A schema theory analysis of the evolution of size in genetic programming with linear representations

Nicholas Freitag McPhee[1] and Riccardo Poli[2]

[1] Division of Science and Mathematics, University of Minnesota, Morris; Morris, MN, USA
mcphee@mrs.umn.edu, http://www.mrs.umn.edu/~mcphee
[2] School of Computer Science, The University of Birmingham, Birmingham, B15 2TT, UK
R.Poli@cs.bham.ac.uk, http://www.cs.bham.ac.uk/~rmp/

Abstract. In this paper we use the schema theory presented in [20] to better understand the changes in size distribution when using GP with standard crossover and linear structures. Applications of the theory to problems both with and without fitness suggest that standard crossover induces specific biases in the distributions of sizes, with a strong tendency to over sample small structures, and indicate the existence of strong redistribution effects that may be a major force in the early stages of a GP run. We also present two important theoretical results: An exact theory of bloat, and a general theory of how average size changes on flat landscapes with glitches. The latter implies the surprising result that a single program glitch in an otherwise flat fitness landscape is sufficient to drive the average program size of an infinite population, which may have important implications for the control of code growth.

1 Introduction

The phenomenon of bloat, or code growth, has been observed in genetic programming (GP) from the very beginning [5], and over the years the study of bloat has probably been one of the most active areas of foundational GP research [3, 13, 15, 24, 21, 11, 6, 8, 7]. Numerous theories have been proposed to explain bloat, but these have typically been qualitative rather than quantitative in nature. It is likely that none of these proposed mechanisms is the complete story, and that they in fact interact in complex ways that vary over the course of a GP run. Their qualitative nature, however, makes it difficult to perform the kinds of quantitative study that would allow one to begin to tease out these complex relationships. In this paper we show how recent results in GP schema theory can be used to take what may be the first steps towards providing a unified theoretical framework for understanding code growth in GP.

In recent work [20, in this proceedings] we present an exact schema theory for Genetic Programming (GP) using standard crossover on linear representations. We also show how that theory can be used to understand the significant biases standard crossover introduces on the distribution and sampling of programs of different lengths even when the fitness landscape is flat. In this paper we apply that schema theory to the problem of better understanding changes in the distribution of lengths under standard crossover. We find that in problems both with and without fitness, standard crossover induces very specific biases in the distribution of sizes, with a strong tendency to over sample the smaller structures. There also appear to be significant redistribution effects that may

be a major force in the early stages of a GP run. We then generalize these results, yielding two important theoretical results. The first is an exact theory of bloat which gives a formula for the change in average program size for infinite populations of linear structures when using standard crossover without mutation. The second is a general theory of how average sizes change on a flat landscape with "glitches" (sets of strings with above or below average fitness), which implies the surprising result that a single program glitch is sufficient to drive the average program size of an infinite population.

While the work reported here is all on GP with linear structures, the schema theorem used is a special case of a more general GP schema theorem [17, in this proceedings]. We have chosen in these early applications to focus on linear structures because the theoretical analysis is more manageable and the computations are more tractable. This has yielded a number of important results for the linear case, and preliminary results further suggest that many of the key ideas here are also applicable (at least in broad terms) to the non-linear tree structures typically used in GP.

In the remainder of this section we will briefly survey three key theories of bloat, and discuss why theoretical tools are crucial in our attempts to understand foundational phenomena like bloat. In Sec. 2 we will present the schema theorem for GP using linear structures and standard crossover. In Sec. 3 we will summarize *theoretical* results from [20] showing that bloat does not happen on flat fitness landscapes when using standard GP crossover (for related empirical results see, e.g., [9]), and in fact standard crossover heavily over samples the shorter programs; we also present the exact schema theorem there. We then apply the theory in Sec. 4 to a problem that does bloat and use the theory to both predict and better understand the changes in the distribution of sizes. In Sec. 5 we show how the schema theory can be used to derive some general results relating variations in the fitness landscape to changes in the average program size, and then finish with some conclusions and ideas for future research (Sec. 6).

1.1 Three theories of bloat

As mentioned earlier, bloat in GP is an old problem that has received quite a lot of attention over the years. Many techniques have been proposed to combat bloat but, as noted in [11], these are mostly *ad hoc*, preceding rather than following from knowledge of the causes of bloat. To date there are three major theories that propose to at least partially explain bloat:

1. Replication accuracy, or protection against bad crossover
2. Removal bias
3. Nature of program search spaces

It is not generally claimed that any one of these is sufficient, or even that this set is complete, but each of these does appear to explain some interesting facet of the problem, and in combination they provide a valuable set of tools for understanding code growth. They are, however, primarily qualitative rather than quantitative in nature, which ultimately limits their predictive power.

The replication accuracy theory [13, 3, 15] is based on the idea (common in evolutionary biology) that an important component of the success of an organism (in our case a GP individual) is its ability to reproduce accurately, i.e., have offspring that are functionally similar to the parent (see, e.g., [4, 1]). This would suggest that if an evolutionary

computation system could evolve towards representations (including specific sizes and shapes) that increased replication accuracy then, all other things being equal, it would do so. One way this can happen in GP is through the evolution of large blocks of code that do not contribute to the semantics of the individual. The larger these blocks, the greater the chance that crossover points will be chosen there, ensuring that the offspring are semantically equivalent to the root parent.

One drawback of the replication accuracy hypothesis is that it is presented solely in terms of individuals avoiding disruption, and does not consider creation effects. Creation effects, however, are crucial, because without them one cannot have the evolution of new structures. Replication accuracy is typically assumed to be most important later in the run, when there's little or no change in the best fitness and creation effects may indeed be minimal. It tells us little, however, about the changes in program size we observe in the early stages of runs. The replication accuracy hypothesis also provides no way of judging the relative importance of potentially competing forces. If, for example, there is a fitness induced bias favoring smaller trees, and replication accuracy favors larger trees, how will these biases interact? In many cases these interactions may change over the course of a run, and this hypothesis currently provides little help in understanding these phase transitions.

