University of Kent

Kent Academic Repository

Öztürkeri, Can and Johnson, Colin G. (2014) *Self-repair ability of evolved self-assembling systems in cellular automata.* Genetic Programming and Evolvable Machines, 15 (3). pp. 313-341. ISSN 1389-2576.

Downloaded from

https://kar.kent.ac.uk/51431/ The University of Kent's Academic Repository KAR

The version of record is available from

https://doi.org/10.1007/s10710-014-9216-2

This document version

Publisher pdf

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in *Title of Journal* , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

Genet Program Evolvable Mach (2014) 15:313–341
DOI 10.1007/s10710-014-9216-2

Self-repair ability of evolved self-assembling systems in cellular automata

Can Öztürkeri · Colin G. Johnson

Received: 19 April 2013 / Revised: 1 October 2013 / Published online: 12 March 2014
© Springer Science+Business Media New York 2014

Abstract Self-repairing systems are those that are able to reconfigure themselves following disruptions to bring them back into a defined normal state. In this paper we explore the self-repair ability of some cellular automata-like systems, which differ from classical cellular automata by the introduction of a local diffusion process inspired by chemical signalling processes in biological development. The update rules in these systems are evolved using genetic programming to self-assemble towards a target pattern. In particular, we demonstrate that once the update rules have been evolved for self-assembly, many of those update rules also provide a self-repair ability without any additional evolutionary process aimed specifically at self-repair.

Keywords Cellular automata · Robustness · Repair · Self-repair · Self-assembly

1 Introduction

Self-repairing systems are those that are able to reconfigure themselves when damage occurs to a part of the system, to bring the functioning of the system back to some defined normal state. This paper investigates the ability of systems that have been evolved to *self-assemble* to also *self-repair* without any additional learning stage specifically aimed at self-repair; self-repair comes "for free" as a *side-effect* of self-assembly. These experiments are carried out using a form of cellular automata (CA), with the update rules evolved using a form of *genetic programming*

C. Öztürkeri · C. G. Johnson (✉)
School of Computing, University of Kent, Canterbury, Kent CT2 7NF, UK
e-mail: c.g.johnson@kent.ac.uk

C. Öztürkeri
e-mail: canozturkeri@gmail.com

called Cartesian Genetic Programming (CGP), guided by a fitness function that promotes self-assembly of the states on a grid to a target pattern.

We show that these behaviours can be obtained for some kinds of cellular automata-like systems, which we term developmental cellular models (DCMs), as they are inspired by the notion of development in biology. The main way in which these systems differ from classical CAs is that the change of state at a particular cell also affects the state in neighbouring cells.

Furthermore, these experiments demonstrates a particular kind of self-repair behaviour where there is no symbolic-level distinction between the *detection* of a fault and the *repair* of that fault. Instead, the usual behaviour of the system is able to demonstrate its robustness by providing self-repair as part of the regular functioning of the system.

This paper is divided into four main sections. In Sect. 2 a number of relevant areas of the literature are reviewed: self-repair, developmental systems, and genetic programming. Section 3 explains the model used in the experiments, whilst Sect. 4 gives details of the experimental methods used. The results of the experiments themselves, consisting of testing the self-repair ability of evolved self-assembling structures, are detailed and discussed in Sect. 5 This is then followed by a conclusion and suggestions for future work.

2 Literature review

A number of areas of the research literature are relevant to this paper. In this section, we review three areas: the general idea of self-repairing systems, the use of biological development as an inspiration for building self-repairing systems, and a particular technique—CGP—that is used to evolve update rules in our work below.

A *self-repairing* system is a system that is able to reconstruct a faulty part so that function is restored when a fault occurs. This is a stronger concept than that of *fault-tolerance*, which is where a system is able to carry on functioning normally despite faults (perhaps within a tolerance level in terms of amount of time spent in a non-functioning state). In a self-repairing system the original system is reconstructed by modification of the system, usually whilst it is still active. Hence, self-repairing systems are, by definition, fault-tolerant.

Existing self-repairing systems in computer science (e.g. [11, 13–17, 19, 33]) do not physically reconstruct, but they rather reconfigure available parts to reconstruct the original circuit/logic/etc. In a computational system such as those studied below, this will consist of modifying the state of components in the system to bring the system back to its original state. Another approach would be to reconstruct around the fault, e.g. in a hardware system where a fault might be a physical fault that cannot be reconfigured.

Fault *detection* and fault *repair* are two distinct activities [32], and a simplistic view is that detection is a precondition for repair: first a fault is detected, then it is repaired. However, the approach that we will explore later in this paper does not involve an explicit detection step; instead, fault repair occurs by the reconstruction of a pattern, and every unit is constantly ready to reconfigure itself depending on

changes to its neighbouring states. In such systems, there is no need for an explicit fault-detection step.

