Using schema theory to explore interactions of multiple operators

Nicholas Freitag McPhee
Division of Science and Mathematics
University of Minnesota, Morris, USA
mcphee@mrs.umn.edu
320-589-6300

Riccardo Poli
Department of Computer Science
University of Essex, UK
rpoli@essex.ac.uk
+44-1206-872338

Abstract

In the last two years the schema theory for Genetic Programming (GP) has been applied to the problem of understanding the length biases of a variety of crossover and mutation operators on variable length linear structures. In these initial papers, operators were studied in isolation. In practice, however, they are typically used in various combinations, and in this paper we present the first schema theory analysis of the complex interactions of multiple operators. In particular, we apply the schema theory to the use of standard subtree crossover, full mutation, and grow mutation (in varying proportions) to variable length linear structures in the one-then-zeros problem. We then show how the results can be used to guide choices about the relative proportion of these operators in order to achieve certain structural goals during a run.

1 Introduction

Most (if not all) Genetic Programming (GP) operators have a variety of biases with respect to both the syntax and the semantics of the trees they produce. These biases can work against or in favor of the biases implied by the fitness function, which makes understanding these biases crucial to understanding the behavior of and relationships among the various operators.

These interactions can be quite complex, however, and consequently understanding them can be difficult. While there is a considerable literature examining the interactions of mutation and crossover in areas like Genetic Algorithms (GAs), there is much less reported work on the interactions of operators in GP. Notable exceptions include the work of O'Reilly [10], Banzhaf, *et al* [1], and Luke and Spector [5, 6, 4]. These studies are primarily experimental in nature, and all suggest that understanding operator interactions is difficult. It would thus be useful to have a theoretical approach to these problems that might allow us to better understand operator interactions, and choose combinations of operators in a more principled manner.

In the last few years work on schema theory for GP has made huge progress, generating not only an exact theory, but also one applicable to a variety of operators used in practice, including: one-point crossover [12, 14, 11, 13], standard and other subtree-swapping crossovers [14, 16, 7], different types of subtree mutation and headless chicken crossover [15, 8], and the class of homologous crossovers [17].

In [16, 7] we showed how these recent developments in GP schema theory can be used to better understand the biases induced by the standard subtree crossover when genetic programming is applied to variable length linear structures. In particular we showed that subtree crossover has a very strong bias towards oversampling shorter strings and, in some senses, works against bloat. In [15, 8] we derived exact schema equations for subtree mutation on linear structures, using both the full and grow methods to generate the new, random subtrees. Iterating those equations on both a flat fitness landscape and a needle-in-a-haystack style problem, called the one-then-zeros problem, we showed that both of these subtree mutation operators have strong biases with regard to the population's length distribution. Similar to the bias of subtree crossover, we found that these mutation operators are strongly biased in favor of shorter strings in both these fitness domains.

In this paper we combine the schema theory for different operators and apply them to the problem of better understanding the behavior produced by their interaction. Studying these complex interactions is particularly easy using the schema formalization because we can simply use a weighted sum of the schema equations generated for each operator in isolation. We also show how the theory can be used to design competent GP systems by guiding the choice of combinations of operators together with their parameter settings.

The work reported here is all on GP with linear structures (not unlike those used in, e.g., [9, 2]), although the schema theory on which it is based is much more general. We have chosen in these 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 Sec. 2 we will introduce the schema theorem for GP using linear structures, standard crossover and mutation, and we will show how easily the theory for different operators can be combined. We then apply the theory in Sec. 3 to the one-then-zeros problem and use the theory to both predict and better understand the changes in the distribution of fit individuals and of sizes (Sec. 4). We finish with some conclusions and ideas for future research (Sec. 5).

