Evolution of Self-Assembling Patterns in Cellular Automata using Development

Can Öztürkeri and Colin G. Johnson

University of Kent, Canterbury, UK

Published in *Journal of Cellular Automata,* 6(4):257-300, 2011

Corresponding author: Colin Johnson, C.G.Johnson@kent.ac.uk

Evolution of Self-Assembling Patterns in Cellular Automata using Development

Can Öztürkeri and Colin G. Johnson

University of Kent, Canterbury, UK

Abstract. This paper is concerned with the application of ideas inspired by developmental biology to the evolution of cellular automata rules using genetic programming. In particular, it is focused on so-called *self-assembling* patterns. The application of development in computing is reviewed, as is the evolutionary technique used in the paper—*Cartesian Genetic Programming.* A novel developmental algorithm, termed the *Developmental Cellular Model* is introduced, and five sets of experiments on various self-assembly problems are detailed and the results examined.

Keywords. Self-Assembly, Emergence, Genetic Programming, Development, Bio-Inspired Computing, Evolvability.

1. Introduction

A long-term aim in computer science is to create complex computational systems that are able to assemble and repair themselves. This paper makes a contribution towards this aim by describing a number of experiments into the evolution of patterns in an extension of cellular automata (CA). The ideas that underpin these extensions to the traditional CA model are the notion of *development* in biology, i.e. the process by which an organism grows and forms structure from a single cell; and the computational process called *Cartesian Genetic Programming*, which is an algorithm for the automated generation of program code.

In particular, we aim to experimentally demonstrate that this class of computational developmental systems on a cellular structure are *naturally* fault-tolerant and evolvable. By naturally we mean that the system is not fault-tolerant by explicit design, nor due to evolutionary pressure, but rather that the framework in which the system is built should ensure a high probability of fault-tolerance as an emergent property.

The main contributions of the paper are a method for encoding an evolving program, which we term the Developmental Cellular Model (DCM); and, a set of five series of experiments in which we carefully investigate the evolution of programs using this method and evaluate it.

This paper is structured as follows. Section 2 gives a review of relevant literature. Section 3 discusses the methodology used for our experiments. Section 4 presents the core experimental results and discussion thereof. Finally, Section 5 provides concluding discussion and ideas for future work.

2. Literature Review

In this section we review the three topics that are of most relevance to the work in this paper: cellular automata and variants thereof; artificial development, with a particular emphasis on developmental systems that generate patterns and which self-assemble; and, Cartesian Genetic Programming (CGP), an algorithm for evolving programs, which we use in this paper to evolve developmental systems for self-assembly.

2.1 Cellular Automata

Cellular Automata (CAs) are dynamical systems in which state, space, and time are discrete. The canonical CA system consists of a set of identical finite state automata (called an *update rule*), distributed as nodes on a regular grid. Each cell within the grid can be in one of a finite number of states. This state is updated at each discrete timestep in accordance with the update rule, which takes as its input the current state of the cell and the state of other cells. The cells that provide this input to the update rule are called the *neighbourhood* of the cell, and are typically cells that are adjacent within the grid. An overview of the important properties of CAs can be found in Wolfram [1].

The earliest CA models are found in work by von Neumann [2], which were focused on developing abstract models of self-reproduction. The initial idea was to build a model of a three dimensional factory with a robot building another robot. The final models developed distilled this down to a two-dimensional grid model, based on a model developed by Ulam [3] for the growth of crystals using a simple lattice network. Ulam's crystal growth model can also be considered a form of self-replication, however; it is surely not self-reproduction. Von Neumann's simplified CA model had 29 states for each cell and had complicated rules specifically set up to simulate operations of computers and mechanical devices, which could algorithmically implement a universal copier and constructor with which he was able to mathematically prove that machine self-reproduction was possible. Since his model used a configuration string to create any structure including the constructor itself, it was capable of creating machines of any complexity thereby also answering the question of whether it is possible for a machine to construct another machine as complex as itself.

Burks later argued that self-replicating automata should also be capable of universal computation [4]. In 1973 Herman showed a very simple self-reproducing CA structure that was also capable of universal computation [5]. Since the universal Turing machine is actually a very simple device, copying its operation with CA was simpler than von Neumann's 29-state, 200,000 cell machine.

The Game of Life was a two-state two-dimensional CA with just a few simple rules [6]. Although the system was extremely simple, it fluctuated between apparent randomness to order. Moving cell groups called gliders were frequently observed. After much effort and with the help of the interactions of these gliders with one another, it was shown that the Game of Life was capable of emulating a universal Turing machine.

The next main body of work in this area began with the work of Wolfram [7,8,9], who investigated an unknown class of simple, one dimensional CA, which he termed *Elementary Cellular Automata*. Wolfram suspected that the complex behaviour displayed by these simple rules closely resembled the complexity in natural processes. He extended his earlier results to a broader range of systems [10].

With the advance of electronics and computer technology, the use of cellular systems for parallel and robust computing is becoming commonplace. Specifically, CA have found uses in error correction, cryptography, simulation of biological phenomena, calculation of differential equations and thermodynamics.

2.2 Artificial Development and Self-Assembly

Developmental biology is the study of how biochemical processes in a growing organism give rise to a complex multicellular organism. An overview of development in biology can be found in books such as that by Wolpert [11]. In broad terms, development consists of five processes [12]; these occur roughly in this order over time, but there is some overlap:

- Embryonic Cleavage, where the fertilised cell divides into a ball of undifferentiated cells, which then form a hollow ball.
- Pattern Formation, where this ball of cells get a notion of position via a process of chemical diffusion.
- Morphogenesis, where a broad three-dimensional shape is created.
- Cellular Differentiation, whereby the different cells take on different types depending on their position within the three-dimensional structure.
- Cellular Growth, whereby the now-differentiated cells grow and divide.

These ideas have been used in a number of ways to inspire novel computational methods. In this section we will review such approaches.

Development-Inspired Computing

A*rtificial development* entails the application of principles and ideas from developmental biology into computer science. The term artificial development indicates all artificially created systems (e.g. mechanical, electronic, or software) where a developmental process is involved. In this overview we focus on issues of complexity, genotype-phenotype mappings in evolutionary algorithms and the application of developmental methods in computer software

The human genome contains around 3×10^9 nucleotides, and the number of cells in a human body is about 5×10^{13}. Even if we use a conservative estimate of a cell's information content, say 1 Mbit, the information content of the whole human body is still 10^8 times the genome [13]. This makes the developmental system the most sophisticated manufacturing system in the world; it has been suggested that a suitable abstraction of this process could solve many of the complex software/hardware design and construction problems [14]. Unfortunately, how this is exactly done is largely unknown.

Traditionally, complex human-built architectures are made by combining simpler modules. These modules are thoroughly tested and therefore the end product made from these is expected to perform to the same standards. However, as the complexity of these architectures grow, for instance the number of components on a silicon chip and their connections grow, or the number of lines in a computer program increases, the design process gets harder and harder. There will eventually be an upper limit to how complex a human built architecture can be. Another problem, due to the complexity of the technology, is the ever-increasing need for its maintenance due to the possibility of errors in its many components.

To solve these problems, we need methods for automatically creating solutions for problems. For example, evolutionary algorithms work with a genotype-to-phenotype mapping process. The genotype refers to a set of parameters that represent the solution, and the phenotype refers to the actual solution. The function or process that turns the genotype into the phenotype is called the mapping function. However, most evolutionary algorithms fail to scale up to more complex architectures. This happens since the data required to represent these architectures also increases and eventually it will get to a point where it cannot be efficiently stored or evolved. This is because most evolutionary algorithms use a direct (one-to-one) genotype-phenotype mapping. The terms *direct* or *one-to-one* indicate that each parameter in the genotype codes for one component in the phenotype. Despite important advantages seen in natural organisms, the only class of mainstream bio-inspired algorithms that makes the crucial distinction between the genotype and the phenotype are genetic algorithms. Even in that case, the mapping is fixed and not a true developmental process.

Development combined with evolution provides an appealing approach for solving complex problems with evolutionary algorithms. Developmental systems use complex genotype-phenotype mappings, which, unlike direct mappings, use a set of instructions to construct the phenotype. This enables smaller, more compact genotypes and greater complexity. Developmental systems have a number of characteristics:

- The distinction between its genotype and its phenotype. However, this distinction is not enough to classify a system as developmental.
- In a developmental system it should be possible that a change in a gene could affect multiple phenotypic traits or a single phenotypic trait could be controlled by the interaction of two or more genes.
- A developmental system should have a modular nature. For instance it may be constructed from cells or some other form of building blocks. Just as in nature these cells or building blocks that make up a developmental system can be different from each other with separate functions and morphologies.
- During the construction of the organism, the modules and the programming that underlies the developmental process interact with each other leading to an increase in complexity. In biological organisms this behaviour does not stop after the organism reaches an adult size. This research is based on this assumption that the developmental system and the system that maintains the size and functions of an adult organism is the same.
- Unlike most direct mappings, with developmental mappings the timing of an instruction to be carried out is as important as what the instruction is. This has analogies in biology: for example, the production of a protein for too long a time can have dramatic effects on the organism.

Computer scientists have been looking at biology for inspiration on a number of computational design problems. They have seen evolution's ability to create highly complex, yet robust organisms and applied the principles of natural selection to computer science; developmental systems add another layer of sophistication to such problem-solving methods.

Development in its most general sense has been applied to a wide range of problems ranging from biological modelling [16,17] to abstract developmental systems [17,18,19] via morphogenesis [20,21,22]. The specific reasons for this interest are multiple, ranging from genotype compactness, adaptability and scaling problems [22,23] to the biological understanding of development [24,25]. Other studies have exploited development as a means for self repair [18,26,27,28]. Therefore artificial development presents ways to achieve complexity and create new, powerful, robust computational algorithms.

Developmental Pattern Generation

The work in this paper is concerned with *developmental pattern generation*. In this section, we review work in this area, in particular that concerned with *self-assembly* of structures.

One of the first works on understanding the principles of development was conducted in the 1950s by Alan Turing [29]. The mathematical model he created involved a set of partial differential equations and modelled the behaviour of what he termed as reaction-diffusion systems. The chemicals in the system, *morphogens* (generators of morphology/pattern), reacted with each other and diffused to create spatial concentration patterns. Since Turing, much work has applied reaction-diffusion systems to explain patterns seen in biological organisms [30,31,32].

In biological development, cells have to function differently, or differentiate based on their location within the organism. Wolpert proposed that the positional information might be given by chemicals or morphogens within the cells and the diffusion of these chemicals towards the boundaries of the organism [11]. Furthermore, other chemicals called maternal factors are deposited into dividing cells starting with the zygote. The unequal division of the maternal factors provide positional information to the cells. Wolpert suggested that the problem is similar to growing a French flag, where each cell determines what colour it needs to become based on the positional information gained from the maternal factors [33]. This developmental method should be able to grow recognisable flags of *arbitrary* size without any increase in the genotype.

As suggested by Wolpert the growth of a French flag pattern has been used as a benchmark problem for many of the experiments presented in this thesis. Similar morphologies have been used as benchmarks by a number of people [21,34,35].

In 2001 IBM launched their *Autonomic Computing Initiative* [36] to solve the problems associated with increasing complexity in modern software. They realized the large cost of software maintenance, and indicated that we need to take fundamental actions to be able to increase the complexity of our software and keep it reliable. To that end, they have proposed a novel software construction architecture—fight complexity with complexity (FCWC)—in which self-monitoring and self repairing is an integral part [13].

Miller and Thomson argue that IBM's FCWC concept is fundamentally flawed and that we need to take more fundamental actions to solve this problem [13]. They correctly indicate that the complexity crisis is caused by the top-down methodology of software construction, since with this methodology one needs to understand and control all the interactions between the components. This rapidly causes a combinatorial crisis leading to huge verification problems. They further argue that one will not need to know everything about the software and that such software will be evolved or grown rather than constructed component by component [13]. This methodology is similar to selective breeding, or grafting of plants to get an organism with favourable properties.

Miller has conducted one of the few successful works involving the benchmark French flag pattern. He has managed to evolve CGP (Cartesian Genetic Programming) programs running inside cells of a cellular automata grid [21,37]. Miller used in his model cells that have 2-bit states that determine whether they are white, red or blue. The white cells are stem cells that can differentiate into other cell types. The model also adopts a chemical diffusion system, where each cell determines how much chemical it is going to produce at each time step, and this chemical diffuses into the CA grid. The evolved organisms create growing patterns of French flags and other regular patterns.

Another interesting study was made by Rothermich and Miller investigating the emergence of multicellularity [38]. The model used was similar to the one with the French flags, where a CGP program runs in each cell in a CA grid.

Liu et al. [26] took an approach that is somewhat similar to Miller's original approach. They managed to evolve a stable 6×6 French flag organism on a hardware simulation similar to an FPGA. Similarly the model uses CGP and also includes a chemical diffusion system.

In another approach, de Garis has used rule based decision makers to generate growing patterns of cells in a classical CA environment [35]. He has managed to evolve rule based systems for convex shapes earlier [39], and in his 1999 paper he attempted to create non-convex shapes but found that they are difficult to evolve. He has managed to evolve CA rules for basic 'L' shaped patterns (non-convex) with correctness of up to 80% of target pattern. The generation of the 'L' pattern involves dividing it into two parts *A* and *B*. The genotype in this experiment consists of two genes which de Garis termed *operons*. One operon controls the growth of part *A* and the other operon controls the growth of part *B* of the pattern.

The system starts with the seed cells and the first operon instructs them to grow until they fill part *A*. After that, the first operon switches off and the second operon kicks in to instruct the cells at the border of part B to grow.