There are a number of methods used for the explicit detection of faults. One of the most well-established methods is to run a number of designed-identical systems in parallel and use a majority voting system to decide which output to trust (as in triple module redundancy [12] and pair-and-spare [8] methods). Another is to carry out regular offline testing against a test pattern [7]. These existing approaches have problems—resource overhead in the voting system, and downtime in the offline testing approach. Furthermore, these systems run into the problem that the voting/testing system might itself develop a fault. By contrast, the framework that is presented here avoids these weaknesses through a continuous growth and maintenance process.

2.1 Self-repair in developmental systems

As noted above, the detection of faults is a difficult problem; one method to solve this problem is to use a system that is continuously renewing itself, that is, a system that is stable in the desired working state, and unstable in any other state. After suffering damage, this system in theory should redevelop, re-stabilize, reconstruct into the desired stable working state [23]. This approach, though still rather undeveloped, has been hinted at to a greater or lesser extent in a certain number of previous studies [2, 4, 13, 14, 18, 36].

Indeed, the idea of a self-repairing system goes back to the early days of cybernetics. Ashby's Homeostat [3, 6] was a hardware device where a number of systems interacted, with the behaviour of each system changing to another behaviour if inputs varied beyond a threshold value, with the aim of producing dynamic stability. Interestingly, this was later described as being able to "survive something that no computer program, however adaptive, could survive—an attack with a pair of wire cutters" [29]. Another early system was that of Pask [27], where conductive threads were "grown" on a surface, potentially avoiding faults in the substrate and perhaps capable of being developed into a system that could rewire itself as new faults emerged.

One source of inspiration for such systems is the process of *development* in biology; that is, the growth of an organism from a single cell to maturity [38]. This is a complex process controlled by various chemical signalling processes between cells, where cells behave in a different manner according to the signalling context in which they find themselves. Much of development is concerned with *pattern formation*—the creation of stable patterns of cells from undifferentiated masses.

A number of computational systems have been created that draw on ideas from development. In particular, development is often combined with a learning process such as an evolutionary algorithm. This provides the learning algorithm with a large, structured search space to explore, without the need to evolve every fine detail of the final system. Developmental systems have been used in a number of applications, from creating realistic-looking biological structures in computer graphics systems [31] to learning neural network topologies [9]. A good overview is the edited book by Kumar and Bentley [10].

One key idea in developmental systems is that they are *self-regulating*; that is, there is a stable state (or states) of the system, and the system is continually regulating itself to bring itself to that stable state. This is achieved in an emergent manner by local interactions between cells and between cells and signalling gradients.

In summary, some of the characteristics of developmental systems of relevance to computer systems are as follows. The system should be sufficiently rich to allow the development of *complex* structures, though this does not usually mean that the individual components are themselves complex, indeed this is usually an emergent property of interactions. Biological developmental systems are based around interactions between *discrete cells*; in the biological system these will have some degree of movement, whereas in computational systems it is more likely that the cells will be in a fixed position with respect to each other [e.g. cells in a cellular automaton, circuit elements in an field programmable gate array (FPGA)]. The main communication is *local*; that is, interactions between neighbouring cells, though in some developmental processes larger chemical gradients across the mass of cells are important [38].

Another problem with self-repair is the difficulty of designing the reconstruction process. If the system uses development to construct itself in the first place, reconstruction after errors might be easier [24]. It may be possible to use development's modular, self-assembling structure to have self-repair built-in. Most approaches to self-repair utilize principles from developmental systems. We are now going to review some of the important contributions within this field.

A very interesting approach to self-repair came from Macias and Durbeck [13] where they represented the design of a special kind of FPGA called the cell matrix. The system is designable "by hand" just as with a normal FPGA; the part that separates it from a regular FPGA based system is its capability of reconfiguring itself on spare parts upon encountering faults. While a regular FPGA can only be configured from outside, the cells in the cell matrix can configure themselves and their neighbours. Any circuit with higher functions can be constructed from these cells. Using this system Macias and Durbeck created higher entities they termed supercells from a collection of these cells. The supercells were designed to be capable of testing nearby regions of the cell matrix for faults and configuring more supercells in those regions. Using this technique the entire chip's surface can be mapped, and only the non-faulty regions are rendered active. During the normal operation of the system, whenever a fault is detected, this process is repeated to build yet another functioning system. However, Macias and Durbeck did not give any details on how the faults are recognized after the circuit becomes operational. Faults will have to be determined from the input output patterns, and the self-repair process will have to be manually triggered. Furthermore, this process is carried out and the entire chip's surface is re-evaluated even if only one of the cells becomes faulty, resulting in an inefficient self repair process. Nevertheless, Macias and Durbeck's research represents a significant contribution to the self-repair field as it involves the self-repair operation on the hardware level, unlike many other software-only approaches.

