Graph Grammars as a Representation for Interactive Evolutionary 3D Design

James McDermott

EvoDesignOpt, CSAIL, MIT
jmmcd@csail.mit.edu

Abstract. A new interactive evolutionary 3D design system is presented. The representation is based on graph grammars, a fascinating and powerful formalism in which sub-graphs, nodes and edges are iteratively rewritten by rules analogous to those of context-free grammars and shape grammars. The nodes of the resulting derived graph are labelled with Euclidean coordinates: therefore the graph fully represents a 3D beam design. Results from user-guided runs are reported, demonstrating the flexibility of the representation. Comparison with results using an alternative graph representation demonstrates that the graph grammar search space is rich in appealing, organised designs. A set of numerical graph features are defined in an attempt to computationally distinguish between good and bad areas of the search space, leading to the definition of a computational fitness function and non-interactive runs.

1 Introduction

3D design is an interesting and difficult application area for evolutionary computation (EC). It has a very large number of real-world applications, including product design and branding, structural engineering, conceptual architecture, furniture design, sculpture and even artificial lifeform bodyplan design.

In this paper we describe a project in the area of exploratory and conceptual architectural design. In this area, one may draw three important distinctions. The **purpose** of the system may be either *optimisation* (e.g. of structural properties [1]) or open-ended *exploration*. Our system emphasises the latter since the search space includes a very wide of possible designs. It cannot, for example, be reduced to a parametric model. The **representation** may be *sculpting* or *structured*. Our representation is structured in that it is composed of elements which have individual identity and function. This is in contrast to (e.g.) representations using voxels [3], where the primitive elements are very small and do not have individual function. The distinction is analogous to that in 2D between pixel-based (e.g. [20]) and vector-based art (e.g. [9]). Finally the **fitness evaluation** may be entirely *aesthetic* and subjective (as in Dawkins' seminal *Biomorphs* [4]), entirely *computational*, or somewhere in between. In our system fitness is primarily aesthetic, but we also set out a framework intended to derive a computational measure of fitness.

P. Machado, J. Romero, and A. Carballal (Eds.): EvoMUSART 2012, LNCS 7247, pp. 199–210, 2012.

In the current work we are interested in designs composed of simple beams of uniform width and differing lengths, a domain which is sufficient to raise interesting representational issues and suitable for exploratory evolutionary design. We avoid obvious problems of beams "suspended in mid air" or failing to connect perfectly by basing our representation on graphs, i.e. sets of nodes and edges. With Euclidean coordinates attached to each node, a graph fully specifies a 3D beam design. The space of such graphs then contains an extremely wide range of valid and useful designs, in addition to many disorganised and ugly ones. One could run EC over the space of graphs using an edge-set or adjacency-matrix encoding, with additional numerical parameters for Euclidean coordinates: however such representations contain no bias towards structured, organised, architectural designs. Instead it is useful to introduce a bias using a *grammar*.

Visual grammars of various types have been widely used in design, and perhaps the best-known variation is the *shape grammar*, though truly general implementation is very difficult. In this paper we aim to solve of the problems of shape grammars by using instead grammars defined over the space of graphs. Such *graph grammars* retain many of the advantages of shape grammars.

The structure of the paper is as follows. The concepts of shape grammars and graph grammars, with motivation and previous work, are presented in Sect. 2. Our graph grammar representation is described in detail in Sect. 3. Experiments and results are in Sect. 4, and conclusions and future work are in Sect. 5.

2 Background

We begin by introducing the concept of shape grammars, leading to the motivation for graph grammars as a constrained alternative. Within each topic, reference is made to relevant previous work in EC and design.

2.1 From Context-Free Grammars...

The formal string-rewriting grammars of Chomsky [2], such as the context-free grammars, allowed the description and generation of formal languages, and led to the development of modern linguistics. Grammar-based EC has been successful in many domains, including in structural design [1].

2.2 ...to Shape Grammars...