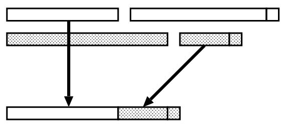
2 Schema theory for GP on linear structures

2.1 Operators

In this paper we will consider three common GP operators: the standard subtree-swapping GP crossover operator, and the full and grow mutation operators. Each operator acts by removing a non-empty suffix of an individual and replacing it with a new suffix, with the production of that suffix being the primary difference between the operators.

More formally, 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 $c_0c_1 \ldots c_{N-1}$ where $c_i \in \mathcal{F}$ for $i < N-1$ and $c_{N-1} \in \mathcal{T}$. Each of the operators, then, starts by removing a non-empty suffix $c_jc_{j+1} \ldots c_{N-1}$ (where j is chosen uniformly such that $0 \leq j < N$) and replacing it with a new non-empty string.[1]

In the case of crossover, the new string is taken to be a suffix $d_{j'}d_{j'+1} \ldots d_{N'-1}$ of another parent $d_0d_1 \ldots d_{N'-1}$, where j' (which could differ from j) is chosen uniformly such that $0 \leq j' < N'$.



[1]The requirement that suffixes be non-empty while prefixes are allowed to be empty comes from standard practice in GP. It does, however, create a number of mild but annoying asymmetries which often clutter up the analysis (see, e.g., [18]).

Both full and grow mutation generate the new suffix randomly, and they differ in how the new random subsequences are generated, and in particular how their sizes are determined. In full mutation, the subsequence has a specified length D; thus non-terminals are selected uniformly from $\mathcal{F}$ until length $D-1$ is reached, at which point a terminal is selected uniformly from $\mathcal{T}$. In grow mutation, on the other hand, one chooses from the set of *all* functions and terminals every time, only terminating the creation of the subsequence when a terminal is chosen; thus for grow mutation there is no *a priori* limit on the size of the resulting sequences.

2.2 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.

Just as we defined a linear GP structure to be a sequence of symbols, we will also define a linear GP *schema* as the same kind of sequence $c_0c_1 \ldots c_{N-1}$ 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 symbol 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*

[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 [14].

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

of the selection operators. With fitness proportionate selection, for example, $p(H,t) = \phi(H,t) \times f(H,t)/\overline{f}(t)$, where $f(H,t)$ 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 an instance of the schema H will be constructed in the process of creating a new individual for the population at time $t+1$ out of the population at time t. This will typically be a function of $p(K,t)$, the various schemata K that could play a role in constructing H, and the details of the various recombination and mutation operators being used.*

Definition 4 (Creation probability) $\pi_{mut}(H,t)$ *is the probability that some GP subtree mutation operator will generate a new, random subtree that is an element of the schema H in generation t.*

To clarify which operator we are working with, we introduce specialized forms of the transmission probability function α, namely α_{xo} for the transmission probability due specifically to crossover, α_{FULL} for the transmission probability due specifically to subtree mutation using the full method, and α_{GROW} for the transmission probability due specifically to subtree mutation using the grow method.

We can now model the standard evolutionary algorithm as the transformation

$$\phi(H,t) \overset{\text{select}}{\longrightarrow} p(H,t) \overset{\substack{\text{mutate}\\ \text{XO}}}{\longrightarrow} \alpha(H,t) \overset{\text{sample}}{\longrightarrow} \phi(H,t+1).$$

Here the arrows indicate that some new distribution (on the RHS of the arrow) is generated by applying the specified operation(s) to the previous distribution (on the LHS). So, for example, the process of selection can be seen as a transformation from the distribution of schemata $\phi(H,t)$ to the selection probability $p(H,t)$. A crucial observation is that, for an *infinite* population, $\phi(H,t+1) = \alpha(H,t)$ for $t \geq 0$, which means we can iterate these transformations to *exactly* model the behavior of an infinite population over time.

To formalize the creation of instances of a linear schema $H = c_0 c_1 \ldots c_{N-1}$ we define

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

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 property of u and l is that if one uses 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.

