

2.2 BLOATING

After a period GP (or any other stochastic search technique) will find it difficult to improve on the best trial solution it has found so far and instead most of the trial solutions it finds will be of the same or worse performance. Selection will discard those that are worse, leaving active only those that are as good as the best-so-far. In the absence of bias, the more plentiful programs with the current level of performance are more likely to be found [Langdon and Poli, 1997b]. The distribution of these is similar to the distribution of trees, therefore we expect the search to evolve in the direction of the most popular tree shape.

2.3 EVOLUTION OF DEPTH

In a variety of problems in earlier work [Langdon, 2000] program depth in the population in runs using standard crossover grew rapidly but apparently linearly. While problem dependent, on average trees grew roughly one level per generation. (Recent work suggests the shape of the initial trees is important).

2.4 EVOLUTION OF SHAPE

In the absence of size or depth limits, program shape evolves towards the ridge in the distribution of programs [Langdon *et al.*, 1999].

2.5 SUB-QUADRATIC BLOAT

If the programs within the population remain close to the ridge in the number of programs versus their shape and they increase their depth at a constant rate this leads to a prediction of sub-quadratic growth in their lengths'. (For modest size programs we expect size $O(\text{gens}^{1.3})$ rising to a limit of quadratic growth for $|\text{program}| \gg 1000$ cf. [Flajolet and Oldyzko, 1982, Table II]. Up to generation 50 [Langdon, 2000, Table 5] reports good agreement on average.

3 EXPERIMENTS

As we intend to run artificial evolution for hundreds of generations on rapidly bloating populations here we restrict ourselves to two problems, one Boolean (6-multiplexer [Koza, 1992, page 187]) and one continuous (symbolic regression of the quartic polynomial [Koza, 1992]). Note for speed the quartic problem is simplified and only uses ten test cases. As the distribution of the number programs' for each size and shape is known for binary trees we use only binary functions. Accordingly we introduce a new variation on the Boolean 6-multiplexer problem by replacing Koza's function set with his usual four binary Boolean operators. Using a 64 bit C++ compiler we can evaluate all the 6-multiplexer fitness cases in parallel [Poli and Langdon, 1999].

Apart from the binary function set, quartic problem population size, the absence of size or depth restrictions and the use of tournament selection our GP runs are essentially the same as [Koza, 1992]. Parameters are summarised in Tables 1 and 2.

Table 1: GP Parameters for the Quartic Symbolic Regression Problem

Objective:	Find a program that produces the given value of the quartic polynomial $x^2(x+1)(x-1) = x^4 - x^2$ as its output when given the value of the one independent variable, x, as input
Terminal set:	x and 250 floating point constants chosen at random from 2001 numbers between -1.000 and +1.000
Functions set:	$+ - \times$ % (protected division)
Fitness cases:	10 random values of x from the range -1 ... 1
Fitness:	The mean, over the 10 fitness cases, of the absolute value of the difference between the value returned by the program and $x^4 - x^2$
Hits:	The number of fitness cases (between 0 and 10) for which the error is less than 0.01
Selection:	Tournament group size of 7, non-elitist, generational
Wrapper:	none
Pop Size:	50
Max program:	10^6 program nodes
Initial pop:	Created using "ramped half-and-half" with depths between 8 and 5 (No uniqueness requirement)
Parameters:	90% one child crossover, no mutation. 90% of crossover points selected at functions, remaining 10% selected uniformly between all nodes.
Termination:	Maximum number of generations 600 or maximum size limit exceeded

Table 2: GP Parameters for Multiplexor Problem (as Table 1 unless stated)

Objective:	Find a Boolean function whose output is the same as the Boolean 6 multiplexor function
Terminal set:	D0 D1 D2 D3 A0 A1
Functions set:	AND OR NAND NOR
Fitness cases:	All the 2^6 combinations of the 6 Boolean arguments
Fitness:	number of correct answers
Pop size:	500
Max program:	10^6 program nodes
Initial pop:	Ramped half-and-half max depth between 2 and 6

