

Covariant Tarpeian Method for Bloat Control in Genetic Programming

Riccardo Poli
Department of Computer Science and Electronic Engineering
University of Essex, UK
rpoli@essex.ac.uk

March 28, 2010

Abstract

In this paper a simple modification of the Tarpeian bloat-control method is presented which allows one dynamically set the parameters of the method in such a way to guarantee that the mean program size will either keep a particular value (e.g., its initial value) or will follow a schedule chosen by the user. The mathematical derivation of the technique as well as its numerical and empirical corroboration are presented.

1 Background

Many techniques to control bloat have been proposed in the last two decades (for recent reviews see (Poli et al., 2008; Luke and Panait, 2006; Alfaro-Cid et al., 2010; Silva, 2008)). One of the very few with a theoretically-sound basis is the Tarpeian method introduced in (Poli, 2003). This is the focus of this paper.

The Tarpeian method is extremely simple in its implementation. All that is needed is a wrapper for the fitness function like the following algorithm:

Tarpeian Wrapper:

```
if size(program) > average_program_size and random() < p_t then
    return( f_low );
else
    return( fitness(program) );
```

where p_t is a real number between 0 and 1, `random()` is a function which returns uniformly distributed random numbers in the range $[0, 1)$ and f_{low} is a constant which represents an extremely low (or high, if minimising) fitness value such that individuals with such fitness are almost guaranteed not to be selected.

A feature of this algorithm is that it does not require *a priori* knowledge of the size of the potential solutions to a problem. If programs need to grow in order to improve fitness, the Tarpeian method will not prevent this. It will occasionally hit some individuals that, *if evaluated*, would result in being fitter than average and this may slow down a little the progress of a run. However, because the wrapper *does not* evaluate the individuals being given a low fitness, very little computation is wasted. Even at a high anti-bloat intensity, p_t , a better-than-average longer-than-average individual has still a chance of making it into the population. If enough individuals of this kind are produced (w.r.t. the individuals which are better-than-average but also shorter-than-average), eventually the average size of the programs in the population may grow. However, when this happens the Tarpeian method will immediately adjust so as to discourage further growth.

After its proposal, the Tarpeian method has started being used in a variety of studies and applications. For example, Mahler et al. (2005) studied its performance and generalisation capabilities, while it was compared with other bloat-control techniques in (Luke and Panait, 2006; Wyns and Boullart, 2009; Alfaro-Cid et al., 2010). The method has been used with success in the evolution of bin packing heuristics (Burke et al., 2007; Allen et al., 2009), in the evolution of image analysis operators (Roberts and Claridge, 2004), in artificial financial markets based on GP (Martinez-Jaramillo and Tsang, 2009), in predicting protein networks (Geffner et al., 2008), in the design of passive analog filters using GP (Chouza et al., 2009), in the prediction of protein-protein functional associations (Garcia et al., 2008) and in the simplification of decision trees via GP (Garcia-Almanza and Tsang, 2006).

In all cases the Tarpeian method has resulted to be a solid and efficient choice. All studies and applications, however, have had to determine by trial and error the value of the parameter p_t best suited to their problem(s).¹ This is not really a drawback of this method: virtually all anti-bloat techniques require setting one or more parameters. For example, also the parsimony pressure method (Koza, 1992; Zhang and Mühlenbein, 1995, 1993; Zhang et al., 1997) requires setting one parameter (the parsimony coefficient).

In recent research (Poli and McPhee, 2008), we developed a method, called *covariant parsimony pressure*, that allows one to dynamically and optimally set the parsimony coefficient for the parsimony pressure method in such a way to completely control the evolution of the mean program size. The aim of this paper is to achieve the same level of control for the Tarpeian method. We will do this partly by following the tracks of (Poli and McPhee, 2008). We therefore start our journey by briefly summarising the main ideas of that led to the covariant parsimony pressure method.

2 Covariant Parsimony Pressure

Let us start by considering the size evolution equation developed in (Poli, 2003; Poli and McPhee, 2003), which, as shown in (Poli and McPhee, 2008), with trivial manipulations can be rewritten as follows

$$E[\mu'] = \sum_{\ell} \ell p(\ell) \quad (1)$$

where the index ℓ ranges over all program sizes, μ' is a stochastic variable which represents the average size of the programs at the next generation and $p(\ell)$ is the probability of selecting a program of size ℓ from the current generation. The equation applies to GP systems with independent selection and symmetric sub-tree crossover.

If $\phi(\ell)$ represents the proportion of programs of size ℓ in the current generation, then, clearly, the average size of the programs in the current generation is given by $\mu = \sum_{\ell} \ell \phi(\ell)$. Thus one can simply express the expected change in average size of programs between two generations as

$$E[\Delta\mu] = E[\mu'] - \mu = \sum_{\ell} \ell (p(\ell) - \phi(\ell)). \quad (2)$$

In (Poli and McPhee, 2008), we showed that if we restrict our attention to fitness proportionate selection, we can express $p(\ell) = \phi(\ell) \frac{f(\ell)}{\bar{f}}$, where $f(\ell)$ is the average fitness of the programs of size ℓ and \bar{f} is the average fitness of the programs in the population. Then, with some algebraic manipulations, one finds that Equation (2) is actually equivalent to Price's theorem (Price, 1970). That is

$$E[\Delta\mu] = \frac{\text{Cov}(\ell, f)}{\bar{f}}. \quad (3)$$

¹In principle also f_{low} needs to be set. However, this is normally easily done (more on this later) and requires virtually no tuning.