The removal bias theory [22, 11] is based on the observation that the semantically irrelevant nodes of a GP tree tend to be in the lower parts of the tree and are therefore the root of subtrees that are smaller than the average subtree for that individual. If it is indeed the case that successful crossovers will tend to replace semantically irrelevant nodes, then the removed subtree will tend to be a smaller than average subtree. There is no size bias with respect to the inserted subtree, however, since that subtree will have no semantic impact. This means that fitness neutral crossover events will tend to replace smaller subtrees with larger subtrees, which will lead to bloat.

The removal bias hypothesis is closely related to the replication accuracy hypothesis, and not surprisingly it suffers essentially the same drawbacks: Failure to consider creation effects, no way of predicting the interactions of competing forces, and a primary applicability to the later stages of a run. It further hinges, though, on important assumptions about the structure of program trees, such as the assumption that the semantically important nodes tend to be clustered near the root of the tree. While there is evidence that this is true in many cases, it is unclear how well this generalizes, e.g., to problems with side-effecting primitives.

The nature of program search spaces theory [11] is based on the experimental observation that, for a variety of test problems (e.g., [10, 6]), above a certain problem dependent size, the distribution of fitnesses does not vary a great deal with the size of the programs. Since there are more (syntactically) different long programs, the number of long programs of a given fitness is greater than the number of short programs of the same fitness. Thus, all other things being equal, over time one is more likely to sample the long programs simply because there are more of them.

This theory has the advantage of being quite general. Since it talks about the search space rather than the search mechanism, it potentially applies to a whole variety of variable length search techniques (see [11] for examples). It relies, however, on two crucial assumptions. First it requires that the distribution of fitnesses is roughly independent of length. As mentioned above, this has been observed experimentally on a variety of problems, and [6] provides a theoretical analysis which shows that certain general classes

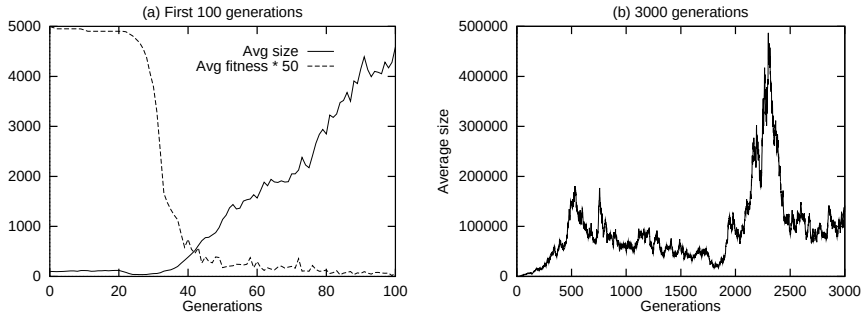

Fig. 1. Graphs of the average size over time of a representative run of the INC-IGNORE problem. Graph (a) (based on [13]) also includes the average fitness (multiplied by 50 so the ranges coincide). The population size is 200.

of problems will always have this property. It is, however, easy to construct (artificial) problems where this assumption does not hold, yet bloat still occurs (e.g., the one-then-zeros problem in Sec. 4). Second, this hypothesis assumes that the search operators sample the space in a manner that is neutral with regard to size. Recent schema theory results on flat fitness landscapes and linear structures [20] raise serious doubts about the veracity of that assumption when one is using standard crossover. While there are GP crossover operators that are neutral with regard to size (e.g., one-point GP crossover [18]), standard crossover clearly is not, being instead heavily biased (on flat fitness landscapes over linear structures) towards an over sampling of the smaller structures.

1.2 Why theory matters

Before proceeding to the presentation of the schema theory and its applications to bloat, it is worth looking at an example of why theoretical tools are so important.

Six years ago McPhee and Miller [13] examined bloat in the INC-IGNORE problem, an artificial problem designed specifically to help isolate the causes of bloat. In this problem there is a single terminal, 0, and two unary functions, INC and IGNORE, defined by $\mathrm{INC}(x) = x + 1$ and $\mathrm{IGNORE}(x) = 0$. Thus INC adds one to its argument, and IGNORE returns a constant 0 regardless of its argument. The goal is to generate a string having value 100; the fitness is the absolute value of the difference between the string's value and the target of 100, with low values being better. Thus a correct individual would be a string of 100 INCs followed by either the terminal 0, or an IGNORE which could then be followed by any number of nodes of any type (since their value would be ignored).

We re-implemented this problem and did new runs, getting very similar behavior to that reported in [13].[1] Fig. 1(a) shows the changes in the average fitness and the average size over time in a representative run, and one can see a clear relationship between the abrupt discovery of a solution and the onset of bloat.

[1] To get the fairly flat fitness one sees in the early generations in Fig. 1(a) it is necessary to bias the initial population so there are many more IGNORE nodes than INC nodes; a ratio of $20:1$ was used both here and in [13].

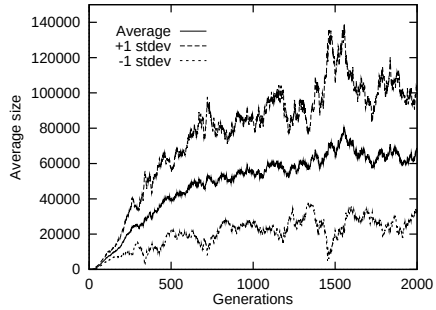

Fig. 2. Graph of the average size over time in the INC-IGNORE problem averaged over 20 runs. Also plotted are the average ± one standard deviation. The population size is 200.