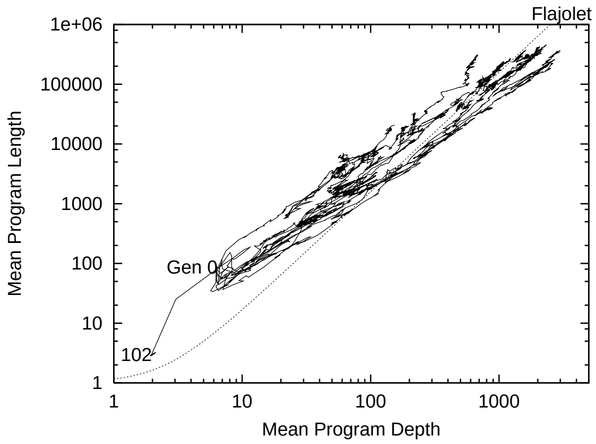

Figure 3: Evolution of tree shape in ten runs of the quartic symbolic regression problem. Note log scales.

4 RESULTS

4.1 QUARTIC SYMBOLIC REGRESSION

In nine of ten independent runs the population bloats. (In run 102 at generation 7, GP finds a high scoring program of one function and two terminals, which it is unable to escape from and the population converges towards it. After generation 35 approximately 90% of the population are copies of this local optima. Similar trapping is also reported in [Langdon, 1998b]). All populations complete at least 400 generations. However three runs stop before 600 generations when they reach the size limit (one million).

As expected in all runs most new generations do not find programs with a better fitness than found before. I.e. changes in size and shape are due to bloat. Figure 3 shows, while there is variation between the remaining runs, on average each population evolves to lie close to the ridge and moves along it, as predicted.

Figure 4 shows the average population depth varies widely between runs and in several runs the mean depth does not increase monotonically at a constant rate. However the mean of all ten runs is better behaved and increases at about 2.4 levels per generation.

Figure 5 shows the coefficient obtained by fitting a power law to the evolution of mean size of programs from generation 12 to later generations. Again there is wide variation between runs but on average the exponents start near 1.0 (generations 12–50) and steadily rises to 1.9 (between generations 12 and 400). I.e. towards the predicted quadratic limiting relationship between size and time.

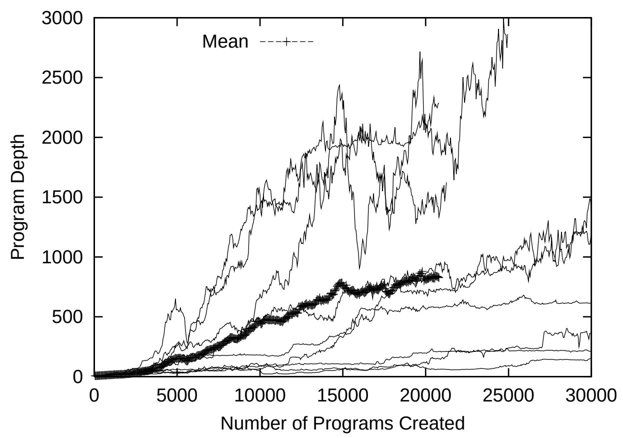

Figure 4: Evolution of tree depth in ten runs of the quartic symbolic regression problem.

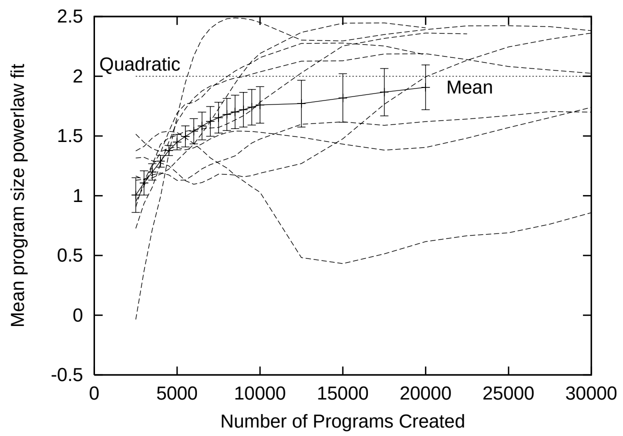

Figure 5: Evolution of power law coefficient in nine runs of the quartic symbolic regression problem (excludes run 102). Error bars show standard error.

