A Novel Developmental System for the Study of Evolutionary Design

Adrian Grajdeanu
Evolutionary Computation Lab.
George Mason University
Fairfax, Virginia, USA
agrajdea@cs.gmu.edu

Sanjeev Kumar
Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, New York, USA
sk525@cornell.edu

Abstract

A novel developmental system designed to facilitate the study of generative encoding-based evolutionary design is presented. An aim of this work is to explore the simplest system and genetic encoding capable of exhibiting phenomena akin to size regulation and self-repair in developmental biology. Here we show that in the absence of complicated developmental mechanisms - such as ligand-receptor mediated cell signaling, asymmetric or mitotic spindle-based cell division, cell motility, or cellular forces - a simple rule-like GRN encoding and a simple internal/external chemical exchange mechanism confer the ability to generate patterns that are size regulated, and able to self-repair when damaged. In addition, we analyze the mechanisms that achieve size regulation and self-repair.

Introduction

Nature is replete with a plethora of organisms whose form and structure have fascinated biologists. Exactly how these complex forms and structures arise from humble beginnings has been the subject of scrutiny and debate for millenia. The fields of Genetics, and Molecular and Cell Biology have helped to uncover very detailed mechanisms of development for the chick limb or the body of Drosophila (Wolpert 1998), and although it can be said that we understand the development of aspects of these biological systems, our understanding remains poor for aspects of other systems.

The fields of Evolutionary Computation (EC) and Evolutionary Design (ED) have similar concerns, for example, how to build complex technology capable of self-adaptation, self-maintenance, and self-repair. A problem common to developmental biology and evolutionary computation is how the genetic encoding maps to the phenotype. In biology, the mapping is a complex non-linear process defined by DNA and cytoplasmic constituents that interact with the environment to yield a complex phenotype with remarkable abilities. Traditionally, in EC the mapping from genotype to phenotype is either extremely simple or non-existent, thus the choice of encoding is of paramount importance. The last decade, or two, has seen much research into the computational modeling of biological processes and mechanisms, which has led to the emergence of relatively new fields such as systems biology and a subfield of EC known as computational development (Kumar 2004).

Computational development centers around a genetic encoding and a model of a developmental system that includes analogues of cells, receptors, an environment and chemicals that diffuse through an embryo. Over time, cells divide and communicate with other cells in order to grow form and structure. Crucial to the developmental system, however, is the genetic encoding.

This paper presents a novel developmental system designed to facilitate the study of generative encoding-based evolutionary design. In particular, we investigate an instance of a generative encoding known as a Genetic Regulatory Network (GRN). GRNs come with an inherent complexity: they mandate graph like structures with more or less explicit links denoting the ability of genes to regulate (turn on/off) the activity of other genes. A specific aim of this work is to keep the core system and the genetic encoding simple, while still being able to generate complex phenomenon such as size regulation and self-repair.

The remainder of this paper is structured as follows: the next section presents related work, followed by a detailed description of the novel developmental system and design choices made. The evolutionary algorithm at the heart of the system is then introduced and the final section presents two sets of experiments and results that explore pattern formation, size regulation, and self-repair.

Related Work

Empirical studies in the field of (EC) show advantages to using generative encodings over explicit encodings (Eggenberger 1997; Bentley & Kumar 1999; Hornby 2003; Roggen & Federici 2004; Federici 2004). The set of studies that use a form of generative encoding may be subdivided into several categories. As such, an important class consists of the study of L-Systems (see (Lindenmeyer 1968; Prusinkiewicz & Lindenmayer 1990; Prusinkiewicz 1993; Prusinkiewicz, Hammel, & Mjolsness 1993)). Another class focuses on the use of cellular automata (Wolfram 1994; 2001; Ilachinski 2001; de Garis 1992). Yet a third class of studies, and the one we focus on in this paper, revolves around Genetic Regulatory Networks (GRNs) (Kumar & Bentley 2003).