Due to practical limitations, the runs reported in [13] were only taken out 100 generations. What effect did this limitation have? Fig. 1(b) shows the development of the average size in a new run taken out to 3000 generations. In this extended run bloat appears to continue steadily for several hundred generations, but with an increasing amount of noise. After that initial steady rise, though, the behavior becomes very erratic. By showing more generations, Fig. 1(b) shows a much more complex picture than that in Fig. 1(a). This raises questions about whether or how long the code growth would continue if we ran more generations. The overall trends are a little easier to read in Fig. 2, where we show the average of 20 runs of INC-IGNORE out to 2000 generations. While there is clearly a change in the rate of growth after several hundred generations, the average size past that point is still remarkably noisy despite the averaging, and it's unclear whether, for example, the long term behavior will be asymptotic.

The point here is not to criticize the methodology in [13]; all empirical papers are subject to practical limits, and any such paper might yield new ideas and information if one were able to extend the experiments. The point is that any empirical study is unavoidably limited, and can perforce only provide a small window on the large and complex picture of GP behavior. How do we get around those limitations? In this paper we propose to use schema theory.

2 Schema theory for GP on linear structures

In recent work the GA schema theory in [23] has been extended to GP [16, 17]. Unlike most previous schema theory work, which provides lower bounds on the number of instances of a schema, this new work provides exact formulas for the transmission of schemata from one generation to the next.

GP is a highly complex process and, as might be expected, the schema theory to describe its behavior is also relatively complex. To make matters more tractable in this paper, we will only consider applications of the theory to a specific simplified domain, namely that of linear structures; thus we will restrict our attention to problems with only unary operators. See [17] for a more general treatment.

2.1 Schema theory definitions

In this section we will present a series of crucial definitions that allow us to represent schemata, and count and build instances of schemata.

In a linear-structure GP where $\mathcal{F}$ is the set of non-terminal nodes and $\mathcal{T}$ is the set of terminal nodes, individuals can be seen as sequences of symbols $c_0 c_1 \ldots c_{N-1}$ where $c_i \in \mathcal{F}$ for $i < N-1$ and $c_{N-1} \in \mathcal{T}$. We will then define a linear GP *schema* as the same kind of sequence $c_0 c_1 \ldots c_{N-1}$ except that a new "don't care" symbol '$=$' is added to both $\mathcal{F}$ and $\mathcal{T}$.[2] Thus schemata represent sets of linear structures, where the positions labelled '$=$' can be filled in by any element of $\mathcal{F}$ (or $\mathcal{T}$ if it is the terminal position). A few examples of schema are:[3]

- $(=)^N$: The set of all sequences of length N.
- $1(=)^a$: The set of all sequences of length $a+1$ starting with a 1.
- $1(0)^a$: The singleton set containing the string 1 followed by a 0's.

Now that we can represent schemata, we present a series of definitions that allow us to count instances of schemata.

Definition 1 (Proportion in population). *$\phi(H,t)$ is the proportion of strings in the population at time t matching schema H. For finite populations of size M, $\phi(H,t) = m(H,t)/M$, where $m(H,t)$ is the number of instances of H at time t.*

Definition 2 (Selection probability). *$p(H,t)$ is the probability of selecting an instance of schema H from the population at time t. This is typically a function of $\phi(H,t)$, the fitness distribution in the population, and the details of the selection operators. With fitness proportionate selection, for example, $p(H,t) = \phi(H,t) \times f(H,t)/\overline{f}(t)$, where $\underline{f}(H,t)$ is the average fitness of all the instances of H in the population at time t and $\overline{f}(t)$ is the average fitness in the population at time t.*

Definition 3 (Transmission probability). *$\alpha(H,t)$ is the probability that the schema H will be constructed in the process of creating the population for time $t+1$ out of the population at time t. This will typically be a function of $p(K,t)$, for the various schemata K that could play a role in constructing H, and of the details of the various recombination and mutation operators being used.*

Given these definitions, we can model the standard evolutionary algorithm as the repeated application of the cycle

$$\phi(H,t) \stackrel{\text{selection}}{\longrightarrow} p(H,t) \stackrel{\substack{\text{mutation}\\ \text{crossover}}}{\longrightarrow} \alpha(H,t) \ .$$

For an *infinite* population $\phi(H,t+1) = \alpha(H,t)$ for $t \geq 0$, which means we can iterate these equations to *exactly* model the behavior of an infinite population over time.

[2] This new '$=$' symbol plays a role similar to that of the '#' "don't care" symbol in GA schema theory. For historical reasons, however, '#' has been assigned another meaning in the more general version of the GP schema theory [17].

[3] We will use the superscript notation from theory of computation, where x^n indicates a sequence of n x's.

To formalize the creation of instances of a linear schema we define

$$u(H,i,k) = c_0 c_1 \ldots c_{i-1}(=)^{k-i}$$
$$l(H,i,n) = (=)^{n-N+i} c_i c_{i+1} \ldots c_{N-1}$$

Here $u(H,i,k)$ is the schema of length k matching the leftmost i symbols of H, and $l(H,i,n)$ is the schema of length n matching the rightmost $N-i$ symbols of H.[4] The important thing about u and l is that if you use standard crossover to crossover *any* instance of $u(H,i,k)$ at position i with *any* instance of $l(H,i,n)$ at position $n-N+i$, the result will be an instance of H, provided[5] $k+n > N$, and $0 \uparrow (N-n) \leq i < N \downarrow k$. Further, these are the *only* ways to use standard crossover to construct instances of H, so these definitions fully characterize the mechanism for constructing instances of H.