4.2 BINARY 6-MULTIPLEXOR

In all runs most new generations do not find better programs and changes in size and shape are due to bloat. Figure 6 shows, while there is variation between runs, on average each population evolves to lie close to the ridge and moves along it, as predicted.

Figure 7 shows the average population depth varies widely between runs, as with the continuous problem, and in several runs the mean depth does not increase uniformly at a constant rate. However the mean of all ten runs is better behaved and increases at about 0.6 levels per generation.

Figure 8 shows the coefficient obtained by fitting a power law to the mean size of programs. Again there is wide variation between runs, however on average the

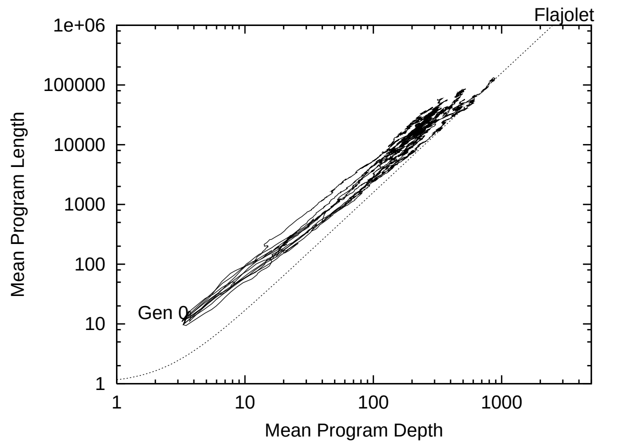

Figure 6: Evolution of tree shape in ten runs of the binary 6-multiplexor problem. Note log scales.

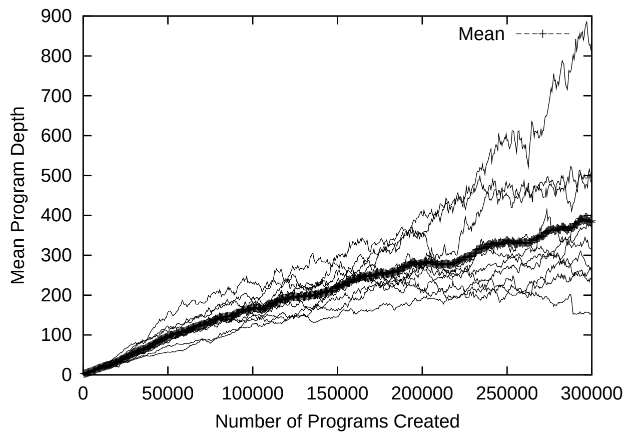

Figure 7: Evolution of tree depth in ten runs of the binary 6-multiplexor problem.

exponents start at 1.25 (generations 12–50) and rise. E.g. between generations 12 and 100 it has reached 1.4. By the end of the runs (generations 12–600) it reaches 1.5. This is approximately the same as for 6 multiplexor with the traditional, multi-arity, function set [Langdon, 2000].

5 CONVERGENCE

We consider but reject two possible explanations for the failure of the 6-multiplexor runs to reach a quadratic exponent:

Firstly, the power law coefficients for the two problems are statistically different. I.e. the difference (1.5 v. 2.0) is unlikely to be due to random variations.

