Grammatical Evolution of L-systems

Darren Beaumont and Susan Stepney
Department of Computer Science, University of York, YO10 5DD, UK

***Abstract*— L-systems are parallel generative grammars that can model branching structures. Taking a graphical object and attempting to derive an L-system describing it is a hard problem. Grammatical Evolution (GE) is an evolutionary technique aimed at creating grammars describing the legal structures an object can take. We use GE to evolve L-systems, and investigate the effect of elitism, and the form of the underlying grammar.**

I. Introduction and Background

L-systems are parallel generative grammars [1] used to model plant development, branching tree structures, and other iteratively-defined and fractal artefacts. Starting from an axiom string, or seed, the grammar rules are applied in parallel to each element of the string, for several iterations (generations). For example, consider the following L-system:

$\omega : F$
$p : F \rightarrow F[+F]F[-F]F$

This has axiom F, and each iteration the production p replaces every F in the current string by the RHS of p. So, starting from the axiom string as iteration 0, successive iterations are:

0) F
1) $F[+F]F[-F]F$
2) $F[+F]F[-F]F[+F[+F]F[-F]F]F[+F]F[-F]F$
$[-F[+F]F[-F]F]F[+F]F[-F]F$

The resulting string is typically rendered graphically, by interpreting the elements as turtle graphics commands [2][3] (although other interpretations are possible [4]). For example, interpreting F as 'forward distance d, drawing a line', $\pm$ as 'turn through $\pm 27.5°$', and [ ] as 'start/end branch', then subsequent iterations of this example L-system render as an increasingly-branched fractal 'weed' (figure 1).

Given an L-System definition, it is relatively easy to render the resulting structure. However, given a structure, it is non-trivial to derive a corresponding L-System.

Grammatical Evolution (GE) [5] is a grammar-based evolutionary search technique. GE uses a string of numbers to represent an individual. This genome is used to index into a Backus Naur Form (BNF) grammar to generate a resulting text. The genome represents an individual's genotype, which is manipulated by the search algorithm[1] (see [5] for details). The resulting text represents an individual's phenotype and is used to test an individual's fitness.

Since L-systems are defined using grammars, GE seems an ideal approach for evolving particular L-system descriptions.

[1]Note that the GE mapping process is independent of the search algorithm. Usually a genetic algorithm is used to explore the search space (and is used here), but it is possible to use other search techniques, such as simulated annealing.

Fig. 1. The example L-system after 1, 2, 3, and 4 iterations. (The pictures are scaled each iteration to make their overall size constant.)

For example, [6] use GE to evolve fractal curves described by a particular restricted L-system grammar. Here we evolve tree-like structures based on a more flexible BNF description of context-free L-system grammar.

The paper is structured as follows. Section II outlines the approach followed. A detailed set of tests and experimental results are presented on elitism in sections III, on the effect of changing the structure of the grammar in section IV, and on more complex grammars in section V. Conclusions and further work are given in section VI.

II. Design and Implementation

A. The default BNF Grammar

Figure 2 shows the default BNF grammar we use to capture our context-free L-system grammar.

On decoding, each individual starts as an `lsys`, then each codon number in its genome is used in sequence to index into the BNF grammar, to determine how to expand the productions. If there are productions remaining when the end of the genome is reached, decoding wraps around to start from the beginning of the string again. Decoding stops when no non-terminal symbols remain, or when the maximum allowed number of genome wrap-arounds is reached.

To simplify the problem of deriving an L-system we follow [6] and initially assume the axiom is fixed as F. The default grammar also assumes that F is the only LHS allowed in a production. These assumptions are relaxed in section V. As a further simplification, we fix the number of derivation iterations, and the rotation angle.

A context-free non-probabilistic L-system requires each symbol (here, `variable`) to appear as the LHS in at most

```
<lsys>  ::=  <axiom> <productions>

<axiom>  ::=  F

<productions>
  ::=  <lhs> <str>
   |   <lhs> <str>
       <productions>

<lhs>  ::=  <variable>

<str>
  ::=  <symbol>
   |   <branch>
   |   <symbol><str>

<branch>  ::=  [<str>]  |  [<str>]<str>

<symbol>  ::=  <variable>  |  <rotate>

<variable>  ::=  F

<rotate>  ::=  +  |  -
```

Fig. 2. Default BNF grammar used

one production. With our BNF, a genome could potentially derive an L-system grammar where more than one production has the same LHS symbol. We therefore require the L-system turtle renderer to use only the first production with a given LHS, and ignore all subsequent productions with that LHS. (This allows easy future extension to probabilistic L-systems, where multiple productions with the same LHS are permitted.)

We ensure automatic scaling of the rendered images (as in fig 1), to ensure that the phenotypes being assessed all have the same size.