Operons have one condition block and one function block. If a condition is met (which, in this case are the states of the cells around the origin cell) the corresponding action is triggered. Later in his paper, de Garis introduced a technique he termed *evolution shaping*, which is setting mid goals to evolution to make it easier to evolve. Shaping was applied to the evolution of the 'L' pattern by evolving the first operon until it could grow part *A* well, and then used the converged population as the initial population for the evolution of the second operon. This resulted in an increased success in achieving the target pattern, however, still not 100%. To increase the model's ability in evolving non-convex shapes, de Garis later included mechanisms such as a chemical gradient, and generation count that provides positional information to the cells.

Fleischer and Barr developed a very sophisticated multicellular developmental test bed that includes realistic models of chemicals diffusion, cellular movement, collision, drag and adhesion [40,41]. Their purpose was to investigate the development of cell patterns, and examine the effects of mechanical forces on cells during morphogenesis. Fleischer's first attempt involved evolution of artificial neural networks, but he later changed the direction of his research towards the investigation of multiple mechanisms of development during morphogenesis. His 1995 paper is one of the best works investigating the multiple mechanisms of development [42].

Fleischer and Barr's developmental model is mostly based on differential equations with rules that are used as decision makers within cells. The genotype consists of parameters for a set of these conditional differential equations. The genotype is designed to be used with an evolutionary algorithm. The equation set controls the cellular activity such as growth, movement, division, death and external effects such as variation of chemical concentration, and cell signalling.

Their experiments show that it is possible to evolve developing organisms that have specific shapes. However, they also note that the design of the genotype that develops into the specific shape is difficult. The developmental test bed is a plausible model of biology, and it reproduces many developmental phenomena; however, the model was not used for biological simulation. Their experiments show that stability or regulation of the size of the pattern is important, and difficult to achieve. They also note that developmental models are more robust to errors.

A different kind of sophisticated, biologically realistic model was created by Eggenberger [43,44,45]. The model is based on a genetic regulatory network system that Eggenberger termed as differential gene expression. Using this model he has evolved developing cellular patterns of three dimensions, which is a considerably harder task than with patterns of two dimensions.

Eggenberger's model uses a realistic three-dimensional cellular model that uses a protein concentration based gene expression mechanism to control cell functions such as division, death, movement, differentiation, and protein synthesis. The system has other features like cell signalling, morphogens, and the latest version also includes mechanical forces [44,45]. The model uses Rechenberg's evolutionary strategy (ES) for evolving the organisms [46]. Eggenberger suggested that the complex genotype phenotype mapping usually seen in developmental systems allow the reduction of the size of the genotype without losing the complexity of the phenotype and that the organisms genetic information does not necessarily need to grow as the number of cells it has increases [43].

Another sophisticated model of development was created by Hogeweg using cells with complex internal dynamics in cellular automata [15,47]. Furusawa created a model of internal cell dynamics and studied the emergence of multicellularity [48]. Federici evolved and used an ANN controlling the cells of a cellular automaton [49]. He managed to get his system to develop into several target patterns. Bentley and Kumar have explored a number of embryogenies including a developmental one, on a tessellating tile pattern problem [50]. They found that their indirect developmental mapping evolved correct patterns much quicker than their other embryogenies. Harding and Miller also compared direct and indirect embryogenies [51].

2.3 Cartesian Genetic Programming

Genetic Programming (GP) is the application of evolutionary algorithms to the evolution of computer programs and similar executable structures. An overview of GP methods can be found in the books by Poli et al. [52] and Banzhaf et al. [53].

In this paper we use the variant known as *Cartesian Genetic Programmming* (CGP). This is a specific kind of Genetic Programming, which was originally designed to evolve digital circuits [37] and thus is well suited for problems involving binary inputs and outputs. CGP uses nodes in a directed graph to represent a program. A CGP genotype is a list of integers that represents the connections and functions of the graph nodes. Contrary to the tree structure of Genetic Programming, this representation results in a genotype that needs to be decoded into the phenotype (executable program). Genetic operators for tree based GP are applied directly to the phenotype whereas in the case of CGP they are applied to the genotype. This property alone helps combat the detrimental effects of bloat as unused code would not be translated into the phenotype and thus is not executed.

Genotype-phenotype mapping in CGP does not require all nodes to be connected to each other; this results in a bounded, variable length genotype [54]. This upper limit to the genotype size limits the search space; however, it also ensures that CGP never suffers from the bloat problem. It has been suggested that the unused nodes that are left due to the variable-sized genotype combined with the evolutionary algorithm used increases the algorithm's search ability considerably due to the neutrality effect [37,55]. The neutrality effect is caused by neutral mutations that have no effect on the fitness of the organism. While adaptive mutations help the system to exploit beneficial changes, the neutral mutations enhance the exploratory capabilities of the algorithm [56]. Therefore CGP has a good balance between exploration and exploitation. Miller has suggested that CGP algorithm has a natural tendency to compress the genotype as well as to utilize unused, dormant nodes when necessary [57]. It has also been shown that CGP is at least as good at finding solutions as GP with Automatically Defined Functions [58] over a series of problems [37,59].

CGP nodes can implement a range of functions. The number of inputs a node has is determined by the node's function. CGP nodes take their inputs from the output of a *previous* node or from the system inputs. This method of connectivity forces the system to form a feed-forward acyclic graph. The connections and the functions of the nodes are sequentially indexed by integers and form the CGP genotype. The system inputs are numbered from *0* to *k-1* where *k* is the number of system inputs. The nodes (Node ID) are then numbered sequentially from *k* to the user-determined size of the genotype. At the end of the genotype there is a sequence of *n* integers determining the connections of the outputs of the system where *n* is the number of system outputs. In figure 1 a sample CGP genotype and its corresponding CGP circuit visualization is shown.

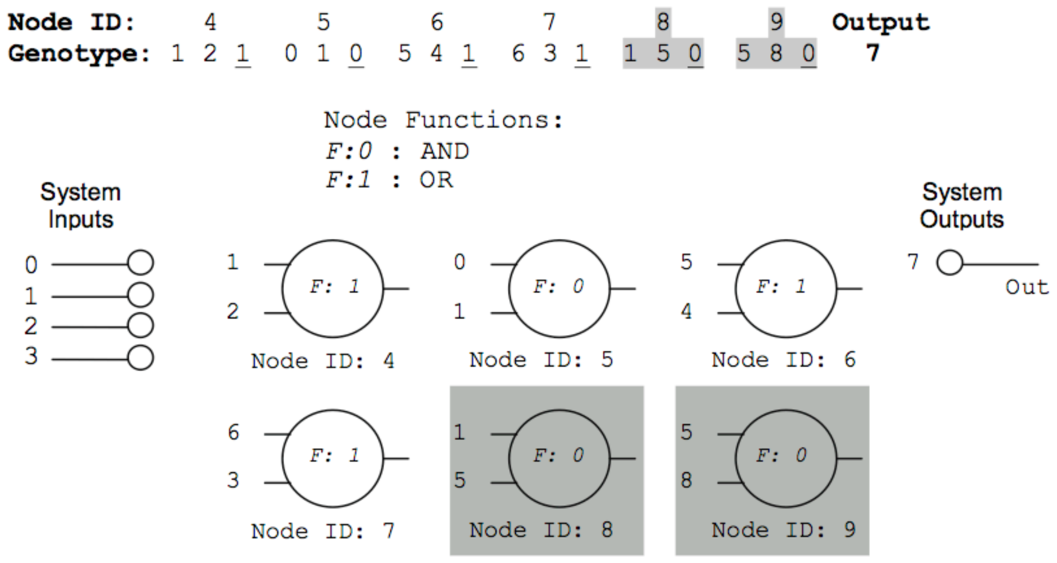


Figure 1. This is a simple logic circuit consisting of three OR and three AND gates. The possible CGP representation and chromosome for this circuit is given above. The first node (Node ID: 4) has inputs connected to system inputs 1, 2 and it implements an OR (F: 1) function. The second node (Node ID: 5) is similar, its inputs are connected to 0, 1 and it implements an AND (F: 0) function. The third node (Node ID: 6) is connected to the second and the first nodes' outputs (5, 4) and it implements an OR (1) function. The fourth node (Node ID: 7) is connected to the third node and the system input 3 (6, 3) and it also implements an OR function. The system's output is connected to the fourth node's output (Node ID: 7). As you can see, although they are connected to each other, the nodes with the grey shading are not connected to the system output. Because of this they are considered as dormant nodes or junk DNA.

The grey areas that are shown in the genotype and the CGP circuit visualization indicate dormant nodes or junk DNA. Although the dormant nodes are connected to each other and also to active nodes, they are not connected to the system output. Since the conversion of the CGP program to the executable code starts from the outputs and moves towards the system inputs, the dormant nodes are never executed, and thus unlike in regular GP they are less of a drain in system resources. The resulting phenotype from the above sample genotype is given in figure 2. We can see that nodes 8 and 9 are not translated into the phenotype.

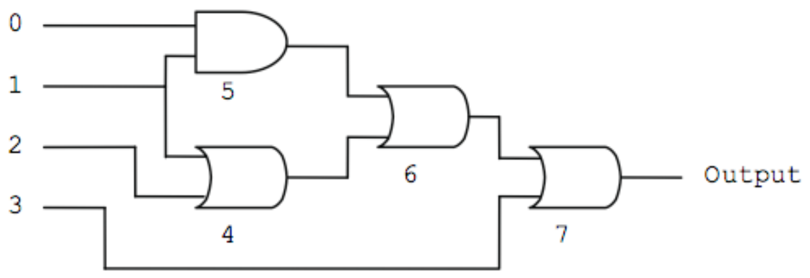


Figure 2. This digital circuit is the phenotypic representation of the CGP genotype given in figure 2.8. The numbers from 0 to 3 indicate the system inputs and the numbers from 4 to 7 indicate the *node IDs*.

Each CGP node, as it is used in the experiments in this paper, has three inputs and one output, and is one of four types of basic operators: if *input*0 then output *input*1 else output *input2*; if not *input*0 then output *input*1 else output *input2*; if *input*0 then output not *input*1 else output *input2*; if *input*0 then output *input*1 else output not *input*2. As highlighted previously by Miller, this allows for any boolean function of three variables. The maximum number of nodes available for each program is fixed to a number as specified in the experimental parameters, and these are conserved in the genetic material whether used or not.

CGP is used, as is often the case, with a small population and a large number of generations. One has to keep in mind that the number of sampled solutions is still extremely small. The population size that is most commonly used with CGP is only 5. Only the best individual is selected, and usually 4 new mutants are created from it to maintain the population size. This evolutionary algorithm is known as the 1+λ evolutionary strategy where λ=4. The algorithm is detailed in figure 3.

```
1  Generate initial Population at random with size n
2  Repeat
3    Evaluate the fitness of each individual
4    Select parent from the population based on its fitness and using the following
     rules:
     •  If there is only one individual with a higher fitness, that one is
        selected
     •  Otherwise an offspring is selected randomly among the individuals with the
        highest fitness
5    Generate the new population by mutating the selected individual. The new
     population is made from the selected individual and n-1 offspring.
6  Until finished
```

Figure 3. Pseudo-code for the CGP algorithm.

In accordance with the description of the neutral effect described earlier, if a new individual has the same fitness as the current best, it replaces it. This is in great part imposed onto us by the very small population size. Mutation is thus the only genetic operator used. The mutation operator mutates an equal percentage of the following: Inputs to the CGP nodes, the function type of the nodes, and the output table that is separate. The only thing worth mentioning here is that the mutation operator makes sure a cell gets its inputs from previous cells or system inputs, entailing a loop free, easily executable organism. We refer the reader to the earlier papers by Miller and Thomson [37,60] for a more extensive explanation of the generic principles.

3. The Developmental Cellular Model

In this section we introduce the Developmental Cellular Model (DCM), which is used as the experimental platform for all of the experiments in this paper.

3.1 Introduction

Developmental growth is usually defined in biology as a purely biological unfolding of events involved in an organism changing gradually from a simple to a more complex form. In this paper we define developmental growth to be the unfolding of events that leads a system, more specifically a cellular system, to configure its global state in a *complex*, *stable*, and *useful* pattern. *Complex* here indicates that the 'adult' state of the system should contain more information than the beginning of the developmental process, and that the system begins with a minimal amount of environmental information. *Stable* means that the growth process should stop by itself, or rather that the process continues, but does not alter the global state of the system anymore. This effect is very much like biological development: once an organism reaches its adult size, cells are renewed continually, but the organism as a whole remains more or less unchanged. Finally, *useful* means that this final configuration of the whole system fits some prerequisite need, which could be a pattern to display, but also configuration of an electronic circuit, or even that given some input and output cells, the system is able to compute some useful functions. As we have discussed above, stability (self-regulation) and context sensitivity are important keys to building highly scalable and robust developmental models.

3.2 Context Sensitivity and Stability (Self-Regulation)

For a developmental model to create complex systems we usually need a context sensitive process. In essence, context sensitivity can be described as the local interaction of the basic components of a system. More than anything else, this characteristic allows the rise of emergent complexity in developmental systems. The earliest developmental system, the L-system developed by Lindenmeyer [61], is not a context sensitive one, but it can be made context sensitive through the extension of the rules to include preconditions that allow the local neighbourhoods of the characters to be rewritten. Interestingly, this allowed the model to generate not only more complex, but also visually more realistic plants. The context sensitive L-systems model is a good step towards reaching the goal of designing an abstract developmental model. However, since the model's growth phase is controlled by a user provided growth limit parameter, it cannot be considered a stable, self-regulating model.

