Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

Evolving Families of Designs Using L-Systems

Elena Popovici
epopovic@gmu.edu

Technical Report GMU-CS-TR-2005-8

Abstract

Evolutionary computation has proven its utility in automating the process of engineering design. However, little attention has been paid to the scalability of generated designs, which is an important issue. This paper addresses this issue and proves the viability of evolving families of designs using parameterized L-Systems as a representation. The rest of the paper is organized as follows: first, an introduction as to why scalability is important and difficult; second, a review of existing work on evolving L-Systems; the third section contains a description of the application domain used for this feasibility study, details on the L-Systems and the EA used; section four presents experiments conducted and their results; the paper ends with a discussion, drawing conclusions and setting goals for future work.

1 Introduction

Present engineering design has moved significantly towards automation by computers and evolutionary computation (EC) is a popular technique for generating new designs. Evolutionary design is already a field in itself. One of the reason for the success of using EC in design is its ability to generate novel designs, and designing is a lot about creativity.

However, designing is also about scalability and this can become an issue for complex designs with multiple interdependent components. Proportionally scaling everything isn't always an option and it may not be obvious which components should be scaled and how and what the impact on the whole will be. Consider for example a table design: suppose you want a new similar table with a bigger surface. Simply stretching the original to enlarge the surface is not a good idea, as resistance in the middle will decrease. Scaling on all dimensions will make a table that is also higher and maybe that is undesired. Stretching the surface and adding additional legs / supports might be required. If evolution was used to generate the initial design, one alternative is re-running the process and selecting for bigger surface. However, this has the major drawback that, since evolutionary algorithms are stochastic processes, it is very likely that the obtained design will bear no similarity with the old one.

The solution proposed here is to use EC for generating families of designs rather than individual ones. The idea is to evolve parameterized design builders – programs for constructing designs that take input parameters signifying desired values for some features of the generated designs. The evolution process is run only once and the output program is stored. When a new design from that particular category is needed, the program is run with the new required parameters. For different input values such design builders might have to apply different design techniques. They might have to scale different components at different rates, not scale some at all or add new components.

Judging from existing work in the evolutionary design field, using parameterized grammars as a representation for this kind of task seemed highly promising. This is because the language that a grammar produces is strongly biased by its rules. That is to say the words of the language have similar properties and structures as a result of the common underlying rules.

This very regularity is what makes grammars suitable for design. One can give grammar symbols meanings either directly in a design representation space, or in a design-construction actions space. The result is a language whose elements are directly designs or are mapped into designs. Languages produced by grammars are often infinite and very large in size. Therefore, the result is a big number of different designs that nonetheless have similar features. Enhancing grammars with parameters bears the hope of using them as generators for families of designs.

The particular kind of grammars used in this work is L-Systems, introduced by Aristid Lindenmayer in [1]. The next section summarizes previous work on evolving L-Systems.

The issue of evolving families of designs with L-Systems has also been investigated by Hornby in [17]. However, the idea and some of the work for this study originate before Hornbys publication. Additionally, this paper touches some issues which were not discussed in [17].

2 Background

L-Systems have been successfully used as a representation for evolutionary algorithms (EAs) for many problem domains, from plants to robots or architecture. EAs and L-Systems seem to be a very good match. The following presents a summary of previous work on evolving L-Systems, grouped by domain areas.

Lindenmayer introduced L-Systems with the purpose of modeling plant structure and growth. He did not use evolution to discover L-Systems that produce desired structures or growth patterns, rather his studies were based on hand designed systems. But following in his footsteps, others (Ochoa [2], Jacob [3-5], Mock [6]) did use EAs to find L-Systems that would generate artificial vegetation, either in 2D or 3D, using selection criteria based on measures such as dimensions, surface, stability, symmetry, branching, etc.

More practical applications of generating structures can be found in the area of engineering designs. L-Systems were used to evolve tables, robot morphologies (Hornby [7]), flytraps, wind tunnels and floor plans for living spaces (Broughton, Coates, Jackson [8-10]).

For computational applications, evolving neural networks has resorted to using L-System representations in quite a number of application domains: the encoder-decoder problem for which Kitano in [11] used an interesting approach of having the symbols of the L-System be matrices; character recognition tasks (Boers and Kuiper [12, 13] – terminals of the L-System turn into neurons); the odd-n-parity problem (Hornby [7]); control systems for robots/artificial life (Mautner and Belew [14, 15]).

Combining structure and control lent itself to evolving L-Systems as well, as can be seen in [7]. Another application which doesnt fit very well in any of the above categories but is none the less interesting is the evolution of L-Systems for modeling blood vessels in the human retina presented by Kokai et. al. in [16].