Secondly, the multiplexor programs are shorter, so the Flajolet limit does not apply. However [Flajolet and

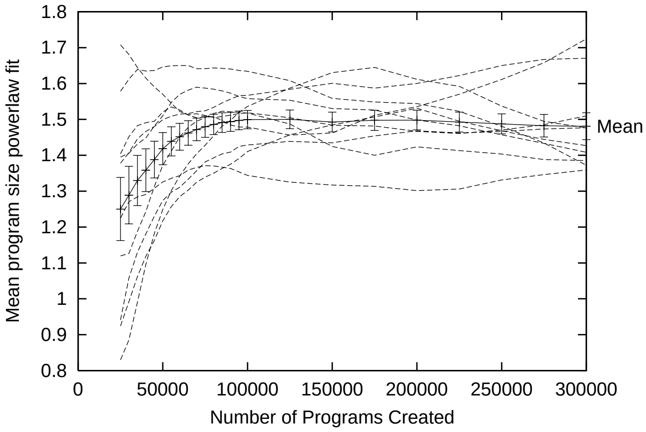

Figure 8: Evolution of power law coefficient in ten runs of the binary 6-multiplexor problem. Error bars indicate standard error.

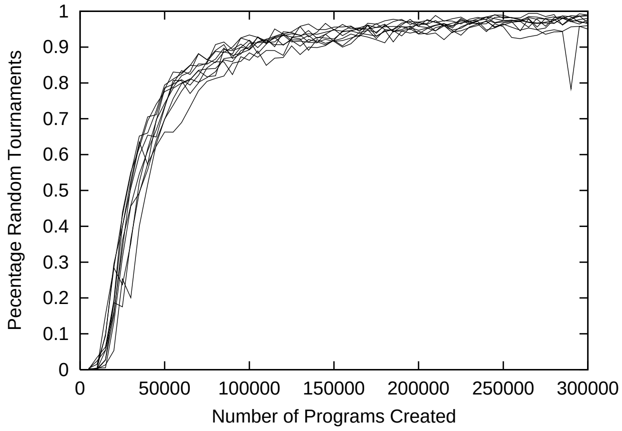

Figure 9: Fraction of selection tournaments where all 7 potential parents have the same fitness. For clarity we plot the mean of ten generations at a time. Ten runs of the binary 6-multiplexor problem.

Oldyzko, 1982, Table II] shows, the parabolic estimate is within 10% of the actual mean for programs of more than 1000 nodes. By generation 90 on average the bulk of the populations exceed 1000. I.e. for at least 500 generations the bulk of the 6 multiplexor populations are reasonably close to the Flajolet limit.

Having rejected these our proposed explanation is, in discrete problems, crossover may cease to be disruptive when the programs become very large. (Similar inability to effect big trees is reported in [Langdon and Nordin, 2000, Section 2]). In fact there are whole generations when every program in the population has an identical fitness. Therefore the selection pressure driving bloat falls as the populations grow in length and we suggest this is why the quadratic limit is not reached. [Tackett, 1994] also points to the importance of selection pressure on bloat. Figure 9 shows the fraction of parent programs selected entirely at random rises towards 100% in all ten runs.

6 DEPTH AND SIZE LIMITS

In this last experimental section we present an experiment on the quintic symbolic regression problem (parameters as [Langdon, 2000]). This shows we can approximately predict when depth and size limits commonly used in GP will have an impact on the evolving population. This is so early that we anticipate these limits usually have an effect. Depth limits will tend to cause bushier solutions to be produced. Bushier trees are more likely to solve some problems (e.g. the parity problems) than random trees [Langdon, 1999b]. Hence a depth limit could have an unanticipated benefit. Conceivably there are problems when the bias introduced by a size limit could be helpful. However [Gathercole, 1998, Langdon and Poli, 1997] show when the whole population presses against size or depth limits, they constrain subtree crossover possibly resulting in premature convergence.

Using the average depth of the initial population and rate of increase in depth from [Langdon, 2000, Table 4] we can estimate how long it will take for bloat to take the population on average to the depth limit (17). $\Delta\text{gens} = (17 - 3.64)/1.2 = 12$.

Figure 10 plots the mean sizes and depths for: no limits, a conventional depth limit (17) and a size limit of 200. They lie almost on top of each other until the fourth tick mark (corresponding to generation 12). Co-incidentally this is also the point where the size limited population diverges from the unlimited population. Given the variability between runs, the agreement between the prediction and measurements is surprisingly good.

