Elitism Reduces Bloat in Genetic Programming

Riccardo Poli
Department of Computing and Electronic Systems
University of Essex
Colchester, UK
rpoli@essex.ac.uk

Nicholas Freitag McPhee
Division of Science and Mathematics
Univ. of Minnesota, Morris
Morris, MN, USA
mcphee@morris.umn.edu

Leonardo Vanneschi
Department of Informatics, Systems and Communication
University of Milano-Bicocca,
Milan, Italy
vanneschi@disco.unimib.it

ABSTRACT

Elitism is commonly used in generational GP to ensure that the best individuals discovered in a generation are not lost, and are made available for possible further improvements to new generations. Using two GP systems and four problems, we show how elitism reduces the growth of mean program size.

Categories and Subject Descriptors: I.2.2 [**Artificial Intelligence**]: Automatic Programming

General Terms: Algorithms, Performance

Keywords: Genetic Programming, Bloat, Elitism

1. INTRODUCTION

Elitism is a technique, commonly used in GP, where one or more of the highest-fitness individuals are copied, unchanged, from one generation to the next. The ratio N/M between the elite size, N, and the population size, M, is called the *elite fraction*. In a recent study of the evolution of robustness in GP, Piszcz and Soule [4] used different degrees of elitism in one of their four sets of experiments (Experiment 3, a symbolic regression problem with $\sin(x)$ as target function). In that experiment they analysed how different levels of elitism affected the robustness of the best-of-run individuals, and found that the average size of such individuals decreased as the elite fraction was increased from 0 to 1% in steps of 0.2%.

In this paper we extend this study and show that elitism modulates the dynamics of the mean program size of a GP population with two different GP systems and four different problems, using elite fractions from 0 to 50%. We find that elitism often substantially slows the rate of bloat [5] in the later generations of a run.

2. GP SYSTEMS AND PROBLEMS

Our first system is a linear generational GP system. It initialises the population by repeatedly creating random individuals with lengths uniformly distributed between 1 and 100 primitives; the primitives are drawn randomly (uniformly) from each problem's primitive set. The system uses fitness proportionate selection and crossover applied with a rate of 100%. Crossover creates offspring by selecting a random crossover point in each parent, taking the first part of the first parent and the second part of the second w.r.t. their crossover points. We used populations of size 1,000, performing 100 independent runs. Runs lasted 100 generations.

With the linear GP system we used two families of test problems: `Polynomial` and `Lawn-Mower`. `Polynomial` is a symbolic regression problem where the objective is to evolve a function which fits a degree d polynomial of the form $x + x^2 + \cdots + x^d$, for x in the range $[-1,1]$. Polynomials of this type have been widely used as benchmark problems in the GP literature. In particular we considered degree $d = 6$, and we sampled the polynomial at the 21 equally spaced points $x \in \{-1.0, -0.9, \ldots, 0.9, 1.0\}$. We call the resulting problem instance `Poly-6`. Fitness was the sum of the absolute differences between target polynomial and the output produced by the program under evaluation over these 21 fitness cases. For this problem we considered a primitive set including the following instructions: `R1 = RIN`, `R2 = RIN`, `R1 = R1 + R2`, `R2 = R1 + R2`, `R1 = R1 * R2`, `R2 = R1 * R2`, and `Swap R1 R2`. The instructions refer to three registers: the input register `RIN` which is loaded with the value of x before a fitness case is evaluated and the two registers `R1` and `R2` which can be used for numerical calculations. `R1` and `R2` are initialised to x and 0, respectively. The output of the program is read from `R1` at the end of its execution.

`Lawn-Mower` is a variant of the classical Lawn Mower problem introduced by Koza in [2]. As in the original version of the problem, we are given a square lawn made up of grass tiles. In particular, we considered lawns of size 10×10. The objective is to evolve a program which allows a robotic lawnmower to mow all the grass. In our version of the problem, at each time step the robot can only perform one of three actions: move forward one step and mow the tile it lands on (`Mow`), turn left by 90 degrees (`Left`) or turn right by 90 degrees (`Right`). Fitness (to be minimised) was measured by the number of tiles left unmowed at the end of the execution of a program. We limited the number of instructions allowed in a program to a small multiple of the number of tiles available in the lawn (400 in these experiments).