Let us imagine that a parsimonious fitness function, $f_p = f - c\ell$, is used, where c is the parsimony coefficient, ℓ is the size of a program and f is its fitness. Feeding this into Equation (3), then setting its l.h.s. ($E[\Delta\mu]$) to zero and solving for c , one finds

$$c = \frac{\text{Cov}(\ell, f)}{\text{Var}(\ell)}. \quad (4)$$

This value of c guarantees that, in expectation, the size of the programs in the next generation will be the same as in the current generation (as long as the coefficient c is recomputed at each generation).

In (Poli and McPhee, 2008) we also showed that with simple further manipulations of Equation (3) one can even set c dynamically in such a way as to force the mean program size to vary according to any desired function of time, thereby providing complete control over the evolution of size.

3 Covariant Tarpeian Method

Let us now model the effects on program size of the Tarpeian method. In the Tarpeian method the fitness of individuals of size not exceeding the mean size μ is left unaffected. If p_t is the Tarpeian rate, on average individuals of size bigger than the mean will see their fitness set to a very low value, f_{low} , in a proportion p_t of cases, while fitness will be unaffected with probability $1 - p_t$. In order to see what effects the Tarpeian method has on the expected change in program size $E[\Delta\mu]$, we need to verify how the changes in fitness it produces affect the terms in the size evolution equation (Equation (2)). In other words, we need to compute

$$E[\Delta\mu_t] = \sum_{\ell} \ell (p_t(\ell) - \phi(\ell)) \quad (5)$$

or

$$E[\Delta\mu_t] = \frac{\text{Cov}(\ell, f_t)}{\bar{f}_t}. \quad (6)$$

where $\Delta\mu_t = \mu'_t - \mu$, μ'_t is the average program size in the next generation when the Tarpeian method is used, $p_t(\ell)$ is the probability of selecting individuals of size ℓ when the Tarpeian method is used, f_t is the fitness of individuals after the application of the Tarpeian method, and \bar{f}_t is the mean program fitness after the application of the Tarpeian method.

Unfortunately, when attempting to study Equations (5) and (6) for the Tarpeian method things are significantly harder than then for the parsimony pressure method. Under fitness proportionate selection, we have that $p_t(\ell) = \phi(\ell) \frac{f_t(\ell)}{\bar{f}_t}$ where $f_t(\ell)$ is the mean fitness of the programs of size ℓ after the application of the Tarpeian method. In the absence of Tarpeian bloat control (i.e., for $p_t = 0$), these quantities are *constants* (given that we have full information about the current generation). However, as soon as $p_t > 0$, they become *stochastic variables*. This is because the Tarpeian method is stochastic and, so, we cannot be certain as to precisely how many individuals will have their fitness reduced by it, how many individual in each length class will be affected and how many individuals in each fitness class will be affected. If $f_t(\ell)$ and \bar{f}_t are stochastic variables then so are the selection probabilities $p_t(\ell)$ and, consequently, also the quantity $E[\Delta\mu_t]$ on the l.h.s. of Equations (5) and (6)

In other words Equations (5) and (6) give us the expectation of the change in mean program size from one generation to the next *conditionally* to the Tarpeian method modifying the fitness of a particular set of individuals. In formulae,

$$E[\Delta\mu_t | F_t = f_t] = \sum_{\ell} \ell (p_t(\ell) - \phi(\ell)) = \frac{\text{Cov}(\ell, f_t)}{\bar{f}_t}. \quad (7)$$

where F_t is a (vector) stochastic variable which represents the fitness associated to the individuals in the population after the application of the Tarpeian method.

The distribution $\Pr\{F_t = f_t\}$ of F_t depends on the fitness and size of the individuals in the population and the parameter p_t . In principle, we could determine the explicit expression for such a distribution and then compute

$$E[\Delta\mu_t] = \sum_{f_t} E[\Delta\mu_t|F_t = f_t] \Pr\{F_t = f_t\}. \quad (8)$$

However, working out a closed form for this equation is difficult. The reason is that the fitness values f_t appear at the denominator of the selection probabilities $p_t(\ell)$ via the average fitness \bar{f}_t in addition to appearing at the numerators.

To overcome the difficulty and obtain results which allow the application of the theory to the problem of optimally choosing the parameters of the Tarpeian method, we will use the following *approximation*:

$$E[\Delta\mu_t] = E[E[\Delta\mu_t|F_t = f_t]] = E\left[\frac{\text{Cov}(\ell, f_t)}{\bar{f}_t}\right] \cong \frac{E[\text{Cov}(\ell, f_t)]}{E[\bar{f}_t]}. \quad (9)$$

Later in the paper we will get an idea as to the degree of error introduced by the approximation. For now, however, let us see if we can find a closed form for this approximation.