Why could L-Systems be used in so many different areas? First of all, because they are defined at an abstract enough level, they allow a large number of distinct ways of instantiation. The symbols of the L-System can represent anything. There are two main approaches: 1) the symbols represent actual components of the resulting object; and 2) the symbols represent commands that specify how an object should be constructed. It should be noted that while the general definition of grammars distinguishes between terminals and non-terminals in the symbols set, L-Systems sometimes dont. Some of the symbols that serve as non-terminals through the iterative rule application process can be considered terminals in the resulting string.

The most popular in the second category is what is known as turtle graphics (a set of commands that move a drawing head (turtle) in a 2D or 3D world. The mark left in the space by the turtle head is considered as the resulting object. All of Ochoa [2]; Jacob [3-5]; Mock [6]; Hornby [7]; Kokai, Vanyi, Toth [16]; Broughton, Coates, Jackson [8-10] use this mechanism. The last of these mentioned works is more special as the world of the turtle is an isospatial grid. Another example of symbols representing commands can be found in [7], where the terminals of the L-System are interpreted as commands for edge encoding (a technique for constructing neural networks introduced in Luke [18]).

L-Systems whose symbols map into parts of the resulting object can be found in the work of Kitano [11] (symbols are matrices representing neural network connectivity), Boers and Kuiper [12, 13] (symbols are network nodes and edges), Mautner and Belew [14, 15] (symbols are rectangles with patterns for the network nodes and edges).

Another reason for the success of L-Systems as an EA representation is the fact that they are a modular concept and each of their components can be customized / enhanced as described in the following to provide the three main features that are responsible for computational power ([19]): **combination** through hierarchical construction of more powerful constructs from simpler ones, **control flow** through iteration and conditionals and **abstraction** through labeling of components for further reuse and parameters. The following reviews the most important features of L-Systems and how they have been used for EA representation. Further details on all these traits can be found in [1].

Symbols of an L-System can have parameters or not. The simpler case, when they dont, is still powerful enough to have been used with good results as an EA representation in [2], [6], [8-10], [12, 13], [11], [14, 15]. Allowing for parameters gives additional strength by facilitating the use of conditions based on the values of the parameters for deciding which rules to apply. The concept has been used in evolutionary systems like those presented in [7] and [16].

Bracketing is a popular concept for L-Systems, in particular those in which the symbols are build commands, allowing the builder to save / recover its state to / from a stack. Most evolved L-Systems are bracketed ones ([2], [3-5], [6], [7], [8-10], [12, 13], [16]), but there are examples of non-bracketed as well ([14, 15], [11]). Additionally, [7] uses repetition, by grouping symbols in the successor in a block and specifying how many times the block should be repeated.

Context sensitivity is another way of providing conditionals, by replacing a symbol with different strings depending on the symbols around it (its context). It becomes fairly complicated to use however when used in combination with bracketing and it is rather seldom used in EAs ([3-5], [12, 13]).

Lindenmayer introduced stochastic L-Systems, in which rewriting rules are chosen based on certain probabilities. To my knowledge however, all L-System based evolutionary systems to date are deterministic. Statistic L-Systems could however potentially be useful for generating similar yet different individuals (e.g. plants of the same species).

Once deciding which of the above features to use for the to-be-evolved L-Systems, there are still some parameters to set and deciding whether its better to fix their values (and if so, to what?) or to evolve them is not always obvious.

One such parameter is the number of symbols. While the number of terminals is domain dependent and generally easy to decide based on desired functionality, the number or non-terminals raises harder questions to answer. How many are enough for covering the search space? Using too few could cut out interesting solutions. Using too many could slow down the search. Evolving the right number of non-terminals could be a solution, however not put in practice yet, as there are a number of issues associated with it as well. Turtle graphics based L-Systems usually have one symbol (called F) that advances the turtle head and additionally serves as a non-terminal plus at least one other terminal for rotating the turtle (generally 2, denoted by + and -) ([2], [8-10]). [6] uses one additional symbol that serves only as non-terminal (and F is used only as terminal). Other work has more than 2 non-terminals, but their number is fixed ([3-5], [7], [12, 13], [14, 15], [16], [11]).

Deciding the number of rules poses the same too many / too few questions. Of course, there should be at least one rule per non-terminal. [2], [6], [8-10], [16] adopt the simplest solution of having a single rule (either because they are non-parametric with a single non-terminal or, like [16], which has more than one, but only one is evolvable). [3-5], [11] and [14, 15] have one rule per terminal and [7] has several condition→successor pairs per non-terminal and the first one whose condition is satisfied is applied. [12, 13] choose to evolve the number of rules.

For how many iterations should an L-System be run? Some systems ([3-5], [14, 15]) run it either until a maximum number of either iterations or rule applications is reached or until there are no non-terminals left in the string. The number of iterations can also be evolved, like in [7], but some measures must be taken to prevent it from growing too much and thus affecting computation time.