2.2 The schema theorem

Given these definitions, we now present the exact schema theorem for linear structures from [20] in a more compact form:

Theorem 1 (Schema theorem for linear structures and standard crossover). *For GP on linear structures using standard crossover with probability p_{XO} and no mutation we have*

$$\alpha(H,t) = (1 - p_{XO}) \times p(H,t) + p_{XO} \times \alpha_{XO}(H,t)$$

where

$$\alpha_{XO}(H,t) = \sum_{\substack{k>0 \\ n>0 \\ k+n>N}} \left(\frac{1}{k \times n} \times \sum_{0\uparrow(N-n) \leq i < N\downarrow k} p(u(H,i,k),t) \times p(l(H,i,n),t) \right) .$$

To simplify the calculations in the remainder of the paper we will assume $p_{\text{XO}} = 1$ throughout.

2.3 An exact theory of bloat

One way to calculate the average length of the population at time t, which we shall write as $\mu(t)$, is

$$\mu(t) = \sum_N (N \times \phi((=)^N, t)) . \tag{1}$$

For an infinite population (where $\phi(H, t+1) = \alpha(H,t)$), we can rewrite Eq. (1) as $\mu(t+1) = \sum_N (N \times \alpha((=)^N, t))$. A surprising result from [20], though, shows that you can replace α by p to get the average size for the next generation:

[4] u and l are based on operators U and L (see, e.g., [17]) which match the *upper* and *lower* parts of general, non-linear, GP schemata.

[5] We will use $\uparrow$ as a binary infix *max* operator, and $\downarrow$ as a binary infix *min* operator.

Theorem 2. *For GP using infinite populations, linear structures, standard crossover, and no mutation we have*

$$\mu(t+1) = \sum_N (N \times p((=)^N, t)) \ . \tag{2}$$

Implicit it these results, but clarified here for the first time, is:

Theorem 3 (Exact theory of length change for infinite populations). *For GP using infinite populations, linear structures, standard crossover, and no mutation we have*

$$\mu(t+1) - \mu(t) = \sum_N (N \times \alpha((=)^N, t)) - \sum_N (N \times \phi((=)^N, t)) \tag{3}$$

or equivalently

$$\mu(t+1) - \mu(t) = \sum_N (N \times p((=)^N, t)) - \sum_N (N \times \phi((=)^N, t)) \ . \tag{4}$$

For infinite populations Theorem 3 gives an exact quantitative theory of the changes in the average size of a population over time.[6] Previous theories of bloat are then in some sense approximations of this result, usually based on some simplifying assumptions (e.g., that a run has "plateaued"). This result also makes it clear that it would be rather surprising if there wasn't some sort of change in the average size, since this would require an unlikely balance between the selection probability and the length distribution.

3 Flat fitness landscapes

In [20, in this proceedings] we applied the Schema Theorem from the previous section to the case of a flat fitness landscape; due to space limitations we won't repeat those results here. A key result in this context, however, was that standard crossover on a flat landscape heavily over samples the shorter programs. This sampling bias in favor of shorter programs raises serious questions about the "all other things being equal" assumption in the "Nature of program search spaces" theory of bloat (Sec. 1.1). That theory depends crucially on at least an approximately uniform sampling of the search space by the operators, and here, at least, we clearly do *not* have uniform sampling by standard crossover. So while that theory may well be valuable in helping explain behaviors in other contexts, it is not clear how it can be applied in this setting.

4 The one-then-zeros problem

We will now apply the Schema Theorem in a setting where there is bloat, namely the *one-then-zeros problem*. We will start by defining and motivating the problem, and we will then use the Schema Theorem to derive length distribution results similar to those reported in [20] for the flat fitness landscape case. In contrast to the flat fitness case, in the one-then-zeros problem we do observe bloat, as well as some important length redistribution effects in the early generations.

[6] We can, in fact, extend this to the finite population case, but to do so we need to replace the exact value on the left hand side with an expectation, namely, $E[\mu(t+1) - \mu(t)]$.

4.1 One-then-zeros problem definition

In this problem we have $\mathcal{F} = \{0+, 1+\}$ and $\mathcal{T} = \{0\}$. Both $0+$ and $1+$ are unary operators that add 0 and 1 respectively to their argument. This gives us a problem that is essentially equivalent to studying variable length strings of 0's and 1's, with the constraint that the strings always end in a 0. Note that $1+$ is INC, and $0+$ can be seen as DON'T-INC, so this problem is structurally quite similar to the earlier INC-IGNORE problem. Fitness in this problem will be 1 if the string starts with a 1 and has zeros elsewhere, i.e., the string has the form $1(0)^a$ where $a > 0$; fitness will be 0 otherwise.

Both the replication accuracy and the removal bias hypotheses (Sec. 1.1) suggest that this problem might exhibit bloat. For replication accuracy, increasingly long tails of 0's could act as a mechanism for increasing the probability of constructing "correct" offspring (avoiding bad crossover). Similarly there is a slight removal bias since the one "bad" crossover involves removing the entire root parent and replacing it with a substring consisting of all 0's, which would on average be shorter than the removed string. Note, however, that the nature of program search spaces hypothesis does not appear to apply here, as the proportion of fit individuals is not constant with respect to length. There is in fact just a single "correct" string for any given length, so the density of solutions drops dramatically as length increases. In this case the program search space hypothesis would not appear to suggest bloat, and might instead be seen as suggesting the opposite, since the odds of finding a correct short individual would seem to be much higher than the odds of finding a correct long individual.

4.2 Calculating $\alpha_{\text{xo}}((=)^N, t)$ and $\alpha_{\text{xo}}(1(0)^a, t)$

We will start[7] by deriving schema equations for $\alpha((=)^N, t)$ so we can study the evolution of length.

