

Balancing Act: Variation and Utility in Evolutionary Art

Jon McCormack

Centre for Electronic Media Art
Monash University, Caulfield East, Australia
Jon.McCormack@monash.edu
<http://jonmccormack.info>

Abstract. Evolutionary Art typically involves a tradeoff between the size and flexibility of genotype space and its mapping to an expressive phenotype space. Ideally we would like a genotypic representation that is terse but expressive, that is, we want to maximise the useful variations the genotype is capable of expressing in phenotype space. Terseness is necessary to minimise the size of the overall search space, and expressiveness can be loosely interpreted as phenotypes that are useful (of high fitness) and diverse (in feature space). In this paper I describe a system that attempts to maximise this ratio between terseness and expressiveness. The system uses a binary string up to any maximum length as the genotype. The genotype string is interpreted as building instructions for a graph, similar to the cellular programming techniques used to evolve artificial neural networks. The graph is then interpreted as a form-building automaton that can construct animated 3-dimensional forms of arbitrary complexity. In the test case the requirement for expressiveness is that the resultant form must have recognisable biomorphic properties and that every possible genotype must fulfil this condition. After much experimentation, a number of constraints in the mapping technique were devised to satisfy this condition. These include a special set of geometric building operators that take into account morphological properties of the generated form. These methods were used in the evolutionary artwork “Codeform”, developed for the Ars Electronica museum. The work generated evolved virtual creatures based on genomes acquired from the QR codes on museum visitor’s entry tickets.

Keywords: Evolutionary Art, Aesthetics, Artificial Life, genotype-phenotype mapping.

1 Introduction

A major challenge in designing evolutionary systems is defining efficient genotype representations and the phenotypes they encode for. In evolutionary art this problem is forefront when critical attention is placed on the evolutionary system and the artistic intent of what it produces.

This paper describes an artwork titled “Codeform”, created by the author. The work was developed for the Ars Electronica museum in Linz, Austria, and

runs on the museum’s *Deepspace* facility¹, a bespoke, large-scale virtual reality system with 16m × 9m stereoscopic projections onto the wall and floor using eight high-brightness stereo projectors. The space can accommodate up to several hundred participants who wear active polarising glasses to experience an immersive 3D virtual environment, generated in real-time.

1.1 Utility and Diversity

While this paper will describe the technical and implementation details of Codeform, the main purpose is to highlight a common problem for evolutionary art: achieving a good balance between the size of the genotype space and its expressive power or aesthetic² utility. This is a well known “open problem” for Evolutionary Music and Art [9]. In basic terms, we want a generative system that can express phenotypes that are *useful* and *diverse*. Useful in the sense that they satisfy some creative or artistic criteria (e.g. have interesting aesthetics or semantics), and diverse in the sense that they occupy distinct areas in the feature space of the phenotype. Hence we want to avoid representations that result in a large number of unappealing or highly similar phenotypes, making the evolutionary search difficult (e.g. a “needle in a haystack” problem[7, p. 21]).

In [10], this problem was described in terms of a quality factor, Q , for an evolutionary image generating system. Q was defined as the ratio between the normalised value of useful images output by the system to the size of the parameter phase-space (genotype space). A reinterpretation of this measure can be expressed:

$$Q = \frac{\nu}{\psi} \log \gamma \quad (1)$$

where ν is the total number of “interesting” phenotypes in the entire phenotype space, ψ is the total size of the phenotype space, and γ is the total size of the genotype space. The goal is to maximise Q .

“Interesting” is of course a subjective or context dependent measure and its definition can change the value of Q significantly. Here is a simple example. A 4-bit genome is used to generate integers (the phenotype) using a direct interpretation of the bit-string’s integer value. If we define “interesting” as the phenotype being prime then $\nu = 7$, $\psi = \gamma = 16$, so $Q \approx 1.21$. Defining “interesting” as being divisible by 5 results in $Q \approx 0.52$. In this example the size of genotype space is equal to phenotype space due to the direct 1-to-1 mapping, but this does not need to be the general case (for any system $\psi \leq \gamma$ must be true).