B. Cost function

The 'cost' is the difference of the candidate image from the target image, as measured on an image rendered on a fixed size canvas. If an individual has a cost of zero, then every pixel of the individual's and target's tree are identical (at the given rendering resolution; it may be that a finer resolution could distinguish them).

We want branches that are close in position to the target branches to have a low cost, so we calculate the cost function in terms of the distance apart of corresponding pixels. To calculate the difference of image1 from image2, we consider every 'on' pixel in image1, find the Euclidean distance to the nearest 'on' pixel in image2, and sum these distances. The cost is the difference of image1 from image2, plus the difference of image2 from image1. (If we calculated this difference in only one direction, either any image would perfectly match a black target, or a black image would perfectly match any target.) Figure 5 shows a range of different cost results: lower cost trees do indeed look more similar to the target than higher cost trees.

If an individual's L-system draws nothing (e.g. one with a production F→[+]) then there are no nearest pixels to those

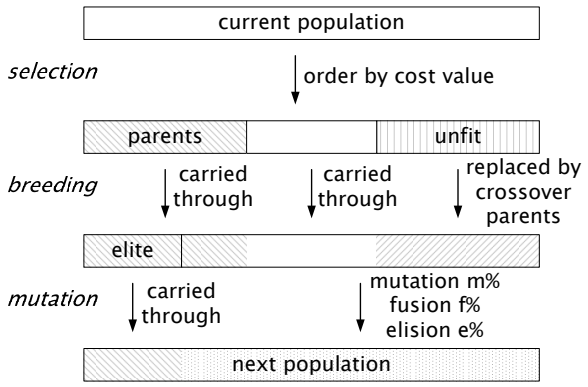

Fig. 3. The breeding process

on the target tree. In this case each pixel in the target tree is assigned a nearest distance equal to the maximum of the width or height of the window. (See appendix A for pseudocode.) Hence individuals with L-systems that draw something (no matter how poor) are favoured over those with L-systems that draw nothing. For our image size, a simple 'stick' figure has a cost of 518257: this is the worst outcome (highest cost) of our experiments.

The cost function also punishes individuals that reach the maximum number of genome wraps (and hence do not have a complete L-system grammar) with a high cost values (the Java maximum integer value, $2^{31} - 1$). This attempts to minimise the number of invalid individuals in each generation.

C. Genetic Operators

A genome is a string of codons. Following [6], a codon is an integer (rather than a bitstring, say). A new generation is created from the current generation as follows (summarised in figure 3):

1) Selection: We use a simple selection strategy. Individuals in the population are ordered according to their cost value with the fittest (cost value closest to zero) first, down to the least fit. A percentage of the best individuals are identified as parents, while a percentage of the worst individuals are replaced.

2) Breeding: We use one point crossover to generate new individuals from the parents, to replace the removed individuals. Each new individual is created thus: randomly select two parents; randomly select a crossover point on each parent's genome (these may be different points); cross over the genomes at these point to create a new child genome.

Our crossover may randomly choose the same genome for both parents. Whereas [6] use the same crossover point on both parents' genome, we allow the crossover point of each parent's genome to be different, to allow greater variation in the genome length of children.

3) Mutation: Elitism allows a certain number of the fittest individuals to go through to the next generation without being mutated.

cost evaluations	time (mins)	time/eval (secs)
1,000	5.9	0.354
10,000	100	0.60

Fig. 4. Mean time taken to perform cost evaluations

Mutation lets each codon be mutated to a new integer value with some user defined mutation rate. Appendix B gives pseudocode for the mutation function.

We use two further mutation operators, based on those used by [6]. The first is fusion, which adds one codon to a random point in the genome. The second is elision, which removes one codon from a random point in the genome. Elision is applied only if the resulting genome has length greater than 0.

4) Default parameter values: All of these genetic operators are controlled via parameters. Parameters allow settings to be turned on and off, while controlling the degree to which an operator is used. For example, parameters give the percentage of individuals in the population that are replaced by crossover and the percentage of individuals which can used as parents from which to create children. An instance of the parameter file can be seen in appendix C.