2.3 The schema theorem

[7, 8] provide schema theorems for each of our three operators when used in isolation. Here we extend these results to the case where all three operators can be used in the same run, each with specified proportions. Since we use exactly one operator to generate any given individual, the probability that we construct an instance of a schema (i.e., $\alpha(H,t)$) is simply the sum of the probabilities of each specific operator constructing such an instance, each weighted by the likelihood of choosing that operator. This leads to the following:

Theorem 1 (Schema theorem for combined operators) *For GP on linear structures using standard crossover with probability p_{xo}, full mutation with length D and probability p_{FULL}, and grow mutation with probability p_{GROW}, such that $p_{xo} + p_{FULL} + p_{GROW} = 1$, we have*

$$\begin{aligned} \alpha(H,t) &= p_{xo} \times \alpha_{xo}(H,t) \\ &+ p_{FULL} \times \alpha_{FULL}(H,t) + p_{GROW} \times \alpha_{GROW}(H,t) \end{aligned}$$

where

$$\begin{aligned} \alpha_{xo}(H,t) &= \sum_{\substack{k>0\\ n>0\\ k+n>N}} \Big(\frac{1}{k \times n} \\ &\times \sum_{0 \uparrow (N-n) \leq i < N \downarrow k} p(u(H,i,k),t) \\ &\qquad \times p(l(H,i,n),t) \Big), \\ \alpha_{FULL}(H,t) &= \sum_{\substack{k>0\\ 0 \leq i < N \downarrow k}} \Bigg(\frac{1}{k} \times p(u(H,i,k),t) \\ &\qquad \times \pi_{FULL}(c_i c_{i+1} \ldots c_{N-1}) \Bigg) \\ \alpha_{GROW}(H,t) &= \sum_{\substack{k>0\\ 0 \leq i < N \downarrow k}} \Bigg(\frac{1}{k} \times p(u(H,i,k),t) \\ &\qquad \times \pi_{GROW}(c_i c_{i+1} \ldots c_{N-1}) \Bigg), \end{aligned}$$

and $q = |\mathcal{F}|/(|\mathcal{F}| + |\mathcal{T}|)$.

[4] u and l are based on operators U and L (see, e.g., [14]) 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.

Due to space restrictions we simply report the general expressions for the quantities α_{xo}, α_{FULL}, and α_{GROW} for the linear case without providing any proofs. The interested reader can find these, together with extensive characterizations of the behavior of crossover and mutation when used separately, in [7, 8].

3 The one-then-zeros problem

We will now apply the Schema Theorem to the *one-then-zeros problem*. We will start by defining and motivating the problem, and then show how the schema theorem can be used to better understand the effects of multiple operator interaction on this problem.

3.1 One-then-zeros problem definition

In this problem we have $\mathcal{F} = \{0, 1\}$ and $\mathcal{T} = \{0\}$, where both 0 and 1 are unary operators. 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. 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.

One of the reasons for studying this problem is that under selection and crossover this problem induces bloat [7], whereas this does not happen when using the full and grow mutation operators [8]. The key advantage of this problem is that in order to fully and exactly describe the length-evolution dynamics and the changes in solution frequency of infinite populations, it is necessary to keep track of only two classes of schemata: those of the form $(=)^N$ and those of the form $1(0)^a$. Unfortunately most problems are not so restricted, and one is typically forced to track the proportion of many (possibly intractably many) more schemata.

3.2 Analyzing one-then-zeros

To apply the schema theorem to the one-then-zeros problem one needs to calculate the probabilities α_{xo}, α_{FULL}, and α_{GROW} for both of the schema $(=)^N$ and $1(0)^a$, and the probabilities π_{FULL} and π_{GROW} for both of the schema $1(0)^a$ and $(0)^a$. These can be calculated from the equations reported above and are also provided in explicit form in [7, 8], so we will not re-derive these results here.

If we assume an infinite population, we can numerically iterate the equations in the Schema Theorem to better understand the behavior of an infinite GP population on this problem. Tracking these distributions over time becomes expensive in terms of computational effort.[6] 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.