Let us start from computing $E[\bar{f}_t]$:

$$\begin{aligned} E[\bar{f}_t] &= E\left[\sum_{\ell} \phi(\ell) f_t(\ell)\right] \\ &= \sum_{\ell \leq \mu} \phi(\ell) E[f_t(\ell)] + \sum_{\ell > \mu} \phi(\ell) E[f_t(\ell)] \\ &= \sum_{\ell \leq \mu} \phi(\ell) f(\ell) + \sum_{\ell > \mu} \phi(\ell) [p_t \times f_{low} + (1 - p_t) \times f(\ell)] \\ &= \sum_{\ell} \phi(\ell) f(\ell) - \sum_{\ell > \mu} \phi(\ell) f(\ell) + \sum_{\ell > \mu} \phi(\ell) [p_t \times f_{low} + (1 - p_t) \times f(\ell)] \\ &= \bar{f} + \sum_{\ell > \mu} \phi(\ell) [p_t \times f_{low} + (1 - p_t) \times f(\ell) - f(\ell)] \\ &= \bar{f} + \sum_{\ell > \mu} \phi(\ell) [p_t \times f_{low} - p_t \times f(\ell)] \\ &= \bar{f} - p_t \sum_{\ell > \mu} \phi(\ell) (f(\ell) - f_{low}) \\ &= \bar{f} - p_t \phi_{>} \sum_{\ell > \mu} \frac{\phi(\ell)}{\phi_{>}} (f(\ell) - f_{low}) \\ &= \bar{f} - p_t \phi_{>} (\bar{f}_{>} - f_{low}) \end{aligned} \quad (10)$$

where $\phi_{>} = \sum_{\ell > \mu} \phi(\ell)$ is the proportion of above-average-size programs and $\bar{f}_{>}$ is the average fitness of such programs.

Let us now compute the expected covariance between ℓ and f_t :

$$\begin{aligned}
E[\text{Cov}(\ell, f_t)] &= E\left[\sum_{\ell} \phi(\ell)(\ell - \mu)(f_t(\ell) - \bar{f}_t)\right] \\
&= \sum_{\ell} \phi(\ell)(\ell - \mu)E[(f_t(\ell) - \bar{f}_t)] \\
&= \sum_{\ell} \phi(\ell)(\ell - \mu)(E[f_t(\ell)] - E[\bar{f}_t]) \\
&= \sum_{\ell} \phi(\ell)(\ell - \mu)(E[f_t(\ell)] - \bar{f} + p_t\phi_{>}(\bar{f}_{>} - f_{low})) \\
&= \sum_{\ell} \phi(\ell)(\ell - \mu)(E[f_t(\ell)] - \bar{f}) \\
&+ p_t\phi_{>}(\bar{f}_{>} - f_{low}) \underbrace{\sum_{\ell} \phi(\ell)(\ell - \mu)}_{=0} \\
&= \sum_{\ell} \phi(\ell)(\ell - \mu)(E[f_t(\ell)] - \bar{f}) \\
&= \sum_{\ell \leq \mu} \phi(\ell)(\ell - \mu)(E[f_t(\ell)] - \bar{f}) \\
&+ \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)(E[f_t(\ell)] - \bar{f}) \\
&= \sum_{\ell \leq \mu} \phi(\ell)(\ell - \mu)(f(\ell) - \bar{f}) \\
&+ \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)[p_t f_{low} + (1 - p_t)f(\ell) - \bar{f}] \\
&= \sum_{\ell} \phi(\ell)(\ell - \mu)(f(\ell) - \bar{f}) - \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)(f(\ell) - \bar{f}) \\
&+ \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)[p_t f_{low} + (1 - p_t)f(\ell) - \bar{f}] \\
&= \text{Cov}(\ell, f) \\
&+ \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)[p_t f_{low} + (1 - p_t)f(\ell) - \bar{f} - f(\ell) + \bar{f}]
\end{aligned}$$

Thus

$$E[\text{Cov}(\ell, f_t)] = \text{Cov}(\ell, f) - p_t \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)(f(\ell) - f_{low}). \quad (11)$$

If $\mu_>$ is the average size of the programs that are longer than μ , we can write

$$\begin{aligned}
& \sum_{\ell > \mu} \phi(\ell)(\ell - \mu)(f(\ell) - f_{low}) \\
&= \sum_{\ell > \mu} \phi(\ell)(\ell - \mu_> - \mu + \mu_>)(f(\ell) - f_{low}) \\
&= \sum_{\ell > \mu} \phi(\ell)(\ell - \mu_>)(f(\ell) - f_{low}) - (\mu - \mu_>) \sum_{\ell > \mu} \phi(\ell)(f(\ell) - f_{low}) \\
&= \sum_{\ell > \mu} \phi(\ell)(\ell - \mu_>)(f(\ell) - \bar{f}_> - f_{low} + \bar{f}_>) - (\mu - \mu_>) \phi_>(\bar{f}_> - f_{low}) \\
&= \sum_{\ell > \mu} \phi(\ell)(\ell - \mu_>)(f(\ell) - \bar{f}_>) + \sum_{\ell > \mu} \phi(\ell)(\ell - \mu_>)(\bar{f}_> - f_{low}) - (\mu - \mu_>) \phi_>(\bar{f}_> - f_{low}) \\
&= \phi_> \text{Cov}_>(\ell, f) + (\bar{f}_> - f_{low}) \underbrace{\sum_{\ell > \mu} \phi(\ell)(\ell - \mu_>) - (\mu - \mu_>) \phi_>}_{=0} (\bar{f}_> - f_{low}), \\
&= \phi_> [\text{Cov}_>(\ell, f) + (\mu_> - \mu)(\bar{f}_> - f_{low})].
\end{aligned}$$

where $\text{Cov}_>(\ell, f)$ is the covariance between program size and fitness *within* the programs which are of above-average size. Thus, we finally obtain

$$E[\text{Cov}(\ell, f_t)] = \text{Cov}(\ell, f) - p_t \phi_> [\text{Cov}_>(\ell, f) + (\mu_> - \mu)(\bar{f}_> - f_{low})]. \quad (12)$$