D. Performance Issues

A quick initial experiment was run to check the speed of execution, and to enable estimates on the amount of time an experiment would run. Figure 4 shows the mean time taken over 10 random runs, with population sizes of 100. Note that:

$$\#\text{ cost evaluations} = \text{pop size} \times \#\text{ generations}$$

At 0.6 seconds per evaluation, 10,000 evaluations have a 41% increase in time per evaluation taken over 1,000 evaluations: the more evaluations we do, the longer the average time to perform an evaluation. This is because there is more opportunity for extremely long evaluations: 94% of evaluations occur in under 0.2 seconds, while 2% take in excess of 1.0 second. These long evaluations are ones involving a large number of variables in the production. Consider the example of applying the production $F \rightarrow F[+F]F$ to a word with 10 Fs in it. After one cycle the word has 30 Fs; after two cycles it has 90 Fs; after 3 cycles, 270 Fs, and so on. After n cycles, the number of Fs in a derived string equals

$$(\#\ F\text{s in the axiom}) \times (\#\ F\text{s in the production})^n$$

We do not want our processing time dominated by these potentially exponentially long evaluation-time individuals. Rather than limit the number of symbols allowed in a production (as done by [7]), we limit the maximum time allowed for parsing and rendering an individual. The default value used is 1.0 second. Once an individual reaches the maximum allowed time, it is terminated, and assigned a high cost value, as if it had rendered a blank image.

E. Experimental parameters

The costs of evolved individuals were measured. Algorithm parameters are factors affecting these costs. The experimental design used here changes parameter values (or combinations of parameter values) in isolation and keeps other parameters constant.

Experiments use the genetic operators in section II-C. Unless otherwise stated, all experiments use the default parameter values given in appendix C. To summarise these defaults, a total of 10,000 cost function evaluations were performed per run. For the experiments in section III-A and III-B a default population of 50 individuals was run for 200 generations. Results from further experiments (not reported here) resulted in the default values being revised to a population size of 40, run for 250 generations for later experiments. Mutation, elision and fusion rates are based on the values used in [6]; these values may not be optimal, but provide a sound basis for the investigations. A mutation rate of 5% per codon is used, with fusion affecting 25% of genomes and elision affecting 5% of genomes. The choices are made independently, so an individual's genome may be affected by any combination of mutation, fusion, and elision. The least fit 25% of the population are removed and the fittest 25% of the population are used as parents to generate new individuals. Initially elitism is set to 0. The effect of elitism is investigated in section III-B and subsequently the default value for elitism is set to 20% of the population size.

This experimental design exposes the effect that these particular parameter changes have on the GE outcomes; it does not necessarily find optimal parameter sets. Because we have not fully investigated the parameter space, we draw fairly cautious conclusions about our results. For example, we are not claiming that 20% elitism is necessarily the optimal value, but it does gives good results for the other default parameter values used.

Due to the stochastic nature of GE, and evolutionary algorithms in general, multiple runs of the same program (identical except for different random seeds) produce different responses. To ensure reliable statistics, a minimum of 50 runs were conducted for each experiment.

With the exception of section V-C all experiments aim to evolve the target tree shown in figure 1 after 4 iterations. The result of a run is the best individual at the last generation.

F. Statistical Analysis

We wish to investigate the effect of several factors on the GE process, such as the effect of elitism, or of changing the underlying BNF grammar in certain ways. To this end, we cast various null hypotheses, that the changes in fact have no effect, and attempt to refute these using statistical tests. We test whether the distribution of final best fitnesses over several runs (usually 50 or 100 runs) are significantly different at the 95% confidence level.

It is common to use a t-test to test whether two distributions differ. However, this test assumes that the underlying distributions are normal. As can be seen from later plots, our distributions are highly skewed, and so not normal. So we use non-parametric tests [8]. Non-parametric statistics include the median (and the corresponding interquartile range) of the

data, rather than the more usual mean and standard deviation (which are more suited to an underlying normal distribution).

We use notched box and whisker plots[2] to present the results of the experiments. We use the Wilcoxon rank-sum test (also known as the Mann-Whitney U test) to test whether the medians of two samples are significantly different, using Matlab's `ranksum` function. If the p-value is less than 0.05, we can say that the null hypothesis is rejected at the 95% confidence level because the medians differ. In cases where this test fails to reject the null hypothesis (so we cannot state that the medians differ significantly) we also perform a two-sample Kolmogorov-Smirnov (KS) test, to test whether the distributions of the two samples are significantly different in any way, using Matlab's `kstest2` function. If the p-value is less than 0.05, we can say that the null hypothesis is rejected at the 95% confidence level because the distributions differ in some way.

III. Basic Experiments

A. Can GE find a simple branched L-system?

The aim is to minimise the cost value, with a target cost of zero. Using the cost function in section II-B and the default parameter values detailed above, our GE approach successfully evolved a range of zero cost L-systems. These perfect solutions include:

- $F[+F]F[-F]F$
- $[F+F]FF[-F]F$
- $F[+F]F[-[F]]F$
- $[[[F]F[+]F]FF][F]F[[[F[F][[-F]]]+F]][+]FF-+$
- $F[[++]+[[F]]]F[-F]F$
- $F[F--+F][[[FF]-+]][+F][[-]]FF$

As is often the case in evolved solutions, some of these are non-intuitive. There is nothing in our cost function to encourage parsimonious solutions.