The seed with which the L-System is started can also be fixed ([2], [6], [8-10], [16]) or evolved ([3-5]). [7] is a special case where the seed is fixed, but its parameters are evolved.

An original approach is taken by [16] which uses a GA to evolve the L-System production and evolutionary strategies to tune the parameters.

3 Evolving for Scalability

3.1 The Domain – What Is a Table?

The domain used to test the hypothesis was evolving design builders for 2D tables of customizable height. The goal was to evolve table builders that when given as input parameter a certain number would construct a table whose height is somehow related to this number. In particular, in the current experiments, the builders were selected for their ability to construct tables of height equal to four times the input parameter.

Quantifying the table-likeness of a 2D structure is not obvious. After preliminary experiments with various formulas, the measure for a structure s that was used for the results presented here was as described below unless otherwise specified: $tableness(s) = surface(s) * base(s) * stabilityX(s) * stabilityY(s)$, where:

- $surface(s) \in 1..W$: the number of pixels at the maximum height level
- $base(s) \in 1..W$: the width at the base level
- $stabilityX(s) \in [0, 1)$: $1-$(the absolute horizontal distance between the center of gravity of the structure and the center of the base)$/(base(s)/2)$
- $stabilityY(s) \in (0, 1)$: (the height of the center of gravity)$/height(s)$

. Tables were not allowed to be wider or taller than $H = W = 100$ pixels. If they exceeded these limits they were penalized and given a fitness of 0.

This is obviously not a perfect definition of what a 2D table is, so in the rest of this paper every time the word table is used one should really think stable 2D structure with big surface and wide base support. With respect to this definition, the evolved designs are fairly good.

3.2 The Representation – L-Systems Used.

Parameterized L-Systems with a turtle graphics interpretation seemed a good choice to represent such builders. The input parameter would be used as the first (and possibly only) parameter of the L-System seed. The selection pressure included a measure of the correlation between the input parameter and the actual height of the generated structure.

The L-Systems used were very similar to those in [7]. They had bracketing and repetition with no embedding. The number of non-terminals and their arity was varied across experiments and will be specified in the results section. For cases when the number of parameters of the seed was bigger than 1, values for the parameters starting at the second position were encoded in the chromosomes and evolved. For every non-terminal there were three condition $\rightarrow$ successor pairs and the first one whose condition was satisfied was applied. The L-Systems were initialized such that each successor would have anywhere between 2 and 6 blocks, each block with at most three symbols. Bracketed, repeating and simple blocks were generated with equal probability. Terminals and non-terminals were also generated with equal probability. There was no limit put on the size of the successors during evolution. The terminals used were $F(x)$ (move forward in the current direction x pixels), $+$ (change direction by rotating 90^o right), $-$ (change direction by rotating 90^o left). The expressions passed as parameters to the symbols of arity greater than zero were of type $operand_1 operator operand_2$, where $operand_1$ and $operand_2$ were either integer constants or variables (denoted by $x_0, x_1, \ldots$) and $operator$ was any of: $+ - * /$. The conditions were always of the type: variable $\geq$ constant. Integer arithmetic was used. Non-terminals were numbered and the first was always taken as seed. Figure 1 shows an evolved L-System whose graphical output can be seen in the second snapshot of Figure 2.

3.3 The Algorithm

Since the goal was to evolve design builders that generate tables of some specified height, the fitness had to include both a measure of the table-likeness (from hereon referred to as utility) of the produced structures and a measure of correlation between the desired height and the actual height (from hereon referred to as correlation). The utility measure used, $U(design)$, was described in a previous section.

Since these design builders can actually be considered functions and the aim was for generality, they had to be tested on several inputs and given a grade representing overall performance. Several issues arise from this. The first issue is how to combine the utility and the correlation from all tests into one number to be used as fitness of a design builder, db. Several approaches were tried, the one used for most of the experiments being: $fitness(db) = \frac{\sum_{i=1}^{n} U(design(db, input_i))}{n} + \frac{\sum_{i=1}^{n} corr(db, input_i)}{n}$, where n is the number of inputs and $design(db, input)$ is the design obtained by running the design builder db started with input $input$. Fitness is to be maximized. Where anything is different from this, it will be noted.

The second is how to compute correlation. Unless otherwise specified, in the results presented this was given by the following formula: $corr(db, input) = H - |actualHeight(design(db, input)) - desiredHeight(input)|$.

A third challenge is how many inputs to use for testing and what these inputs should be. Many inputs can give a better idea of overall quality of a design builder, but increase computational effort. And there is always the danger of over-fitting. For the experiments described in the next section, three input values were used: 5, 10 and 15, corresponding to desired heights of 20, 40 and 60.

Yet another thing to decide was for how many iterations to run each L-System before evaluating the generated structure. Two approaches were tried: one in which the number of iterations was fixed (to 5) across all individuals and all inputs and one where the number of iterations was encoded and evolved in each individual but still kept constant over all inputs.