Substituting Equations (12) and (10) into Equation (9) we obtain

$$E[\Delta\mu_t] \cong \frac{\text{Cov}(\ell, f) - p_t \phi_> [\text{Cov}_>(\ell, f) + (\mu_> - \mu)(\bar{f}_> - f_{low})]}{f - p_t \phi_>(\bar{f}_> - f_{low})}. \quad (13)$$

With this explicit formulation of the expected size changes, following the same strategy as in the covariant parsimony pressure method (see Section 2), we can find out for what value of p_t we get $E[\Delta\mu_t] = 0$. By setting the l.h.s. of Equation (13) to 0 and solving for p_t , we obtain:

$$p_t \cong \frac{\text{Cov}(\ell, f)}{\phi_> [\text{Cov}_>(\ell, f) + (\mu_> - \mu)(\bar{f}_> - f_{low})]}. \quad (14)$$

This equation allows one to determine how often the Tarpeian method should be applied to modify the fitness of above-average-size programs as a function of a small set of descriptors of the current state of the population and of the parameter f_{low} .

We should note that for some values of f_{low} the method is unable to control bloat. For such values, one would need to set $p_t > 1$ which is clearly impossible (since p_t is a probability). Naturally, we can find out what such values of f_{low} are by setting $p_t = 1$ in Equation (14) and solving for f_{low} obtaining

$$f_{low} \cong \bar{f}_> - \frac{\text{Cov}(\ell, f) - \text{Cov}_>(\ell, f) \phi_>}{\phi_>(\mu_> - \mu)}. \quad (15)$$

However, since we normally don't particularly care about the specific value of f_{low} , as long as the method gets the job done, the obvious and safe choice $f_{low} = 0$ is perhaps the most practical one.

What if we wanted $\mu(t)$ to follow, in expectation, a particular function $\gamma(t)$, e.g., the ramp $\gamma(t) = \mu(0) + b \times t$ or a sinusoidal function? The theory helps us in this case as well.

What we want is that $E[\mu'_t] = \gamma(g)$, where g is the generation number. Note that $E[\mu'_t] = E[\Delta\mu_t] + \mu$. So, adding μ to both sides of Equation (13) we obtain:

$$\gamma(g) \cong \frac{\text{Cov}(\ell, f) - p_t \phi_> [\text{Cov}_>(\ell, f) + (\mu_> - \mu)(\bar{f}_> - f_{low})]}{f - p_t \phi_>(\bar{f}_> - f_{low})} + \mu. \quad (16)$$

		Trials									
<i>Size</i>	<i>f</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>	<i>f_t</i>
5	9	0	0	0	0	0	9	0	9	0	0
2	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2
7	8	0	0	8	0	8	0	0	0	8	0
$E[\Delta\mu]$	1.35	-2.00	-2.00	1.64	-2.00	1.64	0.25	-2.00	0.25	1.64	-2.00
Average $E[\Delta\mu] = -0.46$											

Table 1: Size and fitness of the programs in a small sample population and ten repetitions of the application of the Tarpeian method with optimal p_t to the population.

Solving again for p_t yields:

$$p_t \cong \frac{\text{Cov}(\ell, f) - [\gamma(g) - \mu][\bar{f} - p_t\phi_{>}(\bar{f}_{>} - f_{low})]}{\phi_{>} [\text{Cov}_{>}(\ell, f) + (\mu_{>} - \mu)(\bar{f}_{>} - f_{low})]} \quad (17)$$

Note that, in the absence of sampling noise (i.e., for an infinite population), requiring that $E[\Delta\mu] = 0$ at each generation implies $\mu(g) = \mu(0)$ for all $g > 0$. However, in any finite population the parsimony pressure method can only achieve $\Delta\mu = 0$ *in expectation*, so there can be some random drift in $\mu(g)$ w.r.t. its starting value of $\mu(0)$. If tighter control over the mean program size is desired, one can use Equation (17) with the choice $\gamma(g) = \mu(0)$, which leads to the following formula

$$p_t \cong \frac{\text{Cov}(\ell, f) - [\mu(0) - \mu][\bar{f} - p_t\phi_{>}(\bar{f}_{>} - f_{low})]}{\phi_{>} [\text{Cov}_{>}(\ell, f) + (\mu_{>} - \mu)(\bar{f}_{>} - f_{low})]} \quad (18)$$

Note the similarities and differences between this and Equation (14). In the presence of any drift moving μ away from $\mu(0)$, this equation will actively strengthen the size control pressure to push the mean program size back to its initial value.

4 Example and Numerical Corroboration

As an example, let us consider the small population in the first two columns of Table 1 and let us apply Equation (3) to it. We have that $\text{Cov}(\ell, f) = 6.75$ and $\bar{f} = 5$. So, in the absence of bloat control we will have an expected increase in program size of $E[\Delta\mu] = 1.35$ at the next generation. This is to be expected given the strong correlation between fitness and size in our sample population.

Let us now compute p_t using Equation (14). Since in our population $\mu = 4$, we have that $\phi_{>} = 0.5$, the programs of size 5 and 7 being of above-average size. Their average size is $\mu_{>} = 6$ and their average fitness is $\bar{f}_{>} = 8.5$. Finally, the covariance between their size and their fitness is $\text{Cov}_{>}(\ell, f) = -0.5$. Using these values and the covariance between size and fitness which we computed previously, and taking the safe value $f_{low} = 0$, we obtain $p_t \cong 0.818182$.