To better describe the *stability* characteristic, we will focus on one of the simplest of context sensitive models, the elementary (1 dimensional) CA model. Although the elementary CA model and the CA rules that generate the various patterns on the model are very simple, some of the patterns that emerge from it are far from simple. This behaviour is the direct result of the interaction of the simple local rules. The CA model is by its nature context sensitive. It is also capable of supporting stable patterns. Figure 4a shows the development of a one-dimensional pattern on a CA in 5 time steps with respect to the CA rules given in figure 4b. The pattern is generated from three *seed* cells, and that stays the same forever. This 3-celled pattern is a stable pattern because even if the developmental cycle continues, it stays the same. Figure 4c illustrates the error correction ability that is made possible through the continuous developmental cycle. At *t*=6 an error is introduced that turns the middle cell into white. The error is corrected at the next time step. It is of course possible that the error could have stricken at a different cell, and the pattern could be permanently damaged. However, we should look at this as an example model that illustrates the benefits of context sensitivity and stability in a basic way. Elementary CA are very simple and have a very limited range of rules (256 in total), we will see in the rest of the chapter that it is not the case with two-dimensional CA with multiple states.

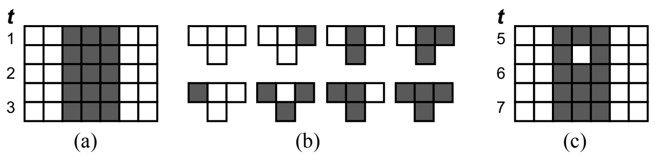


Figure 4. Figure (a) shows the development of a 1 dimensional pattern in 5 time steps (t). The pattern starts with three cells at t=1, and keeps the pattern stable. (b) Shows the rule table for the CA pattern shown on (a). Figure (c) shows the further development of pattern in (a). At t=6 an error is introduced and the middle cell becomes white. The error is corrected at the next time step.

3.3 The Developmental Cellular Model Defined

Following the bio-inspired route laid down in the earlier chapters of this thesis, we will now propose a framework based on development, evolution and cellularity to obtain an evolvable, resilient computing system. The developmental cellular model (DCM) that is defined here is:

1. *abstract*, i.e., we do not pursue biological realism;
2. *complex*, in the sense that it is able to self-configure from minimal amount or no environmental information through the local interaction of the basic units (context-sensitive);

3. *stable*, in the sense that it is actually continuously configuring itself, the working configuration being a stable state of development;
4. *cellular*, it is based on a cellular structure where each unit is relatively simple, and all units are identical, though they may differentiate;
5. *statically connected*, i.e., the neighbourhood of the units is fixed.

While nature was the starting point for our inspiration, our purpose here is not biological plausibility. The system that was developed for this work aims to extract some of the quintessential abstract principles of biological growth process while still remaining computationally tractable. The eventual goal here is to gain fault-tolerance, self-assembly, and self repair capabilities in computing systems.

The DCM is divided into two main parts and we will describe these separately in the next two subsections. The first part the cellular framework of the system. It is inspired by the model of Miller [21]. The second part will give the specifics of the evolutionary algorithm that was used and most importantly, the parameters used.

The Cellular Framework

The cellular model that is used in the DCM is an extension to cellular automata. To be more precise the cellular system lies on and develops along a two dimensional discrete topology. Each cell separately executes the same program and according to its surroundings the program decides the cell's outputs.

Each cell lies at a separate vertex of a two dimensional lattice (Figure 5c) and possesses an x-bit state that is readable by any one of the neighbouring cells where x is dependent on the given problem (Figure 5a). The cell neighbourhood consists of 8 cells (the so-called *Moore neighbourhood*). All cells act according to the same *memoryless* program whose inputs are its own state bits and the state bits of its 8 neighbours. A single *false* state bit is added as the last input to provide a static value to the cell program. Though not stored as a look-up table, this deterministic internal program execution decides the new state of the cell just like regular CA. What distinguishes this from a regular CA is that *the cell program generates outputs that overwrite other neighbouring cells' state bits* as well as its own (Figure 5b).

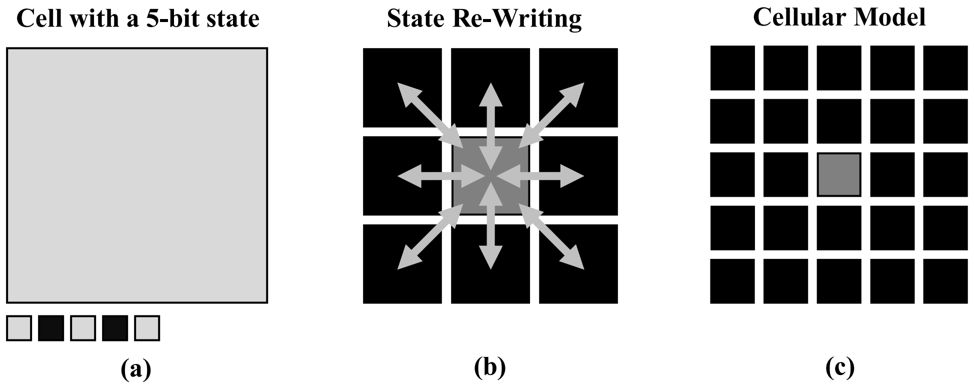
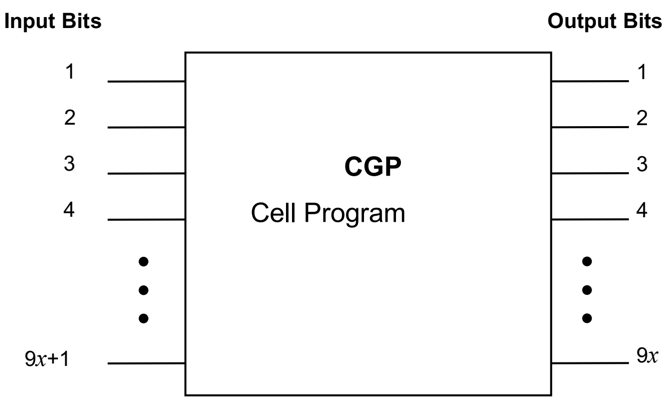


Figure 5. (a) Displays a sample cell with a 5-bit state ("10101" corresponding to grey and black cells). (b) Shows how the grey cell can rewrite its neighbours' state and how in turn its neighbours can write its state. (c) Displays the visual representation of the cellular developmental layer.

Given that all the state bits of the original cell and the neighbouring cells are considered individually, this internal cell program could best be described as a $9x+1$-bit input, $9x$-bit output function (Figure 6). The most important difference between this and the system described by Miller [21] is that our DCM model does not explicitly include a chemical diffusion mechanism. Miller's system relies on the chemical diffusion mechanism to send messages from cells to cells and therefore control the growth patterns of his organisms.

Our earlier experiments removed the chemical diffusion mechanism from our model believing that lack of a diffusion mechanism would make our system simpler, and because of that it would be easier to evolve organisms that fulfil the controlled growth (stability) and regeneration tasks in our experiments [34]. Since then we have discovered that because of the state overwriting capability of our model, it can evolve and simulate a chemical diffusion mechanism, therefore if needed our model can make use of such a mechanism. This makes our model not simpler than Miller's but in fact richer in terms of the methods and components available in the environment for evolution to exploit.



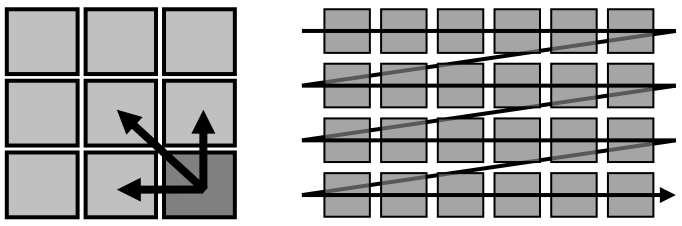
(a)

Cell	Bit
Origin	1 … x
North	x+1 … 2x
N. East	2x+1 … 3x
East	3x+1 … 4x
S. East	4x+1 … 5x
South	5x+1 … 6x
S. West	6x+1 … 7x
West	7x+1 … 8x
N. West	8x+1 … 9x
False	9x+1

(b)

Figure 6. (a) Shows the internal CGP cell program which could be described as a 9x+1-bit input 9x-bit output function. The number 9 comes from the 8 neighbours of the origin cell plus itself and x is the number of bits that each cell has. Therefore 9x becomes the total number of state bits that the origin cell sees not including the *false* bit value that is supplied to increase computability. Table (b) displays the mapping of the bit string to the cell program inputs and outputs.

The neighbour's state rewriting property, which is at the heart of the whole developmental process, entails a series of questions on the order of update of cells. It forbids a democratic update, meaning some cells have the last say in what states the corresponding part of the grid will be in. The process requires each cell to read its neighbourhood and try to write its own state and the states of its 8 neighbours in parallel. However, to solve the problem caused by many cells wanting to rewrite the same cell's state, only the right-most, bottom-most neighbour cell has priority (see Figure 7a). This makes the system a fully deterministic and parallel model but with hard coded update priorities.



(a) (b)

Figure 7. Figure (a) shows the update priority of a cell whereas (b) shows the order of execution of the cells within the grid. The outputs are written to a temporary grid which is turned into the new grid after the last cell is executed to make the system parallel.

The developmental aspects rely on this rewriting property. In effect the state of all cells in this grid starts with 0^x state, where x is the bit-length of the state of each cell. The disruption in uniformity, necessary to obtain positional information and more importantly non-uniform configuration of the grid is achieved by this rewriting priority which implicitly defines a relative order (see Figure 7b). This disruption is sufficient in many cases to obtain any kind of configuration. The fact that the system configures itself from minimal environmental information is believed to be essential to obtain a resilient system. Development in this context is thus really self-organisation.

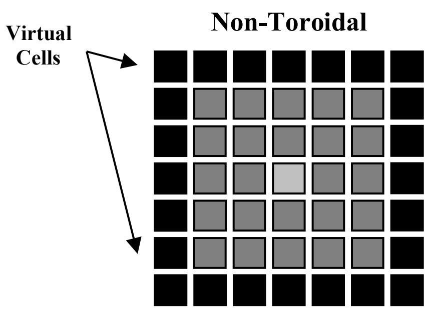


Figure 8. Figure displays a 5×5 cell grid. The cells at the borders can see and overwrite the virtual cells. However virtual cells keep a permanent 0^x state.

Our developmental model is designed to work on a cellular grid with fixed borders. To the cellular program, the virtual cells beyond the borders appear to have a uniform 0^x state that cannot be altered where x is the total bit-length of the state of each cell (Figure 8). Since this 0^x state is not specific to the virtual cells, it does not allow the cellular program to easily distinguish the borders. However, we should note that fixed borders do nevertheless help create non-uniformity in the grid. In the experiments below we contrast this model against two other models. One of them is the classical CA model with the extension of CGP. The other one is the *toroidal* version of our model. In the toroidal model, the only difference is instead of using a fixed border; the grid is extended in each direction to form a toroid shape.

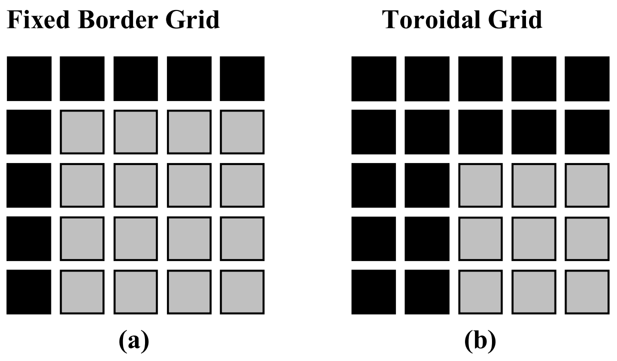


Figure 9. (a) and (b) show the grey "key" cells in sample 5×5 grids. Although each cell runs a separate copy of the same CGP program, due to the update priorities as explained earlier, the grey "key" cells end up being responsible for configuring themselves and the rest of the grid. The black cells therefore can be considered inactive.

Upon closer examination of the update priority and the fixed borders, one can see that beyond creating non-uniformity, they allow for certain "key" cells to emerge within the grid (Figure 9). The execution of the cellular program in the "key" cells directly configures the grid. For instance, for a 5×5, fixed bordered grid, only bottom-most and right-most 4×4 block of cells configure themselves and the rest of the grid (Figure 9a). If the same grid was a toroidal grid, the key cells would make up a 3×3 block (Figure 9b). In effect, the black cells in figures 5.6a and b are therefore rendered inactive.

The Evolutionary Algorithm

As we have detailed in the above section, each cell in the grid operates according to a cellular program. In essence our system uses artificial evolution to evolve this program for use with each cell to generate the desired pattern and attributes. The algorithm that we use for evolution is the CGP algorithm discussed earlier.

CGP is used, as is often the case, with a small population and a large number of generations. The population size used in all experiments in this thesis is only 5. Only the best individual is selected, and 4 new mutants are created from it to maintain the population size. This evolutionary algorithm is known as the $1+\lambda$ evolutionary strategy where $\lambda=4$. Hence, it is an extremely elitist algorithm. This is in great part imposed onto us by the very small population size. We should also note here that this algorithm allows for *neutral* exploration of the search space.

The details of the genotype are given in section 2.3 above. Basically, the CGP algorithm is able to build programs out of any set of three-input one-output boolean functions.

In an attempt to investigate the effects of increased population size with CGP, a series of experiments using a system related to the one in this paper experiments has been performed with a population of 50 and one of these experiments is detailed in the first author's PhD thesis [62]. These experiments demonstrated that, whilst fewer generations were required, the total number of fitness evaluations was much increased. Therefore, in the experiments below, we use 5 as our population size.