Provided the definition of “interesting” is held constant, it is reasonable to compare different representation schemes in terms of their potential Q value.³ However due to the subjective nature of its definition, comparing Q values with different criteria for “interesting” makes no sense. In the case of the work described in this paper, “interesting” has multiple dimensions and constraints.

¹ <http://www.aec.at/center/en/ausstellungen/deep-space/>

² “aesthetics” in the sense of 1, 2, 6 and 7 from [12].

³ Assuming ν can be measured or estimated, which is not always possible.

Key amongst these was the requirement that all phenotypes have basic biomorphic appearances (discussed in Section 2.1), and that the aesthetic experience of the work conforms to a particular and recognisable artistic style consistent with the author’s previous works.

Our goal is to maximise Q , that is to devise a system that can consistently produce useful phenotypes over the largest possible genotype space. In general terms this is often problematic because increasing the size of the genotype does not automatically result in a system that is capable of producing a higher proportion of interesting phenotypes – after a certain point the reverse is more probably the case.

This issue will be discussed in more detail in Section 3 using the Codeform artwork as the example. But before doing so we briefly review some previous work in this area, then explain Codeform’s representation scheme, generative mechanism and genotype and phenotype representations.

1.2 Related Work

Many researchers have devised schemes for evolving 3D, articulated biomorphic shapes, a well-known example is that of Sims [14] who evolved virtual articulated creatures, constructed from cubic blocks, using a graph representation that coded for morphology, sensors and motor control. Hornby looked at a wide variety of representation schemes for generative design [4], including “virtual creatures” generated using a variant of L-systems. Hornby and others [2] have observed that the generative reuse of parameterised elements in encoded designs improves the ability of an evolutionary algorithm to search large design spaces. The cellular encoding technique, developed by Gruau [3] used transformational graph grammars to generate neural networks and is related to the graph-building mechanism described in this paper. Cellular encoding provided a useful generative mechanism for generating neural networks (essentially graphs), exploiting modularity and reuse in the encoding mechanism.

2 Generative Mechanisms

The project called for a simple method to convert museum visitor tickets to a unique id to use as the basis of a generative, evolutionary artificial “creature” (phenotype) that is specific to each visitor. Fortunately the tickets already had a QR code printed on them (Fig. 1), which resolved to a unique 12-digit decimal number. This number serves as both an identifier (to the ticket and its owner) and as a generator of each creature. Ticket numbering is often in sequential batches, so there was a requirement that numbers in close proximity don’t generate phenotypes of similar appearance.

The ticket number was therefore used as a hash to both reference the phenotype and to generate it. Generation is a three-step transformational process:

$$ticket\ number \rightarrow genome\ string \rightarrow graph \rightarrow phenotype \quad (2)$$

