Beyond the Complexity Ceiling: Evolution, Emergence and Regeneration

Julian Francis Miller[1] and Peter Thomson[2]

[1]Department of Electronics, University of York, Heslington, York, UK, YO10 5DD
jfm@ohm.york.ac.uk

[2]School of Computing, Napier University, Edinburgh, EH10 5DT, UK
p.thomson@napier.ac.uk

Abstract. We argue that there is an upper limit on the complexity of software that can be constructed using current methods. Furthermore, this limit is orders of magnitude smaller than the complexity of living systems. We argue that many of the advantages of autonomic computing will not be possible unless fundamental aspects of living systems are incorporated into a new paradigm of software construction. Truly self-healing and maintaining software will require methods of construction that mimic the biological development of multi-cellular organisms. We demonstrate a prototype system which is capable of autonomous repair and regeneration without using engineered methods. A method for evolving programs that construct multi-cellular structures (organisms) is described.

1 Introduction

In 2001, IBM launched their so-called Autonomic Computing Initiative[1] because of the growing problems associated with the complexity of modern software. They recognized the enormous costs associated with software maintenance, and correctly indicated that unless fundamental steps were taken to deal with this we would find that we would no longer be able to increase the complexity of software and have it remain reliable. To quote: "*Consider this: at current rates of expansion there will not be enough skilled I/T people to keep the world's computer systems running...Some estimates for the number of I/T workers required globally...put it at 200 million, or close to the entire population of the United States. Even if we could somehow come up with enough skilled people, the complexity is growing beyond human ability to manage it...Without new approaches, things will only get worse...*". The systems IBM are considering already utilize all the received wisdom about the process of efficient software construction (i.e. OO, reuse,

[1] http://www-306.ibm.com/autonomic/index.shtml

design patterns). Their proposed solution to this problem was to build a software construction architecture in which self-monitoring and self-healing was an integral part. The idea was to fight complexity with complexity (FCWC). There are some fundamental problems with this approach. Foremost among these, is that we believe the complexity crisis is caused by the top-down methodology of software construction. At present all the capability of software and hardware is engineered into the system at the design stage. This means that one has to understand and control all the interactions between the components. This rapidly causes a combinatorial crisis leading to enormous verification problems. Secondly, and more fundamentally the manpower required to construct ever more complex software grows with this engineered complexity. These problems will not be solved by using a cleverer *engineering* approach. The third problem is related to the concept of FCWC. It is difficult to see how engineering more complex software or hardware can possibly help reduce the growing problem of software construction and maintenance since by assumption the proposed system is even more complex. Finally it will prove to be very difficult to extract human knowledge of software and system maintenance to build a sufficiently sophisticated expert system.

In this paper we are looking beyond the immediate technological horizon towards the time when we will require software with much greater complexity than we have at present. This drive for increased complexity of software will come from the desire for it to be ever more intelligent and autonomously adaptive. Although we believe the motivation of the Autonomic Computing movement is well founded, it will ultimately fail unless attempts are made to develop a new way of constructing and refining software. This new paradigm will not only abandon the formal verification of software systems but also even have to abandon Boolean logic and other formal programming constructs at the level of the software programmer. Such software will be created in a manner more akin, for example, to the way a horticulturist might create a new type of rose. They do not need to understand the enormous complexity of a living plant but just need to know how to cross-fertilize, graft and nurture and selectively breed roses with the characteristics they require. When they prune the plant, the plant responds autonomously: seals up the cut site and starts growing again. We believe that software in the future will have to have characteristics like this. Thus, to begin the process of creating this new paradigm of software construction we have turned to nature and looked at the mechanisms of biological development.

Living systems, such as plants, do not simply exist in a constant stable and static state, but continually develop through cell replication and death. Animals, on the other hand, reach maturity - at which point growth ceases - but continue to develop through chemical and cell renewal, for example almost all proteins in the body are destroyed in hours while most cells die and are replaced (e.g. red blood cells (erythocytes) have a half life of 120

days)[2]. This impressive level of self-organisation is what the software systems of the future will have to be capable of in order to enable them to *autonomously* adapt to cope with either damage or a shifting problem definition. This may be thought of as analogous to a plant surviving and thriving within a changing environment. This paper outlines an approach for the development of a simple system that embodies these ideas of self-organisation and emergence.