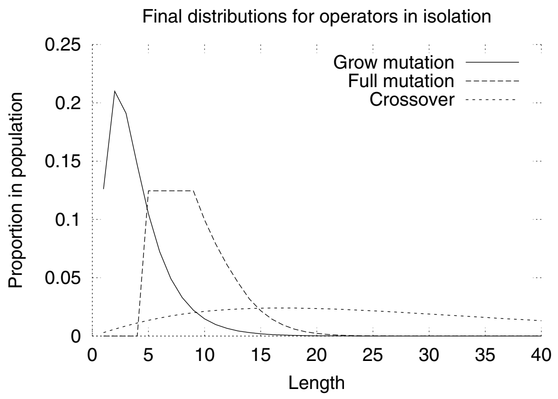


Figure 1: The distributions of lengths after 50 generations when using the three recombination operators individually on the one-then-zeros problem. The tail of the crossover distribution continues past the right hand side of the graph, with lengths above 300 still having proportions above 10^{-10}.

4 One-then-zeros results

We know (see, e.g., [7, 16, 8, 15]) that each of these operators has significant biases when used on its own, and Fig. 1 summarizes some of the earlier results by presenting the final length distributions for each of the operators when acting alone on the one-then-zeros problem. This makes it clear that the three operators all have very different length biases, which suggests that they may indeed demonstrate interesting behaviors when used in combination.

We can now iterate this new combined schema equation to study these combined interactions and their biases, and to use such results to guide the choices of the proportions of operators to help satisfy a variety of goals. As an example in this paper we will consider the following goals:

1. Avoid both bloating and shrinking, by having the average size after 50 generations be as close as possible to the initial average size.

[6]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.

2. Avoid both bloating and shrinking (as above), but also maximize the number of correct individuals.
3. Maximize the proportion of small solutions (as opposed to just short strings).
4. Reach a state where the proportion of $1(0)^{30}$ exceeds 0.01 as early as possible.

In all these simulations we will be applying the three operators discussed earlier (standard subtree crossover, full mutation, and grow mutation) on the one-then-zeros problem. A depth limit $D = 5$ will be used for full mutation. Our initial population will consist of equal proportions (10% each) of the strings $1(0)^i$ for $1 \leq i \leq 10$; thus the average length in the initial population is 6.5.

To study the interaction of the operators, the schema equations from Theorem 1 were iterated 66 different times, using each of the legal combinations of proportions of (standard) crossover, grow mutation, and full mutation with values from the set $\{0, 0.1, 0.2, 0.3, \ldots, 0.9, 1\}$. We'll use triples of the form $(p_{xo}, p_{\mathrm{FULL}}, p_{\mathrm{GROW}})$ to indicate a combination of parameter settings where the first is always the proportion of crossover, the second the proportion of full mutation, and the third the proportion of grow mutation.

4.1 General observations

While the majority of these iterations had converged after 50 generations, there were several which had not. These were typically those with sufficiently high crossover probabilities that bloat was occurring and the average lengths were clearly still growing after 50 generations. As an example, the configuration (0.8, 0, 0.2) has an average length of 7.98 after 50 generations, and is thus not a terrible solution to the problem of avoiding bloat and shrinkage as defined in Sec. 4.2 below. It seems highly likely, however, that if we were to continue iterating the equations with these parameters for another 100 generations we would get higher average length, thereby doing a worse job of meeting the goal of avoiding bloat and shrinkage. This isn't necessarily a concern, however, since actual GP runs always have a finite number of generations. Thus if we know we're likely to run our GP for 100 generations, we can iterate these schema equations and try to find settings that meet our goals (whatever they happen to be) at the end of 100 generations *regardless of whether further generations would take us away from our goals*.