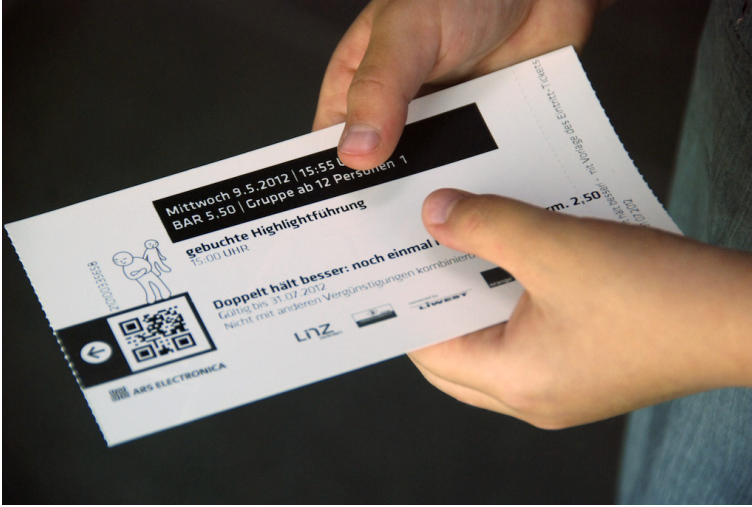


Fig. 1. Museum entry ticket showing the QR code on the left side of the ticket

To transform the ticket to the genome, the ticket number is used as the initial bit pattern in a modified linear congruential generator algorithm [6], which creates bit strings of lengths that vary between supplied minima and maxima. This ensures sufficient variation while keeping the bit string within acceptable limits. Thus the genotype generated is a bit string of length l : $l_{min} \leq l \leq l_{max}$.

The bit string genome (I) is interpreted as a series of programming instructions to a machine that builds graphs. This is somewhat similar to the cellular encoding technique developed by Gruau, which used sequences of graph transformations to build neural networks [3]. Here the instructions are represented as a binary string of machine instructions. Each instruction is 4 bits in length, consisting of a 2-bit opcode and a 2-bit parameter.

From these instructions, I , the machine, M , builds a graph $G = (N, E)$: a set of nodes (N) and unidirectional edges (E), i.e.

$$M \xrightarrow{I} G \quad (3)$$

At all times M maintains a *current node*, c , and a *root node*, r , both initially set to \emptyset (empty).

Allowing 2 bits for an opcode gives 4 possible instructions, which are summarised in Table 1. Four instructions were chosen as they provide the minimum set of operations necessary to construct a graph from scratch. Opcodes 00 (add child) and 01 (add sibling) have special behaviour defined when $c = r = \emptyset$: they will add the node, n , as the root and set $c = r = n$. Instructions to create edges or shift the current node when $c = \emptyset$ are ignored.

Table 1. Instructions for the graph building machine and their interpretation

opcode	instruction	description
00pp	<i>add child</i>	add a child node of type <i>pp</i> to the current node (<i>c</i>) and set <i>c</i> to be the new child
01pp	<i>add sibling</i>	add a sibling node of type <i>pp</i> (a new child to the parent of <i>c</i>)
10pp	<i>add edge</i>	create an edge $c \rightarrow (c - pp) \bmod N $
11pp	<i>shift current</i>	set $c = (c + pp) \bmod N $

Allowing 2 bits for the parameter values means each node can be of four distinct types. For the moment, let us label these types *A*, *B*, *C* and *D* for the parameter values 00, 01, 10 and 11 respectively. Similarly, when creating edges or shifting the current node the 2-bit parameter allows for only 4 possible values, being the number of nodes to move from the current node in the node set *N*. The choice of using only 2 parameter bits places limits on possible phenotypes that the system can generate. This issue is discussed in Section 3.

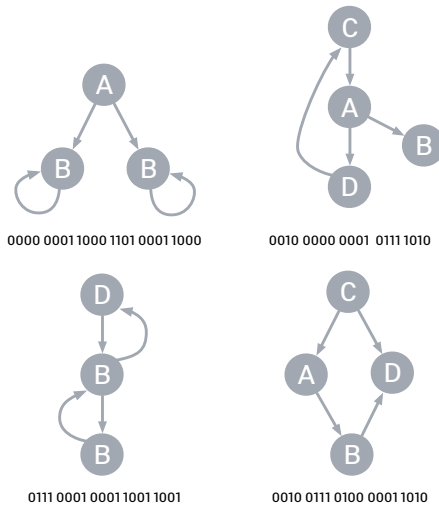
**Fig. 2.** Some example genome program strings and the graphs they generate

Figure 2 shows some hand-made example bit strings and the graphs they generate.

2.1 Graph Interpretation

The next stage is generating the phenotype: essentially using the graph as a generative specification for an animated 3D form. This is achieved by traversing the

graph from the root node in a breadth-first traversal, interpreting information in the nodes and edges to construct time-varying 3D geometry and transformations.