Theorem 4 (Length distribution for one-then-zeros). *Using the Schema Theorem (Theorem 1) we can show that for GP on the one-then-zeros problem using standard crossover and no mutation, the distribution of lengths is characterized (for $N > 0$) by*

$$\alpha_{xo}((=)^N, t) = \sum_{\substack{k>0 \\ n>0 \\ k+n>N}} \left(\frac{N \downarrow k-0 \uparrow (N-n)}{k \times n} \times p(1(0)^{k-1}, t) \times p(1(0)^{n-1}, t) \right) .$$

In the flat fitness case, the formula for $\alpha((=)^N, t)$ depended only on probabilities of the form $p((=)^a, t)$, so if we wanted to iterate the equation we only needed to keep track of $\phi((=)^a, t)$ for all the legal a at each generation. In this case, though, $\alpha((=)^N, t)$ depends on probabilities of the form $p(1(0)^a, t)$, so we also need to know $\phi(1(0)^a, t)$ at each generation. Thus we need to use the Schema Theorem a second time to compute $\alpha(1(0)^a, t)$ for $a > 0$.

[7] The derivation of the results in this section are more lengthy than illuminating and will be omitted as a result. See [12] for the details.

Theorem 5 ($1(0)^a$ distribution for one-then-zeros). *Using the Schema Theorem (Theorem 1) we can also show that for $a > 0$*

$$\alpha_{XO}(1(0)^a, t) = \sum_{k>0} \Big[\frac{p(1(0)^{k-1}, t) \times p(1(0)^a, t)}{k \times (a+1)} + \sum_{\substack{n>0 \\ n+k>a+1}} \frac{(a+1) \downarrow k-1 \uparrow (a+2-n)}{k \times n} \times p(1(0)^{k-1}, t) \times p(1(0)^{n-1}, t) \Big] .$$

Fortunately we find here that $\alpha(1(0)^a, t)$ only depends on probabilities of the form $p(1(0)^d, t)$, so no further calculations are needed. Unfortunately most problems are not so restricted, and it is possible for this process to quickly balloon until one is forced to track the proportion of every different string. This is essentially intractable, though, since the number of different strings grows doubly exponentially with the generation t. As a result it is, at this point in the development of the theory, still a rare problem where one can do this level of exact analysis.

4.3 One-then-zeros results

We can numerically iterate the equations in Theorems 4 and 5 to better understand the behavior of an infinite GP population on this problem. Note, however, that tracking these distributions over time becomes expensive in terms of computational effort. In this case, for example, generating both $\alpha((=)^N, t)$ and $\alpha(1(0)^a, t)$ are $O(2^{3t})$ where t is the generation.[8] A crucial point, though, is that these equations only need to be run once, and have no stochastic effects. They are *exact* calculations of the relevant quantities (up to the limitations of the floating point representation), and once computed need never be computed again. This is in contrast to typical empirical results in evolutionary computation, where combinations of large populations and multiple runs are necessary to smooth out the stochastic effects, and even then there is no guarantee that any two sets of runs will have similar behavior.

Here we calculated $\alpha((=)^N, t)$ and $\alpha(1(0)^a, t)$ for 75 generations, with an initial population consisting entirely of the only fit individual of length 3, namely "100" (so $p(100, 0) = 1$). The key results are summarized in Figures 3 and 4. Fig. 3(a) shows that after a few generations, the average length begins to grow quite steadily, at least for those 75 generations. As we saw in Sec. 1.2, though, there are significant risks in over generalizing from a small view like this, so it is by no means clear that this growth will continue. Our eventual goal is to develop something like a closed form for the average size at time t to better understand the long term behavior.

Both Figures 3 and 4 show interesting behavior in the first few generations that deserves attention. In Fig. 3(a) we see a flat section at the beginning, which then changes to steady growth. In Fig. 3(b) we see a *very* high proportion of short individuals at the beginning, which then drops quite steeply. In Fig. 4 we see a surprising dip in the proportion of fit individuals in the first few generations, which is then followed by the

[8] We have found, though, that ignoring values of α below some small threshold (we have used 10^{-10}) seems to have little impact on the numeric results and can greatly speed up the calculations since it significantly slows the growth of the number of strings that need to be tracked.

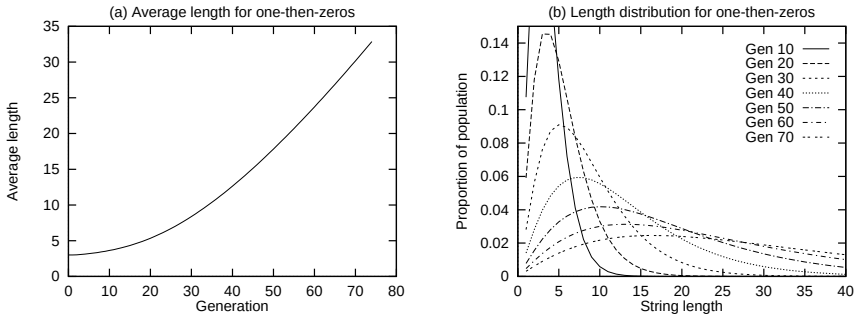

Fig. 3. Graph (a), on the left, shows the growth in the average length of the (infinite) population in the one-then-zeros problem. Graph (b), on the right, shows the distribution of lengths plotted for every 10 generations. The general shapes resemble the gamma distributions for the flat landscape in [20], but here the height of the distributions clearly drops and the peak moves to the right over time, reflecting the increase in average size shown in (a).

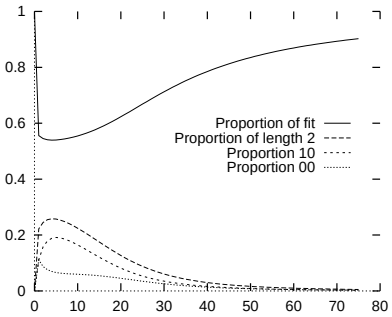

Fig. 4. The proportion of fit individuals (or, equivalently for this problem, the average fitness) in the one-then-zeros problem as a function of time, plotted alongside the proportion of individuals of length 2 and the proportion of both "10" and "00". (Note that *every* individual in the initial population is fit.)

expected rise. While we have no definitive explanation for these behaviors, we can suggest some possibilities. It seems likely, for example, that the rise in the proportion of strings of length 2 is simply a move in the early generations towards a gamma-like distribution with a mean length near 3 (the mean length of the initial distribution). This may, in turn, be the indirect cause of both the lack of initial bloat and the initial drop in the proportion of fit individuals.