It should also be noted that the initial uniform distribution of lengths is very unstable in the sense that any combination of operators will generate a very different distribution immediately in the first generation. As an example, the settings (0.1, 0.7, 0.2) have an average length after 50 generations that's very close to the average length in the initial distribution (6.5). The distribution itself (shown in Fig. 3) is far from uniform, however. This seems to be a general property of "interesting" operators, namely that they have a favored length distribution that they move to quite quickly, and while fitness can modify that tendency, it rarely eliminates it entirely.

4.2 Avoid bloat and shrinkage

In our first example the goal will be to avoid both bloating and shrinkage by searching for a collection of operator probabilities such that the average length after 50 generations is as close as possible to the initial average size.

Out of our 66 configurations, five had a final average fitness that was less than 0.15 away from the initial average of 6.5 (see Table 1); the next closest combination of parameter settings had an absolute difference of over 0.23. Note that in each case the proportion of grow mutation was 0.2. In fact the 20 configurations whose final average lengths were closest to 6.5 all had small non-zero proportions for grow mutation (between 0.1 and 0.4); at the same time, however, those 20 configurations had a broad range of full mutation proportions (ranging from 0 to 0.9) and crossover proportions (from 0 to 0.8). Those combinations where the proportion of crossover was over 0.5, however, all had average lengths that were still climbing after 50 generations, so it's likely that they would continue to diverge from 6.5 if we iterated the equations for more generations. Thus the crucial factors for long-term size stability *in this problem* seem to be a small non-zero proportion of grow, and a crossover proportion of at most 0.5 so the sizes don't bloat above 6.5.

Most (but not all) of the configurations where the proportion of grow was 0.2 had final average lengths close to 6.5; the smallest average length after 50 generations was 6.36 (for $(0.4, 0.4, 0.2)$), and the largest was 7.98 (for $(0.8, 0, 0.2)$). As discussed above, however, those parameter sets with higher crossover proportions probably hadn't converged after just 50 generations, and their final averages would likely continue to grow if we iterated more generations. Taking that into account the range of final average lengths is quite small, being from 6.35 to 6.46 when the proportion of grow is 0.2 and the proportion of crossover is at most 0.5.

Looking at Fig. 2, we can see that in each of these cases there was an initial jump away from 6.5 (caused by the instability of the initial uniform length distribution), followed by a fairly rapid convergence to an average value close to 6.5. The slowest to converge was the case where we had 50% crossover, and that curve in fact looks similar to the bloating seen in [7], with an asymptote close to 6.5.

Fig. 3 shows the final distribution of lengths for each of these five parameter settings. While each of these distribu-

XO	Full	Grow	Diff. from 6.5	Prop. fit
0.1	0.7	0.2	0.081	0.19
0.2	0.6	0.2	-0.045	0.26
0.3	0.5	0.2	-0.131	0.31
0.4	0.4	0.2	-0.147	0.37
0.5	0.3	0.2	-0.045	0.43

Table 1: Parameter settings for the five configurations that came closest to having the same average length after 50 generations as the average length of the initial distribution. "XO" is the proportion of crossover, and "Full" and "Grow" are the proportions of full and grow mutation. "Diff. from 6.5" is the difference between the actual final average length for this set of parameters and the initial average length (6.5); negative values mean that the final average length was less than 6.5. "Prop. of fit" is the proportion of the individuals produced in the last generation that were fit.

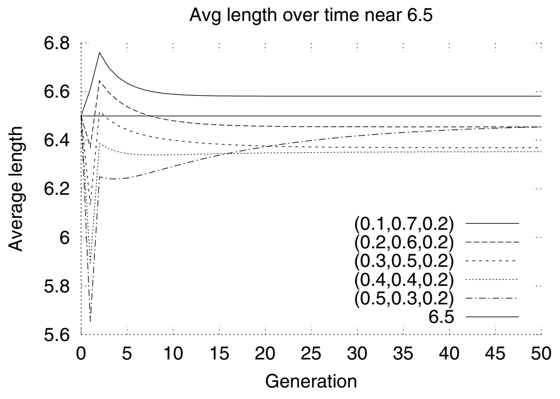


Figure 2: Average lengths over time for the five collections of parameter settings leading to a final average length closest to the initial average length (6.5).

tions has an average length that is nearly equal to that of the initial uniform distributions, none of these distributions is remotely uniform. They instead exhibit combinations of features seen in earlier studies of using single recombination operators on this problem (see Fig. 1). In each case, for example, we see a peak at length=5 which is due to full mutation with depth 5, and the height of the peak is clearly correlated to the full mutation probability.