Both mutation and crossover were used. Several types of mutation were given equal probabilities: perturb one of the conditions or the equation passed as parameter to a symbol (by replacing the operator with a new random one, replacing a variable with a new random one or slightly increasing / decreasing a constant), replace a symbol with a new random one (and random equations as parameters), add a new random block of symbols, delete a block of symbols, slightly increasing / decreasing the number of repeats of a block. When the number of iterations or values for the parameters of the seed were encoded in the chromosomes, two new mutation types were introduced that would slightly alter these values.

Crossover produced one child out of two parents by making a copy of the first parent and then replacing a block of symbols in one of the successors of a rule with a block from one of the successors of the corresponding rule from the second parent. When the number of itera-

$$P_0(x_0) \rightarrow \begin{cases} (x_0 \geq 1): & P_0(-1 - -1)F(-6/x_0)\{P_0(x_0 - -2)-\}(1)F(x_0 * -1) + -F(-12/x_0)F(-1 - x_0 \\ &)P_1(-4 * -1)F(x_0/ - 2)F(-1 - x_0)P_1(-2 - x_0) \\ (x_0 \geq -3): & \{P_0(x_0 - -2)-\}(1)P_1(-2 + x_0)F(3/ - 3)\{P_0(x_0 - -2)-\}(1) \\ (x_0 \geq 5): & \{P_0(x_0 - -2)-\}(1)P_1(-4 * -1)\{P_0(x_0 - -2)-\}(1)\{P_0(x_0 - -2)-\}(13) \end{cases}$$

$$P_1(x_0) \rightarrow \begin{cases} (x_0 \geq 24): & \{F(x_0 - 0)\}(3) + -P_1(2 - x_0)P_1(-3 * -2)F(x_0 + x_0)F(-3/x_0)P_0(x_0 - 3)P_0(2/ \\ & x_0)F(-4 * 3)P_1(x_0 + x_0)P_0(x_0 * x_0)F(x_0/x_0) - P_1(x_0/2)P_0(4 * x_0)P_1(3 - x_0)F(\\ & -4/1)P_1(x_0 - x_0) \\ (x_0 \geq 4): & P_1(0 + 3)F(x_0/x_0) + -\{F(x_0 - -1)\}(3) \\ (x_0 \geq 2): & P_1(1 - 1)P_0(0/x_0)P_0(2 - 2)P_0(x_0 - 3)P_0(2/x_0)F(-4 - 3)P_1(x_0 + x_0)P_1(1 - 1)P_0 \\ & (0/x_0)P_0(2 - 2)P_1(3 - x_0)F(-4/1)P_0(x_0 - 3)P_0(2/x_0)F(-17 - x_0) \end{cases}$$

Figure 1: Rules for the L-System that produces the designs in the second row of Figure 2.

tions or values for the parameters of the seed are encoded in the chromosomes, two new crossover types were introduced allowing to copy the number of iterations or a seed from the second parent.

A generational EA was used in all cases. Unless otherwise specified, its settings were as follows: non-elitism, tournament selection of size two, 0.3 crossover rate and 0.5 mutation rate. Population size was varied across experiments, with smaller numbers used for more computationally expensive L-System types. Runs were conducted for anywhere between 500 and 3000 generations. The EA behavior was observed in the GUI and runs were stopped at different points based on performance (fitness growth pattern, solution structure). Details will be provided for each case in the following section.

4 Results

Given that the purpose of this study was to show feasibility of an idea and not to compare methods, few runs (up to ten) were executed for each experimental setting. Best of run individuals were offline tested on three additional inputs to check for generality. Multiple settings were not meant to determine which is better or to investigate effect of various components, but to show that in all cases some acceptable results are produced.

The simplest configuration tested was with L-Systems with 2 non-terminals of one parameter each and the number of iterations fixed to 5 for all individuals. This setting was run with populations of 500 individuals. The photos in Figure 2 show some of the best of run individuals obtained. Generated designs are shown both for the three input values used for fitness computation (5, 10, 15) and for three additional input values (8 and 12 for checking interpolation and 17 for checking extrapolation). The framing rectangles show maximum allowed size of structures (one extra pixel on each side) and the red horizontal lines mark desired height for the respective input (one extra pixel down).

As can be noted in these images, the evolved design builders do a fair amount of generalization, although they are not perfect. In particular, they seem to have bigger problems with extrapolating. The reason for this can be that they seem to generate designs whose width is correlated with the height and as height increases, width increases as well up to the point where it goes outside the maximum allowed size.

A positive thing that can be observed is that the design builders successfully scale only some of the components of the structures, or they scale different components at different rates. Additionally, another desirable behavior is the one exhibited by the design builder in the first image, which is the ability to fully change the principle ruling the design when the desired size changes.