Explictly inspired by this powerful formalism, Stiny and Gips [21] created *shape grammars*, an analogous formalism for the description and generation of shapes. Instead of rewriting strings of symbols, shape grammar rules add, delete, or edit shapes, which are formally defined in terms of points, lines, planes and solids. Shape grammars have been successfully applied for four decades in both the analysis of existing designs and the generation of new ones, where "design" is interpreted broadly to include architecture [14], fine art [13], and other areas.

Formalisms for design, graphics, and geometry are of obvious interest to the evolutionary design community, and in fact (constrained) shape grammars have been used in combination with EC, e.g. [16,19].

However, a significant obstacle to the use of shape grammars is that computer implementation is relatively difficult [7]. Although shape grammars have been in use for over 40 years, there is no complete and fully general computer implementation. This is in contrast to the world of textual grammars, where simple algorithms exist for deriving sentences from grammars. Such algorithms are general—they can work with any grammar provided as an input—and complete—they can perform the entire range of functionality implied by the formalism.

The lack of complete and general shape grammar implementations is not for the want of trying: in fact a large number of implementations have been created, e.g. [18,22] (see also `http://www2.mech-eng.leeds.ac.uk/users/men6am/men6am/WorkshopReport.htm`), but all are tied to a particular grammar, unable to apply some rules in situations where they are implied by the formalism to be applicable, or otherwise limited.

One of the well-known obstacles to computer implementation is the *subshape recognition problem*, illustrated in Fig. 1. In (a) a shape grammar rule is defined. The single arrow denotes rule definition. In (b), a surprising consequence (the double arrow denotes derivation): the original "K" shape contains infinitely many smaller copies of itself, any of which may be used as the left-hand side of the rule defined in (a). This is an obstacle to computer implementation both because the copies must be recognised, and because there must be some method of choosing to which copy the rule should be applied. *Emergence* [12] is a second well-known obstacle: it is difficult to recognise (for use as rules' left-hand sides) new shapes which "emerge" through the combination of multiple rules' actions.

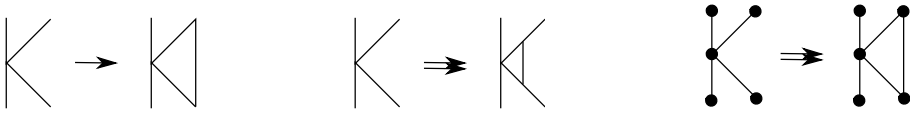

Fig. 1. Shape grammars (a) are sometimes more flexible than we want (b). Graph grammars (c) are more constrained.

2.3 . . . to Graph Grammars. . .

In response to the above problems, some researcher draw inspiration from a very different stream of research. *Graph grammars* [5], again inspired partly by Chomsky's original work, are a formalism originally of mathematical interest. Here the rules rewrite *graphs*, i.e. sets of nodes and edges. Graph grammars have been used in combination with EC for problems such as symbolic regression and circuit design [15].

Graph ideas have also been used to solve some of the problems of shape grammars. For example, Keles et al. [11] describe an implementable algorithm based on graph ideas (though they do not use the term "graph grammar"). Fig. 1(c) demonstrates that the existence of nodes allows the definition of a rule similar in intent to that of (a) but without the problem of subshape recognition. Edges in a graph grammar correspond to lines in a shape grammar. Thus, a graph in which nodes have associated Euclidean coordinates can serve as a representation for a beam or wire design in 3D space.

As in all grammars, graph grammar rules are composed of left- and right-hand side parts (LHS and RHS). Several different types of rules are possible, but in general the idea is to find, in the current graph, a sub-graph which matches (in some sense) the LHS of a rule, and then to replace that sub-graph with the RHS graph, gluing edges together in some predefined way. We use a simple algorithm for matching LHS sub-graphs and adding (never removing) RHS graphs. It is described fully in Sect. 3. Briefly, each rule is composed of two parts: a LHS predicate called a *selector* which, when applied, selects some nodes (and hence a sub-graph); and a RHS *action* which adds new material to the selected sub-graph. Each action includes an *envelope*, a function specifying a curve which can be used to set the Euclidean coordinates of any new nodes added.