7 DISCUSSION

[Tackett, 1994] and [Altenberg, 1994] both suggest Andy Singleton first proposed what is now often called the intron explanation for bloat. I.e. programs become longer not because longer programs are better but because they harbour more ineffective code (introns) than shorter ones. Since crossover chooses a point in the parent at random, having a higher ratio of ineffective to effective code, means there is more chance of crossover altering the ineffective code. In which case the children behave exactly like their parents. In a

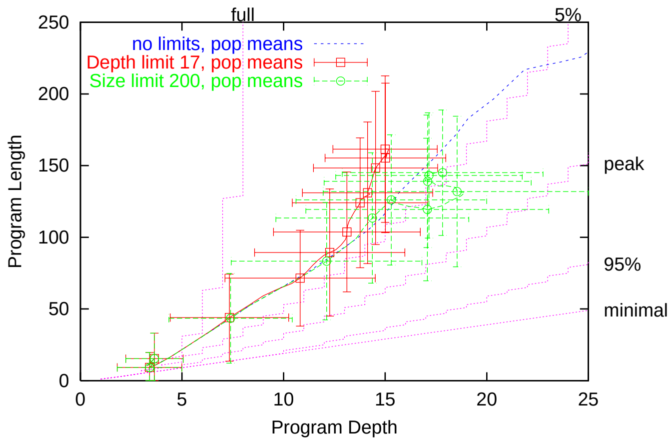

Figure 10: Evolution of population size and depth in a run of the quintic regression problem. Standard deviations shown every 5 generations.

conventional GP, this means children of longer parents will have a higher chance of having the same fitness as their parents than the children of shorter programs. Once bloat is underway, almost no improvements in fitness are made, so the struggle for survival goes not to the children who are better than their parents but to those that are the same as their parents. I.e. children of long programs have more chance of themselves being selected to have parents than children of shorter programs. (This implicitly assumes that long programs tend to produce long children). Since the same preference for children of long programs occurs in each generation, on average program size increases with time. As observed.

This is attractive but as Soule points out it is not the full story [Soule and Foster, 1997, Soule and Foster, 1998, Soule, 1998, Langdon *et al.*, 1999]. He suggests there is at least one additional mechanism involved. While, contrary to speculation in [Altenberg, 1994], subtree crossover on average introduces no size change. But there is a "removal bias" which means on average children which are smaller than their parents are less fit than children which are bigger. Soule also investigated the evolution of tree shape.

Note these mechanistic explanations don't tell us anything about the rate of bloat or predict ultimate shape. Indeed the iterative nature of the intron explanation has lead to false predictions of exponential growth.

[Rosca, 1997] suggests GP evolves programs from the root down. He and Soule point out (in side-effect free programs) introns can exist only at the edges of trees. I.e. all the code beneath an intron must also be ineffective. Stressing the importance of the code near the root, in the GP process. Poli also shows we can nullify the advantage of a high ineffective code ratio by having multiple operations (actually mutations). Using a constant mutation rate per *tree node*, the chance of altering effective code depends upon its size and is independent of the amount of ineffective code.

Often GP analysis assumes the population is made of full binary trees. This is obviously wrong but simple. Section 2 points out that GP populations tend to become like random trees. While apparently more complex than full binary trees, random trees do have nice mathematical properties that might make analysis of the evolution of populations of them tractable. In GP we often use function sets containing functions of more than one arity. This considerably complicates analysis. Fortunately its does not appear to dramatically change the properties of random trees or GP and it may be possible to continue to approximate the behaviour of real GP populations by assuming only binary functions. Or even the apparently extreme position of assuming totally random arities. The mathematics of such random trees has also been analysed.

The theory in Section 2 does not deal particularly with mechanisms and so is simple. It can be applied to non-GP search techniques and other types of search operator and gives a simple approximate prediction of average evolution of program shape. By making the assumption of linear increase in depth, Section 2 gives us a domain independent prediction of sub-quadratic bloat which can be used to give quantitative predictions in a given problem.