Experimental Setup

In the experiments detailed in this thesis, these CGP programs are evolved as decision makers for cells. The aim of the evolution experiments presented in this chapter is to evaluate the evolvability of different variants of the developmental cellular systems presented above, in order to establish a precise framework within which such systems are evolvable. For all evolution experiments in this chapter, the evolutionary runs were repeated 100 times to establish some statistical validity. All experiments were performed on three CA models: the toroidal overwriting, the fixed-bordered overwriting, and the classical CA (i.e. non-overwriting) on a fixed bordered grid as a control. The toroidal classical CA is not used since, given that all cells start from a uniform 0^x state, and that there are no borders, there would be no break in uniformity, therefore self-configuration in a non-uniform pattern would not be possible

4. Developmental Pattern Evolution Experiments

4.1 Introduction

The aim of the evolutionary experiments in this paper is to find an organism able to develop and stabilize by itself into a given target pattern. This is certainly the most straightforward aim for such developmental systems, as it includes no restriction beyond the most generic cellular framework. Nevertheless, as shown in previous work [21,35] this can prove very tricky to obtain. For all the experiments below, we used CGP and DCM as described earlier, with a population of 5 and a maximum number of generations 100,000.

The following evolution experiments are designed to evolve a program that would successfully configure the grids to reach the respective target patterns and stabilize within seven CA time steps. The period of seven time steps was specifically chosen to enable the system to easily propagate positional information and stabilize. Stabilization periods other than seven have also been investigated [62]. To ensure that evolution is guided towards pattern stability, the fitness F is defined as $F = |P_T| - (H(P_7,P_T)+H(P_8,P_T))/2$, where $|P_T|$ is the total number of bits in the pattern $P_T, H(A,B)$ is the hamming distance between patterns A and B, P_i is the pattern of the grid after i time steps, and P_T is the target pattern. As one can see a perfect organism that develops into the target pattern, and is stable (i.e., pattern does not change within 2 time steps) will have a perfect fitness of $|P_T|$. In section 4.2 we will show evolutionary results with random target patterns, in section 4.3 will show the results of our experiments that used regular target patterns such as diamond shapes and French flags.

4.2 Random Pattern Evolution

Our first two experiments are concerned with evolving randomly-generated patterns. These experiments take place on a 5×5 grid, each of which contains a 5-bit value. The aim is to evolve a 9×5-bit+1 input, 9×5-bit output program, which will be applied to each cell in the grid. The success criterion is the evolution of a program that, when applied to a grid where all cells have 0^5 value, reaches the desired (25 cells × 5 bits) 125-bit pattern within 7 time-steps.

Each of the two experiments is carried out using the three variants on the DCM model outlined earlier, *viz.* the toroidal, fixed bordered, and classical CA.

Experiment 1 is the evolution of a program to configure a 125-bit pattern; Experiment 2 is the same, except that in this case a spare capacity of 2 bits is introduced. The effects of the spare capacity will be discussed in the analysis of results.

Experiment 1: Evolution of 125-bit Random Patterns

This experiment was set up to determine our model's ability to generate a stable, complex, and random pattern. To allow for a broader range of patterns a set of ten random 125-bit long patterns were generated as targets. Each bit of the patterns was randomly chosen with probability ½. For each pattern, a series of ten evolutionary runs were performed, using the parameters in Table 1.

Grid Size:	5×5=25 cells
Cell State Bit-Length:	5 bits
Total Configuration Bit-Length:	25×5=125 bits

Target Pattern Bit-Length:	25×5=125 bits
Time Allowed for Development:	7 time steps
CGP Node Size:	100 nodes
Mutation Percentage:	1%
Maximum Number of Generations:	100,000
Population Size:	5
Experiment Repeated per Model:	10×10=100 times

Table 1. The parameters for Experiment 1.

As expected from previous works on the evolution of patterns on cellular systems, this basic approach leads to very poor results. For the classical CA model and the toroidal grid, none of the 100 evolutionary runs produced a successful result. For the model with fixed borders, only one pattern was evolved twice out of the 10 runs, four patterns only once, and five patterns were not evolved at all (Figure 9).

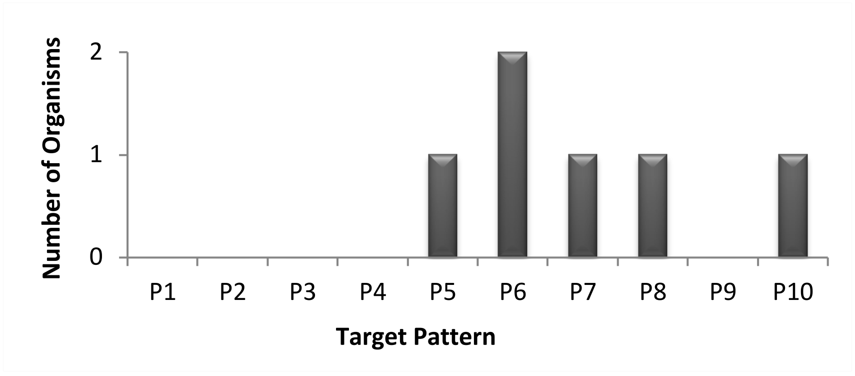


Figure 10. This chart is displaying the evolutionary outcome of the 10 random patterns on the fixed bordered model. The patterns are all 125-bit long. A total of 6 organisms were evolved.

The developmental steps for an evolved organism that generates the stable target pattern P7 can be seen in Figure 11. The system starts with an empty grid, and develops after 7 time steps into the desired target pattern; also, the pattern also stabilizes after *t*=7.

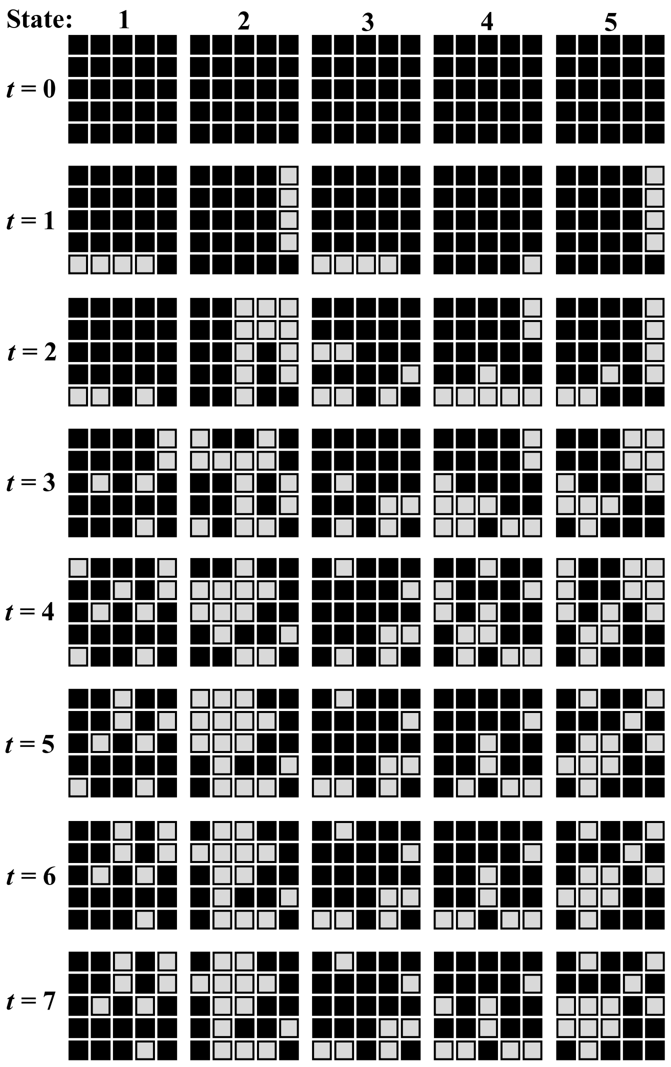


Figure 11. The development of the evolved organism on the fixed bordered model for pattern P7. The time steps are shown from top to bottom, and the separate bits of the cells are shown from left to right.

Experiment 2: Evolution of 125-bit Random Patterns using Spare Bits

The previous experiment indicated that the work required from the cell program is rather ambitious. It should develop from minimal environmental information into a complex pattern using as "memory" for its computation only the bits used in the final pattern. In other words, it must either compute using information as required in the final pattern, or at least none of the information used to grow into the final pattern should be left over when it has reached its stable state. This restriction is quite drastic and it was decided following the positive results of this experiment that we should allow the program two extra bits per cell for use with the developmental process. With the spare bits, only the first 5 of the now 7-bit states are used in the fitness measure. The idea here is to provide the cell program a 2-bit long space in each cell to store and read positional and developmental information separate of the 5 bits that configure the actual target pattern. Therefore the fitness is measured as before, but taking into account only the first 5 bits of each state.

The same target patterns were used as in Experiment 1, and the parameters used are given in Table 2.

Grid Size:	5×5=25 cells
Cell State Bit-Length:	7 bits
Total Configuration Bit-Length:	**25×7=175 bits**
Target Pattern Bit-Length:	25×5=125 bits

Time Allowed for Development:	7 time steps
CGP Node Size:	100 nodes
Mutation Percentage:	1%
Maximum Number of Generations:	100,000
Population Size:	5
Experiment repeated per Model:	10×10=100 times

Table 2. The parameters for Experiment 2.

The results for the classical CA and the toroidal model both give a success rate of 0/100. However, for the fixed bordered model a total of 22/100 runs were successful (details in Figure 12). It can be noted that only one pattern was unsuccessful to be evolved for with this grid. As this pattern was successfully evolved in the previous experiment, it can be said that this is the result of a statistical anomaly rather than the ability to evolve a pattern within this framework.

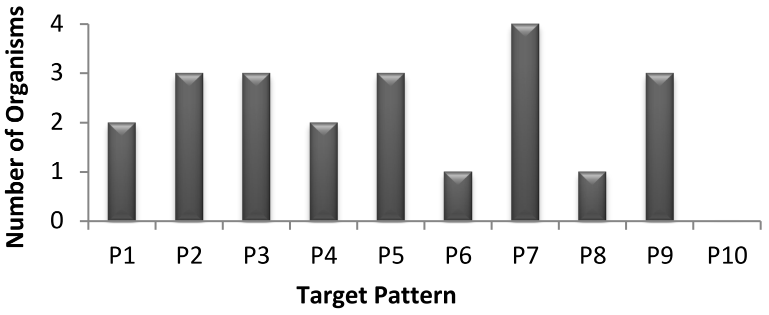


Figure 12. *The success rate of Experiment 2 on each of the ten patterns.*

Figure 13 displays a sample evolved organism with 2 spare bits using the fixed bordered model that generates pattern P7.

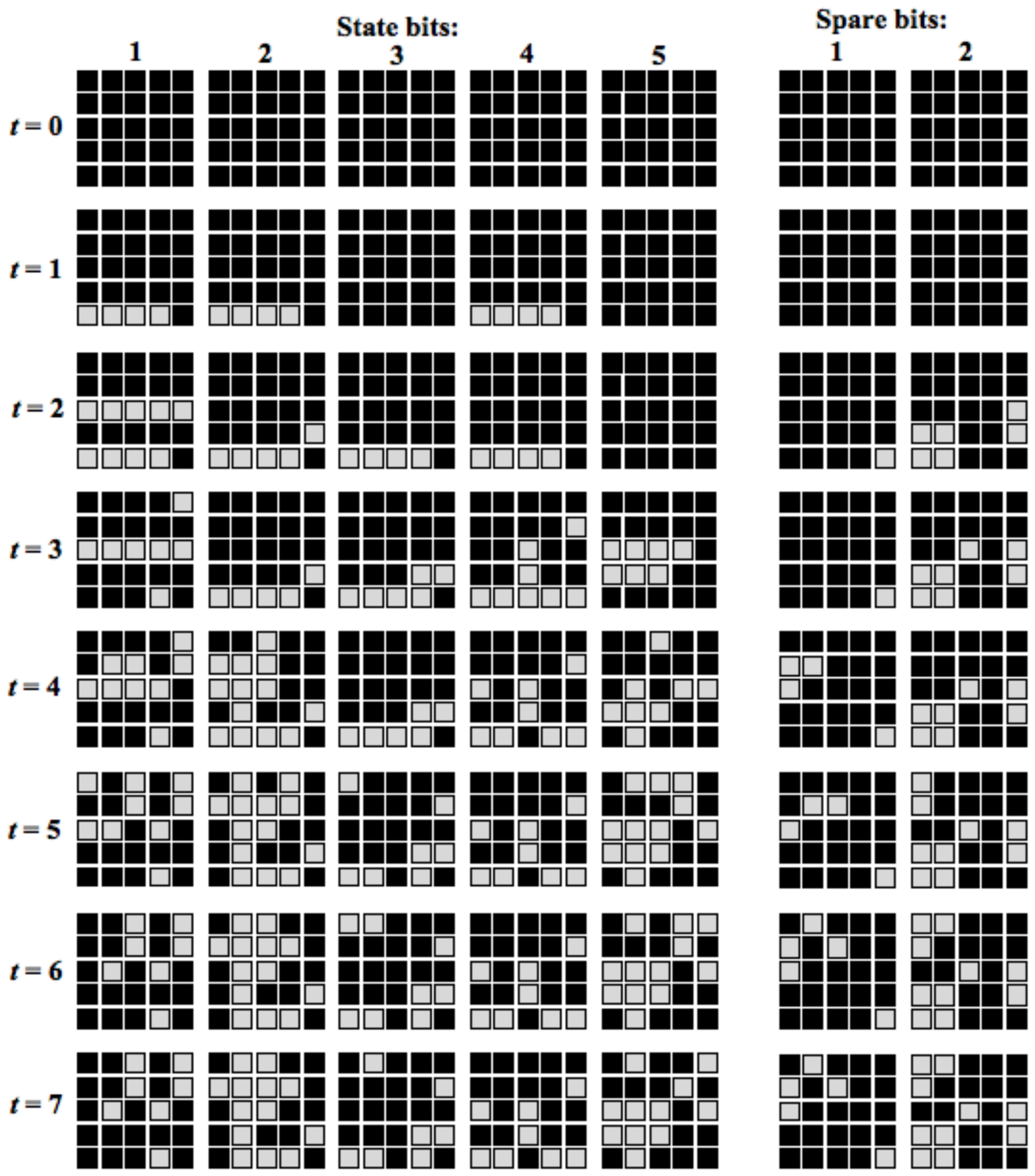


Figure 13. The development of the evolved organism with 2 spare bits on the fixed bordered model for pattern P7. The time steps are shown from top to bottom, and the separate bits of the cells are shown from left to right. The 2 rightmost grids indicate the spare bits.