2.4 . . . and Back Again

Our graph grammars consist of multiple rules, and each rule consists of a selector (LHS) and an action (RHS), the latter including an envelope. All three components are readily implementable directly as computer code, as described in the next section. Since CFGs can generate code, it is possible to define a space of graph grammars using a CFG; which brings us back to where the section began.

3 Representation and Implementation

Our graph grammars are composed of multiple rules. At each step, a sub-graph is selected (by a rule's left-hand side or LHS), and some new nodes and edges are added (by a rule's right-hand side or RHS). In addition to Euclidean coordinates, every node is labelled with two variables: the derivation time-step t at which it was added, and the index k indicating *which* rule added it.

For each rule in the grammar, the LHS is specified as a *selector*, that is a fragment of code which applies a predicate to the nodes of the graph. Some possible predicates include matching (on t) all nodes which were created in the previous step, or (on k) all nodes which were ever created by application of the current rule. It is also possible to match nodes which, for example, exceed a certain z-value, or exceed a certain distance from the $x = y = 0$ axis. The sub-graph on the selected nodes is returned.

The RHS is specified as an *action*, a fragment of code with a simple function such as adding a new node connected to an existing one. It assigns the new node's x, y and z values with reference to an *envelope*, a numerical function such as a

straight line, exponential curve, or sinusoidal curve in 3D space. Each action has one envelope. Since the envelope sets the coordinates of the new nodes, they are arranged into well-ordered lines and curves, as in Fig. 2. The action is performed *for each node* in the sub-graph returned by the selector.

Fig. 2. Action envelopes. New nodes' coordinates are set according to functions of t. In (a), x is linear, y is exponential; in (b), x is linear, y is sinusoidal.

This simple graph grammar algorithm has some useful properties. In addition to avoiding the problem of subshape recognition, it allows rules to be specified in quite a generic way. For example it is possible to define a rule whose LHS selects a cycle, and whose RHS adds and connects a new cycle of the same number of nodes, *without specifying that number*. See Fig. 3.

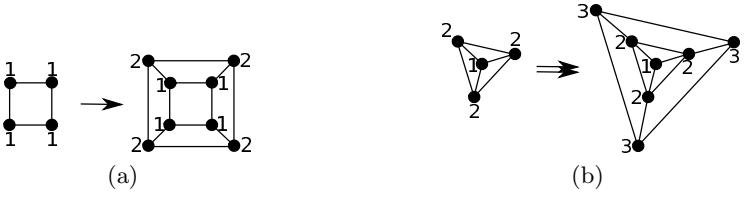

Fig. 3. A cycle rule. Nodes are labelled with t. The rule illustrated with four nodes in (a) is generic, capable of applying to a cycle of any number of nodes. In (b), it is applied to an existing graph, selecting just those parts which form a 3-cycle with the appropriate value of t, and correctly adding a new 3-cycle.

As stated above, the selector, the action, and the envelope can each be simply implemented as program code. This suggest the possibility of using a non-deterministic CFG to represent a large space of possible graph grammars. A highly simplified fragment of our grammar is as follows:

```
<design> ::= GraphGrammar(<init>, <rules>).run(<steps>)
<init> ::= <cube> | <star> | <single_node>
<rules> ::= [<rule>, <rule>, <rule>]
<rule> ::= [<selector>, <action>]
<selector> ::= filter(k=current_k) | filter(t=current_t - 1) | [...]
<action> ::= [<function>, <envelope>]
<function> ::= add_node | add_node_and_edge | [...]
<envelope> ::= <exponential> | <sinusoid> | <line>
<steps> ::= 2 | 3 | 4
```