Here we describe related work that centers around developmental systems, genetic regulatory networks, and those works that investigated the phenomena of size regulation and self-repair.

In (Jakobi 1995) Jakobi presents a rather complex developmental system, that is heavily inspired by biology in order to evolve neural network controllers for Kephera robots. His work included intricate genetic mechanisms and processes such as the analogue of the TATA box found in biological genomes (these mark the start of a gene on the genome).

Hutton's work (Hutton 2004) is entirely dedicated to the concept of membranes in artificial chemistries. A membrane is a selectively permeable separator that encloses the cell body isolating it from the exterior, while allowing the passage of certain chemicals.

Miller's work (Miller 2003) on size regulation and self-repair is perhaps more closely related to the work presented here. The similarities range in the fact that it involves a chemical agent as a means for communication with the environment and among cells. This chemical signal obeys a diffusion rule and finally it attempts to evolve a certain pattern: that of the French flag as described in (Wolpert 1998). However, while Miller used Cartesian Genetic Programming as the generative encoding to evolve the cell control program, the work presented here uses a much simpler rule-based GRN encoding.

In contrast to the approaches described above, the current work does not use the notion of fixed differentiation (user specified and fixed at the start of each run). Instead, cells differentiate only by what role they play in development and do not present an otherwise recognizable label chosen from a set *a priori* decided by the experimentalist.

Finally we provide specific mechanisms that show how pattern formation, self-maintenance, and self-repair are achieved via the evolved regulatory networks.

The Artificial Developmental System (ADS)

This section describes the ADS in detail. First we provide an overview of how the ADS functions; this is followed by a description of the ADS's components.

System Overview

One of the aims of this work is to keep the infrastructure as simple as possible. Yet at a minimum, it should provide an adequately rich 'environment' for development. The environment consists of a few mechanisms deemed absolutely necessary for the purposes of constructing complex structures. At a high level, development starts from a single cell (akin to a zygote) and - in reaction to environmental conditions - undergoes cell division to produce a set of cells that together may be considered a colony or a body. The resulting colony is the phenotype that is assessed for fitness. The spatial aspect of development is captured by organizing the environment as a lattice of locations.

In the current implementation of the ADS, each cell must occupy a location, and each location may host at most one cell (i.e no overlap). Furthermore, environmental interaction is reduced to one of its simplest aspects: chemical sensitivity. The system maintains abstract chemical ingredients, called *proteins*. They do not engage in chemical reactions, but are present only in the form of concentrations maintained at each location in the form of *external* concentrations and within cells in the form of *internal* concentrations.

Proteins present at locations (in the form of external concentrations) are subject to diffusion. The cells have the ability to alter internal concentrations and also to exchange proteins with their substrate location. There is therefore a minimal form of communication between adjacent cells. In addition, cells may form a preference for a certain direction of growth based upon comparatively assessing external protein concentrations in their neighboring locations. This is the only means by which directed growth may emerge within the system, and also the only case in which a cell has access to information contained in a place other than its own body or location. Note that cells cannot explicitly access a certain neighboring location (only a comparative aggregate of *all* neighboring locations).

Cells

Cell To Cell Communication While nature goes to great lengths to implement cell to cell communication such mechanisms are not included in the ADS. However, cell to cell communication is not precluded. Each cell may absorb/eliminate proteins from/into its external space, thereby creating chemical imbalance between adjacent spatial locations. Proteins diffuse across spatial locations carrying a chemical signal. Ultimately, another cell may absorb/eliminate the protein from/to its own external space, and thus be able to receive the signal.