4.3 Avoid bloat and shrinkage, maximizing correct proportion

In the preceding example we looked for parameter settings that avoided both bloat and shrinkage. It's possible, however, that this goal was met at the expense of correctness. A given collection of parameter settings could, for example, generate the desired average size, but have a very low proportion of fit individuals. This would in turn greatly reduce the effective population size since most of the generated individuals can never be selected for recombination.

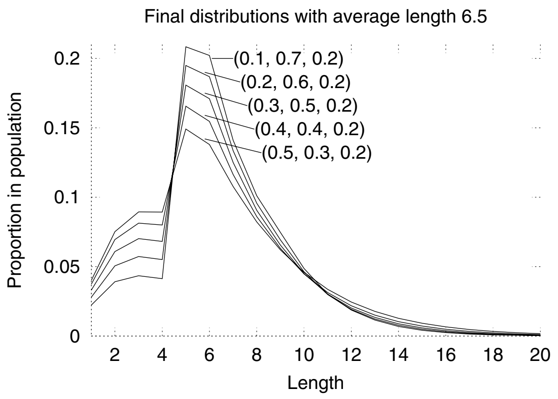


Figure 3: Final length distributions of the five parameter settings whose final average length was closest to the initial average length (6.5).

We can assess this by looking at the proportion of correct individuals in the final generation for each set of parameter values, and we indeed see that there are substantial differences among these five configurations (see Table 1), with the values ranging from 0.19 to 0.43. It's also clear that increased probabilities of crossover correspond with increased proportion of fit individuals. This is not surprising since increased probabilities of crossover also correspond to decreased probabilities of full mutation, and full mutation is rarely going to produce a fit offspring in this problem [8]. Crossover on the other hand, has a high probability of generating correct offspring, especially when given two correct, fairly long individuals as parents [7].

Another approach to optimizing these two criteria would be to start by identifying the configurations with high proportion of fit individuals in the final generation, and then choosing from those the parameter settings that also lead to final average lengths near 6.5. The settings with the highest proportion of fit individuals are those with high crossover probabilities, but these configurations also have the highest final average lengths (because high crossover probabilities lead to bloat in this problem [7]). One of the best settings is (0.8, 0.0, 0.2), which has a final average length of 7.98 (nearly 1.5 nodes longer than the original average of 6.5) but a final proportion of fit individuals of 0.65 (about 0.22 higher than the proportion generated by (0.5, 0.3, 0.2)).

4.4 Maximize proportion of small solutions

Now consider the case where we want to minimize the average size of the *fit* individuals (i.e., those of the form $1(0)^a$ for $a > 0$). There are quite a few combinations of parameter settings that lead to average size of fit individuals that are just above 3. The three smallest are (0, 0, 1), (0.1, 0, 0.9), and (0.2, 0, 0.8) with final average sizes of fit individuals 3.02, 3.04, and 3.07 respectively. This suggests that (for this problem) the best way to make short correct solutions is to primarily use grow mutation, with small amounts of crossover being acceptable as well. Adding small amounts of full mutation doesn't lead to much bigger average sizes (the average size for (0, 0.1, 0.9), for example, is 3.13) despite the fact that depth mutation always generates strings of length at least $1 + D$ (or 6 in our case). This is probably due to the fact that full mutation will very rarely generate correct individuals in this problem.