In order to use a CFG in an EC context, it is necessary to define an encoding: that is, a genotype data structure and a mapping from the genotype to an output string. We use the method of grammatical evolution (GE) [17]. GE defines a mapping from an integer-array genome, via a CFG, to a derivation string. The GE mapping is relatively simple: at each step in the derivation, the first non-terminal symbol in the derivation string is chosen, and the corresponding rule is identified; an integer is read from the genome; it determines which of the rule's RHS productions will be used to rewrite the non-terminal. The process repeats until all non-terminals have been replaced by terminals. (For fuller explanation see [17]; for previous applications of GE in the area of design see [1,10].) The result is a string, in our case a program defining a deterministic graph grammar. When run, this program executes a graph grammar derivation, eventually giving rise to a graph whose nodes have x, y and z values. Feeding such a graph into a 3D rendering program results in an image.

In all the runs described in this paper, GE's int-flip mutation was carried out with per-gene probability 0.015, and the one-point crossover probability was 0.9. Tournament selection was used with tournament size 5. In interactive runs the population size was 12 and the number of generations unlimited. In non-interactive runs the population size was 100 and the number of generations 50.

3.1 Aesthetic and Computational Fitness

In our system, fitness is primarily aesthetic. In typical runs, a user interacts with evolution, assigning a "good" or "bad" fitness value to each individual at each generation. Individuals have "bad" fitness by default, so that the user is required only to click the images of good individuals with the mouse.

However, the system can also be run in non-interactive mode. We derive a (necessarily incomplete and imperfect) fitness measure by defining a set of three numerical features which can be calculated over graphs, observing their values on "good" and "bad" individuals as evaluated by interactive users on preliminary experiments, and then defining fitness as the achievement of the "good" ranges.

Graph Features. We define three numerical features over graphs. Each is intended to measure some aspect of the complexity the graph, including the Euclidean labels of the nodes.

Coordinate complexity measures the complexity of the x, y and z coordinates of the graph's nodes. For each coordinate, the *unique ratio* is the number of unique values of that coordinate divided by the number of nodes in the graph. Coordinate complexity is the mean of the x, y and z unique ratios. In graphs where most nodes' coordinates are unique, the value will be high. Where many nodes are aligned along the principal axes, the value will be low.

Clustering is based on the graph-theoretic concept of clustering, where a highly-clustered node is a node most of whose neighbours are themselves also neighbours of each other. That is, it depends on the existence of many "triangles" involving that node. It can be calculated for node u as: $c_u = \frac{2T(u)}{\deg(u)(\deg(u)-1)}$, where $T(u)$ is the number of triangles of u and the denominator is twice the

maximum possible number of triangles of u. Clustering is then the mean value of c_u over all nodes. Graphs where nodes cluster in several almost-independent subgraphs will therefore achieve high values, while graphs where all nodes are uniformly connected to other nodes will achieve low values.

Finally, *size* is a simple measure of the number of nodes and edges in a graph. The two values are summed and the log is taken on the grounds that the difference between (say) 10 and 20 graph elements is as significant as the difference between 100 and 200. In order to normalise the value to $[0, 1]$, the log-value is divided by log(1000), taking 1000 graph elements as a plausible upper limit. For very large graphs, then, the normalised value must be clamped to avoid exceeding 1.

None of the three features is intended as an "aesthetic" measure, and indeed it is not clear that aesthetics can be satisfactorily measured in this domain. However, it is hypothesized that certain ranges of the measures will turn out to be more appealing to some users, in some circumstances, than others. Experiments using these features are presented in Sect. 4. The overall process executed by the system is depicted in Fig. 4.