Cell Division Placement An important aspect of development is cell motility. Since the ADS does not currently implement movement, a colony may migrate only via directed division at one end and death at the other. Directed division means that when a cell divides, the daughter cell is placed in an adjacent location according to specified preferences. The ADS designates such preference of placement of the daughter cell as *neighbor affinity*. Each cell maintains its own neighbor affinities. When it divides it performs a roulette drawing among all empty neighbors. If there are no empty neighboring locations the division does not take place. The *neighbor affinities* are used as weights in the roulette drawing, thus biasing the placement of the daughter cell towards locations that have higher affinity. Incidentally, this roulette drawing is a source of stochasticity in the developmental phase. It causes the fitness evaluation to be noisy; consequently, special measures are employed in the evolutionary algorithm to deal with this.

Proteins

Arithmetic Operations During development, the system maintains a set of protein concentration values. These values are constrained to the $[0, 1]$ range. Ensuring such range constraints can be achieved by applying naive operations and performing clipping afterward. This is not, however, the approach taken here. Instead, special operations are set in place that inherently keep the concentration values in the appropriate range.

Each operation works on a single protein and has an extra operand called amount, formally in the $[-1, 1]$ range. There are two groups of operations: produce/consume and absorb/eliminate. The first group deals only with the internal concentration of a given protein, altering its value. The second group deals with both the internal and external concentrations of the encoded protein altering both values. Within each group the two operations are each other's duals. Specifically, a 'produce' operation with a negative amount is equivalent to a 'consume' operation (and vice versa) with the corresponding absolute value amount. An identical dual relationship between absorb/eliminate operations also exists. As such, it is enough to describe the operation details for positive argument amounts.

A 'consume' operation is conceptually inspired by the cell using up some of its available protein, thereby decreasing its internal concentration. With an amount argument of, say, 0.5 it would reduce the internal concentration by half. With an amount argument of 0.3 it would reduce the internal concentration by a roughly a third. Similarly, a 'produce' operation is inspired by the cell synthesizing protein, thereby increasing its internal concentration. However, an amount argument of 0.5 does not indicate that the protein concentration must double, but rather indicates that it should approach its maximal value (1) by half. Similarly, an argument amount of 0.3 indicates the concentration should approach its maximal value by about a third of the distance it currently holds. For example, if the existing concentration is, say, 0.1, a 'produce' with an argument of, say, 0.3 would increase concentration by $(1 - 0.1) \times 0.3 = 0.27$, thereby setting the resulting concentration to 0.37. In order to formally define the operations, two functions are introduced first:

$$\begin{aligned} add &: [0,1] \times [0,1] \to [0,1] \\ add(x,a) &= x + (1-x) * a \quad (1) \\ sub &: [0,1] \times [0,1] \to [0,1] \\ sub(x,a) &= x - x * a \quad (2) \end{aligned}$$

Furthermore, let p denote a protein, $int(p)$ and $ext(p)$ denote its internal and external concentrations at the cell and location level and a be an operand amount in $[0, 1]$. In these terms, then the consume and produce operations become:

$$\begin{aligned} prod(p,a) &: int(p) = add(int(p), a) \quad (3) \\ cons(p,a) &: int(p) = sub(int(p), a) \quad (4) \end{aligned}$$

The second group of operations consists of eliminate and absorb functionalities. The 'eliminate' operation provides a mechanism for the cell to expunge some of its internal protein, thereby reducing the internal and increasing the external concentrations. The 'absorb' operation allows the cell to internalize some of the external proteins, increasing its internal concentration and reducing the external one. In terms of above formalism, these two operations are defined as follows:

$$elim(p,a) : \begin{cases} ext(p) = add(ext(p), a \times int(p)) \\ int(p) = sub(int(p), a \times (1 - ext(p))) \end{cases} \quad (5)$$

$$absb(p,a) : \begin{cases} int(p) = add(int(p), a \times ext(p)) \\ ext(p) = sub(ext(p), a \times (1 - int(p))) \end{cases} \quad (6)$$

Code	Type	Parameters
00	Zero	None
11		
01	One	One relational operator, one protein index and one threshold (between 0 and 1)
10	Two	One relational operator, two protein indices and two threshold (between -1 and 1)