For the experiments with tree-based GP we adapted Fraser and Weinbrenner's GPC++ so as to allow the use of elites of any size. We considered two classical problems: the `Ant` problem and the `Even-5 Parity` problem. In both cases we used the standard primitives as described in [1], using populations of size 1,000. Runs lasted 300 generations for `Ant` and 500 generations for `Even-5 Parity`. We used fitness proportional selection and Koza's subtree crossover applied with 100% probability. In all conditions we performed 100 independent runs.

3. RESULTS

We ran all the configurations outlined in the previous section using six different elite fractions: 0%, 1%, 5%, 10%, 30%, and 50% of the population.

Figure 1 shows the results we obtained with our linear GP system on the Poly-6 symbolic regression problem for the different elite percentages (all plots are averages over 100 independent runs). The main thing to note here is that, while all systems behaved rather similarly in the early generations, elitism seems to induce a reduction in the rate of bloat in the later generations, with the runs using the bigger elites bloating more slowly than those with smaller elites and than "no elite" case. By generation 100, the runs with $\leq 10\%$

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

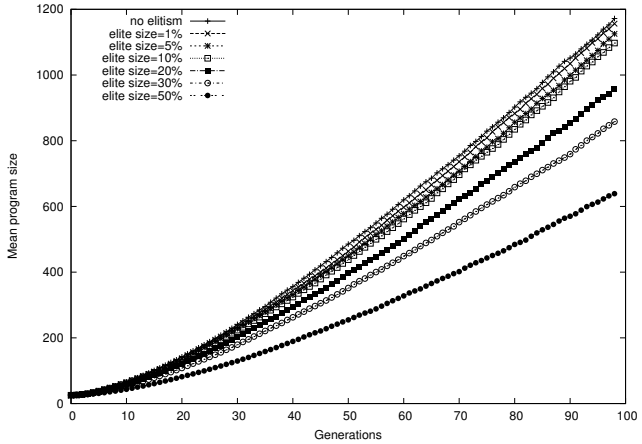

Figure 1: Dynamics of mean program size in a linear GP system solving the `Poly-6` problem for different elite fractions.

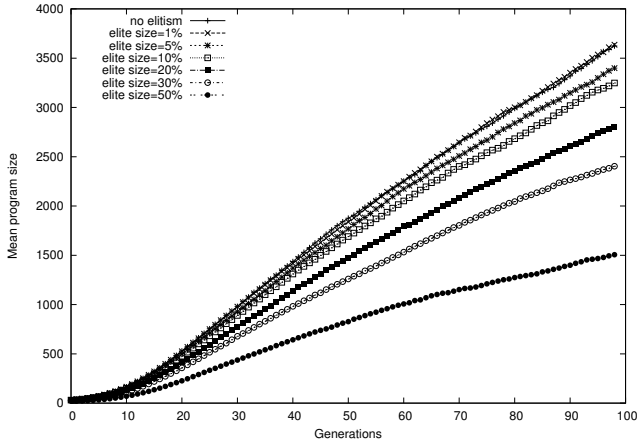

Figure 2: As in Figure 1 but for the `lawnmower` problem.