2 Development, Cellular Automata and Evolutionary Algorithms

Biological development is the process that leads from a fertilized cell to an entire organism. It is the most sophisticated software and hardware construction process on the planet. The human genome contains about $3x10^9$ nucleotides each containing 2 bits of information. Yet the human body is made of roughly 5×10^{13} cells. Even if we use a conservative estimate of a cells information content, say 1 Mbit we find that the information content of the human body is roughly 10^8 times the content of the genome! There is obviously something very clever about the construction process that in our view has to be used for us to build complex software systems of the future [3]. How does nature achieve this feat of engineering? Frank M. Harold explains [17]: "*Genes specify the cell's building blocks; they supply raw materials, help regulate their availability and grant the cell independence of its environment. But the higher levels of order, form and function are not spelled out in the genome. They arise by the collective self-organization of genetically determined elements, affected by cellular mechanisms that remain poorly understood*."

The work we describe in this paper is attempting to follow nature's example and use emergence and self-organization to construct structures that have more information content than the genetic information needed to specify the software cells. It is therefore natural to turn to nature once again to find some algorithmic paradigm that is responsible for creating such exquisite systems. The Cellular automaton [36][41][42] is a computational paradigm in which a grid of identical computational units (cells) are in communication with their local neighbours. In two dimensions there are two commonly defined neighbourhoods: von Neumann and Moore. In the former case each cell communicates with four other cells: north, east, south and west, in the latter cells communicate also with north-east, south-east, south-west and north-west.

Evolutionary Algorithms have been developed from idealizations of Darwinian evolution [11][14][20][35][37]. In this paper we have applied an evolutionary algorithm to the production of computer programs. Such approaches are generally called Genetic Programming [2][25][26][27].

---

[2] Alberts et al. Molecular Biology of the Cell, 4th edition, 2002, page 1292. Proteins are synthesized in 20s to several minutes, pages 349-350. The authors were unable to obtain a source on typical lifetimes of protein molecules.

3 Related work

Fleischer and Barr created a sophisticated multicellular developmental test bed and included realistic models of chemical diffusion, cell collision, adhesion and recognition [10]. Their purpose was to investigate cell pattern generation and found that size regulation is critical and non-trivial. Eggenberger suggests that the complex genotype-phenotype mappings typically employed in developmental models allow the reduction of genetic information without losing the complex behaviour and will scale better on complex problems [9]. Bongard and Pfeifer have evolved genotypes that encode a gene expression method to develop the morphology and neural control of multi-articulated simulated agents [6]. Bentley and Kumar examined a number of genotype-phenotype mappings on a problem of creating a tessellating tile pattern [4]. They found that the indirect developmental mapping (that they refer to as an implicit embryogeny) could evolve the tiling patterns much quicker, and further, that they could be subsequently grown to (iterated) much larger sized patterns. Other researchers are more motivated by fundamental biological aspects of cell behaviour. Furusawa and Kaneko modeled cell internal dynamics and its relationship to the emergence of cell multicellularity [12]. Hogeweg has carried out impressive work in computer models of development and constructed a sophisticated model of cells (biotic) by modeling the internal dynamics by groups of cells in a cellular automaton that are subject to energy minimization [18][19]. The energy minimization automatically leads to cell movement and sorting by differential cell adhesion. The cell genome was modeled as 24 node Boolean network that defined cell signaling and adhesion. She evolved organisms that exhibited many behaviours that are observed in living systems: cell migration and engulfing, budding and elongation, and cell death and redifferentiation. Kumar has recently presented an impressive, biologically well-motivated computational development system involving a genetic regulatory network and cell receptor-mediated signal transduction [29]. Recently, many of the research contributions in computational development have been presented in a single volume [28].

A number of researchers have studied the potential of Lindenmeyer systems [30] for developing artificial neural networks (ANNs) and generative design. Boers and Kuiper have adapted L-systems to develop the architecture of artificial neural networks (ANNs) [5]. They used an evolutionary algorithm to evolve the rules of a L-system that generated feed-forward neural networks. Kitano developed another method for evolving the architecture of an artificial neural network [23] using a matrix re-writing system that manipulated adjacency matrices. Although Kitano claimed that his method produced superior results to direct methods (i.e. a fixed architecture, directly encoded and evolved), it was later shown in a more careful study that the two approaches were of equal quality [38]. Gruau devised an elegant graph re-writing method called cellular encoding [15][16]. Cellular encoding is a language for local graph transformations that controls the division of cells that grow into artificial neural networks. Others have successfully employed this approach in the evolution of recurrent neural networks that control the behaviour of