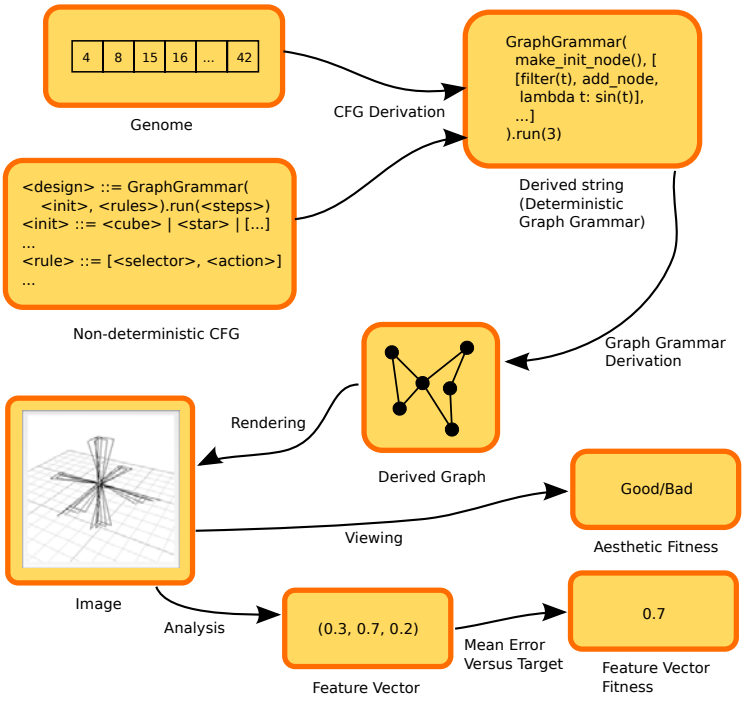

Fig. 4. The genotype to phenotype and fitness process

4 Evaluation

The system is next evaluated in both interactive and non-interactive use.

4.1 Design Outcomes

We begin by presenting, in Fig. 5, 12 designs created in three different ways. First, designs were created by random generation using the graph grammar. That is, many initial populations of random genomes were randomly generated and their corresponding designs rendered. In each population of 12, between 1 and 6 individuals were subjectively marked as "good" and the rest deemed "bad". The feature vectors for all individuals were saved. The rationale for selecting from randomly-generated initial populations in this case, rather than from ongoing evolutionary runs, was to achieve more consistent data. It is well-known [8] that during ongoing runs users tend to award inconsistent values to similar or identical individuals, influenced by boredom and novelty. Running many one-off populations in which the task was to mark good individuals, rather than to steer the evolution, was intended to avoid such effects. Out of about 200 individuals a few are presented as Fig. 5 (a-h). The designs demonstrate that the grammar has been successful in allowing a very wide *range* of designs, and giving each individual design a sense of being *well-formed* or *cohesive*.