The same set of features can be observed for the other settings tried and shown below. A second setting uses an elitist EA with L-Systems with 10 non-terminals, still one parameter each, number of iterations fixed to 5 and population size of 500. Three best of run individuals are shown in Figure 3.

A third setting uses a non-elitist EA with L-Systems with 2 non-terminals, 2 parameters each. The value for the second parameter of the seed was encoded in the chromosome and evolved, as was the number of iterations. This EA was run with population size of 100. Sample best of run individuals are shown in Figure 4.

The first snapshot shows again ability to switch the design concept with increasing size (even twice in this case). All the other ones exhibit the feature of scaling only some design components while keeping others fixed and / or scaling different components at different rates. The last snapshot displays very good scaling, but a rather uninteresting structure.

The final setting uses a different fitness model. The average utility is multiplied with the average correlation. In addition, correlation is now computed differently and takes the form of either penalty or reward. When the actual and desired heights are different, the correlation is a penalty between 0 and 1 (0 means harder penalty – because it will be used in multiplication) obtained by di-

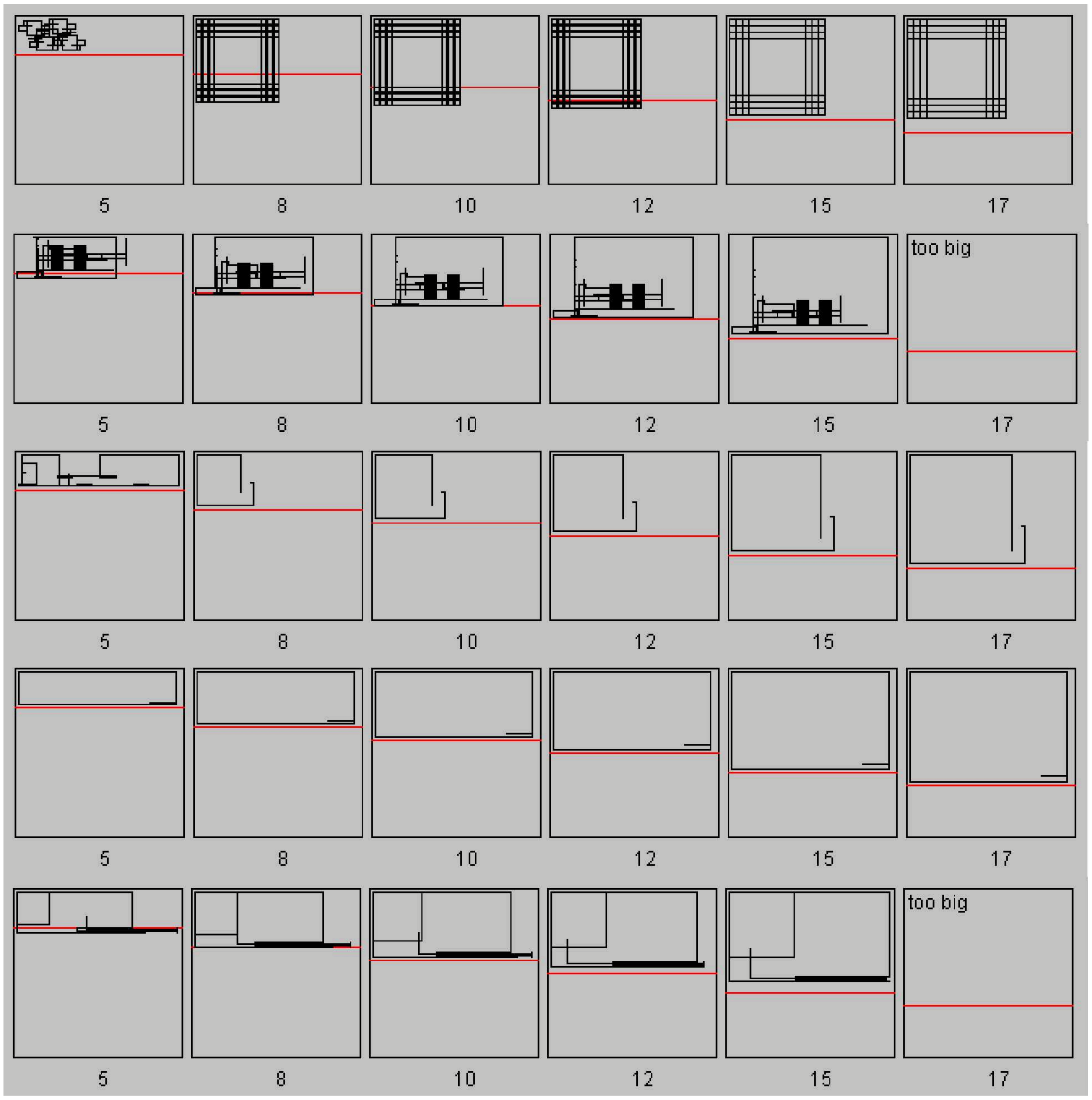

Figure 2: Five best-of-run for L-Systems with 2 non-terminals, 1 parameter each, fixed no. of iterations = 5. Non-elitist EA, tournament selection size 2, population size = 500.