Table 1: Condition Encoding

Code	Type	Parameters
0000	None	None
1111		
0010	Reserved	None
1101		
0001	Modify neighbor affinity	One relational operator, one protein index and one factor (between 0 and 1)
1110		
0100	Divide	None
1000	Die	None
0011	Protein related	One protein index, one amount (between 0 and 1) and one operation of the following: absorb, eliminate, produce, consume
0101		
0110		
0111		
1001		
1010		
1011		
1100		

Table 2: Action Encoding

Note that the way the absorb/eliminate operations are defined, they do not have any biological plausibility. The amount by which internal concentration is affected is unrelated to (and in most of the cases does not match) the amount by which the external concentration is changed.

Generative Encoding

Here we describe the implementation of the Cell Program. At a high level what is evolved is a genome with diffusion coefficients for each of the configured proteins, and a variable number of genes. The diffusion coefficients are non-negative numbers between 0 and infinity. In order to encode them, 5 bits are used. They are decoded (using standard binary mapping) to a number *t* between -5 and 5; the coefficient is obtained by raising 10 to power *t*.

Each gene consists of a condition and an action section. During development, all genes in each cell run sequentially. For a given gene, if the corresponding condition holds true, then the action is executed. This is repeated for all genes in the genome. The gene condition may be one of three types as described in table 1.

Condition type zero is always true. Condition type one is true if the internal concentration of the given protein is

	Condition layout			
Bit	Len	Type Zero	Type One	Type Two
0	2	Type		
2	2	—	Relational operator ($<, <=, >=, >$)	
4	3	—	Protein index	
7	8	—	Threshold	
15	3	—		Protein Index 2
18	8	—		Threshold 2

	Action layout					
Bit	Len	None / Reserved	Division	Die	Modify Neighbor Affinity	Protein related
26	4	Type				
30	2	—				Code
32	3	—			Protein Index	
33	8	—			Factor	Amount

Table 3: Gene Layout

with the also given threshold in the relation indicated by the encoded relational operator. More detail is required for condition type two. It involves two protein concentrations and two thresholds and centers around the $[0, 1] \times [0, 1]$ square that defines the combined universe of concentrations for the given two proteins. On this square a line section is defined by two points that are in turn defined by the two thresholds as follows. The range for the first threshold ($[-1, 1]$) extends from the top-left corner, through bottom-left one, until it reaches the bottom-right corner. The value of this first threshold defines one point that lies consequently either on the left or on the bottom side. Similarly, the range of the second threshold ($[-1, 1]$ too) extends from the same top-left corner, this time through the top-right one, until it also reaches the same bottom-right corner. The value of the second threshold defines the second point that lies this time either on the top side or the right side. Finally, this dividing line splits the possible combined range in two parts, of which the encoded relational operator designates one where the condition holds true. The purpose of this (rather convoluted) type of condition is to be able to emulate the biological phenomenon of activator/inhibitor so often encountered in regulatory networks.

The gene action may be one of six types described in table 2. The reserved action types are for now equivalent to a no operation. The *protein related* and the *die* actions have straightforward meaning.

The *divide* action initiates cellular division. However this will only take place if there are empty neighboring slots. The *modify neighbor affinity* action plays a role in the placement of the daughter cell in case of division. It iterates through all neighbors, looks at external concentration difference for the given protein. Then it modulates this difference by the given factor and finally uses this value to modify that neighbor's affinity.

Table 3 details the binary layout of all gene types, conditions and actions with their respective parameters. Notice that like parameters are stored for different conditions/actions in the same bit position. This is not unintended, but expressly included in the design, with the aim of limiting the disruptive effects of mutations.

Evolutionary Algorithm

The evolutionary algorithm used is an Evolutionary Strategie ES($\mu + \lambda$). A population size of 2 was used, with each parent able to generate 16 offspring. Both parents and offspring are evaluated (parents are re-evaluated due to the stochastic nature of the evaluation function) and compete for survival using truncation selection.