Let us now imagine that we adopt this particular value of p_t and let us recompute the fitness of the members of our population based on the application of the Tarpeian method (with $f_{low} = 0$). Since the method is stochastic we will do it multiple times, so as to get an idea of its expected behaviour. The results of these trials are shown in columns 3–12 of Table 1. Computing the expected change in program size after the application of the Tarpeian method shows that in 5 out of 10 cases it is negative, in 2 cases it is marginally positive and only in the remaining cases it is comparable (in fact slightly bigger) than expected when the Tarpeian method is not used. Indeed, on average we expect a slight contraction in the mean program size of -0.46 . In other words, the estimate for p_t has exceeded the value required to achieve a zero expected change in program size. Errors such as this have to be expected given the tiny population we have used.

Population size M	$E[\Delta\mu]$ without Tarpeian	Estimated optimal p_t	Average $E[\Delta\mu_t]$ with Tarpeian	Standard deviation of $E[\Delta\mu_t]$
10	15.00	0.750	-3.050	10.74
100	16.51	0.795	-0.275	3.64
1000	16.80	0.804	0.026	1.16
10000	16.83	0.805	-0.004	0.36
100000	16.83	0.805	-0.003	0.12

Table 2: Errors in $E[\Delta\mu_t]$ resulting from the approximations in the calculation of p_t for different population sizes and for a fitness function where $f(\ell) = \ell$. Statistics were computed over 1,000 independent repetitions of the application of the Tarpeian method to a population including programs from size 1 to M , M being the population size.

To corroborate the theory presented in the previous section and evaluate how population size affects the accuracy of our estimate of p_t , we need to perform much more trials (so as to avoid small sample errors) with a variety of population sizes. For these tests we will create populations with an extremely high correlation between fitness and size.

Our populations include $M = 10, 100, 1000, 10000$, and $100,000$ individuals. In each population individual i has size $\ell_i = \lfloor \frac{i}{M} \times 100 \rfloor$ and fitness $f_i = \ell_i$. These choices would be expected to produce very strong bloat. Indeed, as shown in the second column of Table 2 we expect to see the mean size of programs to increase by between 15 and 16.83 at the next generation.

We now apply the Tarpeian method with the optimal p_t computed via Equation (14) on our test populations 1000 times. The optimal p_t obtained for each population size is shown in the third column of Table 2. Each time different individuals are hit by the reduction of fitness associated with the method. So, different expected changes in program size $E[\Delta\mu_t]$ will be produced. The fourth and fifth columns of Table 2 show the mean and standard deviations of $E[\Delta\mu_t]$ over the 1000 repetitions of the test. As we can see from these values, in all cases bloat is entirely under control, although, for this problem, Equation (14) consistently overestimates p_t thereby leading to slightly shrinking individuals on average. Note how rapidly the mean error becomes very small as the population size grows towards the typical values used in realistic GP runs. The standard deviations also rapidly drop, indicating that the method becomes almost deterministic for very large population sizes. This is confirmed by the distributions of $E[\Delta\mu_t]$ for different population sizes shown in Figure 1.

5 Empirical Tests

To further corroborate the theory, we conducted experiments using a linear register-based GP system. The system we used is a generational GP system. It initialises the population by repeatedly creating random individuals with lengths uniformly distributed between 1 and 200 primitives. The primitives are drawn randomly and uniformly from a problem’s primitive set. The system uses fitness proportionate selection and crossover applied with a rate of 90%. The remaining 10% of the population is created via selection followed by point mutation (with a rate of 1 mutation per program). Crossover creates offspring by selecting two random crossover points, one in each parent, and 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 and 10,000. In each condition we performed 100 independent runs, each lasting either 50 or 100 generations.

With this system we solved a classical symbolic regression problem: the quintic polynomial. In other words, the objective was to evolve a function which fits a polynomial of the form $x + x^2 + \dots + x^d$, where $d = 5$ is the degree of the polynomial, for x in the range $[-1, 1]$. In particular we sampled the polynomials at the 21 equally spaced points $x \in \{-1, -0.9, \dots, 0.9, 1.0\}$. Polynomials of this type have been widely used as benchmark problems in the GP literature.

Fitness (to be maximised) was $1/(1 + \text{error})$ where **error** is the sum of the absolute differences between the target polynomial and the output produced by the program under evaluation over

Table 3: Primitive set used in our experiments.

Instructions
R1 = RIN
R2 = RIN
R1 = R1 + R2
R2 = R1 + R2
R1 = R1 * R2
R2 = R1 * R2
Swap R1 R2

these 21 fitness cases. The primitive set used to solve these problems is shown in Table 3. 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.

Figure 2 shows the results of our runs for populations of size 1000 and 10,000 in the absence of bloat control and when using the version of the Covariant Tarpeian method in Equation (18). Figure 3 shows the results for a population of size 1000 when using the version of the Covariant Tarpeian method in Equation (17) where $\gamma(g)$ is the following triangle wave of period 50 generations:

$$\gamma(g) = 100 \times \left(0.75 + 0.5 \times \left| \frac{g + 12.5}{50} - \left\lfloor \frac{g + 12.5}{50} + 0.5 \right\rfloor \right| \right).$$

It is apparent that in the absence of bloat control there is very substantial bloat, while the Covariant Tarpeian method provides almost total control over the size dynamics.