Imperfect solutions with cost values as high as 6433 or 8074 superficially look like the target structure (figure 5), but on closer inspection it becomes apparent that branches are slightly higher, lower, or longer, or display other subtle differences.

The higher the cost, the less the evolved structure looks like the target, although all are clearly approximations to it. (This also serves to validate our choice of fitness function.)

B. The effect of elitism

In order to establish the effect of elitism on the default parameters described in section II-E, we carried out nine experiments, using a different number of elite individuals, e, in each experiment: $e \in \{0, 1, 2, 4, 6, 8, 10, 12, 14\}$. All experiments comprised 50 runs. The distribution of data of each experiment is denoted D_e, for $e \in 0, 1, 2, 4, 6, 8, 10, 12, 14$.

[2]These plots illustrate the spread of values in a sample. A box is drawn between the quartiles Q_1 and Q_3, with a midline marking the median Q_2. Whiskers are drawn from the box to cover all the data within $3/2$ of the interquartile range $Q_3 - Q_1$. Data points outside this range (outliers) are shown by dots. If the notches of two boxes do not overlap, this indicates that their medians differ at the 95% confidence level (although a rank-sum test should be performed to test this).

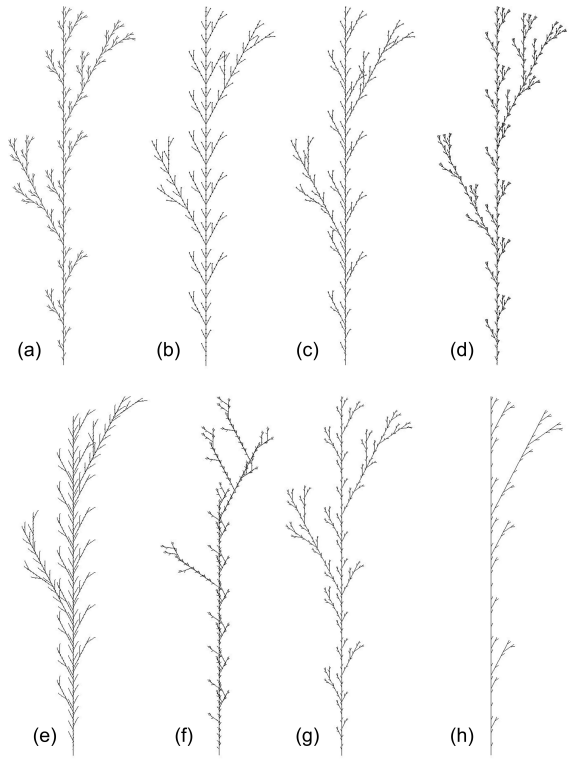

Fig. 5. Example evolved solutions and their costs:
(a) target, cost = 0; $F[+F]F[-F]F$
(b) 6433; $FFF[[FFF[[F]-F]]+F+-]$
(c) 8074; $FF[[F[[+][F]]F[-F-]F][+F]][F]F$
(d) 29240; $FF[+FF]FF[-[F]-[+F[F][-]+[F+][+]+]+]F[F]$
(e) 52056; $F[FF[[F[+F]FFFF-F--]]]FF$
(f) 66009; $FFFFF[[FF-[+]F[F][+[+F]][F++F]]][+-++F]$
(g) 137734; $[[F]][F]F[FF-F][F[F]F][-]F[+F][-]F[F]$
(h) 257518; $F[[F-[F]]]FF$

The null hypothesis H_0 is: *elitism has no effect on the evolution; that is D_i and D_j (for $i \neq j \in 1, 2, 4, 6, 8, 10, 12, 14$) have the same distributions.*

Figure 6 shows the box plots. The rank-sum test results are shown in figure 7. The case of no elite individuals is significantly different from all other cases, and one elite individual is significantly different from 6 or more elites, all at the 95% level. The KS test (figure 8) marginally increases the number of significant differences.

These tests may suggest three different classes of elitism: no elites, a few elites, and several elites. The box plots reveals that more elitism is helping. With no elitism ($e = 0$) more than 75% of L-systems produced are 'sticks'. (In this case, the best cost values seen throughout all generations are significantly better than these final best solutions. This suggests elitism is required to ensure that good solutions are kept and built on, not lost or mutated in the next generation.) Using the interquartile range in figure 6 we can see that when elitism is 6 or more, 25% of the runs result in a close match to the target tree.

Following these experiments, elitism is incorporated into the default parameters, with elite individuals representing 20% of the population.