simulated insects [24]. L-systems have also been used to define three dimensional objects [21]. Jacobi created an impressive artificial genomic regulatory network, where genes code for proteins and proteins activate (or suppress) genes [22]. He used the proteins to define neurons with excitatory or inhibitory dendrites. Others evolved encoded neuron position and branching properties of axonal trees that would spread out from the neurons and connect to other neurons [7][34]. Astor and Adami have created a developmental model of the evolution of an ANN that utilizes an artificial chemistry [1].

4 The Cellular Map

In our work the software (phenotype) is represented as a set of cells arranged in a non-toroidal two-dimensional cellular automaton [32]. However there are several grids one for the software cells, the others for chemicals. The function of each cell is governed by its genotype which is a representation of a feed-forward Boolean circuit. This maps the cell's input conditions to output behaviour. Cells can be dead or alive. Each live cell sees its own state and the states of its eight immediate neighbours. It also sees the amount of chemical in the Moore neighbourhood. Using this information, the cell's program decides on the amount of chemical that it will produce, whether it will live, die, or change to a different cell type at the next time step, whether and how it will grow.

Unlike real biology, when a cell replicates itself, it is allowed to grow in any or all of the eight neighbouring cells simultaneously (this is done to speed up growth, mainly for reasons of efficiency). In all the experiments reported in this paper there are three cell types (represented by colours) and the amount of chemical is represented by an eight-bit binary number. The cell types are represented by two-bit binary codes, with 00 reserved for the absence of a cell (or a dead cell).

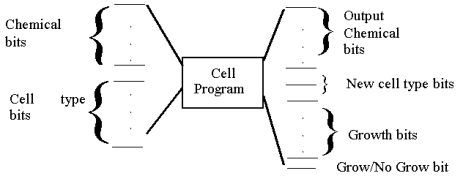

Fig. 1. The cell program's binary inputs and outputs

Only live cells have their programs executed. Initially a single cell is placed in the grid (the zygote) and one or more chemicals are set to an initial value at this location. If two or more cells decide to grow into the same location at the next time step, the last such cell in the scan path overwrites all previous growths. This was chosen as it greatly simplified the process of constructing the newly grown organism. The two dimensional grid is scanned from the top-left corner to the bottom right. The process of constructing the new organism at time t+1 from the organism at time t is the following: Every live cell from the top-left to the bottom-right has its program run (all cells run the same program). A new map (initially empty) is created and filled with cells that have either grown, or not

died, in the map at time t. After all the programs inside the living cells have been run, the map at time t+1 replaces the map at time t. The chemical map is updated in a similar manner. A depiction of the cell's inputs and outputs is shown in Fig. 1. The chemicals obey the diffusion rule (1), where c represents the amount of chemical and N the set of Moore neighbours.

$$(c_{ij})_{new} = 1/2(c_{ij})_{old} + \frac{1}{16}\sum_{k,l \in N}(c_{kl})_{old} \quad (1)$$

This simple diffusion rule was chosen to mimic chemical conservation; however the role of diffusion and its formulation remains for later investigation.

In our initial work the function of the cellular map has been chosen to be purely topological and we have evolved the program inside the cell to grow and become particular patterns such as national flags, regular spots. However in more recent work we have chosen various points on the grid to measure the chemical and use that to define the control signals for a pen. The object being to grow programs that can make the pen draw particular line figures and to recover the desired behaviour autonomously when the cellular program is subjected to severe damage.

4 Training the software: Cartesian Genetic Programming

To train the genotypes to obtain the desired phenotype we use a form of Genetic Programming (GP) called Cartesian Genetic Programming (CGP) that was developed from methods developed for the automatic evolution of digital circuits [33]. CGP represents a program or circuit as a list of integers that encode the connections and functions. The representation is readily understood from a small example. Consider the one bit binary adder circuit (Fig. 2). This has three inputs that represent the two bits to be summed and the carry-in bit. It has two outputs: sum and carry-out. CGP employs an indexed list of functions that represent in this example, various two input logic gates and three input multiplexers. Suppose that in a function lookup table AND is function 6, XOR is function 10 and MUX is function 16. The three inputs A, B, Cin are labeled 0, 1, 2. The output of the left (right) XOR gate is labeled 3 (6). The output of the MUX gate is labeled 5. The AND output is labeled 4. In Fig. 2 a genotype is shown and how it is decoded to a phenotype (the one-bit binary adder). The integers in italics represent the functions, and the others represent the connec-