In Fig. 4 the initial dip in the proportion of fit individuals corresponds closely to very high proportions of individuals of length 2. As the average size grows, however, the proportion of strings of length 2 in the (still gamma-like) distributions soon drops. Note also the significant proportion of the unfit string "00", which contributes to the initial dip in the proportion of fit individuals. After a sharp drop in the proportion of "00" in the first few generations, its proportion drops quite slowly thereafter (especially given that it has fitness 0). In contrast, the proportion of "10" (which has fitness 1) rises for several generations, but then drops *much* more quickly than that of "00", soon being almost equal to it. Since "00" has fitness 0, it can never be selected as a parent,

so neither replication accuracy nor removal bias can tell us anything about why a non-trivial proportion of the population's resources are devoted to sampling this string. It exists strictly as a creation effect, an effect frequently dismissed in other theories.[9]

These observations suggest that it may be common for runs to have an initial period that is heavily influenced (and in some cases dominated) by a redistribution of lengths towards a gamma-like distribution. This redistribution may have little to do with fitness, being instead driven by the size bias of standard crossover. There is, for example, considerable anecdotal evidence that many GP runs are initially dominated by short, moderately fit trees, and that the heavy sampling of these short trees can interfere with the progress of the run. The results presented here suggest this might be especially true if the initial average size is small (as in this case), as the gamma distribution with a small mean has a *very* strong bias towards over sampling small trees. It is possible that further exploration of these ideas could lead to important new understanding in the area of population initialization.

5 Landscape levels and program size

In the previous sections we saw how the schema theory could be used to better understand the evolution of length distributions over time both in cases where there was no bloat and in cases where there was bloat. In this section we will show how the theory can be used to predict some important behaviors in cases where the fitness can be seen as essentially two-valued.[10] What we find is that in the infinite population case the average length of the population moves away from the *average* length of the strings having the lower fitness. Thus the particulars of the fitness landscape are not important; what is important is simply the average length of the strings having the lower fitness and the average length of the population. Quite surprising, perhaps, is the fact that this holds even if one of the two levels is sampled by just a *single* string,[11] which means that even a single string with fitness different from that in an otherwise flat landscape is sufficient to drive the average size of the entire population. We will call small parts of the search space having the lower fitness "holes", and small parts of the search space having the higher fitness "spikes"; we will use the term "glitch" to describe both holes and spikes.

These results are based on the assumption of an infinite population, which ensures that the average size of the strings in the two fitness levels is accurately sampled. In the finite population case stochastic effects make it possible for one of the levels to not be sampled accurately (especially if that level only represents a small proportion of the search space). As an example, consider an a fitness landscape where the fitness of every string is 1 except for the string "00", which has fitness 0. If the population

[9] It is also a small example of the possible use of bloat as a repository for genetic material, as suggested in [2]. In this case the material in question never directly contributes positively to the fitness of individuals it's inserted into, but it does suggest that specific structures could be preserved in bloat and recovered later.

[10] The requirement that the fitness function has only two values might, at first, seem quite restrictive. There are settings, however, where this may not be such a unrealistic model. In a highly converged run of a discrete problem, for example, the selection process may well only turn up a limited number of fitness values (see, e.g., [7]).

[11] Here we mean a single string in the search space and *not* a single individual in the population.

contains many short strings, then the hole is likely to be heavily sampled and, as we shall see shortly, bloat will likely occur. If, however, the population consists of very long strings, then it is much less likely that the hole will ever be sampled, and if it isn't then the system will presumably behave (at least temporarily) as if it's acting on a flat fitness landscape. It is possible that this is at least part of the explanation for the increasingly noisy behavior of the size curves in the INC-IGNORE problem (Fig. 1(b)). As the average size in the small population grows, the sampling of the unfit strings may become increasingly erratic, causing (in part) this highly noisy behavior.

5.1 Computing the mean size

Since one of our key goals is tracking the changes in the mean population size over time, it would be nice if we could find an equation that relates the average size at time $t+1$ in terms of the average size at time t. In this section we will derive such an equation under the assumption that the fitness function has only two values, and the assumption of an infinite population.

Theorem 2 gave us the following formula for the average size of an infinite population at time t:

$$\mu(t+1) = \sum_N (N \times p((=)^N, t)) \ . \tag{5}$$

We will start by extending this result to include schemata other than those of the form $(=)^N$. Assume that the set of possible strings is partitioned into a set S of disjoint, fixed length schemata. We will write $S = \{S_i\}$ where the S_i are the schemata in question, and denote the length of the strings in S_i by $|S_i|$. Further assume that the fitness function is constant within each of these schemata; we will write $f(S_i)$ for the fitness shared by all instances of S_i. Note that the assumptions here are not restrictive since for any fitness function we can always decompose the space into singleton schemata, one for each different string, and those singleton schemata would trivially satisfy these conditions.

Given these new assumptions one can rewrite Eq. (5) as

$$\mu(t+1) = \sum_i (|S_i| \times p(S_i, t)) \tag{6}$$

and Eq. (1) (from Sec. 2.3) as

$$\mu(t) = \sum_i (|S_i| \times \phi(S_i, t)) \ . \tag{7}$$

It is useful in what follows to generalize μ so that we can compute the mean length of strings matching a particular subset Q of the schemata (i.e., $Q \subset S$):

$$\mu(Q, t) = \frac{\sum_i (|Q_i| \times \phi(Q_i, t))}{\sum_i \phi(Q_i, t)} \ . \tag{8}$$

Eq. (7) is now a special instance of Eq. (8) where $Q = S$, in which case the denominator is 1.