6 Conclusions

There are almost as many anti-bloat recipes as there are researchers in genetic programming. Very few, however, have a theoretical pedigree. The Tarpeian method (Poli, 2003) is one of them. In recent years, the method has started becoming more and more widespread, probably because of its simplicity. The method, however, like most others, requires setting one main parameter (and one secondary one) for it to perform appropriately. Until now this parameter had to be set by trial and error.

In this paper we integrate the theory that led to the development of the original Tarpeian method with ideas that recently led to the covariant parsimony pressure method (Poli and McPhee, 2008) (another theoretically derived method), to obtain equations which allow one to optimally set the parameter(s) of the method so as to achieve almost full control over the evolution of the mean program size in runs of genetic programming. Although the complexity of the task has forced us to rely on approximations to make progress, numerical and empirical corroboration confirm that the quality of the approximation is good. Experiments have also confirmed the effectiveness of the Covariant Tarpeian method.

References

- Alfaro-Cid, E., Merelo, J. J., de Vega, F. F., Esparcia-Alcázar, A. I., and Sharman, K. (2010). Bloat control operators and diversity in genetic programming: a comparative study. *Evolutionary Computation*, 18(2). To appear, ISSN: 1063-6560.
- Allen, S., Burke, E. K., Hyde, M. R., and Kendall, G. (2009). Evolving reusable 3D packing heuristics with genetic programming. In Raidl, G., Rothlauf, F., Squillero, G., Drechsler, R., Stuetzle, T., Birattari, M., Congdon, C. B., Middendorf, M., Blum, C., Cotta, C., Bosman,

- P., Grahl, J., Knowles, J., Corne, D., Beyer, H.-G., Stanley, K., Miller, J. F., van Hemert, J., Lenaerts, T., Ebner, M., Bacardit, J., O'Neill, M., Di Penta, M., Doerr, B., Jansen, T., Poli, R., and Alba, E., editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 931–938, Montreal. ACM.
- Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2007). Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J. A., Cliff, D., Congdon, C. B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J. F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K. O., Stutzle, T., Watson, R. A., and Wegener, I., editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1559–1565, London. ACM Press.
- Chouza, M., Rancan, C., Clua, O., , and García-Martínez, R. (2009). Passive analog filter design using gp population control strategies. In Chien, B.-C. and Hong, T.-P., editors, *Opportunities and Challenges for Next-Generation Applied Intelligence: Proceedings of the International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA-AIE) 2009*, volume 214 of *Studies in Computational Intelligence*, pages 153–158, Berlin. Springer-Verlag.
- Garcia, B., Aler, R., Ledezma, A., and Sanchis, A. (2008). Protein-protein functional association prediction using genetic programming. In Keijzer, M., Antoniol, G., Congdon, C. B., Deb, K., Doerr, B., Hansen, N., Holmes, J. H., Hornby, G. S., Howard, D., Kennedy, J., Kumar, S., Lobo, F. G., Miller, J. F., Moore, J., Neumann, F., Pelikan, M., Pollack, J., Sastry, K., Stanley, K., Stoica, A., Talbi, E.-G., and Wegener, I., editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 347–348, Atlanta, GA, USA. ACM.
- Garcia-Almanza, A. L. and Tsang, E. P. K. (2006). Simplifying decision trees learned by genetic programming. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 7906–7912, Vancouver. IEEE Press.
- Geffner, H., Prada, R., Alexandre, I. M., and David, N., editors (2008). *Advances in Artificial Intelligence - IBERAMIA 2008, 11th Ibero-American Conference on AI, Lisbon, Portugal, October 14-17, 2008. Proceedings*, volume 5290 of *Lecture Notes in Computer Science*. Springer.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.
- Luke, S. and Panait, L. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344.
- Mahler, S., Robilliard, D., and Fonlupt, C. (2005). Tarpeian bloat control and generalization accuracy. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J. I., and Tomassini, M., editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 203–214, Lausanne, Switzerland. Springer.
- Martinez-Jaramillo, S. and Tsang, E. P. K. (2009). An heterogeneous, endogenous and coevolutionary GP-based financial market. *IEEE Transactions on Evolutionary Computation*, 13(1):33–55.
- Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217, Essex. Springer-Verlag.
- Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>. (With contributions by J. R. Koza).

- Poli, R. and McPhee, N. (2008). Parsimony pressure made easy. In Keijzer, M., Antoniol, G., Congdon, C. B., Deb, K., Doerr, B., Hansen, N., Holmes, J. H., Hornby, G. S., Howard, D., Kennedy, J., Kumar, S., Lobo, F. G., Miller, J. F., Moore, J., Neumann, F., Pelikan, M., Pollack, J., Sastry, K., Stanley, K., Stoica, A., Talbi, E.-G., and Wegener, I., editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1267–1274, Atlanta, GA, USA. ACM.
- Poli, R. and McPhee, N. F. (2003). General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206.
- Price, G. R. (1970). Selection and covariance. *Nature*, 227, August 1:520–521.
- Roberts, M. E. and Claridge, E. (2004). Cooperative coevolution of image feature construction and object detection. In Yao, X., Burke, E., Lozano, J. A., Smith, J., Merelo-Guervós, J. J., Bullinaria, J. A., Rowe, J., Kabán, P. T. A., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 902–911, Birmingham, UK. Springer-Verlag.
- Silva, S. (2008). *Controlling Bloat: Individual and Population Based Approaches in Genetic Programming*. PhD thesis, Coimbra University, Portugal. Full author name is Sara Guilherme Oliveira da Silva.
- Wyns, B. and Boullart, L. (2009). Efficient tree traversal to reduce code growth in tree-based genetic programming. *Journal of Heuristics*, 15(1):77–104.
- Zhang, B.-T. and Mühlenbein, H. (1993). Evolving optimal neural networks using genetic algorithms with Occam’s razor. *Complex Systems*, 7:199–220.
- Zhang, B.-T. and Mühlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38.
- Zhang, B.-T., Ohm, P., and Mühlenbein, H. (1997). Evolutionary induction of sparse neural trees. *Evolutionary Computation*, 5(2):213–236.