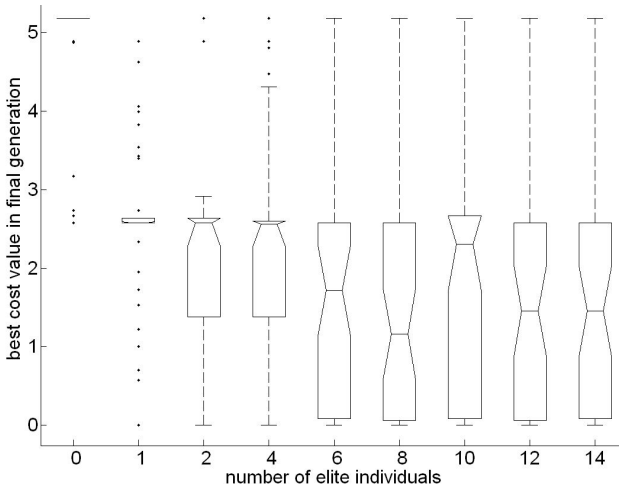

Fig. 6. Best cost values of final generation (divided by 10^5), against the number of elite individuals (50 runs each)

	1	2	4	6	8	10	12	14
0	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
1		0.174	0.059	**0.000**	**0.001**	**0.016**	**0.001**	**0.001**
2			0.648	0.054	**0.040**	0.306	0.055	0.058
4				0.154	0.099	0.596	0.145	0.154
6					0.697	0.544	0.856	0.902
8						0.360	0.819	0.747
10							0.496	0.555
12								0.943

Fig. 7. Rank-sum test for elitism, p values. Bold values reject the null hypothesis at the 95% confidence level.

	1	2	4	6	8	10	12	14
0	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
1		0.199	**0.041**	**0.006**	**0.000**	**0.006**	**0.003**	**0.003**
2			0.935	0.452	**0.022**	0.308	0.123	0.123
4				0.452	**0.022**	0.308	0.123	0.123
6					0.625	0.801	0.935	0.935
8						0.625	0.935	0.801
10							0.935	0.935
12								1.000

Fig. 8. KS test for elitism, p values. Bold values reject the null hypothesis at the 95% confidence level.

IV. BNF Grammar Based Experiments

[9] say that the structure of a GE algorithm's BNF grammar can have a significant effect. Here we explore this claim in the context of evolving L-systems. All experiments use the default parameter values, and are each repeated for 100 runs. Only the BNF grammars are changed between experiments. The default BNF grammar (figure 2) is used as the control grammar, $g0$. Each run returns the best cost value in the final generation.

A. Order of productions

The default BNF grammar builds lists (of productions, of symbols) from left to right. Here we test whether building lists from right to left is any different. The revisions to the default grammar are:

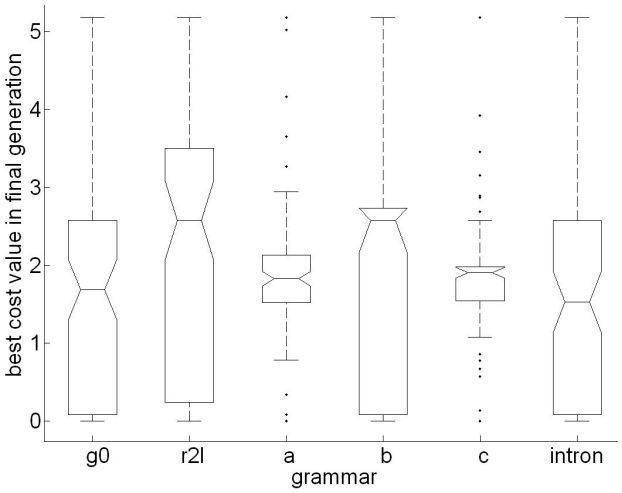

Fig. 9. Best cost values of final generation (divided by 10^5), for different grammar modifications, 100 runs

```
<productions>
  ::=  <lhs> <str>
   |   <productions>
       <lhs> <str>

<str>
  ::=  <symbol>
   |   <branch>
   |   <str><symbol>

<branch>  ::=  [<str>] | <str>[<str>]
```

The null hypothesis H_0 is: *the revised $r2l$ grammar and default grammar $g0$ produce the same distribution.*

Figure 9 shows the box plots for the default grammar $g0$ and the revised order grammar $r2l$. The rank-sum test $p = 0.020$ so the null hypothesis, H_0, is rejected at the 95% confidence level. This effect may be due to the left-to-right order of expanding the productions in the GE process.

B. Size of the Grammar

Here we test whether the number of productions in the grammar affect the solutions. The default grammar treats terminal symbols used for rotation ($\pm$) separately from terminal variables. So the default grammar allows a terminal symbol in a production to be a variable with a probability of 0.5 and to be rotation symbol with probability 0.5 (governed by the parity of the relevant codon). With only one variable compared to two rotation symbols it could be argued that the default grammar has a bias towards the variable over individual rotation symbols.