Initially, we experimented with an interpretation similar to that of Sims [14], whereby each node generates a cuboid of varying dimensions with a series of articulated joints that specify the placement of the parts generated by child nodes. Sims also used a parameter to specify the “recursive limit” of node traversal to limit the number of times a node should generate a part when in a recursive cycle. Edge information was used to specify the placement of child parts relative to the parent, with child parts constrained to the surface of the parent part. Finally, a *terminal-only* flag was used to specify the connection of child parts only at the end of a recursive cycle.

These node and edge parameters were generated using the parameter information (*pp*) associated with each machine instruction. Having only four possible values places some limitation on the parameter variation possible, but gives enough variety to easily distinguish one phenotype from another.

This interpretation allows for a large flexibility in the structure and morphology of possible phenotypes, however for the application described in this paper there was an additional requirement: that the phenotype generated from a ticket have a basic, recognisable biomorphic appearance, including recognisable features such as lateral or radial symmetry, articulated appendage parts and a coherent overall structure. The reasons for this will be explained in Section 3.

After some experimentation, the following interpretation of the graph was used. Each node type defines a unique morphic structure, \mathcal{S} , composed of a *connecting topology*, *shape* (geometry) and an ordered set of *connection points*, \mathcal{S}_c . Connection points have specific locations over the topology that allow other structures to be connected to them and from them, subject to an affine geometric transform applied at each point. This transformation is stored as a homogeneous 4x4 matrix for each point.

Table 2. Node types and their geometric interpretation





node type	description	\mathcal{S}
A n -point radial symmetry	a disk shape with n radially symmetric connections	
B bi-directional branch	branching element with laterally symmetric branch connections	
C uni-directional branch	branching element with uni-directional laterally symmetric branch connections	
D n -phyllotaxic	sphere with n points distributed over the surface in a phyllotaxis pattern	

Table 2 details the interpretation of each node type and shows a graphical representation of \mathcal{S} , including the connection points, shown as either black, white

or red points on an example shape (shown in grey). The difference between the black and white connection points is in the way the point performs its transformation, with the white point applying a reflection transform (lateral mirroring about the symmetry) to the child geometry instantiated at the connection point. The red point defines the location and orientation at which the structure is attached when being placed as a child object. Some examples are shown in Fig. 3. The examples in the figure show topological relationships only – additional geometric and transformational operations are also applied which are not shown in the figure for sake of clarity.

One important feature of this system is that the topologies always connect in any possible combination without generating illegal topologies or self-intersections – an important attribute given the requirement of biomorphism in every phenotype.

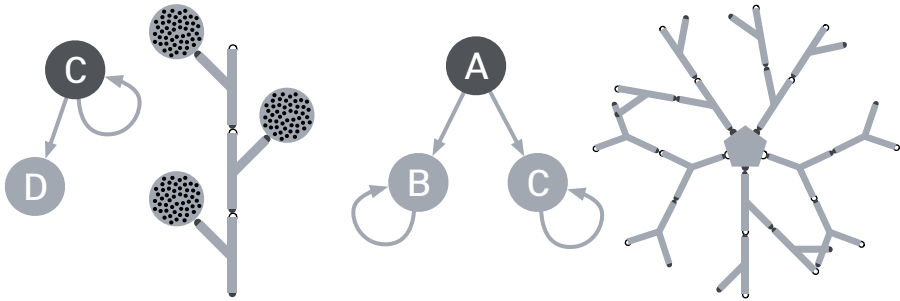


Fig. 3. Example graphs and the topological structures they generate. The root node is shown with darker shading.

The graphs are traversed using a breadth-first traversal, up to the recursive limit imposed by the node (derived from parameter information in the genome). For any given node, as each output edge is traversed an internal counter is incremented and this counter is used as an index to determine the connection point for the child object pointed to by the current edge. So if the counter value is q , the connection point is:

$$\mathcal{S}_c[q \bmod |\mathcal{S}_c|] \quad (4)$$

This means that the child node of each outgoing edge is iterated over the available connection points, the number of connections points ($|\mathcal{S}_c|$) being independent of the number of outgoing edges.