Analysis of Results: Experiments 1 and 2

These two experiments clearly show us that both the classical CA and the model with the toroidal grid are less evolvable than the fixed-bordered model. The success rate of the fixed-bordered model in Experiment 1, which is about 6%, puts it forward as a better candidate than the other two models for an evolvable framework. But this result in general still remains rather poor. Statistically, the classical CA and toroidal models have a success rate between 0 and 4.6% (with 95% confidence, using the methods in [63,64]) and the one with fixed borders has only a success rate of 2.4-13.1% (with 95% confidence), which is not enough to clearly distinguish them.

Looking at figure 11 in more detail, we can see that the first thing the system does is to find the borders of the grid ($t = 1$). In the following time steps we can see the cells are carrying the positional information gathered from the borders towards the centre of the grid. Because this pattern is quite complex, the system manages to generate the pattern using the entire time it is allowed. We will see in the experiments of the next section that this is not the case with regular patterns.

The introduction of the two spare bits in the latter experiment presents a strikingly positive impact on the evolvability of our model. Whilst the overall task is as hard as before,with the introduction of the two spare bits the search space got larger yet the evolvability of the model improved. It now encompasses all the possible cell programs that match a 9×7+1-bit input 9×7-bit output function. It is a space that is 2^{50} times larger than the one in the earlier experiment. On the other hand, it gives two extra bits of information that the program may use to grow and stabilise without impacting the fitness measure, which is really an increase in the computational power of each cell.

For Experiment 2 it can be concluded that in general, the evolvability of patterns using 2 spare bits with a fixed bordered grid is about 14.5-31.6% (with 95% confidence). This can be deemed as a reasonably evolvable model, but not quite satisfying in terms of the effort needed to ensure the evolution of a successful solution. Taking an evolvability of 22%, which is a reasonable hypothesis given the low standard deviation, one would need on average 1,261,582 generations to be 95% sure of finding an individual (based on the fact that a single run is successful in 60,000 generations on average). This weakness is strongly mitigated by the important fact that the algorithm is uniformly successful in almost all of the target patterns.

We should also note that the value 2 for number of spare bits is not at random. In fact, separate experiments with 1 and 3 spare bits were also conducted. The one with the 3 spare bits only marginally improved the results of the fixed bordered model, with 26 successful evolutions, and 1 spare bit did not provide a statistically relevant improvement over the one with no spare bit. Neither experiment managed to produce successfully evolved organisms for the classical CA and the toroidal models.

4.3 Regular Pattern Evolution

As we discussed above, most pattern generation experiments are conducted using a basic target pattern such as a square, a circle, an "L" shape, or a simple flag [21,35,60]. The reason for using basic shapes for these experiments is that such shapes are easier to evolve with developmental systems than random patterns. Basic shapes or regular patterns usually have a repetitive or modular structure. Since developmental representations are compressed versions of their respective adult organisms, they are more suitable for representing regular patterns.

The experiments in this section were designed to investigate the validity of the above statement on our model through comparisons with random pattern evolution experiments. Experiment 3 is designed to give us some idea of regular pattern evolvability within our system by investigating the evolution of a simple square pattern. Experiment 4 covers the evolution of a moderately more complex diamond pattern on a 5×5 grid. In experiment 5, evolution of more complex yet also modular French flag patterns on 8×8 grids using four colours is performed. All sets of experiments were repeated for all three grid types as before to allow us to make a comparison.

Simple Pattern Evolution

The experiments in this subsection were designed to assess our model's ability to generate a simple regular pattern through comparisons with the toroidal and the classical CA models. The experiments were conducted on a 4×4 cellular grid with a target pattern of a 2×2 block of cells as can be seen in figure 14. Parameters are given in Table 3.

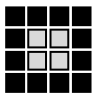


Figure 14. Figure displays the target pattern for the simple pattern evolution experiment.

No spare bits are allowed for the organism to use with the developmental process. This makes the target pattern 16 bits long. Although this is a much simpler task than any one of the random pattern experiments done previously, thanks to its simplicity it is easier to analyse the inner workings of the model.

Grid Size:	4×4=16 cells
Cell State Bit-Length:	1 bit
Total Configuration Bit-Length:	16×1=16 bits
Target Pattern Bit-Length:	16×1=16 bits
Time Allowed for Development:	7 time steps
CGP Node Size:	100 nodes
Mutation Percentage:	1%
Maximum Number of Generations:	100,000
Population Size:	5

Experiment Repeated per Model:	100 times

Table 3. Parameters for Experiment 3.

As expected from an experiment with such a simple target pattern, the evolutionary results for our fixed bordered model are excellent with 100 successful evolutions out of 100 experimental runs. This indicates a 95.3-100% success rate with 95% confidence. What is interesting is that the toroidal model also managed to generate 100 successfully evolved organisms. Furthermore, unlike the random pattern evolution experiments, the classical CA model managed to evolve successful organisms for this pattern. However, as expected it performed the worst with 3 successful evolutions out of 100 which indicates a 0.7-9.1% success rate with 95% confidence as can be seen in table 4.

	Evolutionary	Number of Generations before Evolution is Successful			
	Success Rate	Average	Minimum	Maximum	Std. Dev.
Fixed Bordered	100	1087.75	26	23629	2885.40
Toroidal	100	72.69	1	569	88.51
Classical CA	3	69004.67	33777	94240	31445.65

Table 4. Table shows the success rates out of 100 evolutions and the respective numbers of generations before a successful organism is evolved for each developmental model.

Beyond the evolvability results the behaviour with the toroidal model is clearer when we look at the average number of generations before a successful organism is evolved for this grid type. With the low standard deviation, it clearly shows that toroidal grid provides a much smoother fitness landscape than the other models. However, we should refrain from making any further comparisons with the random pattern evolution experiments because of the simplicity of this task.

The developmental steps for three sample evolved organisms on fixed bordered, toroidal and classical CA respectively can be seen in figure 15. As always, the system starts with an empty grid at *t* =0 and within 7 time steps it is expected to develop into the target pattern. However, in this case, the developmental process concludes within 4 time steps further indicating the simplicity of the task. We should also note that while the fixed bordered model managed to stabilize at *t* = 3, the toroidal model managed to do that in a single time step.

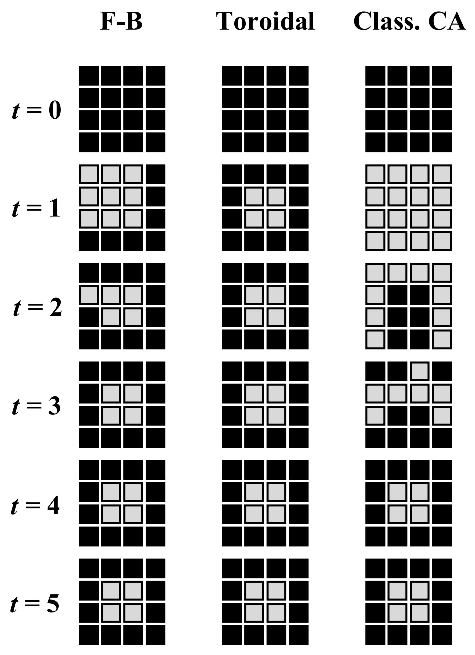


Figure 15. Figure shows the development of three evolved organisms on the fixed-bordered, toroidal and classical CA models respectively. The time steps are shown from top to bottom.

This experiment was later repeated with a 30-node CGP without too much loss in evolvability [Ozturkeri, 2008].

Detailed Analysis of a Sample Evolved Organism

In this section we will present an analysis of the internal workings of a sample CGP organism from Experiment 3. We will first decode the CGP organism and then use graph representation to help us understand the cellular inputs and outputs within the developmental process.

Although the pattern that we have selected is quite simple (Figure 5.11), and the cells only have a single bit, the developmental process is still very complicated. In order to make the analysis of the developmental processes easier, we have repeated the evolution of the simple square pattern using a 30-node CGP program with parameters as in Table 5.

Grid Size:	4×4=16 cells
Cell State Bit-Length:	1 bit
Total Configuration Bit-Length:	16×1=16 bits
Target Pattern Bit-Length:	16×1=16 bits
Time Allowed for Development:	7 time steps
CGP Node Size:	30 nodes
Mutation Percentage:	1%
Maximum Number of Generations:	100,000
Population Size:	5
Experiment Repeated per model:	10 times

Table 5. Parameters for the experiment from which the sample organism was created.

The evolution experiment was successful at every run producing ten organisms each for the fixed bordered and the toroidal models. Within the evolved organisms we have picked one that manages to configure the grid in a minimal amount of time to help us better analyse the developmental process. The organism that we have picked was evolved on the fixed bordered model and managed to develop within 2 time steps into the desired pattern and stabilize. The developmental cycle for this organism is clearly shown in figure 16.

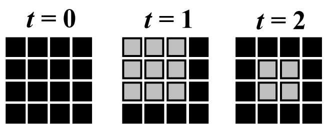


Figure 16. The development of the sample organism on the fixed bordered model.

As we recall from our description of the model above, due to the update priority imposed by the overwriting process, some cells have no effect on the configuration of the grid, and therefore can be thought as inactive. For the fixed bordered model with a 5×5 cellular grid the active cells, which we have termed *key cells,* make up a 4×4 block. For the 4×4 cellular grid that we use here the *key cells* make up a 3×3 block. The white cells in figure 17a indicate the *key cells* for the sample organism. As we also recall from earlier, there are 9 output bits for each state bit of the cells. Since our sample experiment only has a single bit per cell, the number of outputs for the CGP program is also 9. At the CGP level, each of these bits are evaluated separately and therefore these output bits can be interpreted as 9 separate single bit binary functions. Figure 17b shows where each output function writes its output to, with regards to an origin cell. For example, function f_4 outputs to the origin cell, whereas f_0 outputs to the North West cell. Again, due to the imposed update priority, not every output function of every *key cell* has an effect on the cellular layer. For instance, it is easy to see that some of the outputs of the border cells would have no effect. Other than that, since the bottom-most, right-most cell has the highest priority; it overwrites four cells in the grid including itself. From now on we will call these functions, *key* functions. The numbers in the *key cells* shown in figure 17a indicate which *key* functions are executed within the *key* cells. As you can see from the figure, only the first four functions are effective for this grid, the remaining five functions have no effect.

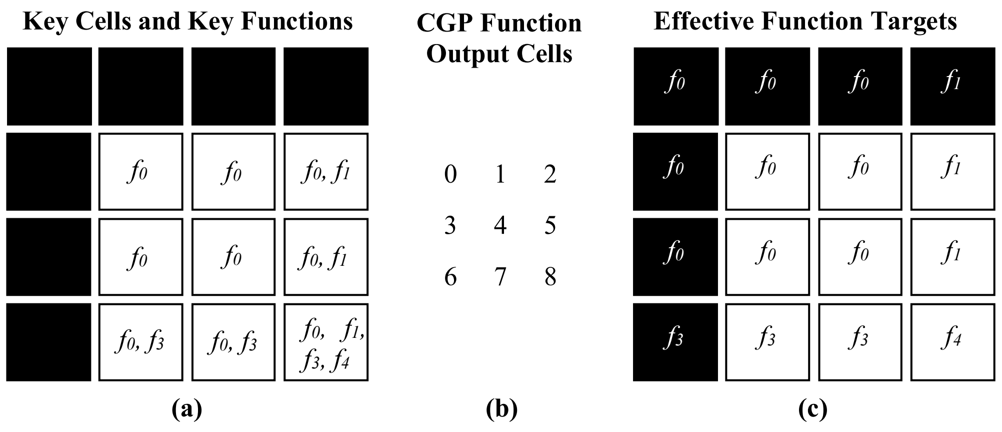


Figure 17. Figure (a) shows the key cells (white) for the 4×4 fixed bordered model. The values in the key cells indicate which key function is effectively executed in the cells. Figure (b) is a table showing where each function (output bit) overwrites which cell in a cellular neighbourhood where cell 4 is the executed cell. Figure (c) indicates which function effectively configures which cell at the end of each developmental cycle due to the imposed update priorities.

Finally, figure 17c shows which function ends up configuring which cell at the end of each developmental cycle. As we can see, function f_4 only configures the cell at position (4, 4) where as f_0 which is the North West output function, is responsible for configuring 9 cells of the grid. Since function f_4 only affects a single cell, and thus should be simpler, this is where we will start our analysis.

According to the decoded CGP program as shown in table 6, function f_4 always outputs the current state of the south east cell for this randomly picked individual. Since f_4 is only executed in cell (4, 4) its south east is outside the fixed borders of the grid and therefore f_4 and consequently cell (4, 4) always stays at state *0* whatever the rest of the grid is. This provides a strong, and correct positional information to the model at all times. Function f_3 which is responsible for the development of bottom-most, border cells except cell (4, 4), is similar. It takes its output directly from outside southern border which also ensures a correct configuration regardless of the rest of the grid.

```
f0    IF( South East = true ) { NOT( South ) }
      ELSE
      {
         IF( North East = true ) { false }
         ELSE { NOT( IF( South West = true ) { West } ELSE { South West } ) }
      }
f1    IF( East = true ) { NOT( North ) } ELSE { false }
f3    South
f4    South East
```

Table 6. The CGP output functions $f_{0,1,3,4}$ in pseudocode form.