This test combines productions in the default BNF grammar to produce a shorter BNF grammar with more options for each production. Three shorter variations of the default grammar are tested. The revised grammars are:

a Combine all terminals into the symbol production

```
<lhs>  ::=  <symbol>
<symbol>  ::=  F  |  +  |  -
```

b Combine the branch production into the str production

	g0/a	g0/b	g0/c	a/b	a/c	b/c
W	0.418	0.252	0.537	0.751	0.896	0.595
KS	**0.000**	0.344	**0.000**	**0.000**	0.794	**0.000**

Fig. 10. Wilcoxon rank-sum test, and KS test, for different grammar sizes: p values. Bold values reject the null hypothesis at the 95% confidence level.

```
<str>
  ::=  <symbol>
   |   [<str>]  |  [<str>]<str>
   |   <symbol><str>
```

c Combine revisions 1 and 2

```
<lhs>  ::=  <symbol>
<str>
  ::=  <symbol>
   |   [<str>]  |  [<str>]<str>
   |   <symbol><str>
<symbol>  ::=  F  |  +  |  -
```

The null hypothesis H_0 is: *the revised grammar X_g ($g \in \{a, b, c\}$) and default grammar $g0$ have the same distributions.*

Figure 9 shows the box plots for default $g0$ and revised grammars a, b, and c. Figure 10 shows the p values for the rank sum test, and the KS test. Although the medians are not shown to be significantly different, the KS test nevertheless shows that some distributions are significantly different.

C. Introns

The default BNF grammar defines a way to create matching brackets, and thus syntactically legal individuals, in the successor of a production. When a set of brackets are to be inserted (with one or more symbols inbetween) into the L-system grammar, two options can occur: the L-system word terminates with the brackets, or the brackets are followed by one or more symbols. In biology an intron is a segment of a gene which is removed before forming the eventual gene. In GE, an intron is used to skip over a codon in the genome and performs no operation to the derivation of the grammar. This experiment alters the design of the BNF grammar, and introduces introns to the structure of bracket. A bracket is preceded and succeeded by zero or more symbols, dependent on the value of the codon. If the codon is odd, then an intron is used before/after the brackets so that no symbols are added.

The aim of this experiment is to discover whether introns, used in this context, help build 'useful' grammars which converge on a solution quicker and find more solutions. The revised intron grammar is:

```
<str>
  ::=  <symbol>
   |   <symbol><str>
   |   <pstr>[<str>]<pstr>

<pstr>  ::=  <str>  |  ' '
```

The null hypothesis H_0 is: *the revised intron grammar and default grammar have the same distribution.*

Figure 9 shows the box plots for the default grammar $g0$ and the intron grammar. The rank-sum test gives $p = 0.641$, and the KS test gives $p = 0.961$. Thus the null hypothesis H_0 is not rejected at the 95% confidence level and we cannot conclude that introns help the evolution.

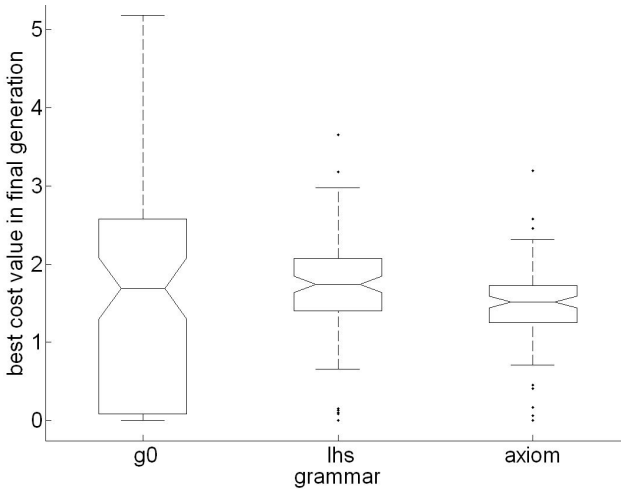

Fig. 11. Best cost values of final generation (divided by 10^5), for grammars that evolve the LHS and axiom, 100 runs

	g0/lhs	g0/ax	lhs/ax
W	0.917	0.311	**0.001**
KS	**0.000**	**0.000**	**0.001**

Fig. 12. Wilcoxon rank-sum test, and KS test, for evolving the LHS and the axiom: p values. Bold values reject the null hypothesis at the 95% confidence level.