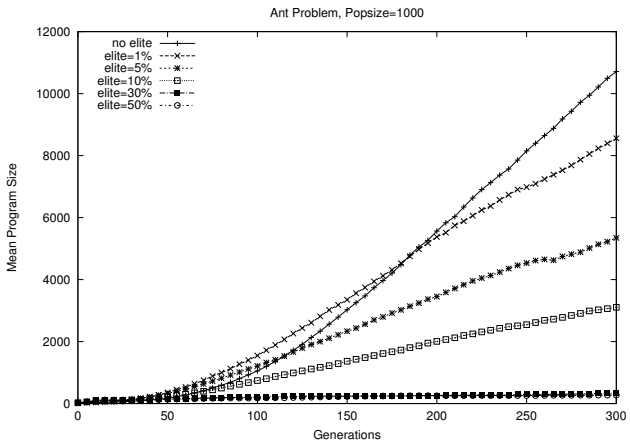

Figure 3: Dynamics of mean program size in a tree-based GP system solving the `Ant` problem for different elite fractions.

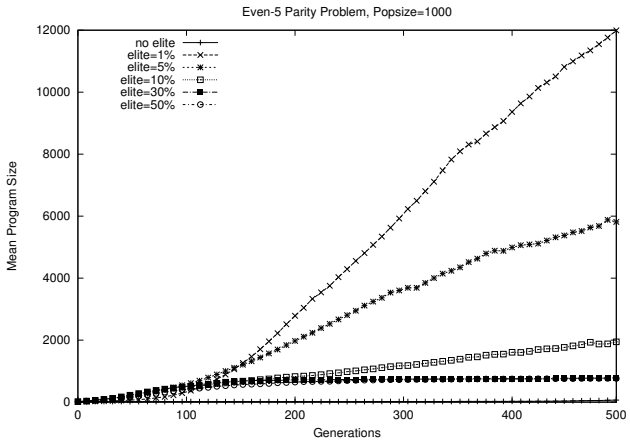

Figure 4: As in Figure 3 but for the `Even-5 Parity` problem.

had average sizes that were nearly twice as large as those using 50% elitism, for example. In the early generations we don't see faster growth in the presence of elitism because in symbolic regression problems of the kind considered here, fitness values across the population have very high variance in the early generations. So, fitness proportional selection is more aggressive than truncation selection. Essentially the same can be seen for the `Lawn-Mower` problem, as illustrated in Figure 2, where initially there is a strong correlation between length and fitness.

For tree-based GP, both for the `Ant` problem (Figure 3) and for the `Even-5 Parity` problem (Figure 4) we see that, when the population settles into a stagnating phase, the rate of bloat is very markedly reduced by increasing the elitism in the system, with elite fractions of 30% and 50% essentially behaving almost identically.

Only one case *appears* to deviate from what we expected. This is represented by the runs of the `Even-5 Parity` problem where no elitism was used. These runs (see Figure 4) appear not to bloat at all, contrary to our expectation that the no-elitism case would be, in general, the fastest growing. The reason for this is very simple. In this problem, with a population of 1,000 individuals, most (if not all) programs in the first generation satisfy 16 out of the 32 fitness cases (e.g., see [3]). When an improvement is found, it typically is a program that satisfies one extra case. With fitness proportionate selection this produces a very small increase in the selection probability for such a program. Since we use 100% crossover, without elitism this improved program is very likely to be destroyed at the next generation. Its offspring (which statistically might be slightly longer then average) are likely to have the same fitness as the rest of the population. So, they are not likely to be reselected. In short, there is very little correlation between size and fitness, so many generations are needed before significant progress can be made on the problem and bloat can really set in. Eventually it does so; notice the small rise in the very bottom right corner of the figure, which corresponds to an average size of 67.1 nodes. The average size in the initial population was 15.3. We conjecture that if given many more generations eventually populations without elitism would overtake those with elitism.

In conclusion, our results make it clear that elitism can have a powerful effect on bloat, generally reducing the level of bloat in the later generations, with larger elite sizes typically controlling bloat more strongly.

4. REFERENCES

[1] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[2] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[3] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, Berlin, Heidelberg, New York, Berlin, 2002.

[4] A. Piszcz and T. Soule. Dynamics of evolutionary robustness. In M. Keijzer *et al.*, editor, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 871–878, Seattle, Washington, USA, 8-12 July 2006. ACM Press.

[5] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).