The graph traversal sends a series of commands to a 3D-turtle [1], which maintains a state, consisting of a local coordinate reference frame, transformation matrix, position, material colour, and so on. Additionally the turtle maintains a local stack and has commands to push and pop this state to and from the stack. The turtle can also instantiate geometry, so traversing the graph results in a series of commands to build a 3D geometric structure. This is a common method used in generating models from L-systems for example [5,8,13].

2.2 Animation and Sound

Connection points also include articulation information (degree of freedom, movement limits, movement function) allowing different parts of the creature to animate. Animation is performed using a simple harmonic sum-of-sines function, which is also sent as a control parameter to the sound generation system to control audio articulation.

The sound system takes control parameter and phenotype specific information (age, size, 3D position and orientation, etc.) to generate a soundscape specific to each creature. This includes generating a specific pitch contour over 8 octaves distinct to each creature and derived from the ticket number used to generate it.

Multiple creatures can exist simultaneously and sounds for each are generated individually and spatially located using a 5.1 surround speaker system. As a creature ages the pitch of the sounds it makes decrease and become slower. The sound component of the system was developed in MAX/MSP⁴. Communication with the main application via OSC over a local network. Up to fifty creatures can generate sounds simultaneously using this system. The sound system keeps track of new creatures that may begin “life” at any time, and those that die off, ending their sound-making in the system.

2.3 Evolution

As detailed at the beginning of Section 2, the visitor’s ticket uniquely identifies the visitor and the phenotype (virtual creature) that the ticket generates. Visitors enter a large, immersive space and have their ticket scanned, generating the creature, which is presented to the viewer as stereoscopic 3D geometry rendered using OpenGL. Multiple tickets can be scanned, creating an ecosystem of creatures which are free to move about, breed and reproduce. Diploid reproduction is performed on each creature’s bit string, I , using single point crossover constrained to instruction boundaries and bit flip mutation with probability inversely proportional to genome length/4. As the bit strings can vary in length crossover points are clamped to whichever string is shorter.

The creatures exist in a virtual 3D ecosystem, the details of which will not be covered in this paper for space reasons, but the interested reader can refer to [11] for details on the ecosystemic approach. Some important considerations are that creatures may live and die, give birth to offspring, and so on. The system is designed with a persistence mechanism and database that keeps track of all tickets scanned and all creatures generated, even those that have died. This mechanism allows for a museum visitor to return to the work at any time (from minutes to years later) and see the fate and “family tree” of their creature. Some example creatures generated from tickets are shown in Fig. 4.

⁴ www.cycling74.com

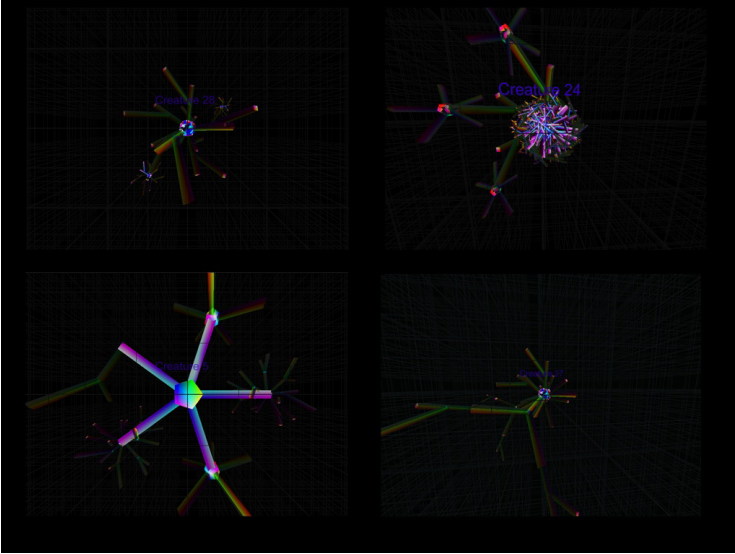


Fig. 4. Example phenotypes generated with the system (in 3D)