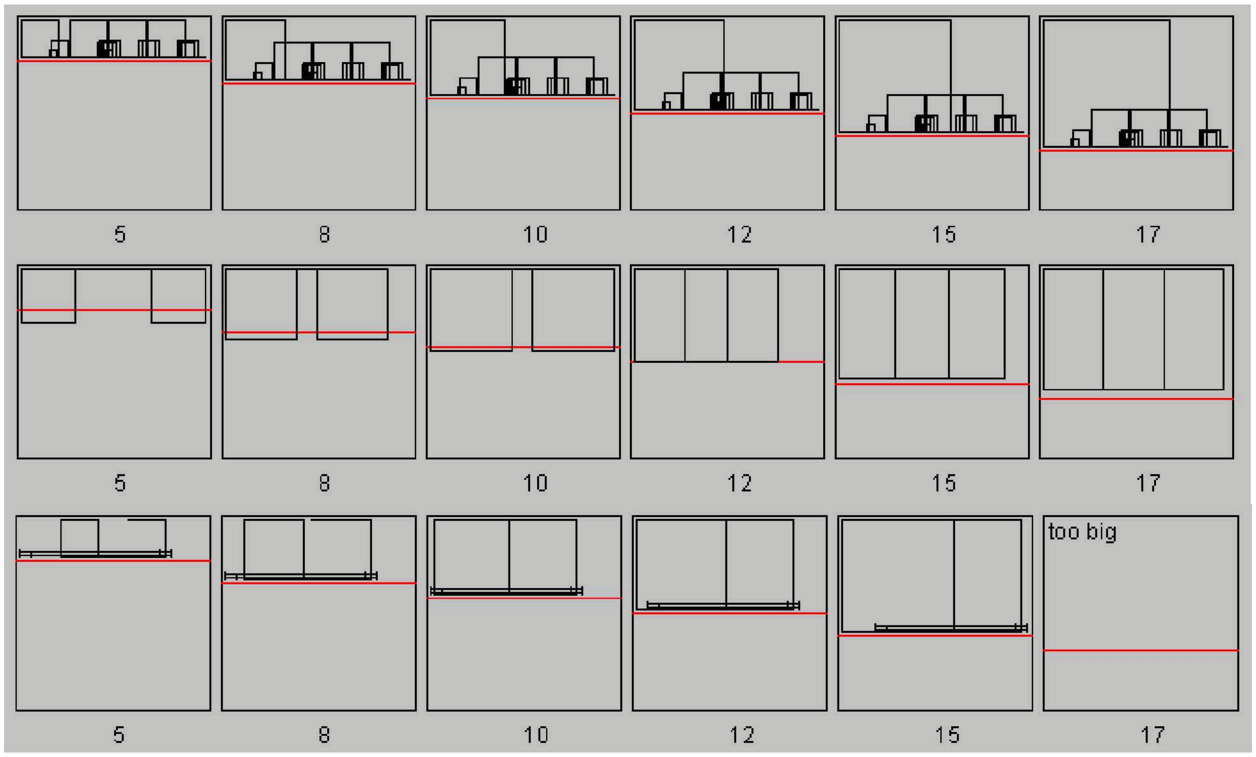

Figure 3: Three best-of-run for L-Systems with 10 non-terminals, 1 parameter each, fixed no. of iterations = 5. Elitist EA, tournament selection size 2, population size = 500.

viding the smaller of the two heights by the larger. When the heights are equal, the correlation is a reward equal to the actual height. L-Systems with 10 non-terminals, 1 parameter each, number of iterations fixed to 5 and an elitist EA with population size of 500 and fitness proportional selection with linear scaling were used. One sample best-of-run individual is shown in Figure 5. While the generated shape is quite interesting and esthetic, the scaling is rather imperfect.

5 Conclusions and Future Work

This paper pushes forward the idea of evolving families of designs and backs it up with experimental results that show that it is practical. Desired qualities of design builders are identified and the results show that they can be accomplished through evolution. Preliminary experiments identified issues as well as leads for further research. The proposed model has many knobs that can be tuned to allow for flexibility, but the effect of each of them needs to be studied more rigorously. For example:

- number of non-terminals;
- number of parameters;
- method of computing correlation;
- method of combining utility and correlation – Pareto optimization could be used;
- number of iterations – it could be made a function of the input parameter and have evolution discover the right function;
- number of input parameters – to control scalability for several features of the designs at once.