Function f_1 is slightly more complex looking. However, if we look closer we can see that since it only applies to the cells at the eastern border, the first part of the IF statement and consequently the output of the function always becomes *false* or state *0*. So far, all the functions that we have detailed had a static output of state *0*. However, this cannot be the case with function f_0 as it will configure the rest of the grid into cells with different states. f_0 starts with an IF statement that tests the south east of each origin cell. The key cells at the borders always have state *0* at their south-east; therefore this part of the function is always *false* for them. The rest of the key cells are dependent on the state of the south and east border cells. The next IF statement tests the north-east value, and since all cells have state *0* to begin with, this statement also returns a *false* value. The third IF statement tests the south-west cell that also returns *false* and the function ends up outputting the inverse of the south-west cell of each origin cell. Since all key cells use f_0 to output to their north west, we end up with a 3×3 block of state *1* cells at the first time step as can be seen in figure 17.

At the next time step, while all the other key functions output exactly the same value as the previous time step, f_0 changes for the non-border key cells. The first IF statement only returns true in cell (2, 2) changing the cell (1, 1) to state *0*. The second IF statement then handles cells (1, 2) and (1, 3). Finally the third IF statement, apart from keeping the rest of the cells configured by f_0 the same, handles cells (1, 2), (2, 2) and (3, 2) turning them back to state *0* which forms the target pattern. In the subsequent time steps these 4 functions keep the pattern quite stable and fixed.

It is interesting here to note that while function f_1 is quite effective, it could have been a lot shorter. Actually, it could be as short as functions f_3 or f_4 since it always outputs the same value, state *0*. In any case, this sample development is a very good example of how the fixed borders provide positional information to the system and how this information is transmitted towards the north east of the grid while configuring it at the same time. It is also interesting to note here that any error that is introduced to the south-eastern corner of the grid would take longer to heal, as this would disrupt the positional information that is continuously being transmitted.

Experiment 4: Diamond Pattern Evolution

In this subsection we will test our system on a more complex regular pattern, a diamond shape on a 5×5 grid, which can be seen in figure 18. Similar to the random pattern experiments we will perform the experiments on a 5×5 cellular grid. Since the size of this pattern is larger than the square pattern in the previous experiment, we will use the spare bit extension with one bit. This makes our target pattern 5×5×1=25 bits and the entire configuration 5×5×2=50 bits.

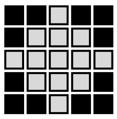


Figure 18. Figure displays the target pattern for the diamond pattern evolution experiment.

The evolutionary runs were repeated 100 times for each grid type using the parameters in Table 7. Although the number of bits used within each cell (2 bits) is small, and the target pattern is made up of cells with only 1-bit state, this experiment is still moderately challenging as its aim is to find an organism that will develop into a stable 25-bit configuration.

Grid Size:	5×5=25 cells
Cell State Bit-Length:	2 bits
Total Configuration Bit-Length:	25×2=50 bits
Target Pattern Bit-Length:	25×1=25 bits
Time Allowed for Development:	7 time steps
CGP Node Size:	100 nodes
Mutation Percentage:	1%
Maximum Number of Generations:	100,000
Population Size:	5
Experiment Repeated per Model:	100 times

Table 7. Parameters for Experiment 4.

Similar to the previous experiment, with the exception of the classical CA model, all models produced perfect results in terms of evolvability. The number of successful evolutions for the fixed bordered, and the toroidal models were both 100 organisms (indicating a 95.3-100% success rate with 95% confidence), as can be seen in Table 8. The classical CA model performed the worst of the three models; however, it was significantly better than the respective result in the simple pattern evolution experiment. It managed to generate 74 organisms, indicating a 64.1-82% success rate with 95% confidence. While the target pattern is only 25 bits long, hence less complex, the results with the classical CA model is significantly better than Experiments 1 and 2, showing that it is easier for this model to evolve regular patterns than random ones. The improvement of the classical CA model over Experiment 3 could suggest a connection to the positive effects of the added spare bit for the developmental process.

Evolutionary	Number of Generations before Evolution is Successful

	Success Rate	Average	Minimum	Maximum	Std. Dev.
Fixed Bordered	100	9108	431	92245	13694
Toroidal	100	1080	24	6163	1088
Classical CA	74	29660	2174	99882	24810

Table 8. Table shows the success rates out of 100 evolutions and the respective numbers of generations before a successful organism is evolved for each developmental model.

Beyond strict evolvability, it is interesting to look into the speed of evolution for each grid type (Table 8). The table clearly shows that with 1080 generations on average, the toroidal model finds it a lot easier, in terms of speed, to evolve the diamond shaped pattern than the fixed bordered and the classical CA model. Although the fixed bordered model manages to evolve a successful organism on each evolutionary run just like the toroidal model, the average number of generations for a successful organism to emerge for the first model is approximately 9 times more than the latter model. Similarly, while the speed of evolution for the classical CA and the fixed bordered model are close, there is considerable difference between them and the toroidal model. From these and the previous experiment's results, it seems that the fitness landscape for the toroidal model is a lot smoother than for the other two models, at least with regular patterns.

Figure 19 displays a sample evolved organism for each of the three developmental models. For each model, the left column displays the development of the pattern, and the right column displays the spare bit. It appears from the development of the fixed bordered and the toroidal models which stabilize at t =4, and t =5 respectively, that these models find it easy to evolve this pattern. The classical CA model only manages to successfully stabilize into the target pattern at t =7, and although these are only sample values, they generally hold for all respective models. An interesting behaviour that is worth mentioning is seen at time steps 3, 4, and 5 of the toroidal model. At t =3 the toroidal model successfully generates the target pattern; however, the spare bits continue to change for the next 2 iterations and finally stabilizing at t =5.

Based on these results, regular patterns seem to be easier to evolve than random patterns. As we have discussed before, this may be due to their ordered, modular nature, which allows easier compression and representation of their structure within a developmental process. However, the target pattern used for this experiment is much simpler than the ones used in random pattern experiments, and this may also be responsible for the better results. The next experiment is therefore designed to investigate how the cellular models are affected by increasing the complexity of the regular pattern.

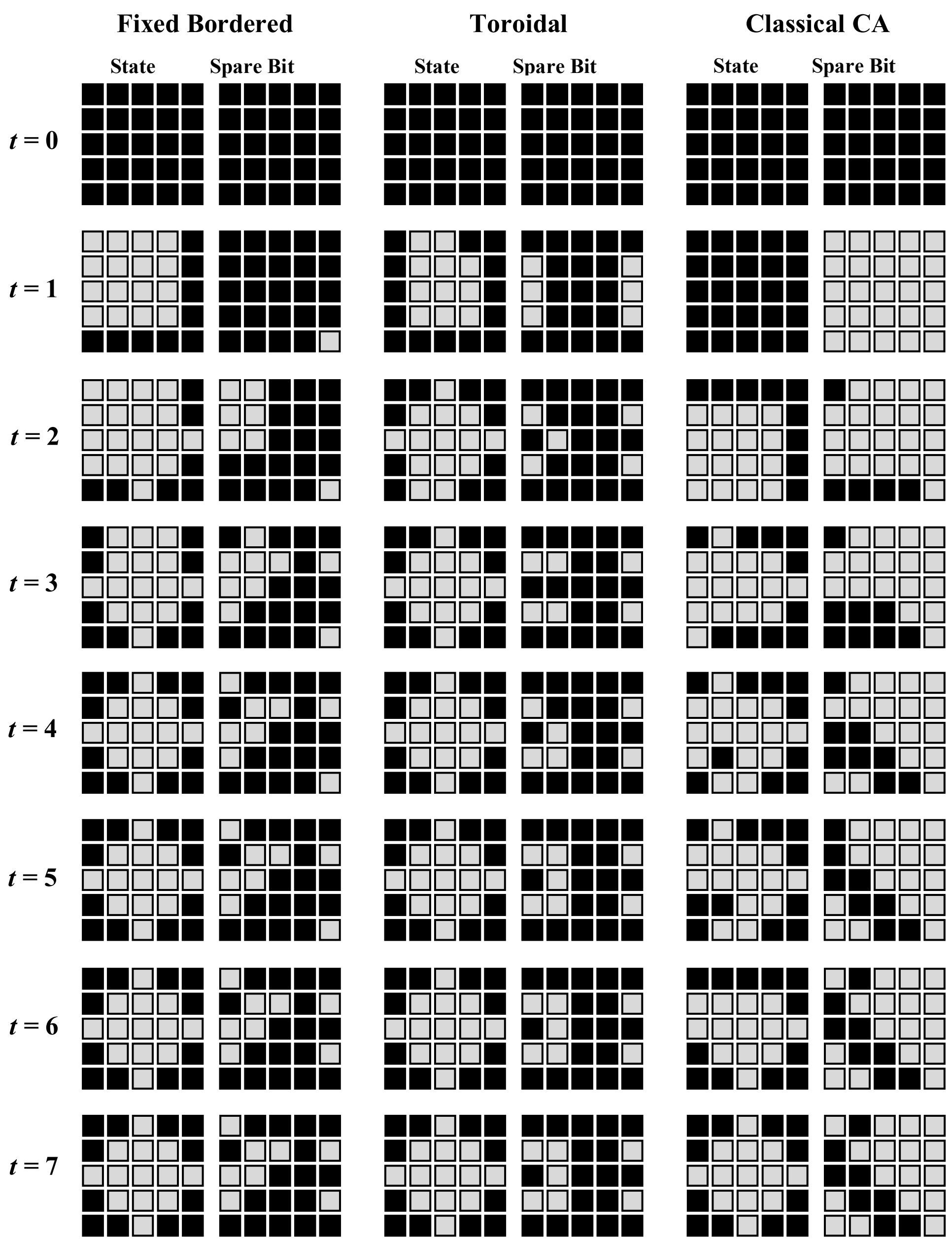


Figure 19. The development of three evolved organisms on the fixed bordered, toroidal and classical CA models respectively. The time steps are shown from top to bottom, and the spare bits of each model is shown on the right columns.

Experiment 5: French Flag Pattern Evolution

The biological developmental process and the differentiation of cells have been likened to the formation of French flags, and these have used as a benchmark experiment in a number of previous computational development studies [21,34,60]. In our experiment here, the French flag requires two functional bits to represent the different colours. Taking into account the results of the previous experiments, we have added two spare bits to facilitate the developmental process so the cells in this experiment have a 4-bit state. To increase the complexity of the pattern, the flag was designed on an 8×8 cellular grid. Therefore the target pattern is 8×8×2=128 bits long, and the cellular grid configuration string is 8×8×4=256 bits long. The 128-bit target pattern is a good candidate for comparison with both the 125-bit random pattern earlier and the indirect evolution experiments in the next section. As before the evolutionary runs were repeated 100 times on each model, with parameters as in Table 9.

Grid Size:	8×8=64 cells
Cell State Bit-Length:	4 bits
Total Configuration Bit-Length:	64×4=256 bits
Target Pattern Bit-Length:	64×2=128 bits
Time Allowed for Development:	7 time steps
CGP Node Size:	100 nodes
Mutation Percentage:	1%
Maximum Number of Generations:	100,000
Population Size:	5
Experiment Repeated per Model:	100 times

Table 9. Parameters for Experiment 5.

The evolutionary results shown in table 10 confirm that , as in Experiments 3 and 4, the fixed-bordered and the toroidal models produce superior results. Since this pattern is more complex, the models cannot evolve a successful organism for each evolutionary run. The number of successful evolutions for the fixed bordered model is 77 organisms, which indicate a 67.3-84.5% success rate with 95% confidence. However, in this case, contrary to random pattern evolution experiments where the fixed bordered model was better and the previous regular pattern evolution experiments where the toroidal and the fixed bordered models were equally good, the toroidal model here gives the best performance. It has managed to produce 83 successful evolutions, indicating a 73.8-89.5% success rate with 95% confidence.

	Evolutionary	Number of Generations before Evolution is Successful			
	Success Rate	Average	Minimum	Maximum	Std. Dev.
Fixed Bordered	77	37522.52	2235	98769	27171.46
Toroidal	83	28751.58	1601	97530	23991.68
Classical CA	5	74845.8	44080	97035	22425.06

Table 10. Table shows the success rates out of 100 evolutions and the respective numbers of generations before a successful organism is evolved for each developmental model.

The evolutionary speed results displayed in Table 10 fit very well with the previous regular pattern evolution experiments with the toroidal model being the quickest to evolve a successful organism.

Figures 20, 21 and 22 display a sample evolved organism for each three developmental models. For each model the left column displays the combined and colour coded two state bits of the cells, and the middle and left columns display the two spare bits of the system. The colour codes are as follows: 00: Black, 01: Red, 10: Blue and 11: White. Similarly with the previous experiments, the evolved French flag organisms also start their developmental cycles by establishing the borders of the grid.

For the fixed bordered model, as can be seen in figure 20, the south and the east borders are established by the first spare bits at t =1. It appears that for this sample fixed bordered model, the establishment of these two borders are adequate to start constructing the bottom part of the pattern. It then seems to use the second spare bits to mark the start of the differentiation and at the next time step the growth process. After the general shape of the pattern becomes evident at t =5 the spare bits lock into their final configuration. From this point on the second spare bits seem to indicate the colours of the flag: a 2×2 block indicating the red column, a 2×1 block for the white column, and none for the blue column.