If one wanted to further maximize the proportion of fit individuals, then the three candidate combinations have progressively increasing proportion of fit individuals; the highest is (0.2, 0, 0.8), which has a proportion of 0.41. If we look more broadly, we find that (0.6, 0, 0.4) has a somewhat higher proportion of fit individuals after 50 generations (0.49), with the average size of fit individuals being only slightly higher (3.54).

4.5 Find solutions of length 31 quickly

For our final example we will consider the goal of finding solutions of the form $1(0)^{30}$ as quickly as possible. There are a variety of motivations for this sort of goal, but one might be that instead of only having a two level fitness function, we might have a three level fitness function: Fitness 0 for individuals that don't have the "one then zeros" pattern, fitness 1 for individuals of the form $1(0)^a$, $a \neq 30$, and fitness 2 for individuals of the form $1(0)^{30}$. If we further assume that our run will terminate as soon as we discover a target individual $1(0)^{30}$, then dynamics of such a run are identical to the original one-then-zeros problem, except they terminate upon discovery of a target individual.

Thus we can use our schema theory results to discover what parameter settings lead most quickly to the discovery of a target individual. Because of the infinite population assumption, however, we may find that early in a run there is a *very* small, but still positive, proportion of target individuals, yet with such small proportions the likelihood is minuscule of actually generating a target individual that quickly in a "real" (finite population) run. We will, therefore, look for the collection of parameter settings that first achieves a proportion of target individuals exceeding 0.01.

Only four of our tested parameter settings ever obtain a proportion of at least 0.01 target individuals (see Table 2), with all crossover (1, 0, 0) reaching the target the most quickly (in 26 generations). Adding small amounts of full mutation still allows the goal to be satisfied, but even a proportion of 0.3 is enough to increase the number of generations by 12. Grow mutation clearly interferes with this goal, as none of the four parameter settings that achieve the goal have any grow mutation.

XO	Full	Grow	First gen to 0.01
1.0	0.0	0.0	26
0.9	0.1	0.0	28
0.8	0.2	0.0	32
0.7	0.3	0.0	38

Table 2: Parameter settings for the four configurations that eventually achieve a proportion of 0.01 target individuals $1(0)^{30}$. The first three columns are as in Table 1. "First gen to 0.01" is the first generation for a given collection of parameter settings where the proportion of target individuals exceeded 0.01.

If we relax the target proportion to 0.001 there are a total of 12 parameter settings that achieve this new goal. Of these only four have non-zero grow mutation probabilities, all of which are the lowest possible value (0.1). Similarly, all but three of these 12 settings have crossover probabilities exceeding 0.5, although one (0.3, 0.7, 0) managed to reach the target of 0.001 in 21 generations despite the low crossover probability. It's interesting to note, however, that two of the settings with non-zero grow mutation probabilities ((0.7, 0.2, 0.1) and (0.6, 0.3, 0.1)) both reached the goal more slowly (in 23 and 29 generations respectively) despite having much high crossover probabilities.

It's not terribly surprising that crossover is useful in increasing the length of fit strings, since we've previously seen that crossover can lead to bloat (presumably due to replication accuracy) [7]. Further, one would expect both mutation operators to at least slow down the process of generating a target string containing thirty 0's, and thus having length 31 (see [18] for details):

- Given a parent string of length l, full mutation generates (on average) a string of length roughly $l/2+D$, so full mutation tends to generate shorter offspring once $l > 2D$. Since $D = 5$ in our examples, full mutation will tend to reduce the size of strings once their lengths begin to exceed 10.
- Given a parent string of length l, grow mutation in the one-then-zeros problems will generate (on average) a string of length roughly $l/2 + 3$. Thus grow mutation will tend to reduce the size of strings once their lengths begin to exceed 6.