Evaluating an Individual Evaluating an individual consists of creating the initial seed and state of the environment, followed by development of the individual and the resulting shape is compared with the target.

One type of experiment is geared toward generating and maintaining shape. In this case, development is allowed to occur for a set number of steps (50) and the final shape is compared with the target. Another type of experiment deals with self repair. While development takes place, the environment randomly kills cells and the individual must detect such events and repair the damage. In these experiments development is run for 150 steps and at each step the current shape is compared with the target, eventually producing an average comparison over all 150 steps.

The method of comparing two shapes (the current one and the target) is to count the differences between them: accumulate 1 for each difference in each location. A difference is reported if a cell was to be present and isn't or is present and was not supposed to be.

Note that the evaluation method described is stochastic. The initial environmental seeding may be fixed, but the development phase involves an element of randomness. As such, evaluation is noisy; to deal with it, each evaluation is repeated four times and the mean result is considered as a measure that the algorithm is ultimately aiming to minimize.

Mutation Operator The mutation operator functions in the context of an ES. In order to implement the operator efficiently, both the bit representation of the genotype is maintained along with its internal representation. The operator uses these bits as well as $3n + 1$ fake bits, where n is the number of genes. The semantic of the fake bits (every time initialized to 0) is as follows. If any of the first $n + 1$ bits is flipped to 1, this designates an insertion point for a randomly generated gene. If any of the next n bits is flipped to 1, this designates a gene to be duplicated. If any of the last n bits

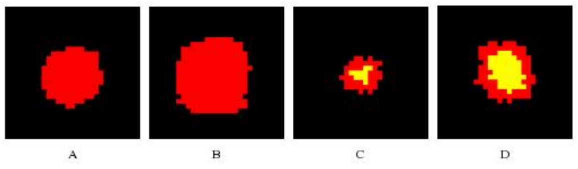

Figure 1: Size Regulation

is flipped to 1, this designates a gene to be deleted from the genome. So the mutation operator proceeds by joining the bits that represent the individual, with the $3n + 1$ fake bits, for a total of L bits. Bits are independently flipped with a probability of $1/L$. The mutation process may have to have multiple passes through the bit sequence until at least one bit has been flipped. If any of the fake bits were affected, implement their semantic (insert, duplicate, or delete gene) at the bit representation level. Finally, from the bit representation re-assemble the new individual.

Experiments

This paper presents two sets of experiments that were performed. The first explores both pattern formation and size regulation while the second is an extension that explores self-repair: a pattern is grown and subjected to random damage which it must then repair autonomously. The experiments involve generating and maintaining a target two dimensional shape in a universe of 27×27 locations. For division, as well as for diffusion purposes, a Moore neighborhood of radius 1 (8 neighbors) was used. The number of proteins was fixed to 4, but their diffusion coefficients were allowed to evolve.

Pattern Formation and Size Regulation

The first experiment explores whether the representation is capable of evolving an individual whose development from an initial seed leads to an approximately circular shape of a given size. The developmental program needs to grow the initial seed, achieve a target size and then maintain this size. The environment is initialized with the seed cell in the center location. The seed cell has all internal protein concentrations set to their maximal state. The environment does not contain any proteins (all external concentrations are zero).

The target shape is illustrated in panel A of figure 1. Randomly generated individuals fall mostly into two categories: stillborn or cancerous. The first successful individual was found after 50 developmental steps and is shown in panel B. The pattern is larger than the target, but this individual is intriguing. How does it achieve this size? Analysis reveals that left to run for more then 50 developmental steps, it will continue to grow until it occupies the entire space. It managed to improve fitness, not by limiting the size of its growth, but by slowing its growth rate. It is opportunistic, as it found a way to achieve good fitness without actually solving the problem.