Bibliography

1. P. Prusinkiewicz and A. Lindenmayer. The Algorithmic Beauty of Plants. Springer-Verlag, 1990.
2. G. Ochoa. On Genetic Algorithms and Lindenmayer Systems. In A. Eiben, T. Baeck,M. Schoenauer, and H. P. Schwefel, editors, Parallel Problem Solving from Nature V, pages 335–344. Springer-Verlag, 1998.
3. C. Jacob. Genetic L-System Programming. In Y. Davidor and P. Schwefel, editors, Parallel Problem Solving from Nature III, Lecture Notes in Computer Science, volume 866, pages 334–343, 1994.
4. C. Jacob. Evolution Programs Evolved. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, Parallel Problem Solving from Nature PPSN-IV, Lecture Notes in Computer Science 1141, pages 42–51, Berlin, 1996. Springer-Verlag.

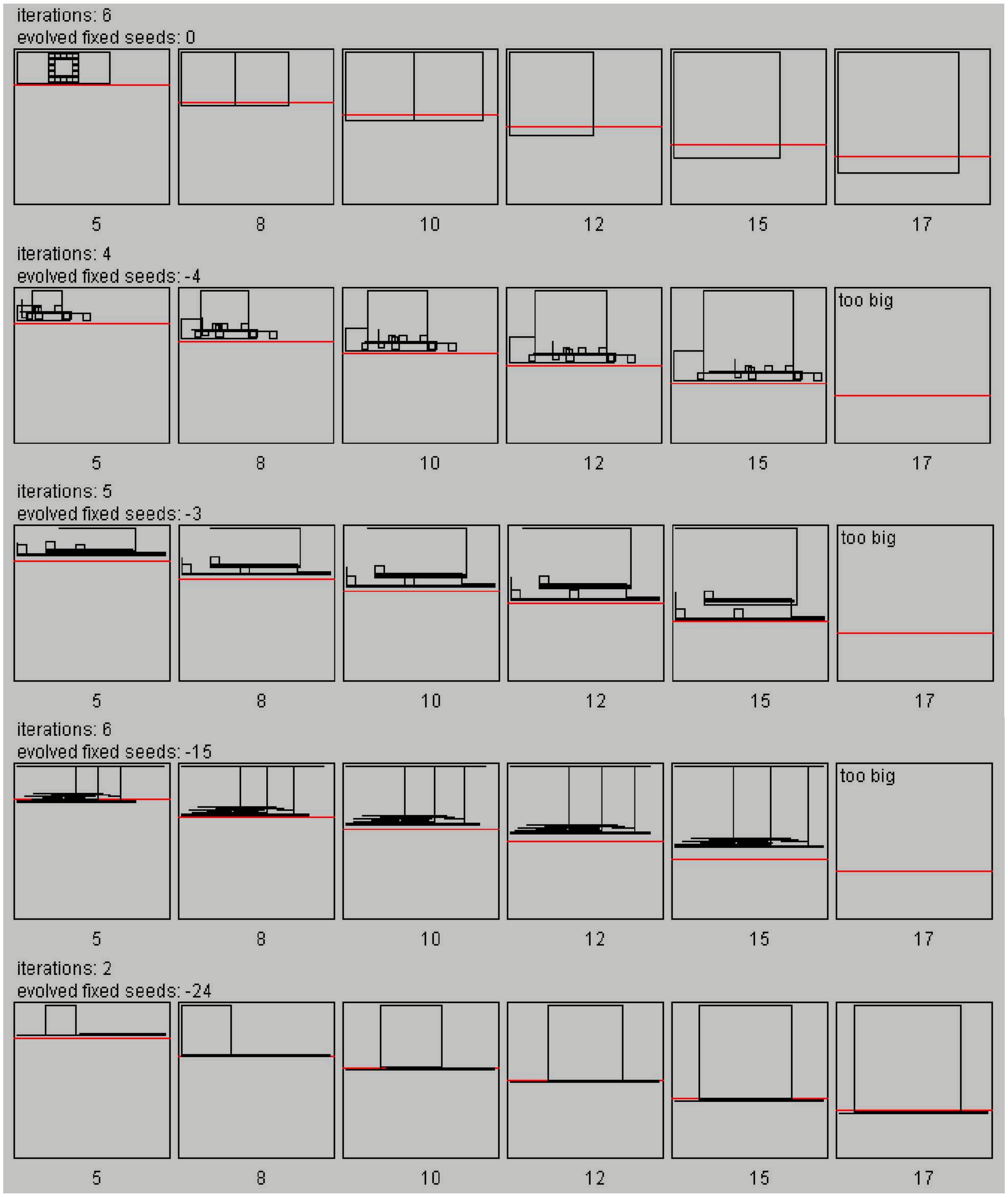

Figure 4: Five best-of-run for L-Systems with 2 non-terminals, 2 parameters each, evolved no. of iterations, evolved second seed parameter. Non-elitist EA, tournament selection size 2, population size = 500.