This is not to say the others are wrong. Far from it. (We and others have measured the growth of introns). Section 2 gives another way of thinking about evolution of trees.

8 CONCLUSIONS

Average growth in program depth when using standard subtree crossover with standard random initial trees is near linear. The rate is about 1 level per generation but varies between problems. When combined with the known distribution of number of programs, this yields a prediction of sub-quadratic, rising to quadratic, growth in program size. There is considerable variation between runs but on the average we observe sub-quadratic growth in size. Which in a continuous domain problem, rises apparently towards a quadratic power law limit.

Discrete		mean length$\leq$O(generations$^{2.0}$)
Continuous	$\lim_{g\to\infty}$	mean length=O(generations$^{2.0}$)

However in large discrete programs we observe a new type of genetic programming (GP) specific convergence: the whole population has the same fitness, even though its variety is 100%. This reduces the selection pressure on the population which, we suggest, is the reason why the growth remains sub-quadratic.

Most GP systems store each program separately and memory usage grows linearly with program size. I.e. O(generations$^{1.2-2}$). Run time is typically dominated by program execution time, which is proportional to its length [Langdon, 1998b, D.8], therefore run time O(generations$^{2.2-3}$).

In other systems the whole population is stored in a directed acyclic graph (DAG) [Handley, 1994]. New links are created at a constant rate. However memory usage may be less than O(generations), since every generation programs are deleted. In the absence of side effects and with a fixed fitness function it may be possible to avoid re-evaluating unchanged code by caching intermediate values. I.e. only code from the crossover point to the root would be execute. Then run time should be proportional to the average height of trees. So run time O(generations^2).

Note we refer to standard sub-tree crossover, other genetic operators and/or representations have different bloat characteristics. For example [Nordin and Banzhaf, 1995] suggests program size increases exponentially with generations in his linear machine code representation and crossover operator. While new mutation [Langdon, 1998a] and crossover operators [Langdon, 2000] can reduce bloat in trees.

GP populations using standard sub-tree crossover (and no parsimony techniques) quickly reach bounds on size or depth commonly used. When this will happen can be readily estimated. We suggest such bounds may have unanticipated (but problem dependent) benefits.

To allow big programs (1,000,000 nodes) to evolve we were restricted to simple problems which and can be solved by small trees. However our results do raise the question of how effective subtree crossover will be on complex discrete problems whose solutions are big programs. It may also be the case that subtree crossover will cease to be effective (i.e. explorative, disruptive) if the program is structured as big trees. GP may need to limit tree size (perhaps by evolving programs compose of many smaller trees) and/or alternative genetic operators may be required.

Acknowledgements

I would like to thank Wolfgang Banzhaf, the people of the Chair of Systems Analysis, University of Dortmund and to acknowledge funding of the "Deutsche Forschungsgemeinschaft", under project B2 of "Sonderforschungsbereich SFB 531".

References

[Altenberg, 1994] Lee Altenberg. Emergent phenomena in genetic programming. In A. V. Sebald and L. J. Fogel, editors, *Evolutionary Programming — Proceedings of the Third Annual Conference*, pp233–241. World Scientific Publishing, 1994.

[Angeline, 1998] Peter J. Angeline. Subtree crossover causes bloat. In John R. Koza *et. al.*, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp745–752, Wisconsin, 22-25 July 1998. Morgan Kaufmann.

[Blickle and Thiele, 1994] 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)*, pp33–38, 1994. Max-Planck-Institut für Informatik (MPI-I-94-241).

[Blickle, 1996] Tobias Blickle. Evolving compact solutions in genetic programming: A case study. In Hans-Michael Voigt *et. al.*, editors, *Parallel Problem Solving From Nature IV. LNCS* 1141, pp564–573, Berlin, 22-26 September 1996. Springer.