Panel C shows a better individual that evolved from this experiment. This time the shape is grown to a certain size and growth is indeed stopped. This individual evolved the mechanism to control its growth size, albeit to a size smaller than the target. How did it achieve this? Analysis shows that the growth stopping criteria is a combination of conditions that mathematically should never happen. As it turns out there is a difference between mathematical sense and numerical implementation. Due to the inherently limited precision of the machine, what mathematically should never happen becomes reality. So the 'solution' found in this case is not really one that was sought.

Finally, panel D shows another evolved individual of even better performance. This individual is just about perfect. It manages to reach the target size and effectively stops the growth process.

```
proteins
[
    p0 4.21697e-05    p1 0.00153993
    p2 0.000365174    p3 4.21697e-05
]
genome
[
    condition-0                                    # 1
        action-protein produce p1 -0.078125
    condition-2 > p3 -0.359375 p1 0.710938         # 2
        action-neighbor-affinity p1 -0.0546875
    condition-2 <= p1 -0.0078125 p2 -0.1875        # 3
        action-divide
    condition-2 <= p1 -0.0078125 p2 -0.1875        # 4
        action-divide
    condition-2 <= p1 -0.0078125 p2 -0.1875        # 5
        action-divide
    condition-0                                    # 6
        action-protein consume p1 0.648438
]
```

The mechanism by which this individual controls its size is fairly easy to understand. The major role is played by protein p1. Notably, while protein p2 does show up in the conditions of the genes, none of the actions alters it. Thus, protein p2 stays at maximal concentration throughout the whole development phase. What the first and last genes of the genome do is to unconditionally reduce the internal concentration of protein p1 inside each cell. The second gene uses the same p1 protein to alter the neighbor affinity, but it has no effect because the initial affinities of zero can not be reduced any further. The approximately round shape of the developed colony is due to the randomness in the placement of the daughter cell in the absence of usable neighboring affinity.

Finally the 3rd, 4th and 5th genes are duplicates that control cell division. Any one of them would achieve the same result, and the fact that there are more than one does not increase fitness. It can be only speculated at this point that having multiple copies of the same gene provides the individual more robustness in the face of the mutation operator. The way that genes 3, 4 and 5 control cell division is to conditionally employ the same protein p1 whose internal concentration is gradually reduced. Protein p2 is kept constant, therefore protein p1 falls until the condition fails to be true. That is when and how growth is stopped. The size at which growth stops depends on the first and last genes: the faster p1 is reduced, the smaller the resulting size. If the concentration of p1 were to be reduced slowly, the resulting size would be larger.

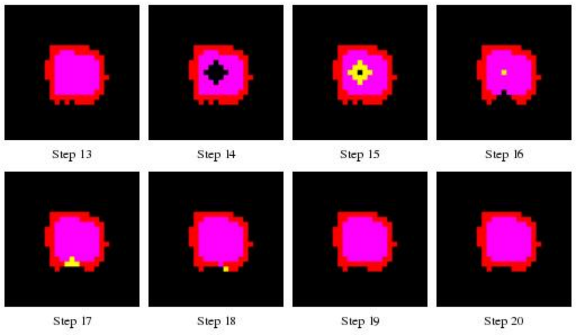

Figure 2: Self Repair

Self-Repair

This set of experiments subsumes those carried out in the pattern formation experiments, in that it requires the generation of the same circular pattern. This time, however, the environment is adversarial: at random intervals, it kills chunks of cells, an event that can be likened to necrosis (cell death due to external factors, i.e. that are not preprogrammed). The evolved individual must repair the damage and reform the target shape, after which it must stop the growth process and maintain the shape. Both the individual and the environment are seeded the same way as in the prior experiment.

Good solutions evolved in all runs of the experiment. Figure 2 presents consecutive developmental steps showing how a typical individual is damaged and repairs itself. There are two kinds of damage: one completely internal, and one affecting the edge. The fit individuals are capable of adequately repairing both types of damage. In the runs performed so far, the size of the evolved best genomes ranges from 24 to 145 genes. The functionality of these genomes is not trivial. Currently we are using ablation studies that selectively turn genes off at set times in the developmental process, aiming to uncover the underlying evolved mechanisms that result in this self-repair behavior.