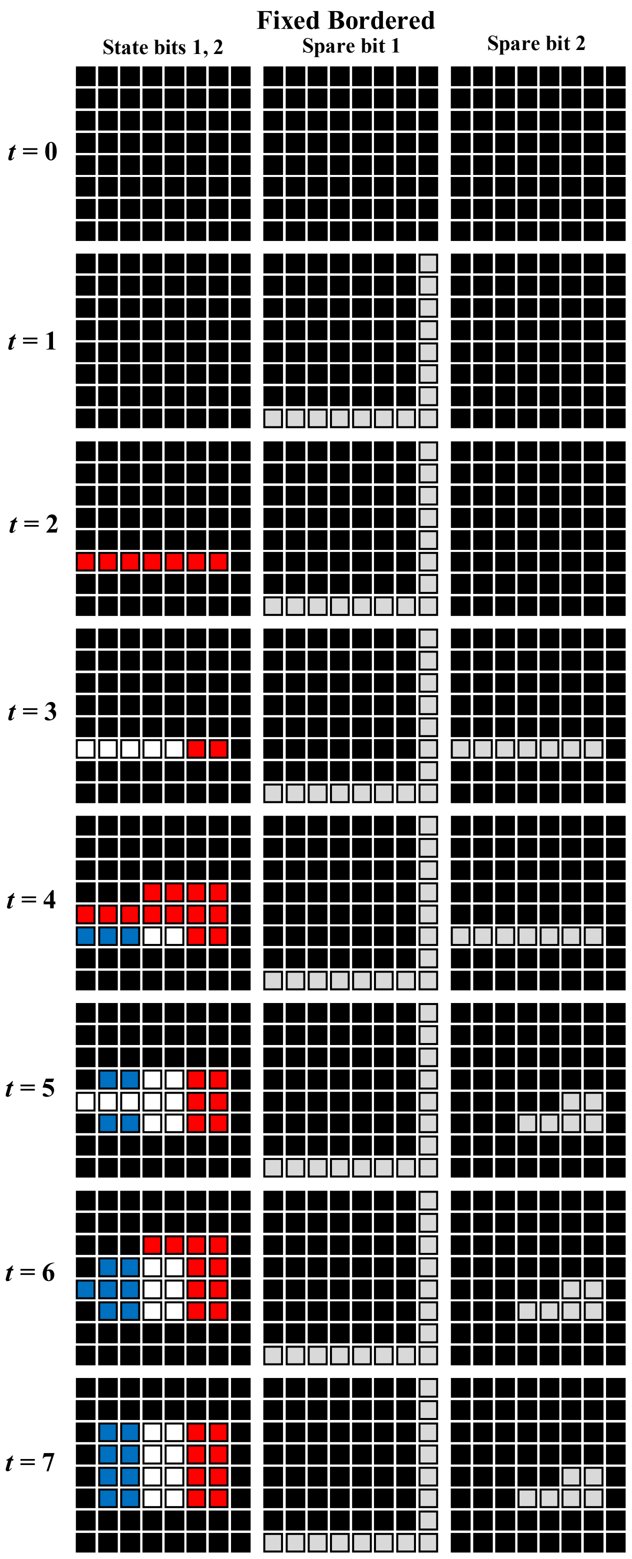


Figure 20. The development of a sample evolved organism on the fixed bordered model. The time steps are shown from top to bottom. The first two bits of each cell are combined, colour coded and shown on the left column and the spare bits are shown on the middle and right columns.

Similarly for the toroidal model, as can be seen in Figure 21, the borders are established by the first spare bits at *t* =1. The difference is, this time all four borders are established, probably due to the toroidal nature of this model. Another difference is that the borders are established by the "0" states rather than the "1" states as before which essentially means that actually non-border cells are established. The effects of this is evident at *t* =2 where a large portion of the non-border cells are filled with red and white cells. After this point the cells start organising themselves and differentiating with the help of the spare bits. The spare bits lock into their final configuration at *t* =6 and the minor bits are corrected forming the target pattern at *t* =7. Unlike the sample organism with the fixed bordered model, this example does not always use the first spare bit to establish and permanently mark the borders, although it still seems to mark the south and the east borders with the partial help by the second spare bit. Another thing that is worth mentioning is that the blue cells emerge and complete their development the last in each case. This is probably connected to the formation of the pattern starting from the south-east corner.

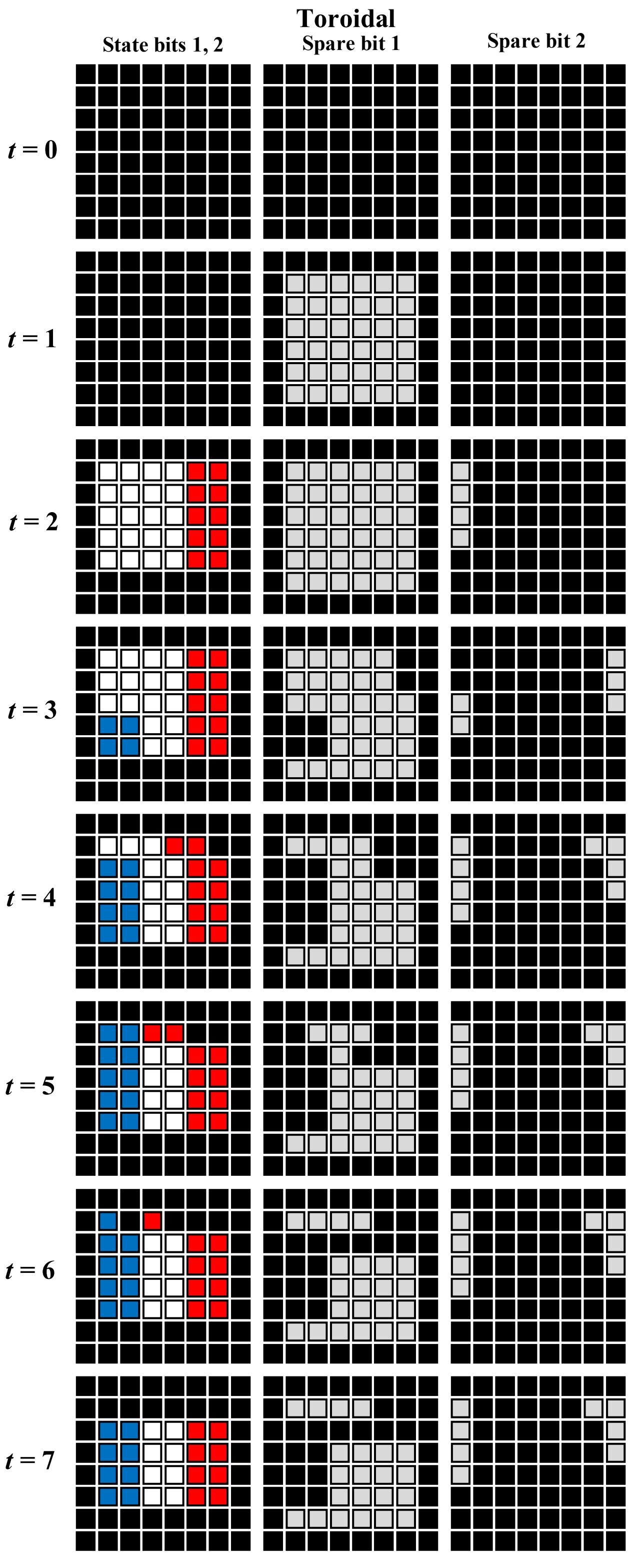


Figure 21. Figure shows the development of a sample evolved organism on the toroidal model. The time steps are shown from top to bottom. The first two bits of each cell is combined, colour coded and shown on the left column and the spare bits are shown on the middle and right columns.

The sample organism with the classical CA model, as can be seen in Figure 22, starts its developmental cycle by initializing the grid with the second spare bits at *t* =1, and marking the borders with the first spare bits at *t* =2. Similar to the fixed bordered model, it uses the first spare bits to permanently mark the borders of the grid. The second spare bits are then used to guide the formation of the target pattern starting with the red cells. Interestingly the second spare bits reach their final configuration at *t* =3 but slightly change at the next time step to presumably initiate the differentiation of the blue cells. It then changes back and locks into its final configuration at *t* =5 where the first white cell differentiates from a red cell. The whole pattern is configured and stabilized at *t* =6. Although blue cells and white cells are configured at the same time in this case, the red cells play an important role in the formation of the pattern similar to the previous models. Another thing that is worth mentioning is that for all three models the state bits 1 and 2 come into existence only after the spare bits establish some sort of positional information within the grids. It appears that the system has evolved the spare bits to have a function of triggering and stopping, therefore in effect controlling the growth of target pattern cells.

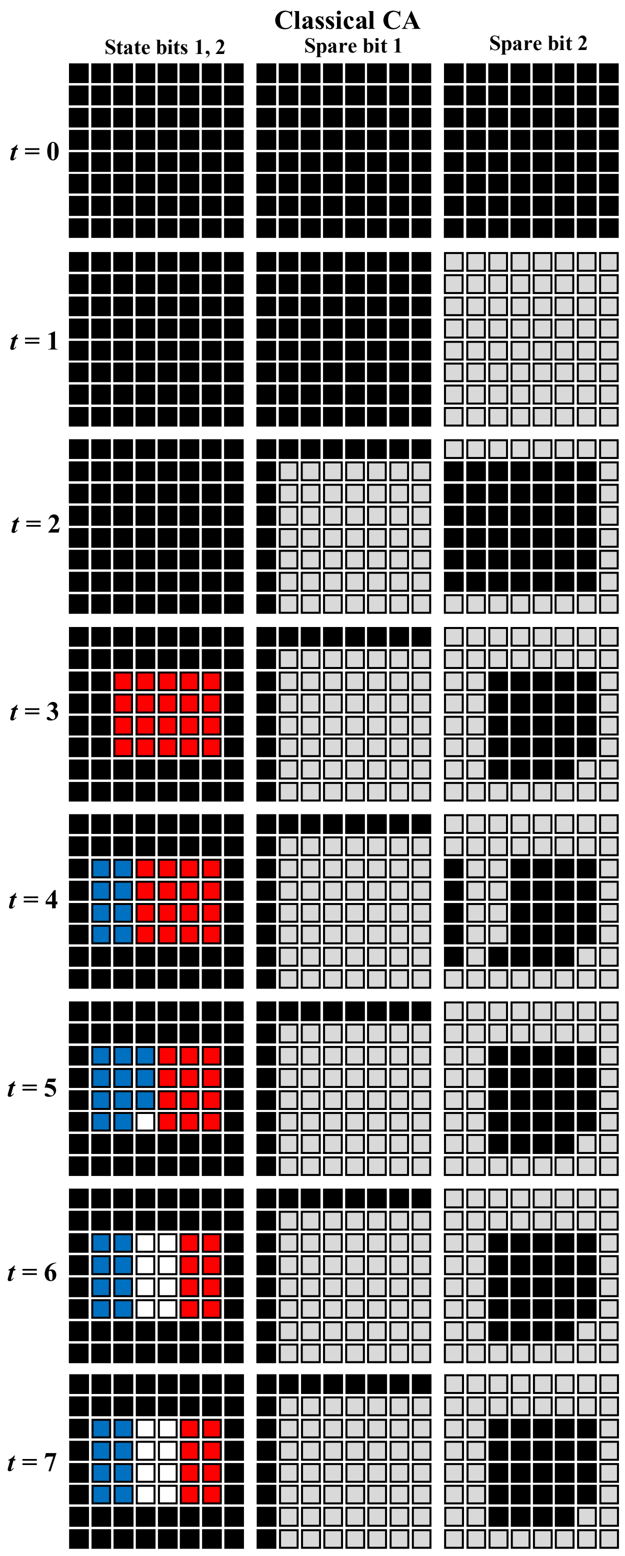


Figure 22. Figure shows the development of a sample evolved organism on the classical CA model. The time steps are shown from top to bottom. The first two bits of each cell is combined, colour coded and shown on the left column and the spare bits are shown on the middle and right columns.

Analysis of Results

The experimental results in this section confirms our predictions on the evolvability of developmental generation of regular patterns over random patterns rather conclusively, at least on these three investigated models. Superiority of our models with the cell overwriting capability over the classical CA model is confirmed in every experiment. These experiments also further affirm the need for spare bits in development.

However, there is a major difference in the success rates of our two models when compared to the random pattern evolution experiments. Unlike its performance in Experiments 1 and 2, the toroidal model is equally successful with the fixed bordered model in Experiments 3 and 4, and it is the most successful model in Experiment 5. We will attempt to analyse the differences between these experiments.

If we take a detailed look at these models, we find that there are three major differences between Experiments 3-5 and Experiment 2. The first and the most obvious difference is the difference in the nature of the target pattern, regular or random. This difference can account for the superior results achieved by Experiment 5; however it cannot by itself account for the superiority of the toroidal model over the fixed bordered model, as shown by the equal performance in Experiments 3 and 4.

The second difference is the total number of bits. Experiment 5 uses a 256-bit configuration string, whereas Experiment 2 uses a 175-bit configuration string to fully configure the grid. However, the target patterns for both experiments are quite similar to the 128-bit string is used in Experiment 5, and a 125-bit string is used for the other making the overall configuration size difference less important. Besides, this difference also applies to all three models equally, and cannot account for the superiority of the toroidal model.

The third and the most important difference is the difference between the grid sizes of the two experiments. Experiment 5 uses an 8×8 grid, whereas Experiments 1 and 2 use a 5×5 grid. The larger grid size required by the Experiment 5 may result in a system where the positional information is harder to be determined or conveyed. We know from our analyses of the developmental cycles of our models that the first thing that happens is the establishment of the borders, and then the delivery of this information towards the central cells. In the case of a larger grid this makes the developmental task much more complex, and consequently the evolutionary task gets harder. However, the increased connectivity introduced by the toroidal model greatly reduces the maximum distance between any two cells. This greatly improves the gathering and delivery of positional information on a larger grid. Although the increased connectivity serves to make the problem more complex, the gains from this outweigh the increased complexity problem in larger grids.

4.4 Discussion

Our experiments have shown us that regular patterns are more evolvable than random patterns in the DCM. Beyond the comparisons between random and regular patterns, some patterns seem to be easier to be evolved whereas others fail to result in a single successful evolution. Therefore we can deduce that while these pattern evolution experiments are quite generic, and the model can be applied to other problems, the fixed target pattern evolution task even with the spare bits creates a big constraint on evolution specifically because some patterns are harder to construct.

5. Conclusions

We opened this paper with the statement of the long-term aim of this research field: to find ways to automatically create and maintain complex computational and electronic systems.