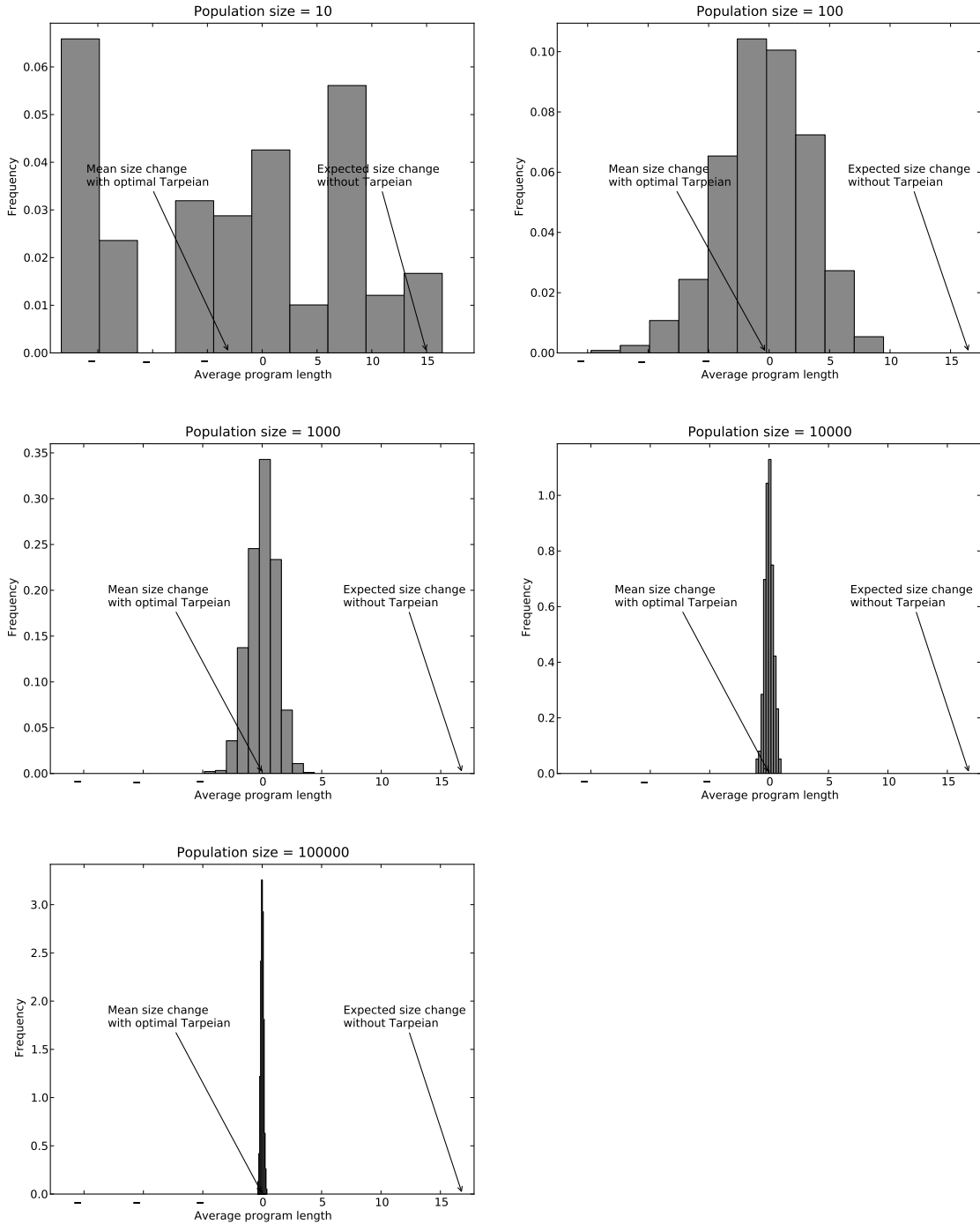
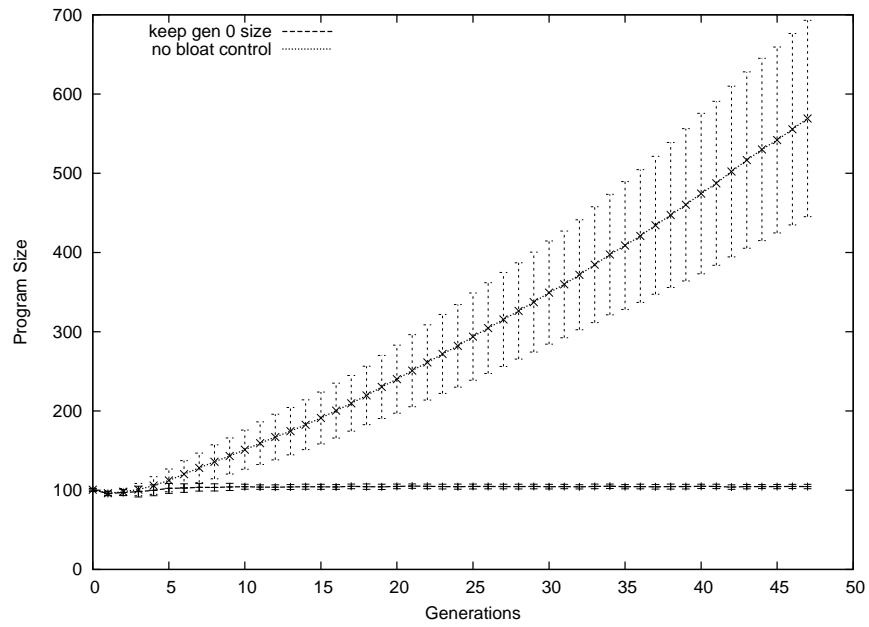
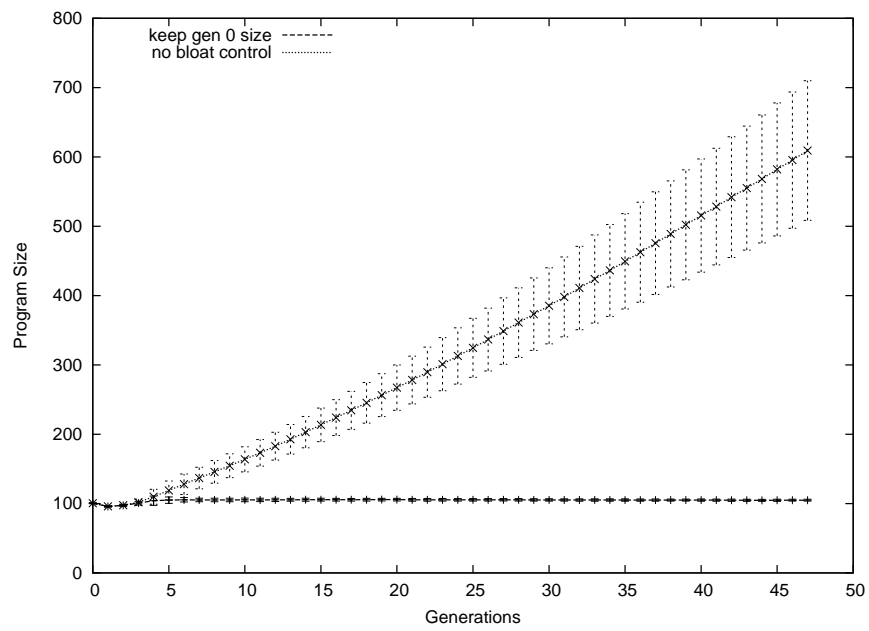