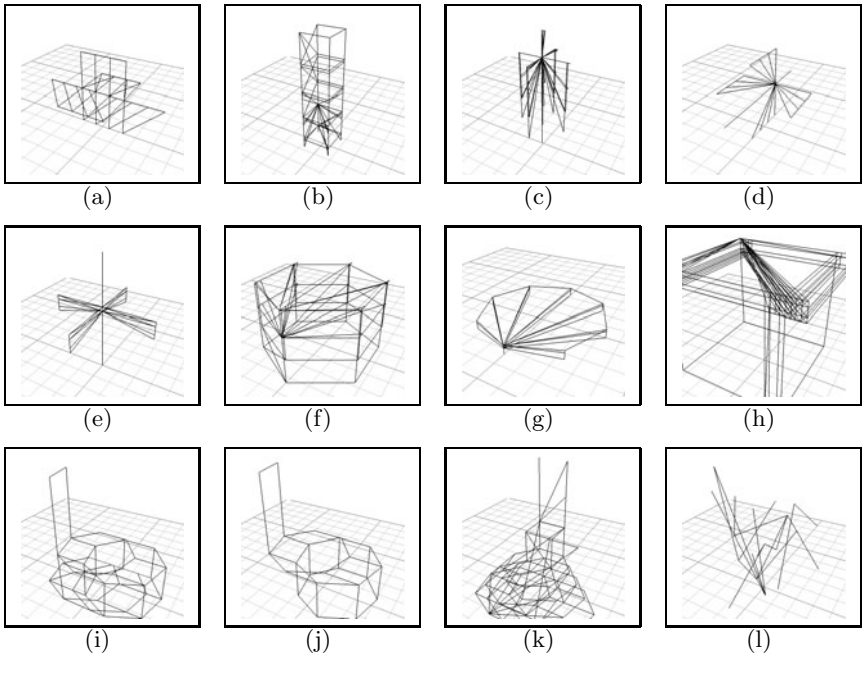

Fig. 5. Designs created by random generation with the graph grammar (a-h); by interactive evolution (i-k); and by a non-grammatical random graph model (l)

Next, designs were created during a single interactive run of 20 generations. Only a single individual was marked good in each generation, and a few representative samples were chosen for presentation as Fig. 5 (i-k). The three designs appear inter-related, as expected from a single run. Many more similar runs have been performed, always producing similarly interesting and organised results, but are omitted to save space.

Finally, Fig. 5 (l) was created using a random graph model. Several random graph models are in common use; one of the best-known is the $G(n, p)$ model [6], in which the graph G has exactly n nodes and each possible edge occurs with probability p. We choose n uniformly in $[20, 100]$ and set $p = 0.03$. Running the algorithm multiple times then yields multiple distinct graphs of different sizes. In each case, we then set the Euclidean coordinates of the nodes randomly, choosing them uniformly in $[0, 3]$ (quite similar to the ranges achieved by the graph grammar individuals). Results are almost always highly disorganised in appearance: Fig. 5 (l) is a typical example. There is a very clear aesthetic contrast with the grammar-derived designs, and this contrast can also be demonstrated computationally as we will see.

4.2 Results with Feature Vectors

Next the feature vectors of the designs created in the previous section are analysed. In Fig. 6 the feature vectors of approximately 250 individuals are presented as scatterplots. These individuals come from four categories as above: $G(n, p)$ random graph individuals (small dark circles), good graph grammar individuals from random populations (dark triangles), bad individuals from the same populations (crosses), and single selected individuals from successive generations in a single interactive run (large light-coloured circles, joined with lines).

The features give some useful information about the four different categories. The random-graph model designs form a tight cluster. This is in contrast to the individuals created using graph grammars. However, the features do not appear to distinguish between good and bad graph-grammar individuals. In Fig. 6 (b), good and bad individuals are intermingled with no discernible pattern. In (a), good individuals seem to achieve higher values for clustering complexity, overall, but the pattern is not clear-cut. The selected individuals from the 20-generation run conform to the distribution of good individuals.

4.3 Non-interactive Runs

In the light of the previous results, it is clearly not possible to define any values or ranges for the features which can be used as a general aesthetic fitness function. Future work will be addressed to finding an improved set of features. In the meantime we test the system's non-interactive evolution component using arbitrary feature vectors as targets. We define the fitness of an individual as the mean error of its feature vector versus a target. We ran experiments with several targets, reporting results from just two: (0.3, 0.3, 0.6) and (0.8, 0.4, 0.8). Each is just outside the ranges achieved in interactive runs. Non-interactive evolution