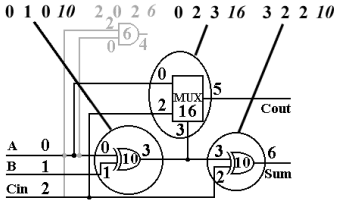

Fig. 2. Genotype and corresponding phenotype (one-bit binary adder)

tions between gates, however, if it happens to be a two input gate then the third input is ignored. It is assumed that the circuit outputs are taken from the last two nodes. The second group of four integers (shown in grey) represent an AND gate that is not part of the circuit phenotype. Since only feed-forward circuits are being considered, it is important to note that the connections to any gate can only refer to gates that appear on its left.

Typically CGP uses point mutation (that is constrained to respect the feed-forward nature of the circuit). Suppose that the first input of the MUX gate (0) was changed to 4. This would connect the AND gate into the circuit (defined by the four grey genes). Similarly, a point mutation might disconnect gates. Thus, CGP uses a many to one genotype-phenotype mapping, as redundant nodes may be changed in any way and the genotypes would still be decoded to the same phenotype. The (1+4)-ES evolutionary algorithm (below) uses characteristics of this genotype-phenotype mapping to great advantage (i.e. genetic drift). Step 3 is a crucial step in this algorithm: if more than one chromosome is equally good then the algorithm always chooses the chromosome that is not the *current_best* (i.e. equally fit but genetically different). This step allows a genetic drift process that turns out be very beneficial [40][44].

1. Generate 5 chromosomes randomly to form the population
2. Evaluate the fitness of all the chromosomes in the population
3. Determine the best chromosome (called *current_best*)
4. Generate 4 more chromosomes (offspring) by mutating the *current_best*
5. The *current_best* and the four offspring become the new population
6. Unless stopping criterion reached return to 2

The mutation rate is defined to be the percentage of each chromosome that is mutated in step 4. In all the experiments described in this paper only four kinds of MUX logic gates were employed defined by the expression f(A,B,C)=AND(A, NOT(C)) OR AND(B, C). The four types correspond to cases where inputs *A* and *B* are either inverted or not. The program outputs are taken from the rightmost set of consecutive nodes.

5 Demonstration tasks

In the biological development of organisms cells have to behave differently according to their position within the organism. Lewis Wolpert [43] proposed that this positional information might be laid down in the formation of chemical gradients relative to organism boundaries. Cells might respond differently according to threshold concentrations. He likened the problem to one of growing a French Flag; the developmental method of construction would be able to produce a recognizable flag of arbitrary size. This illustrates an important property of developmental systems in that they are scale free (i.e. there is no relationship between the genotype size and the size of the phenotype). Wolpert's model was one of the inspirations for the task the maps of cells were to achieve. We defined two

tasks. The first was to produce a growing and always recognizable German flag. The second was to produce a growing then maturing French Flag. In the first experiment target maps representing German and French flags were defined. These were used to compare the evolved cellular map at particular times in the development to the target map. The evolved organism and target organism were compared cell by cell and a cumulative score of correctness was calculated. This was the fitness of the cell genotype used in the evolutionary algorithm. In the German flag experiment the cell Cartesian program was allowed 200 nodes, 20 evolutionary runs with 30,000 generations were carried out. The target maps were a small German flag at iteration 4 and a slightly larger flag at iteration 6. There was a single chemical initialized at the maximum value (255) at the location of the seed cell. The third fittest solution (gGf11) found is shown in Fig. 3. This always looks like a growing German flag. The two fitter solutions looked better at iterations 3 and 5 but rapidly lost their German flag like appearance. The temporal behaviour of the second fittest solution (gGf0) is shown in Fig. 4. In section 7 we will examine the behaviour of a graft of both these maps.

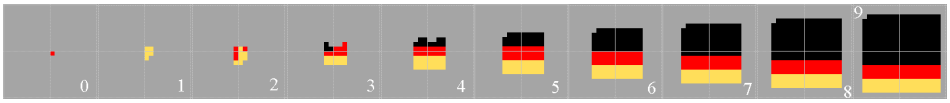

Figure 3. Growth of "third best" multicellular program (gGf11) from a red seed cell (0) to a growing German flag.