For fitness proportionate selection we know that

$$p(S_i, t) = \phi(S_i, t) \times f(S_i)/\overline{f}(t) = \frac{\phi(S_i, t) \times f(S_i)}{\sum_k (\phi(S_k, t) \times f(S_k))} \ . \tag{9}$$

Putting Eq. (9) into Eq. (6), we then get

Theorem 6 ($\mu(t+1)$ in terms of $\phi(t)$). *If the set of possible strings is partitioned into a set of disjoint, fixed length schemata $\{S_i\}$ of uniform fitness, we have, for an infinite population, standard crossover, no mutation, and fitness proportionate selection, that*

$$\mu(t+1) = \frac{\sum_i (|S_i| \times \phi(S_i,t) \times f(S_i))}{\overline{f}(t)} = \frac{\sum_i (|S_i| \times \phi(S_i,t) \times f(S_i))}{\sum_k (\phi(S_k,t) \times f(S_k))} . \quad (10)$$

Note that in Eq. (10), the numerator is nearly Eq. (7) for $\mu(t)$ except for the presence of the $f(S_i)$ term. This suggests that if we make some simplifying assumptions about the fitness, we might be able to write this in terms of $\mu(t)$. Assume, then, that the fitness function only has two values, 1 and $1+\hat{f}$. Then we can split the set $\{S_i\}$ of schemata into a disjoint union $\{S_i\} = \{A_x\} \cup \{B_y\}$ where $\{A_x\} = \{S_i \mid f(S_i) = 1\}$, and $\{B_y\} = \{S_i \mid f(S_i) = 1+\hat{f}\}$. This allows us to further rewrite Eq. (10):

$$\begin{aligned}
& \mu(t+1) \\
= \quad & \langle \text{ Eq. (10); definition of } \{A_x\}, \{B_y\}. \rangle \\
& \frac{\sum_x (|A_x| \times \phi(A_x,t) \times f(A_x)) + \sum_y (|B_y| \times \phi(B_y,t) \times f(B_y))}{\sum_x (\phi(A_x,t) \times f(A_x)) + \sum_y (\phi(B_y,t) \times f(B_y))} \\
= \quad & \langle \, f(A_x) = 1, f(B_y) = 1+\hat{f} \, \rangle \\
& \frac{\sum_x (|A_x| \times \phi(A_x,t)) + \sum_y (|B_y| \times \phi(B_y,t) \times (1+\hat{f}))}{\sum_x \phi(A_x,t) + \sum_y (\phi(B_y,t) \times (1+\hat{f}))} \\
= \quad & \langle \text{ Distributing across } (1+\hat{f}); \{S_i\} = \{A_x\} \cup \{B_y\} \, \rangle \\
& \frac{\sum_i (|S_i| \times \phi(S_i,t)) + \hat{f} \times \sum_y (|B_y| \times \phi(B_y,t))}{\sum_i \phi(S_i,t) + \hat{f} \times \sum_y \phi(B_y,t)} \\
= \quad & \langle \text{ Equations (7) and (8). } \rangle \\
& \frac{\mu(t) + \hat{f} \times \mu(\{B_y\},t) \times \sum_y \phi(B_y,t)}{1 + \hat{f} \times \sum_y \phi(B_y,t)} \\
= \quad & \langle \text{ Factor out } \mu(t). \rangle \\
& \mu(t) \times \frac{1 + \frac{\mu(\{B_y\},t)}{\mu(t)} \times \hat{f} \times \sum_y \phi(B_y,t)}{1 + \hat{f} \times \sum_y \phi(B_y,t)}
\end{aligned}$$

.

This, then, gives us the following:

Theorem 7 (Size evolution equation). *If $\{B_y\}$ is a set of disjoint, fixed length schemata of uniform fitness $1+\hat{f}$, and the rest of the search space has fitness 1, then, for an infinite population, standard crossover, no mutation, and fitness proportionate selection, we have*

$$\mu(t+1) = \mu(t) \times \frac{1 + \frac{\mu(\{B_y\},t)}{\mu(t)} \times \hat{f} \times \sum_y \phi(B_y,t)}{1 + \hat{f} \times \sum_y \phi(B_y,t)}. \quad (11)$$

The numerator and denominator in Eq. (11) differ only by $\mu(\{B_y\}, t)/\mu(t)$. This allows us to fully characterize the change in the (infinite) population's average size from time t to $t+1$ solely in terms of the sign of $\hat{f}$ and whether $\mu(\{B_y\}, t)/\mu(t)$ is greater or less than 1. For "spikes", i.e., when $\hat{f} > 0$, we have

$$\mu(\{B_y\}, t)/\mu(t) > 1 \Longleftrightarrow \mu(t+1) > \mu(t)$$
$$\mu(\{B_y\}, t)/\mu(t) < 1 \Longleftrightarrow \mu(t+1) < \mu(t).$$

For "holes", i.e., $\hat{f} < 0$, the converse is true:

$$\mu(\{B_y\}, t)/\mu(t) > 1 \Longleftrightarrow \mu(t+1) < \mu(t)$$
$$\mu(\{B_y\}, t)/\mu(t) < 1 \Longleftrightarrow \mu(t+1) > \mu(t).$$

Thus if the fitness of the B_i is better than the fitness of the A_i, the average size of the population will move *towards* the average size of the B_i. If, on the other hand, the fitness of the B_i is worse than the fitness of the A_i, then the average size of the population will move *away from* the average size of the B_i.

It is important to note that this latter case could lead to either bloat *or* a shrinking average size depending on whether $\mu(t)$ is above or below $\mu(\{B_y\}, t)$. Thus we see that the same forces that can lead to bloat in one setting can lead to the opposite effect in another setting. It is also important to note that while this tells us that (in these conditions) bloat will continue forever, it does *not* tell us that the magnitude of the bloat is unbounded; it is possible, for example, that the average size will continue to grow towards some asymptotic limit.