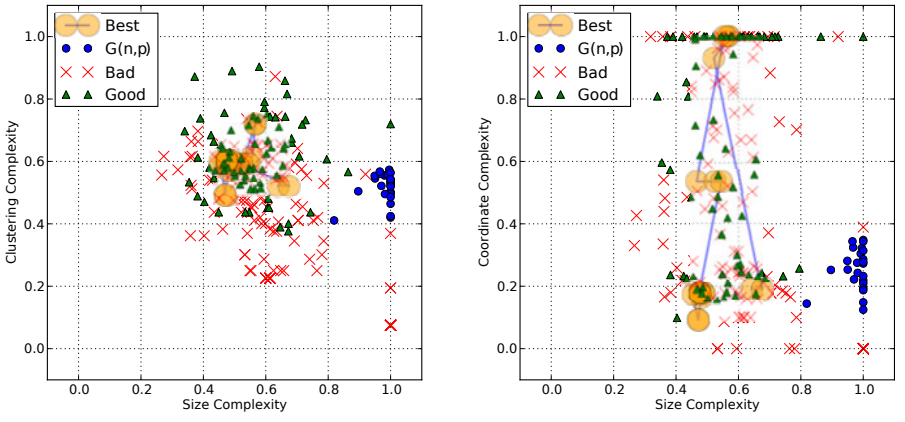

Fig. 6. Feature distributions for random, bad, good, and best individuals: clustering complexity versus size (left); coordinate complexity versus size (right)

turns out to be possible, but the first target appears easy and the second difficult, as shown in Fig. 7 for 10 independent runs each. Explanation of this interesting result requires further analysis of the search space and is left to future work.

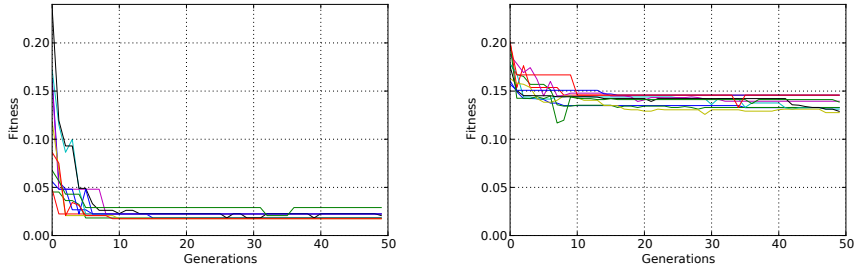

Fig. 7. Non-interactive evolution: an easy feature vector target (left); a more difficult one (right). 10 independent runs for each target. Fitness is minimised.

5 Conclusions and Future Work

A representation for 3D design based on graph grammars has been introduced. It has been demonstrated to introduce a bias on the space of graphs so that disorganised designs are largely eliminated from the search space. The representation is nevertheless capable of generating a very wide range of designs. A set of numerical features over graphs has been defined. A set of experiments has been run in order to characterise subjectively-judged good and bad individuals in terms of

the features and thus to derive a non-interactive fitness function. Although the good and bad individuals do not appear separable using the current features, non-interactive evolution (with an arbitrary target) has been demonstrated.

The success of the graph grammar representation and its limitations, as described above, together suggest that there is potential for future work in improving and understanding the system.

A careful study of the individual and combined effects of the different types of selectors, actions, and envelopes in the graph grammar would help us to better understand what makes it work. It might also point the way to improvements to the grammar. Increasing the grammar's already impressive flexibility would be particularly interesting, but eliminating some classes of undesirable designs from the search space would also be very useful.

The framework for computational fitness—defining graph features, observing their values in user-judged "good" and "bad" individuals, and defining computational fitness in those terms—has the potential to be very useful, but the three features proposed in this work have not proved capable of clearly separating good individuals from bad. In future work, a more thorough study of graph-theoretic and complexity-inspired features will be carried out in the hope of finding more useful features. It would also be interesting to study EC-theoretic aspects of the system, such as the degree of heritability via mutation and via crossover.

The system has been presented in a classroom setting with positive feedback. A formal user study with architecture and design students and practitioners is in progress.

Acknowledgements. Thanks to Jonathan Byrne and Erik Hemberg of the NCRA, UCD, for providing GUI code. Thanks to Terry Knight of MIT and the MIT Visual Computing class. The author is funded by the Irish Research Council for Science, Engineering and Technology under the Inspire scheme.