[Flajolet and Oldyzko, 1982] Philippe Flajolet and Andrew Oldyzko. The average height of binary trees and other simple trees. *Journal of Computer and System Sciences*, 25:171–213, 1982.

[Gathercole, 1998] Chris Gathercole. *An Investigation of Supervised Learning in Genetic Programming.* PhD thesis, University of Edinburgh, 1998.

[Handley, 1994] S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pp154–159, Orlando, 27-29 June. IEEE Press.

[Hooper *et al.*, 1997] Dale C. Hooper, Nicholas S. Flann, and Stephanie R. Fuller. Recombinative hill-climbing: A stronger search method for genetic programming. In John R. Koza *et. al.*, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp174–179. Morgan Kaufmann.

[Iba *et al.*, 1994] 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*, pp265–284. MIT Press, 1994.

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

[Langdon and Nordin, 2000] W. B. Langdon and J. P. Nordin. Seeding GP populations. In Riccardo Poli *et. al.*, editors, *Genetic Programming, Proceedings of EuroGP'2000*, *LNCS* 1802, pp304–315, Edinburgh, 15-16 April 2000. Springer-Verlag.

[Langdon and Poli, 1997] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza *et. al.*, editors, *Genetic Programming 1997*, pp222–230. Morgan Kaufmann.

[Langdon and Poli, 1997b] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry *et. al.*, editors, *Soft Computing in Engineering Design and Manufacturing*, pp13–22. Springer London, 1997.

[Langdon *et al.*, 1999] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In Lee Spector *et. al.*, editors, *Advances in Genetic Programming 3*, pp163–190. MIT Press.

[Langdon, 1998a] W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, pp633–638, Anchorage, 1998. IEEE Press.

[Langdon, 1998b] W. B. Langdon. *Data Structures and Genetic Programming*, Kluwer, 1998.

[Langdon, 1999b] W. B. Langdon. Scaling of program tree fitness spaces. *Evolutionary Computation*, 7(4):399–428, Winter 1999.

[Langdon, 2000] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119, April 2000.

[McPhee and Miller, 1995] Nicholas F. McPhee and J. D. Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pp303–309, 1995. Morgan Kaufmann.

[Nordin and Banzhaf, 1995] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *ICGA95*, pp310–317, 1995. Morgan Kaufmann.

[Nordin *et al.*, 1996] 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*, pp111–134. MIT Press, 1996.

[Nordin, 1997] Peter Nordin. *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, der Universitat Dortmund am Fachereich Informatik, 1997.

[Poli and Langdon, 1999] Riccardo Poli and W. B. Langdon. Sub-machine-code genetic programming. In Lee Spector *et. al.*, editors, *Advances in Genetic Programming 3*, pp301–323. MIT Press, 1999.

[Rosca and Ballard, 1996] Justinian P. Rosca and Dana H. Ballard. Complexity drift in evolutionary computation with tree representations. Tech Report NRL5, University of Rochester, Computer Science.

[Rosca, 1997] Justinian P. Rosca. Analysis of complexity drift in genetic programming. In John R. Koza *et. al.*, editors, *Genetic Programming 1997*, pp286–294. Morgan Kaufmann.

[Ryan, 1994] Conor Ryan. Pygmies and civil servants. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pp243–263. MIT Press, 1994.

[Soule and Foster, 1997] Terence Soule and James A. Foster. Code size and depth flows in genetic programming. In John R. Koza *et. al.*, editors, *Genetic Programming 1997*, pp313–320. Morgan Kaufmann.

[Soule and Foster, 1998] Terence Soule and James A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation*, pp781–186, 1998. IEEE Press.

[Soule *et al.*, 1996] T. Soule, J. A. Foster, and J. Dickinson. Code growth in genetic programming. In John R. Koza *et. al.*, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp215–223, Stanford, 28–31 July 1996. MIT Press.

[Soule, 1998] Terence Soule. *Code Growth in Genetic Programming*. PhD thesis, University Idaho, 1998.

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

[Zhang and Mühlenbein, 1995] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.