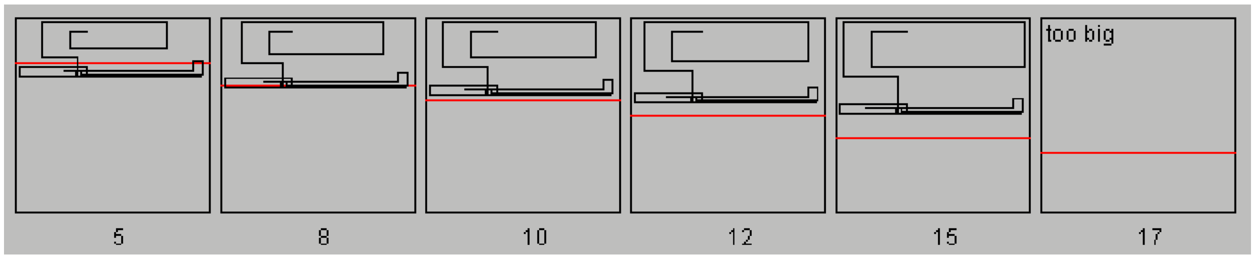

Figure 5: One best-of-run for L-Systems with 10 non-terminals, 1 parameter each, fixed no. of iterations = 5. Elitist EA, fitness proportionate selection with linear scaling, population size = 500. Fitness based on multiplication.

5. C. Jacob. Evolving Evolution Programs. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. Riolo, editors, Proceedings of the First Annual Conference on Genetic Programming, pages 107–115. Morgan Kaufmann, 1996.

6. K.J. Mock. Wildwood: The Evolution of L-system Plants for Virtual Environments. In Proceedings ICEC 98, pages 476-480. IEEE-Press, 1998

7. G. S. Hornby. Generative Representations for Evolutionary Design Automation. PhD Thesis, The Faculty of the Graduate School of Arts and Sciences, Brandeis University, Department of Computer Science, February, 2003

8. T. Broughton, A. Tan, and P. S. Coates. The Use of Genetic Programming in Exploring 3d Design Worlds. In R. Junge, editor, CAAD Futures 1997. Kluwer Academic, 1997.

9. P. Coates, T. Broughton, and H. Jackson. Exploring Three-Dimensional Design Worlds Using Lindenmayer Systems and Genetic Programming. In P. J. Bentley, editor, Evolutionary Design by Computers, 1999.

10. H. Jackson. Toward a Symbiotic Coevolutionary Approach to Architecture. In P. J. Bentley and D. W. Corne, editors, Creative Evolutionary Systems, chapter 11, pages 299–313. Morgan Kaufmann, San Francisco, 2001.

11. H. Kitano. Designing Neural Networks Using Genetic Algorithms with Graph Generation System. Complex Systems, 4:461-476, 1990.

12. E. J. W. Boers and H. Kuiper. Biological Metaphors and the Design of Modular Artificial Neural Networks. Master's thesis, Leidea University, the Netherlands, 1992.

13. E. J. W. Boers, H. Kuiper, B. L. M. Happel, and I.G. Sprinkhuizen-Kuyper. Designing Modular Artificial Neural Networks. In H.A. Wijsho, editor, Proceedings of Computing Science in The Netherlands, pages 87–96, SION, Stichting Mathematisch Centrum, 1993.

14. C. Mautner and R. Belew. Coupling Morphology and Control in an Evolved Robot. In Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiel, and Smith, editors, Genetic and Evolutionary Computation Conference, pages 1350–1357, 1999.

15. C. Mautner and R. Belew. Evolving Robot Morphology and Control. In M. Sugisaka, editor, Proc. of Articial Life and Robotics(AROB99), Oita, 1999.

16. Kokai, G., Vanyi, R., Toth, Z.: Parametric L-System Description of the Retina with Combined Evolutionary Operators. In Proc. GECCO, Genetic and Evolutionary Computation Conference July 13-17, 1999 Orlando, Florida, USA, Vol. 2 pages 1588-1596

17. Hornby, G.: Generative Representations for Evolving Families of Designs. In Proc. GECCO, Genetic and Evolutionary Computation Conference 2003, E. Cantu-Paz et. al. (Eds.), LNCS 2724, pp 1678-1689, Springer-Verlag Berlin Heidelberg 2003

18. S. Luke and L. Spector. Evolving graphs and networks with edge encoding: Preliminary report. In J. Koza, editor, Late-breaking Papers of Genetic Programming 96, pages 117–124. Stanford Bookstore, 1996.

19. H. Abelson, G. J. Sussman, and J. Sussman. Structure and Interpretation of Computer Programs. McGraw-Hill, second edition, 1996.