Figure 4. Growth of "second best" multicellular program (gGf0) from a red seed cell.

In experiment 2, the target maps were a fixed size French flag and fitness was calculated at iterations 7, 8, 9, 10. The cell's Cartesian program was allowed 300 nodes and once again 20 runs of 30,000 generations was carried out. The best solution is shown in Fig. 5. The cellular map stops changing at iteration 8. We examine its behaviour under damage in a sister paper [31] where we will see that it has remarkable powers of autonomous regeneration reminiscent of the pond organism hydra which can reform itself when its cells are dissociated and then re-aggregated in a centrifuge [13].

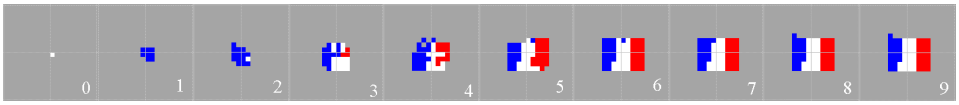

Figure 5. Growth of multicellular program from a white seed cell (0) to a mature French flag at iteration 8.

6 Results: Emergent regeneration

We will examine the behaviour of the growing German flag (gGf11) described in the previous section. Fig. 6 shows what happens when a large hole is cut out of the growing German flag (upper) and when it is subjected to substantial random damage (lower). We see that the cellular program gradually fills in the hole and recovers though it retains "scarring".

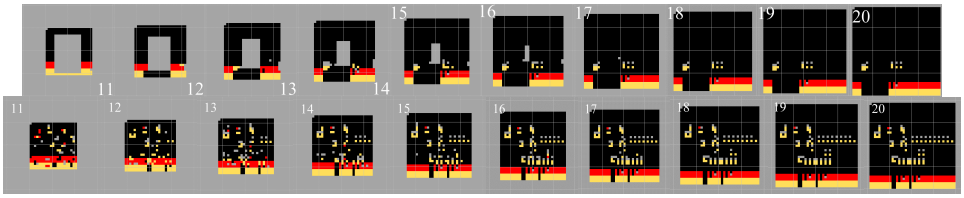

Figure 6. Autonomous recovery of multicellular program for growing German flag from badly damaged initial conditions.

In many other cases we found that the growing German flag can cope with a great variety of damage almost always recovering and rebuilding the German flag. The regenerative powers of the French flag are also impressive and examined in the sister paper [31]

7 Grafting software

Motivated by the rose-breeding metaphor we have investigated the behaviour of cellular maps that have been joined together involving distinct and independently evolved genotypes (i.e. grafting). When a particular cell program decides to grow it replicates its own genotype. The German flag cellular map at iteration 8, gGf11 was bisected. The cells on the right half had their genotypes replaced with those of gGf0 (the cell states were left untouched). The cellular chemical map was left untouched corresponding to gGf11.

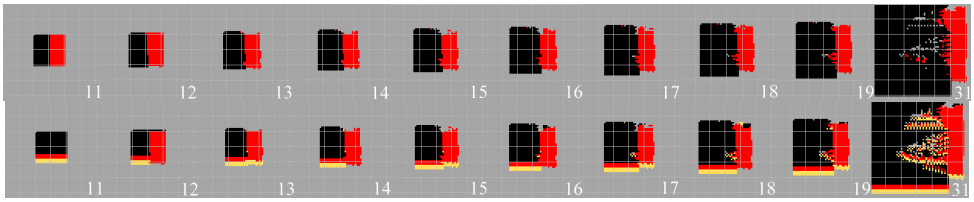

Figure 7. Two different genotypes gGf11 and gGf0 are grafted together at iteration 11, The upper shows the location of each genotype over time and lower shows the phenotype.

Fig. 7 shows what happens to the genotypes and phenotype over time. The black region in indicates the gGf11 genotypes and the red region the gGf0 genotypes. The graft of the two cellular maps behaves in a stable way, with each genotype dominating on each side of the map and mixing taking place in the region around the graft site. Other experiments have been conducted where cells of gGf0 are randomly placed into the German flag of gGf11, early indications show that gGf11 tends to dominate and smother the foreign cells. This might have implications for future work on software immunity in developmental systems.