V. Experiments Evolving L-system Components

A. Evolving the LHS

The default BNF grammar specifies that the LHS of a production must be a variable, and hence can only be F. This experiment extends the grammar so that a LHS can be any terminal symbol $(F, +, -)$. The aim is to discover what effect increasing the search space by allowing productions to be evolved has on the frequency and speed of convergence of solutions. The extended grammar is:

```
<lhs>  ::=  <symbol>
```

The null hypothesis H_0 is: *the extended grammar and default grammar have the same distribution.*

Figure 11 shows the box plots, and figure 12 shows the tests. A rank-sum test does not reject the null hypothesis, but the KS test does reject it, at the 95% confidence level.

Inspection of the box plots shows that the extended grammar finds far fewer good solutions that the more constrained grammar, but also finds rather fewer very poor solutions.

Inspection of the L-systems produced by individuals which are solutions, reveals that only Fs appear as a LHS in the grammar. Thus although the search space has been increased, it does not appear that new potential solutions have been found that would not have existed with the default grammar.

B. Evolving the Axiom

To simplify the derivation of L-systems, the axiom of previous tests is assumed to be F. This experiment considers whether a revised BNF grammar will allow the algorithm to evolve the correct axiom from the full set of terminal symbols.

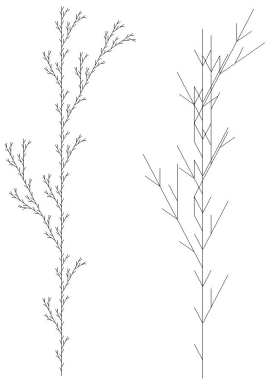

Fig. 13. Evolving the axiom

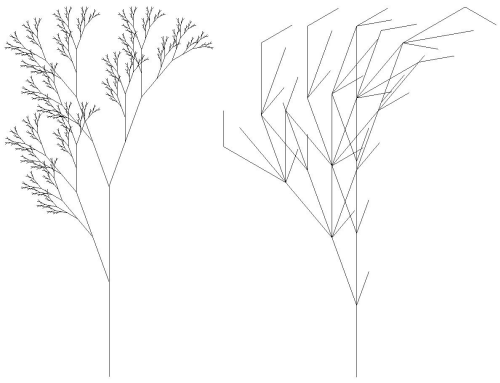

Fig. 14. Evolving L-systems containing two variables.

The grammar is adapted to allow the axiom to be any (single) terminal symbol. Additionally the LHS of a production is defined to allow any terminal symbol, to allow the system to recover if the 'wrong' axiom is used.

```
<axiom>  ::=  <str>
<lhs>  ::=  <symbol>
```

The null hypothesis H_0 is: *the new grammar and default grammar have the same effect.*

Figure 11 and figure 12 show the results. The KS test rejects the null hypothesis at the 95% confidence level. Inspection of the box plots again shows that this extended grammar finds far fewer good solutions that the more constrained grammar, but again also finds rather fewer very poor solutions.

The algorithm can evolve the axiom. Out of the 200 runs, 2 runs returned 'perfect' individuals that drew the target tree. Both solutions identified the correct axiom, F, with respective productions:

$F \rightarrow F[[+[F]]]F[-[F]]F$
$F \rightarrow F[+F]F[-F]F[[-]-]$

Some individuals produced similarities towards the target tree for a different axiom. The best such individual had a cost value of 44893, using axiom $+$ and productions:

$+ \rightarrow F-$
$F \rightarrow F[+F][F][F[F]-]F$
$- \rightarrow [[[-]]F[-[-]F]F] - ++$

Figure 13 shows the rendered version of this L-system. It is clear that although the L-system shares the same basic shape as the target tree, the individual evolved lacks detail. This individual has a genome length of 437 codons. The average genome length of solutions found in this experiment was 145.

C. Evolving More Complex Trees

The final experiment examines whether the techniques discussed above can be applied to larger and more complex problems. All trees evolved thus far involve only one variable terminal symbol. This section aims to evolve a target tree involving two terminal variables, F and X.

```
<axiom>  ::=  <variable>
<variable>  ::=  F  |  X
```

The target L-system for this experiment can be seen in figure 14a. This target tree is defined with axiom X, an angle of rotation of 20°, with the L-system derived for 7 iterations. The target L-system uses two productions:

$X \rightarrow F[+X]F[-X] + X$
$F \rightarrow FF$

Introducing more variables increases the size of the search space since there are a greater number of potential L-systems that can be produced from the grammar. To allow for greater exploration of the increased search space the population size was increased to 50 with 1000 generations used. In total 50,000 cost evaluations were run, compared to the default of 10,000. All other parameters used the default values. Over 200 repeat runs did not generate any perfect solutions. The best individual produced is shown in figure 14b, evolving axiom X and productions:

$X \rightarrow XFF$
$F \rightarrow [[[[+][X] + X[[--F]]]]]$

This best individual clearly has the correct shape, but lacks precise detail.