Conclusions and Future Work

A novel developmental system designed to facilitate the study of generative encoding-based evolutionary design was presented. An aim of this work was to explore the simplest system and genetic encoding capable of exhibiting phenomena akin to size regulation and self-repair in developmental biology.

Consequently, in contrast to other approaches that use more complex generative encodings, in this work we presented a simple rule-based GRN developmental system (without any complicated artificial chemistry governing interactions, complex signaling mechanisms, or asymmetric division) that is capable of giving rise to pattern formation, size regulation, and self-repair.

Preliminary statistical analysis (ongoing) shows that the rule system employed in this work is also quite evolvable. Solutions capable of self-repair evolved with consistent predictability. In addition, the mechanisms governing these evolved phenomena were provided.

As future work, new experiments detailing how a colony of cells is able to move within an environment via chemical gradient following, despite the fact that cell movement is not directly built in (as well as the development, size regulation, and self-repair of more complex patterns) is already underway.

References

Bentley, P. J., and Kumar, S. 1999. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Banzhaf, W.; Daida, J.; Eiben, A. E.; Garzon, M. H.; Honavar, V.; Jakiela, M.; and Smith, R. E., eds., *Genetic and Evolutionary Computation Conference – GECCO-1999*, 25–43. Morgan Kaufmann.

de Garis, H. 1992. Artificial embryology: The genetic programming of an artificial embryo. In Souček, B., ed., *Dynamic, Genetic, and Chaotic Programming*. New York: John Wiley. 373–393.

Eggenberger, P. 1997. Evolving morphologies of simulated 3d organisms based on differential gene expression. In *Fourth European Conference on Artificial Life*.

Federici, D. 2004. Evolving a neurocontroller through a process of embryogeny. In Schaal, S.; Ijspeert, A.; Billard, A.; Vijayakumar, S.; Hallam, J.; and Meyer, J., eds., *From Animals To Animats 8: SAB 2004*, 373–384.

Hornby, G. S. 2003. *Generative Representations for Evolutionary Design Automation*. Ph.D. Dissertation, Brandeis University.

Hutton, J. J. 2004. Making membranes in artificial chemistries.

Ilachinski, A. 2001. *Cellular Automata: A Discrete Universe*. World Scientific Publishing Co.

Jakobi, N. 1995. Harnessing morphogenesis. Technical report, School of Cognitive and Computing Sciences, University of Sussex.

Kumar, S., and Bentley, P. J. 2003. *On Growth, Form, and Computers*. Academic Press, London UK.

Kumar, S. 2004. *Investigating Computational Models of Development for the Construction of Shape and Form*. Ph.D. Dissertation, University College London.

Lindenmeyer, A. 1968. Mathematical models for cellular interaction in development (parts i and ii). *Journal of Theoretical Biology* 18:280–315.

Miller, J. F. 2003. Evolving developmental programs for adaptation, morphogenesis and self-repair. In *Proceedings of ECAL*, 256–265.

Prusinkiewicz, P., and Lindenmayer, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag.

Prusinkiewicz, P.; Hammel, M. S.; and Mjolsness, E. 1993. Animation of plant development. *Computer Graphics* 27:351–360.

Prusinkiewicz, P. 1993. Modeling and vizualisation of biological structures. In *Proceeding of Graphics Interface '93*, 128–137.

Roggen, D., and Federici, D. 2004. Multi-cellular development: is there scalability and robustness to gain? In Yao, X., and al. ed., eds., *Proceedings of PPSN VIII*, 391–400.

Wolfram, S. 1994. *Cellular Automata and Complexity*. Addison-Wesley.

Wolfram, S. 2001. *A New Kind Of Science*. Wolfram Media, Inc.

Wolpert, L. 1998. *Principles of Development*. Oxford University Press.