8 Conclusions and further work

It is easy to argue that the results presented in this paper do not address practical problems however, the research is at an early stage and need not immediately be tested on industrial problems just yet. However, the next phase of the work is to apply these emergent developmental systems to a more practical problem. Work has already begun on the use of such systems to control a robot. The idea being to map inputs (from sensors) and outputs (to effectors) to simulated chemical sources. The developmental genotypes will be evolved to control the robot. Once the control program is sufficiently good, it will be damaged. Investigations will be carried out to see if the developmental control program can *autonomously* recover the desired robot behaviour. Work is also continuing in looking at the application of these ideas to communication network control and balancing. We have discussed the idea that conventionally constructed software will reach a complexity ceiling in the near future and that bottom-up, bio-inspired ideas for new software paradigms will become increasingly important if we are to construct ever more complex, intelligent and autonomous systems. We have shown how it is possible to create emergent systems inspired by developmental biology that can be trained to achieve a higher level goal. It turns out that systems produced in this way are highly robust to damage and are able to regenerate themselves. We feel that software systems that are capable of self-repair will have to consider such approaches. The construction of useful developmental systems is really in its infancy, but we hope to have demonstrated their potential in the task of creating self-maintaining, self-repairing software that may be able to utilize complexity in ways that conventional software designers are unable to.

References

1. J. C. Astor and C. Adami, "A Development Model for the Evolution of Artificial Neural Networks", Artificial Life, Vol. 6, pp. 189-218, 2000.
2. W. Banzhaf, P. Nordin, R. Keller, and F. Francone. Genetic Programming - An Introduction. Morgan Kaufmann, San Francisco, CA, 1998.

3. W. Banzhaf and J. F. Miller, The Challenge of Complexity. In: A. Menon (ed.): Frontiers of Evolutionary Computation. Kluwer Academic Publishers, 2004
4. P. Bentley and S. Kumar, "Three ways to grow designs: A comparison of embryogenies for an Evolutionary Design Problem", in Proceedings of the Congress on Evolutionary Computation, IEEE Press, pp. 35-43, 1999.
5. E. J. W. Boers and H. Kuiper, "Biological metaphors and the design of modular neural networks", Masters thesis, Department of Computer Science and Department of Experimental and Theoretical Psychology, Leiden University, 1992.
6. J. C. Bongard and R. Pfeifer, "Repeated Structure and Dissociation of Genotypic and Phenotypic Complexity in Artificial Ontogeny", in Spector L. et al. (eds.) Proceedings of the Genetic and Evol. Comput. Conference, Morgan-Kaufmann, pp. 829-836, 2001.
7. A. Cangelosi, D. Parisi and S. Nolfi, "Cell Division and Migration in a ' Genotype' for Neural Networks", Tech. report PCIA-93, Inst. of Psych., CNR, Rome, 1993.
8. F. Dellaert, "Toward a Biologically Defensible Model of Development", Masters thesis, Dept. of Computer Eng. and Science, Case Western Reserve University, 1995.
9. P. Eggenberger, "Evolving morphologies of simulated 3D organisms based on differential gene expression", Proc. Of European Conf. on Artificial Life, pp. 205-213, 1997.
10. K. Fleischer and A. H. Barr, "A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis", in Langton C. G (ed.) Proceedings of the 3rd Workshop on Artificial Life, Addison-Wesley, pp. 389-416, 1992.
11. D. B. Fogel, Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, IEEE Press, 1995
12. C. Furusawa and K. Kaneko, "Emergence of Multicellular Organisms with Dynamic Differentiation and Spatial Pattern", in Adami C. et al. (eds.) Proceedings of the 6th International Conference on Artificial Life, MIT Press, 1998.
13. A. Gierer, S. Berking, H. Bode, C. N. David, K. Flick, G. Hansmann, H. Schaller and E. Trenkner, Regeneration of hydra from reaggregated cells, Nature New Biology, Vol. 239, pp. 98-101, 1972.
14. D. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
15. F. Gruau, "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm", PhD thesis, Ecole Normale Supérieure de Lyon, 1994.
16. F. Gruau, D. Whitley and L. Pyeatt, "A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks", in Proc. of the 1st Annual Conference on Genetic Programming, Stanford, 1996.
17. F. M. Harold, The Way of The Cell: Molecules, Organisms and the Order of Life, Oxford University Press, 2001.
18. P. Hogeweg, "Evolving Mechanisms of Morphogenesis: on the Interplay between Differential Adhesion and Cell Differentiation", J. Theor. Biol., Vol. 203, pp. 317-333, 2000.
19. P. Hogeweg, "Shapes in the Shadow: Evolutionary Dynamics of Morphogenesis", Artificial Life, Vol. 6, pp. 85-101, 2000.
20. J. H. Holland. Adaptation in natural and artificial systems. MIT Press, Cambridge, MA, 1992.
21. G. S. Hornby and J. B. Pollack, "The Advantages of Generative Grammatical Encodings for Physical Design", in Proceedings of the Congress on Evolutionary Computation, IEEE Press, pp. 600-607, 2001.