3 Discussion

As outlined in the introduction (Section 1), the topic of this paper is the issue of increasing the Q value from Equation 1. The size of the genotype space (γ) of the system described above is $2^{l_{max}}$. l_{max} is typically 120, making $\gamma \approx 10^{36}$, a number much larger than the 10^{12} possible ticket numbers. This is reasonable because the number of possible phenotypes is necessarily greater than the number of tickets, as genotypes can be created through evolution, not just ticket scanning (you could think of the tickets as mapping to a random initialisation point in genotype space).

One limitation of the graph generation scheme is that different genomes can potentially produce graphs of identical structure. The “add sibling” instruction can be replicated by a sequence of shifts and the “add child” instruction, similar to the way multiplication can be achieved through multiple addition nodes in a GP tree for example. However, graph structure is only one factor in phenotype generation, so two phenotypes with the same graph structure will still be different.

As discussed in Section 2.1, initially the “block creatures” method of Sims [14] was implemented as the way of interpreting the graph generated from the genome string. As is well known, Sims was able to evolve “biomorphic” creatures with characteristics strikingly reminiscent of real biology. However he was not able to get the system to evolve creatures with morphologies like those he was able to easily devise by hand, including a tree, a multi-segmented, multi-legged figure, and a human-like figure, despite these morphologies needing a maximum

of only 3 nodes and 6 edges. One explanation for this difficulty is that, despite a relatively low number of elements, each node and edge contains a large number of parameters, significantly increasing the size of the search space. Additionally, certain fitness measures used (such as the distance moved from a fixed point or light following ability) would not tend to favour the evolution of tree- or human-like shapes over any others.

The application described in this paper required that when a user scans their ticket, they expect the resultant phenotype to have at least some biomorphic properties (such as those which could easily be devised by hand in Sims' system). The nature of the algorithm that converts a ticket number to genome bit-string is essentially mapping the ticket to a random point in genotype space, necessary so that consecutive ticket numbers do not produce very similar genotypes⁵, as would be the case with a direct conversion of the ticket number to bit-string, for example.

Allowing only 2 parameter bits in each instruction places limitations on phenotype diversity: the translation distances in the *add edge* and *shift* instructions (Table 1) and the number of possible node types in any graph. In the case of graph generation this did not appear to be a problem. Multiple *shift* instructions, for example, allow large movements beyond the 4 possibilities allowed by a single instruction. Statistical tests on large numbers of randomly generated genotypes (bit strings) showed that increasing the parameter bits did not result in a significant increase in the variety of graph topologies generated.

While more node types could be added with an increased parameter size, this also increases the size of the search space. The question to ask is: does adding additional types significantly increase the utility *and* diversity of the resultant phenotypes? If the additional functionality gained by introducing a new type can be achieved through a combination of existing types, we are potentially increasing utility (its easier to find good phenotypes) but reducing diversity (more phenotypes will have this composite functionality).

3.1 Composing Structure

Testing the system by generating many thousands of initial bit-strings did reveal that while there was good variation in the graphs, the majority of phenotypes generated just looked like random assemblages of blocks, suggesting the value of ν from Equation 1 was very low. Trying to find even basic "biomorphic" structures that the genotype space is known to possess requires significant amounts of time to evolve, which is not practical in a situation where the time between scanning a ticket and seeing it generated must be less than around 500ms. Hence the solution is to try and increase the value of ν in Equation 1.

To achieve this the system was modified so that the nodes of the graph control a kind of "biomorphic construction kit", meaning that it will always generate

⁵ Tickets are dispensed from rolls in sequentially numbered batches. While there are several different ticketing machines simultaneously in operation each with a different counter, analysis of ticket numbers shows that for any given audience there are many tickets that form sequences.