In response to these results the population was doubled to 100 individuals, and the number of generations increased to 2000, leading to a maximum of 200,000 cost evaluations to be performed. Due to the increased time required to run each experiment (several CPU-days), only 30 repeat runs were performed. The revised experiments returned two perfect solutions, found on average in generation 891. Both solutions evolved axiom X, with one evolving the production set:

$X \rightarrow F[F[+X]-[+]X][[[+X]]F]+$
$F \rightarrow FF$

and the other evolving the productions set:

$X \rightarrow [F]F[+X+]F[+X+]-[[-+]X]$
$F \rightarrow FF$

Once again this highlights how the size of the population

and number of generations the GE algorithm is able to explore the search space is a decisive factor in finding solutions.

VI. Conclusion and Further Work

We have successfully used GE to evolve simple L-systems. In our evolutionary set-up, elitism provides a significant advantage to finding solutions. We have performed simple experiments in altering the BNF grammar being used, and found certain alterations have a significant effect on the results, in line with the experience of [9].

[5] state that GE needs applying to larger scale problems with real-world applications. Here GE has struggled to derive simple two-variable L-systems for the parameter settings we used.

We have also successfully evolved the number of iterations (not reported here). Future work should investigate evolving the branching angle, and multi-symbol axioms. This will likely involve more generations and bigger population sizes.

More sophisticated L-system grammars should be evolved. In particular, context sensitive, parametric, and stochastic L-system grammars should be investigated. For stochastic L-systems, a more sophisticated cost function would be required, to assess statistical similarity of evolved and target structures.

Acknowledgements

We thank Tim Clarke for some early interesting discussions of this work. We thank Simon Poulding for a detailed discussion of the statistical tests.

References

[1] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer, 1996.

[2] A. L. Szilard and R. E. Quinton, "An interpretation for D0L systems by computer graphics," *The Science Terrapin*, vol. 4, pp. 8–13, 1979.

[3] P. Prusinkiewicz, "Graphical applications of L-systems," *Proceedings of Graphics Interface '86 / Vision Interface '86*, pp. 247–253, 1986.

[4] P. Worth and S. Stepney, "Growing music: musical interpretations of L-systems," in *EvoMUSART workshop, EuroGP 2005, Lausanne, Switzerland, March 2005*, ser. LNCS, F. Rothlauf *et al.*, Eds., vol. 3449. Springer, 2005, pp. 545–550.

[5] M. O'Neill and R. Conor, *Grammatical Evolution : Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

[6] A. Ortega, A. A. Dalhoum, and M. Alfonseca, "Grammatical evolution to design fractal curves with a given dimension," *IBM J. Res. Dev.*, vol. 47, no. 4, pp. 483–493, July 2003.

[7] B. Runqiang, Y.-P. P. Chen, K. Burrage, J. Hanan, P. Room, and J. Belward, "Derivation of L-system models from measurements of biological branching structures using genetic algorithms," in *IEA/AIE*, ser. LNCS, T. Hendtlass and M. Ali, Eds., vol. 2358. Springer, 2002, pp. 514–524.

[8] G. W. Snedecor and W. G. Cochran, *Statistical Methods*, 8th ed. Iowa State University Press, 1989.

[9] C. Ryan and M. O'Neill, *How To Do Anything With Grammars*. AAAI, July 2002. [Online]. Available: http://www.grammatical-evolution.org/gews2002/howto.ps

Appendix

A. Cost function pseudocode

```
cost := 0

for each canvas {
  for each pixel in canvas {
    if canvas[pixel] is ON {
      cost := cost +
        findNearestPixel(pixel,otherCanvas)
    }
  }
}
return cost

function findNearestPixel(p,oC) {
  width,height := size of oC

  for d = 0 to max(width,height) {
    for each otherPixel at dist d from p
      if oC[otherPixel] is ON
        return d
  }
  return max(width,height)
}
```

B. Mutation pseudocode

```
for each codon in a genome {
  x := randfloat( 0 .. 1 )
  if x < mutationRate {
    codon := randint( 0 .. maxCodonValue )
  }
}
```

C. Default parameter settings

```
### GE settings ###
populationSize = 50 (later, 40)
numGenerations = 200 (later, 250)
  # hence 10,000 evaluations
numberOfRepeatRuns = 50 (later, 100)

### mutation settings ###
mutationRate = 0.05
fusionRate = 0.25
elisionRate = 0.05

### selection settings ###
crossoverRate = 0.25
  # proportion of worst individuals
  # to remove
crossoverFromRate = 0.25
  # proportion of best individuals
  # to use as parents

### elitism settings ###
elitism = 0 (later, 20)

### Genome Information ###
initialNumberOfCodons = 15
maxCodonValue = 255
maxWrap = 15

### Individual Settings ###
axiom = F
numCycles = 4
angle = 27.5
```