22. N. Jacobi, "Harnessing Morphogenesis", Research Paper 423, COGS, Univ. of Sussex, 1995.
23. H. Kitano,"Designing neural networks using genetic algorithms with graph generation system", Complex Systems, Vol. 4, pp. 461-476, 1990.
24. J. Kodjabachian and J-A Meyer, "Evolution and Development of Neural Controllers for Locomotion, Gradient-Following and Obstacle-Avoidance in Artificial Insects", IEEE Transactions on Neural Networks, Vol. 9, pp. 796-812, 1998.
25. J. R. Koza, Genetic Programming: On the programming of computers by means of natural selection. MIT Press, 1992
26. ___Genetic Programming II: Automatic Discovery of Reusable Subprograms. MIT Press, 1994
27. ___Genetic Programming III, Morgan Kaufmann 1999.
28. S. Kumar and P. Bentley, On Growth, Form and Computers, Academic Press, 2003.
29. S. Kumar, "Investigating Models of Development for the Construction of Shape and Form", Doctoral thesis, UCL, January, 2004.
30. A. Lindenmeyer, "Mathematical models for cellular interaction in development, parts I and II", Journal of Theoretical Biology, Vol. 18, pp. 280-315, 1968.
31. J. F. Miller, "Evolving a self-repairing, self-regulating French flag organism", in Proceedings of GECCO, 2004.
32. J. F. Miller, "Evolving Developmental Programs for Adaptation, Morphogenesis, and Self-Repair", In W. Banzhaf et al (ed.), Proceedings of Advances in Artificial Life (ECAL 2003), Springer Lecture Notes in Artificial Intelligence, Vol. 2801, pp. 256--265, 2003.
33. J. F. Miller and P. Thomson, "Cartesian genetic programming", in Proceedings of the 3rd European Conf. on Genetic Programming. LNCS, Vol. 1802, pp.121-132, 2000
34. S. Nolfi and D. Parisi,"Growing neural networks", Technical report PCIA-91-15, Institute of Psychology, CNR, Rome, 1991.
35. I. Rechenburg. Evolutionsstragie "93. Frommann Verlag, Stuttgart, 1994.
36. Alexander Schatten, Cellular Automata, Digital worlds http://www.ifs.tuwien.ac.at/~aschatt/info/ca/ca.html, 1999.
37. H.-P. Schwefel, Evolution and Optimum Seeking. Sixth-Generation Computer Technology Series. John Wiley & Sons inc., New York, 1995.
38. A. A. Siddiqi and S. M. Lucas, "A comparison of matrix rewriting versus direct encoding for evolving neural networks", in Proceedings of the IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 392-397, 1998.
39. K. Sims, "Evolving 3D morphology and behaviour by competition", in Proceedings of Artificial Life IV, pp. 28-39, 1994.
40. V. K. Vassilev and J. F. Miller, "The Advantages of Landscape Neutrality in Digital Circuit Evolution", 3rd Int. Conf. on Evolvable Systems: From Biology to Hardware, LNCS, Vol. 1801, Springer-Verlag, pp. 252-263, 2000.
41. J. Von Neumann in (A. Burks ed.), Theory of Self-Reproduction Automata, University of Illinois Press, 1966.
42. S. Wolfram, A New Kind of Science, Wolfram Media Incorporated, 2002.
43. L. Wolpert, Principles of Development, Oxford University Press, 1998.
44. T. Yu and J. F. Miller,"Neutrality and the evolvability of Boolean function landscape", in Proceedings of the 4th European Conference on Genetic Programming, Springer-Verlag, pp. 204-217, 2001