References

1. Byrne, J., Fenton, M., Hemberg, E., McDermott, J., O'Neill, M., Shotton, E., Nally, C.: Combining Structural Analysis and Multi-Objective Criteria for Evolutionary Architectural Design. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) EvoApplications 2011, Part II. LNCS, vol. 6625, pp. 204–213. Springer, Heidelberg (2011)
2. Chomsky, N.: Three models for the description of language. Transactions on Information Theory 2(3), 113–124 (1956)
3. Clune, J., Lipson, H.: Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In: Proceedings of the European Conference on Artificial Life (2011), `http://endlessforms.com`
4. Dawkins, R.: The Blind Watchmaker. Longman Scientific and Technical, Harlow, England (1986)

5. Ehrig, H., Korff, M., Löwe, M.: Tutorial Introduction to the Algebraic Approach of Graph Grammars Based on Double and Single Pushouts. In: Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Graph Grammars 1990. LNCS, vol. 532, pp. 24–37. Springer, Heidelberg (1991)
6. Gilbert, E.N.: Random graphs. The Annals of Mathematical Statistics 30(4), 1141–1144 (1959)
7. Gips, J.: Computer implementation of shape grammars. In: NSF/MIT Workshop on Shape Computation (1999), http://www.shapegrammar.org/nsfmit.html
8. Gu, Z., Xi Tang, M., Frazer, J.H.: Capturing aesthetic intention during interactive evolution. Computer-Aided Design 38(3), 224–237 (2006)
9. den Heijer, E., Eiben, A.E.: Evolving art with scalable vector graphics. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, pp. 427–434. ACM, Dublin (2011)
10. Hemberg, M., O'Reilly, U.-M., Menges, A., Jonas, K., da Costa Goncalves, M., Fuchs, S.: Genr8: Architect's experience using an emergent design tool. In: Machado, P., Romero, J. (eds.) The Art of Artificial Evolution. Springer, Berlin (2007)
11. Keles, H.Y., Özkar, M., Tari, S.: Embedding shapes without predefined parts. Environment and Planning B: Planning and Design 37(4), 664–681 (2010)
12. Knight, T.: Computing with emergence. Environment and Planning B: Planning and Design 30(2) (2003)
13. Knight, T.W.: Transformations of De Stijl art: the paintings of Georges Vantongerloo and Fritz Glarner. Environment and Planning B: Planning and Design 16(1), 51–98 (1989)
14. Koning, H., Eizenberg, J.: The language of the prairie: Frank Lloyd Wright's prairie houses. Environment and Planning B 8, 295–323 (1981)
15. Luerssen, M.H., Powers, D.M.W.: Graph design by graph grammar evolution. In: Proceedings of the 2007 IEEE Congress on Evolutionary Computation, pp. 386–393. IEEE (2007)
16. McCormack, J.P., Cagan, J.: Curve-based shape matching: supporting designers' hierarchies through parametric shape recognition of arbitrary geometry. Environment And Planning B: Planning And Design 33(4), 523 (2006)
17. O'Neill, M., Ryan, C., Keijzer, M., Cattolico, M.: Crossover in grammatical evolution. Genetic Programming and Evolvable Machines 4(1), 67–93 (2003)
18. Piazzalunga, U., Fitzhorn, P.: Note on a three-dimensional shape grammar interpreter. Environment And Planning B 25, 11–30 (1998)
19. Reddy, J.G., Cagan: An improved shape annealing algorithm for truss topology generation. Journal of Mechanical Design 117, 315 (1995)
20. Sims, K.: Artificial evolution for computer graphics. In: SIGGRAPH 1991: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, pp. 319–328. ACM, New York (1991)
21. Stiny, G., Gips, J.: Shape grammars and the generative specification of painting and sculpture. In: Petrocelli, O.R. (ed.) The Best Computer Papers of 1971, pp. 125–135. Auerbach (1972); originally published in: C. V. Freiman (ed.), Information Processing 71: Proceedings of the 1971 Congress of the International Federation for Information Processing, Ljubljana, Yugoslavia
22. Tapia, M.: A visual implementation of a shape grammar system. Environment and Planning B 26, 59–74 (1999)