One of the most remarkable implications of this is that a *single* string with higher or lower fitness than an otherwise flat landscape is sufficient (in the infinite population case) to drive the average size of the entire population. Preliminary experiments with finite populations indicate that small holes are also capable of affecting change in the average length for a time, although sampling effects eventually take over. It's possible, therefore, that one might be able to use these ideas to develop new methods for managing the size of individuals during a run.

6 Conclusions and future work

In this paper we have shown how the schema theory for linear-structure GP from [20] can be productively applied to the important challenge of better understanding the changes in size distributions during a GP run. We have seen (Sec. 1.2) that, while experimental work is crucial, one must be very careful in interpreting the data that comes from unavoidably small views on a large and complex process. Applications of the schema theory to problems both with (Sec. 4) and without fitness (Sec. 3) suggest that standard crossover induces some very specific biases in the distributions of sizes, with a very strong tendency to over sample the smaller structures. There also appear to be significant redistribution effects that may be a major force in the early stages of a GP run. We have also generalized these specific results, yielding two important theoretical results. The first is an exact theory of bloat (Sec. 2.3) which gives a formula for the change in average program size for infinite populations of linear structures when using

standard crossover without mutation. The second is a general theory of how average sizes change on flat landscapes with glitches (Sec. 5), including the surprising result that a single string glitch is sufficient to drive the average program size of an infinite population.

These ideas open up numerous possible avenues for future research, both theoretical and applied. While this paper has concentrated on standard GP crossover without mutation, we have extended this elsewhere to include headless chicken crossover and two different types of subtree mutation [19, 14]. The theory could then be extended to combinations of operators, complete with different likelihood of application. Also, since the length distribution for the one-then-zeros problem (Fig. 3(b)) doesn't appear to reach a limit distribution as it did in the flat fitness landscape, finding a closed form for that distribution over time would be of real value. The two-level fitness theory tells us that under certain conditions bloat will happen, and will in fact continue forever, but at the moment it's not clear whether the average size rises towards some asymptotic value, or instead continues without bound.

One of the most intriguing possible applications of these ideas is the possibility of using artificially created "holes" to control code growth. The two-level fitness theory suggests that one might be able to slow or stop bloat either by lowering the fitness of a (possibly small) set of large individuals, or by raising the fitness of a set of small individuals. There are important issues, such as sampling errors, that would also need to be studied, but one might eventually be able to develop a method that would allow us to control code growth in a way which limits the changes made to the fitness landscape, thereby limiting the introduced bias.

Another key application is to the question of population initialization. Most of the existing initialization techniques are fairly ad hoc, and may in fact interact badly with the sampling bias induced by standard crossover. Our theoretical results could be used as the basis for new initialization techniques that minimize the potentially distributive effects of both length redistribution and over sampling of small structures.

In sum, we have seen that the schema theory can be successfully applied to a variety of linear problems, and that the results help answer important questions and suggest interesting new lines of inquiry. It is our expectation that further development will continue to yield valuable results.

Acknowledgements

The authors would like to thank Jonathan Rowe and other members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group at Birmingham for helpful comments and discussion. We are also grateful for the valuable feedback provided by Bill Langdon and the anonymous reviewers.

The first author would like to extend special thanks to The University of Birmingham School of Computer Science for graciously hosting him during his sabbatical, and various offices and individuals at the University of Minnesota, Morris, for making that sabbatical possible.

References

[1] L. Altenberg. The evolution of evolvability in genetic programming. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press, 1994.

[2] P. J. Angeline. Genetic programming and emergent intelligence. In K. E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press, 1994.

[3] T. Blickle and L. 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).

[4] R. Dawkins. *The selfish gene*. Oxford University Press, Oxford, 1976.

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

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

[7] W. B. Langdon. Quadratic bloat in genetic programming. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 451–458, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann.

[8] W. B. Langdon and W. Banzhaf. Genetic programming bloat without semantics. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, volume 1917 of *LNCS*, pages 201–210, Paris, France, Sept. 16-20 2000. Springer Verlag.

[9] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London, 23-27 June 1997.

[10] W. B. Langdon and R. Poli. Why ants are hard. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[11] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, June 1999.

[12] N. F. McPhee. A note on the derivation of transmission probabilities for a flat fitness landscape and for the one-then-zeros problem. 2000. Unpublished; contact the author at `mcphee@mrs.umn.edu` for a copy.

[13] N. F. McPhee and J. D. 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.

[14] N. F. McPhee, R. Poli, and J. E. Rowe. A schema theory analysis of mutation size biases in genetic programming with linear representations. Technical Report CSRP-00-24, University of Birmingham, School of Computer Science, December 2000.

[15] P. Nordin and W. 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.

[16] R. Poli. Exact schema theorem and effective fitness for GP with one-point crossover. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 469–476, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann.

[17] R. Poli. General schema theory for genetic programming with subtree-swapping crossover. In *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, Milan, 18-20 Apr. 2001. Springer-Verlag.

[18] R. Poli and W. B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 278–285, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[19] R. Poli and N. F. McPhee. Exact GP schema theory for headless chicken crossover and subtree mutation. Technical Report CSRP-00-23, University of Birmingham, School of Computer Science, December 2000.

[20] R. Poli and N. F. McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, Milan, 18-20 Apr. 2001. Springer-Verlag.

[21] T. Soule. *Code Growth in Genetic Programming*. PhD thesis, University of Idaho, Moscow, Idaho, USA, 15 May 1998.

[22] T. Soule and J. A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation*, pages 781–186, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

[23] C. R. Stephens and H. Waelbroeck. Schemata evolution and building blocks. *Evolutionary Computation*, 7(2):109–124, 1999.

[24] B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.