We have given detailed explanation of our developmental cellular model (DCM). We have clearly identified its properties and relations to other developmental models. More than anything else we have stressed the importance of stability, and context sensitivity. We have described our evolutionary algorithm and experimental setup.

In the core of the paper, we have detailed the results of our direct, developmental pattern evolution experiments. This section was separated into two major parts where we separately examined and compared the evolution of random and regular patterns. From the random pattern experiments we have learned that the use of spare bits in each cell that are only used by the developmental process without affecting the fitness is vital to improve the evolvability of more complex patterns. What is really interesting in the spare bit experiments is that while the overall task stayed as hard as before (i.e. evolve a 125-bit long pattern), with the introduction of the two spare bits the evolvability of the model improved. In other words, with the introduction of the spare bits, the search space of the experiment got many times larger, but on the other hand, it gave two extra bits of information storage that the program may use to grow and stabilize without impacting the fitness measure, which really is an increase in the computational power of each cell.

Most pattern generation experiments in the literature were conducted using a basic target pattern. The reason for using basic shapes for these experiments is that basic shapes are easier to evolve with developmental systems. Basic shapes or regular patterns usually have a repetitive or modular structure. Since developmental representations are literally compressed versions of their respective adult organisms, they are more suitable for representing regular patterns; hence their evolution was easier compared to the random patterns, as supported by our experimental results.

The results for direct random pattern evolution, which is the most versatile task, suggest only the fixed bordered model presents some evolvability. Referring only to this task would discard the toroidal and the classical CA models straight away. Nevertheless, these results should be mitigated by the fact that practical applications often involve regular patterns. In particular, as demonstrated, compressible and more easily representable regular patterns greatly improve the evolvability of all the models. Our fixed bordered and toroidal developmental models clearly demonstrate their superiority in terms of evolvability, but the classical CA model's performance is also greatly improved. It is also interesting to note that even if the fixed bordered model is more evolvable, and therefore more useful, the toroidal model is significantly faster to evolve.

Beyond the comparisons between random and regular patterns, some patterns seem to be easier to be evolved, whereas others failed to result in a single successful evolution. Therefore, we can deduce that while these pattern evolution experiments are quite generic, and the model can be applied to other problems, the fixed target pattern evolution task even with the spare bits, create a big constraint on evolution specifically because some patterns are harder to construct. To increase the evolvability of our system, one approach is to make the evolutionary process indirect by introducing a disjunction between the fitness function and the cellular pattern.

References

[1] Wolfram, S. (1986) Approaches to Complexity Engineering, *Physica D*, **22**, 385–399.

[2] von Neumann, J. (1966) *Theory of Self-Reproducing Automata*, Urbana/London:University of Illinois Press.

[3] Ulam, S. (1952) Random Processes and Transformations, in *Proceedings of the International Congress of Mathematicians,* Vol 2.

[4] Burks, A.W. (1970) *Essays on Cellular Automata,* Urbana/London:University of Illinois Press.

[5] Herman, G.T. (1973) On Universal Computer-Constructors, *Information Processing Letters,* **2**, 61–64.

[6] Gardner, M. (1970) The Fantastic Combinations of John Conway's New Solitare Game 'Life', *Scientific American,* **223**(4), 120–123.

[7] Wolfram, S. (1983) Statistical Mechanics of Cellular Automata, *Reviews of Modern Physics,* **55**(3), 601–644.

[8] Wolfram, S. (1984) Computation Theory of Cellular Automata, *Communications in Mathematical Physics*, **96**:15–57.

[9] Wolfram, S. (1984) Universality and Complexity in Cellular Automata, *Physica D,* **10**, 1-35.

[10] Wolfram, S. (2002) *A New Kind of Science*, Champaign, IL:Wolfram Media.

[11] Wolpert, L. (1998) *Principles of Development,* Oxford: Oxford University Press.

[12] Kumar, S. (2004) *Investigating Models of Development for the Construction of Shape and Form*, PhD Thesis, University College London.

[13] Miller, J.F. and Thomson, P. (2004) Beyond the Complexity ceiling: Evolution, Emergence and Regeneration, in *Proceedings of the GECCO 2004 Workshop on Regeneration and Learning in Developmental Systems.*

[14] Banzhaf, W. and Miller, J.F. (2004) The Challenge of Complexity, in Menon, A. (ed.) *Frontiers of Evolutionary Computation*, Kluwer Academic Publishers, 73–79.

[15] Hogeweg, P. (2000) Evolving Mechanism of Morphogenesis: on the Interplay between Differential Adhesion and Cell Differentiation, *Journal of Theoretical Biology,* **203**, 317–333.

[16] Streichert, F. at al. (2003) Evolving the Ability of Limited Growth and Self-Repair for Artificial Embryos, in Banzhaf, W. et al. (eds.), *Proceedings of the 2003 European Conference on Artificial Life*, Vol. 2801 of Lecture Notes in Computer Science, Springer-Verlag, 289–298.

[17] Basanta, D. et al. (2004) Evolving and Growing Microstructures of Materials using Biologically Inspired CA, in *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware.* Seattle, Washington. June 24-26th, 2004, IEEE Computer Society.

[18] Mange, D. et al. (2000) Towards Robust Integrated Circuits: The Embryonics Approach, *Proceedings of the IEEE*, **88**(4), 516–541.

[19] Pfeifer, R. (2004) Interacting with the Real World: Design Principles for Intelligent Systems, in *Proceedings of the Ninth International Symposium on Artificial Life and Robotics (AROB)*, Oita University, Vol. 1, 13–18.

[20] Kniemeyer, O. et al. (2004) A Graph-Grammar Approach to A-Life, *Artificial Life,* **10**(4), 413–431.

[21] Miller, J.F. (2003) Evolving Developmental Programs for Adaptation, Morphogenesis and Self-Repair, in *Proceedings of the 7th European Conference on Artificial Life*, Lecture Notes in Computer Science Vol. 2801, Springer-Verlag, 256-265.

[22] Stanley, K. and Miikkulainen, R. (2003) A Taxonomy for Artificial Embryogeny, *Artificial Life*, **9**(2), 93–130.

[23] van Remortel, P. et al. (2004) Gene Interaction and Modularisaion in a Model for Gene-Regulated Development, in *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, 253–260, IEEE Computer Society.

[24] Kim, J. (2001) Transsys: a Generic Formalism for Modelling Regulatory Networks in Morphogenesis, in Kelemen, J. and Sosik, P. (eds.) *Advances in Artificial Life: Proceedings of the 6th European Conference on Artificial Life*, Vol. 2159 of Lecture Notes in Artificial Intelligence, 242–251, Springer-Verlag.

[25] Leung, C.H. and Berzins, M. (2003) *A Computational Model for Organism Growth based on Surface Mesh Generation,* London: Academic Press.

[26] Liu, H. et al. (2004) An Intrinsic Robust Transient Fault-Tolerant Developmental Model for Digital Systems, in *Proceedings of the GECCO 2004 Workshop on Regeneration and Learning in Developmental Systems.*

[27] Macias, N.J. and Durbeck, L.J.K. (2002) Self-Assembling Circuits with Autonomous Fault Handling, in Stoica, A. et al. (eds.), *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* 46–55.

[28] Roggen, D. and Federici, D. (2004) Multi-Cellular Development: is there Scalability and Robustness to Gain?, in *Proceedings of PPSN VIII: The 8th International Conference on Parallel Problem Solving from Nature*, 391–400.

[29] Turing, A. (1952) The Chemical Basis for Morphogenesis, *Philosophical Transactions of the Royal Society B,* **237**, 37–72.

[30] Meinhart, H. (1982) *Models of Biological Pattern Formation*, London:Academic Press.

[31] Meinhart, H. (1998) *The Algorithmic Beauty of Seashells*, Heidelberg:Springer-Verlag.

[32] Murray, J.D. (1993) *Mathematical Biology* (2nd Edition), Heidelberg:Springer-Verlag.

[33] Wolpert, L. (1969) Positional Information and the Spatial Pattern of Cellular Differentiation, *Journal of Theoretical Biology,***25**:1–47.

[34] Öztürkeri, C. and Capcarrere, M.S. (2004) Emergent Robustness and Self-Repair through Developmental Cellular Systems, in *Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, MIT Press.

[35] de Garis, H. (1999) Artificial Embryology and Cellular Differentiation, in Bentley, P. (ed.), *Evolutionary Design by Computer*, 281–295, Morgan Kaufmann.

[36] IBM (2006) *An Architectural Blueprint for Autonomous Computing,* Fourth Edition. Retrieved June 2009, from http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf

[37] Miller, J.F. and Thomson, P. (2000) Cartesian Genetic Programming, in *Genetic Programming: Proceedings of the 2000 Conference on Genetic Programming,* Lecture Notes in Computer Science Vol. 1802, 121–132, Springer-Verlag.

[38] Rothermich, J. and Miller, J.F. (2002) Studying the Emergence of Multicellularity with Cartesian Genetic Programming in Artificial Life, in *Proceedings of the 2002 UK Workshop on Computational Intelligence.*

[39] de Garis, H. (1992) Artificial Embryology: The Genetic Programming of an Artificial Embryo, in Soucek, B. (ed.) *Dynamic, Genetic and Chaotic Programming*, New York: John Wiley, 373–393.

[40] Fleischer, K. and Barr, A.H. (1992) 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,* 389-416, Addison-Wesley.

[41] Fleischer, K. (1996) Investigations with a Multicellular Developmental Model, in Langton, C.G. and Shimohara, K. (eds.) *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, 229–236, MIT Press.

[42] Fleischer, K. (1995) *A Multiple-Mechanism Developmental Model for Defining Self-Organizing Geometric Structures*, Technical Report, California Institute of Technology.

[43] Eggenberger, P. (1997) Evolving Morphologies of Simulated 3D Organisms Based on Differential Gene Expression. In *Proceedings of 4th European Conf. on Artificial Life*, 205–213, Springer-Verlag.

[44] Eggenberger Hotz, P. (2003) Combining Developmental Processes and Their Physics in an Artificial Evolutionary System to Evolve Shapes, in Kumar, S. and Bentley, P. J. (eds), *On Growth, Form and Computers*, 302–318, Academic Press.

[45] Eggenberger Hotz, P. (2004) Asymmetric Cell Division and its Integration with Other Developmental Processes for Artificial Evolutionary Systems, in *Artificial Life IX,* MIT Press.

[46] Rechenberg, I. (1994) *Evolution Strategy '94*, Stuttgart:Fromman-Holzboog.

[47] Hogeweg, P. (2000) Shapes in the Shadow: Evolutionary Dynamics of Morphogenesis, *Artificial Life* **6**, 85-101.

[48] Furusawa, C. and Kaneko, K. (1998) Emergence of Multicellular Organisms with Dynamic Differentiation and Spatial Pattern, in Adami, C. et al. (eds.) *Artificial Life VI,* 79–93, MIT Press.

[49] Federici, D. (2004) Using Embryonic Stages to Increase the Evolvability of Development, in *Proceedings of WORLDS Workshop on Regeneration and Learning in Developmental Systems*, GECCO 2004.

[50] Bentley, P. and Kumar, S. (1999) Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem, in *Proceedings of the Congress on Evolutionary Computation*, IEEE Press, 35–43.

[51] Harding, S. L. and Miller, J. F. (2006) Dead State: A Comparison between Developmental and Direct Encodings, in *Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle.

[52] Poli, R., Langdon, W.B. and McPhee, N.F. (2008) *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, (With contributions by J. R. Koza).

[53] Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998) *Genetic Programming—An Introduction*, San Francisco:Morgan Kaufmann.

[54] Walker, J. A., and Miller, J. F. (2008) The Automatic Acquisition, Evolution and Reuse ofModules in Cartesian Genetic Programming, *IEEE Transactions on Evolutionary Comptuation*, **12**(4), 397–417.

[55] Vassilev, V. K., and Miller, J. F. (2000) The Advantages of Landscape Neutrality in Digital Circuit Evolution. In Miller, J. F., Thompson, A., Thomson, P., and Fogarty, T. C. (eds.), *Evolvable Systems: From Biology to Hardware*, Vol. 1801 of Lecture Notes in Computer Science, pp 252–263, Springer-Verlag.

[56] Yu, T., and Miller, J. F. (2002) Finding Needles in Haystacks is not Hard with Neutrality, in Foster, J., Lutton, E., Miller, J.F., Ryan, C., and Tettamanzi, A.G.B., *Genetic Programming: Proceedings of the Fifth European Conference on Genetic Programming*, 13–25, Springer-Verlag.

[57] Miller, J. F. (2002) What Bloat? Cartesian Genetic Programming on Boolean Problems, in *Late Breaking Papers of the 3rd Genetic and Evolutionary Computation Conference (GECCO'01),* 295–302.

[58] Koza, J.R. (1994) *Genetic Programming: Automatic Discovery of Reusable Programs*, MIT Press.

[59] Miller, J. F. (1999) An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach, in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1135–1142, Morgan Kaufmann.

[60] Miller, J. F. (2004) Evolving a Self-Repairing, Self-Regulating, French Flag Organism, in *Proceedings of the 2004 Genetic and Evolutionary Computation Conference.*

[61] Lindenmeyer, A. (1968) Mathematical Models for Cellular Interaction in Development. Parts I and II, *Journal of Theoretical Biology*, **18**, 280-315.

[62] Öztürkeri, C. (2008) *Investigation of Developmental Methods for Growing Self Repairing Programs,* PhD Thesis, University of Kent, UK.

[63] Agresti, A., and Coull, B. (1998) Approximation is Better than ‘Exact’ for Interval Estimation of Binomial Proportions. *The American Statistician*, **52**, 119–126.

[64] Newcombe, R. G. (1998) Two-Sided Confidence Intervals for the Single Proportion: Comparison of Seven Methods. *Statistics in Medicine*, **17**, 857-872.