Figure 1: Distributions of $E[\Delta\mu_t]$ resulting from the application of the Covariant Tarpeian method for different population sizes with our sample fitness function.



(a)



(b)

Figure 2: Mean program size for populations of size 1000 (a) and 10,000 (b) as a function of the generation number on the quintic polynomial symbolic regression in the absence of bloat control and when using the version of the Covariant Tarpeian method in Equation (18).

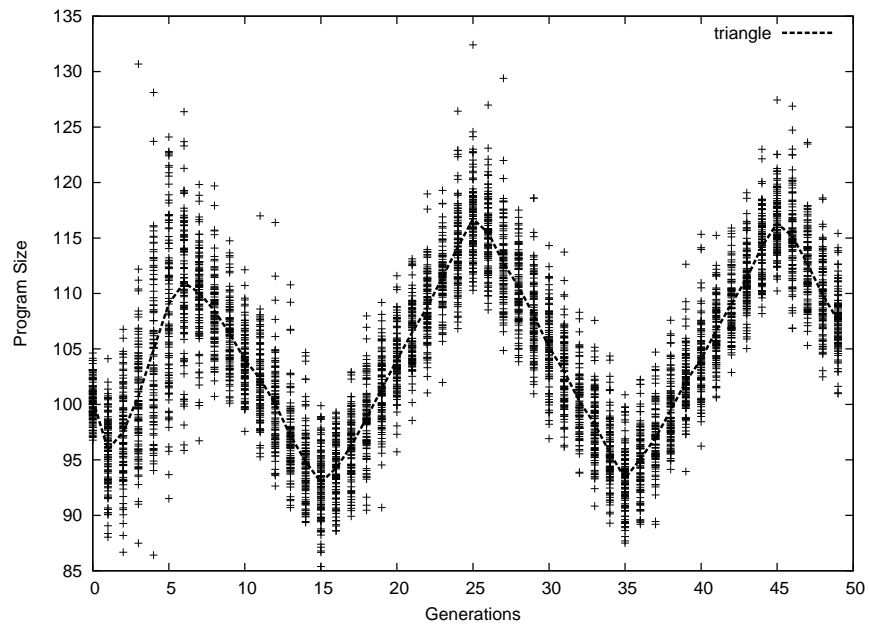


Figure 3: Average program size for populations of size 1000 and runs lasting 100 generations with the quintic polynomial symbolic regression when using the version of the Covariant Tarpeian method in Equation (17) where $\gamma(g)$ is a triangle wave. The dashed line represents the mean average program size across runs.