“acceptable” phenotypes. Interestingly, this was achieved without modification to the size of the genotype space. While this may at first seem counterintuitive, it is analogous to an evolutionary text generation system whose genome controls the assembly of words to form sentences (the phenotype). Allowing every possible word combination gives great scope, but the majority of randomly assembled words will be grammatically incorrect nonsense. Enforcing structure (adjectives precede nouns, verb conjugation, etc.) provides a good starting point for building coherent sentences.

However, by enforcing this structure, we impose a limitation on what can potentially ever evolve – nothing can “break the rules”. In other words (and related to the discussion above regarding tradeoffs when increasing the number of node types), we limit the *diversity* of what is potentially possible while increasing the *utility* or usefulness of what is produced. If we equate both utility and diversity with the “interestingness” of our system (Section 1.1), then arguably we haven’t increased ν as it might first have appeared. This is an inherent problem with trying to formalise the subjective, a problem that we might optimistically hope improves as our understanding of human subjectivity increases.

The experiments discussed here suggest that a system which imposes fewer structural rules has only the *potential* to generate useful diversity, not the certainty. Understanding the sizes and structure of genotype, phenotype and feature spaces allows us to make better judgments in this “balancing act” between the potential and the practical, hence improve the possible of our evolutionary art systems.

Acknowledgements. This research was supported by an Australian Research Council Discovery Project grant, DP1094064. Peter McIlwain developed the MAX/MSP sound component of the work. Thanks also to Matthew Gardiner and Benjamin Mayr from the Ars Electronica Future Lab for supporting the production of “Codeform” and to Gerfried Stocker, Director of Ars Electronica for commissioning the work.

References

1. Abelson, H., DiSessa, A.A.: Turtle geometry: the computer as a medium for exploring mathematics. The MIT Press series in artificial intelligence. MIT Press, Cambridge (1982)
2. Bentley, P.J.: Evolutionary design by computers. Morgan Kaufmann Publishers, San Francisco (1999)
3. Gruau, F.: Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. Phd thesis, l’Ecole Normale Supérieure de Lyon (1994)
4. Hornby, G.S.: Generative Representations for Evolutionary Design Automation. PhD thesis, Boston, MA (2003)
5. Hornby, G.S., Pollack, J.B.: Evolving L-systems to generate virtual creatures. Computers & Graphics 26, 1041–1048 (2001)
6. Knuth, D.E.: The art of computer programming, world student series edition. World student series, vol. 2. Addison-Wesley, Reading (1972)

7. Luke, S.: *Essentials of Metaheuristics*. Lulu Publishing, Department of Computer Science, George Mason University (2009)
8. McCormack, J.: *Aesthetic Evolution of L-Systems Revisited*. In: Raidl, G.R., et al. (eds.) *EvoWorkshops 2004*. LNCS, vol. 3005, pp. 477–488. Springer, Heidelberg (2004)
9. McCormack, J.: *Open problems in evolutionary music and art*. In: Rothlauf, F., et al. (eds.) *EvoWorkshops 2005*. LNCS, vol. 3449, pp. 428–436. Springer, Heidelberg (2005)
10. McCormack, J.: *Facing the future: Evolutionary possibilities for human-machine creativity*. In: Machado, P., Romero, J. (eds.) *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pp. 417–451. Springer (2008)
11. McCormack, J.: *Creative ecosystems*. In: McCormack, J., d’Inverno, M. (eds.) *Computers and Creativity*, ch. 2, pp. 39–60. Springer, Heidelberg (2012)
12. McCormack, J.: *Aesthetics, art, evolution*. In: Machado, P., McDermott, J., Carballal, A. (eds.) *EvoMUSART 2013*. LNCS, vol. 7834, pp. 1–12. Springer, Heidelberg (2013)
13. Prusinkiewicz, P., Lindenmayer, A.: *The algorithmic beauty of plants*. Number xii, 228 in *The virtual laboratory*. Springer, New York (1990)
14. Sims, K.: *Evolving virtual creatures*. In: *Computer Graphics*, pp. 15–22. ACM SIGGRAPH (July 1994)