What's perhaps more surprising is that grow mutation interferes with the process of finding a target string so much more than full mutation does. The likely reason is that grow mutation is more likely to produce fit offspring than full mutation (see [8, 18] for details). Because of the infinite population assumption, generating unfit offspring has no substantial effect on the dynamics of the system, as doing so has no effect on the selection probabilities. Generating short, fit individuals, however, will change the dynamics by increasing the probability that short individuals are selected as parents in the next generation. In our case its likely that grow mutation creates a sufficient number of short, fit strings that it can significantly hamper the process of generating fit strings of length 31.

5 Conclusions and future work

It's clear, then, that there is a fairly complex set of interactions between these three recombination operators, making it quite difficult to guess *a priori* what proportions of operations would aid in satisfying goals that might be important in a particular domain. For this problem, however, we were able to iterate the schema equations on many different combinations of operator proportions, generating a useful map of the interactions.

In this paper the number of different combinations of proportions was small enough to make manual searches for desirable values feasible. With more operators, or a larger variety of different proportions, the number of combinations would quickly grow out of control, making it prohibitive to iterate the equations for every combination and then search the results by hand. Since this is essentially just another parameter optimization problem, one possibility would be to apply a GA, although in many cases something simpler like a hill-climber would probably also work. Another possibility (which could potentially dramatically reduce the number of different combinations that would need to be iterated) would be to use factorial design of experiments [3].

Perhaps the key observation here is that there is clearly no "best" set of operator proportions, and that the desirability of a combination of operators will depend critically on the specific goals. It is therefore particularly important that we have tools that help us understand not only the general interactions of operators, but also understand the more specific interactions in order to guide our choices.

Acknowledgments

Both authors would like the thank the Dagstuhl International Conference and Research Center for Computer Science, and the organizer of Dagstuhl Seminar No. 02031, for providing the wonderful environment that made this paper possible.

References

[1] W. Banzhaf, F. D. Francone, and P. Nordin. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 300–309, Berlin, Germany, 22-26 Sept. 1996. Springer Verlag.

[2] D. S. Burke, K. A. D. Jong, J. J. Grefenstette, C. L. Ramsey, and A. S. Wu. Putting more genetics into genetic algorithms. *Evolutionary Computation*, 6(4):387–410, Winter 1998.

[3] R. Feldt and P. Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 271–282, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[4] S. Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. PhD thesis, Department of Computer Science, University of Maryland, A. V. Williams Building, University of Maryland, College Park, MD 20742 USA, 2000.

[5] S. Luke and L. Spector. A comparison of crossover and mutation in genetic programming. 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 240–248, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[6] S. Luke and L. Spector. A revised comparison of crossover and mutation in genetic programming. 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 208–213, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[7] N. F. McPhee and R. Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. In *Genetic Programming, Proceedings of EuroGP 2001*, LNCS, Milan, 18-20 Apr. 2001. Springer-Verlag.

[8] N. F. McPhee, R. Poli, and J. E. Rowe. A schema theory analysis of mutation size biases in genetic programming with linear representations. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul, Korea, May 2001.

[9] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transaction on Evolutionary Computation*, 5(4), 2001.

[10] U.-M. O'Reilly. *An Analysis of Genetic Programming*. PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 Sept. 1995.

[11] 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*, pages 469–476, Las Vegas, July 2000. Morgan Kaufmann.

[12] R. Poli. Hyperschema theory for GP with one-point crossover, building blocks, and some new results in GA theory. In R. Poli, W. Banzhaf, and *et al.*, editors, *Genetic Programming, Proceedings of EuroGP 2000*. Springer-Verlag, 15-16 Apr. 2000.

[13] R. Poli. Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2), 2001.

[14] 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.

[15] R. Poli and N. F. McPhee. Exact GP schema theory for headless chicken crossover and subtree mutation. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001*, Seoul, Korea, May 2001.

[16] 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.

[17] R. Poli and N. F. McPhee. Exact schema theory for GP and variable-length GAs with homologous crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[18] J. E. Rowe and N. F. McPhee. The effects of crossover and mutation operators on variable length linear structures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.