Another self-repairing hardware-based system is that of Mange et al. [14, 15]. This uses a hierarchical set of reconfigurable hardware components. Self-repair in this system is triggered by the explicit detection of a fault. Once a fault is detected in a system, the functionality of connected components is moved elsewhere on the hardware, the connections remade, and a new, functioning soft-copy of the faulty component remade in the appropriate location. Interestingly, this has a hierarchical aspect. For example, initial attempts at repair will just be concerned with reconstructing the faulty component, and the components that immediately connect to it, somewhere on the same hardware component. If it proves impossible to find a location on the current component, e.g. because there is physical damage in many locations on the component, then the entire functionality of that component can be moved to a different component. This system has been built and tested at scale. In most applications of it, the individual components were hand-designed, and so this cannot be used to test the hypotheses that we are putting forward that evolved systems have self repair as an epiphenomenon, rather than being built in explicitly. Furthermore, explicit fault detection is required, which contrasts with the emergent reconstruction explored in this paper.

Liu et al. [11] have managed to evolve a stable "French flag" pattern on a hardware simulation similar to an FPGA. The evolved flag pattern is shown to have some self-repair capabilities in one sample experiment. This result illustrates the potential, but is a long way from demonstrating the self-repair property, being based on only one experiment on one evolved system.

Miller [19] has managed to evolve CGP programs running inside cells of cellular automata to generate French flag patterns. His model involved a chemical diffusion mechanism that allows the cells to gain positional information. Miller performed some limited self-repair experiments with the French flag pattern, where the evolved system continued to grow and fixed some of the errors, however some scarring was left in the end. The regeneration capabilities of the model were further investigated in another paper [17] where they also had a separate experiment investigating the behaviour of evolved German flags that have been joined together, a process likened to grafting. They showed that some cells dominate others and indicated that this might have implications for future work on software immunity utilizing developmental systems. Although Miller obtained interesting results in terms of morphogenesis and self-repair, he did not try to evolve stability. As we will detail in the description of the model used in this paper, self-repair is easier to achieve if the evolved systems's growth stabilizes after reaching an adult size, and the developmental process continues only for self-regulation or maintenance. Miller later tried to evolve stability with the French flags but he could not achieve a perfectly stable flag although some improvement could be seen [16].

Roggen and Federici [33] made a comparison of three different systems based on their scalability and robustness. The systems that were compared included a directly mapped model, a morphogenetic system, and a model based on cell chemistry. The morphogenetic system was described in a previous paper [34]. and the model based on cell chemistry is an extension to the system in Miller's French flag paper [18]. Unlike Miller's system where he evolved CGP, they evolved artificial neural networks (ANN) running inside cells in a cellular grid. The problem they chose to

perform the comparison with was the evolution of visual patterns. They found that while directly mapped system was superior to others with pattern sizes up to 32×32, the developmental approaches scaled much better due to the reduction in search space. It is important to note here that, while they did not specifically select for the ability of it, the developmental approaches exhibited a good level of self-repair behaviour and thereby provided further evidence that it may be possible to achieve self-repair for free with developmental systems.

Other research in developmental self-repairing systems include works by Streichert et al. [36], who studied growth regulation and self-repair in artificial embryology, and Prodan et al. [30] who discussed the degree of bio-inspiration attained within the field so far.

2.2 Cartesian genetic programming

Genetic Programming (GP) [28] is a well-known technique for automatically generating computer programs (and other executable structures) from a description of the desired behaviour, which draws on inspiration from biological evolution. A population of programs is generated at random, the performance of the programs evaluated, the best-performing ones mutated or crossed-over, and this process of evaluate-select-modify repeated until a sufficiently high-quality program is discovered.

The form of GP used in this paper is *Cartesian GP* [20]. This system uses nodes in a directed graph to represent the program. These nodes are encoded as a string of numbers, each tuple of numbers in the string representing one node, i.e. which function is represented by that node and which other nodes/inputs it obtains its information from. Therefore, the population of programs can be represented simply by lists of numbers, and the standard machinery of genetic algorithms [5] applied to this problem, rather than devising particular GP-specific operators.

An overview of the CGP method can be found in the recent book edited by Miller [20], and details of how CGP has been applied to this problem can be found in our earlier work [25, 26].

3 The developmental cellular model

The DCM is a cellular automata-like model that takes its inspiration from biological development. The key difference from classical CAs is that the update rule can also influence neighbouring cells. In this section we give a description of the model and various variants on it. A detailed description of this model can be found in our earlier work [25, 26].

The aim of the DCM and similar systems is *pattern development*. That is, the system will, through interactions between cells, form a specific pattern from an (in this case uniform) starting configuration. This provides a simple prototype for investigations in this area, where fitness evaluation is fairly quick. If a cellular system on this kind of problem can be shown to have the properties that we are

interested in, then it provides some *prima facie* evidence that it is worth exploring more complex developmental systems, such as developmental circuits or programs.

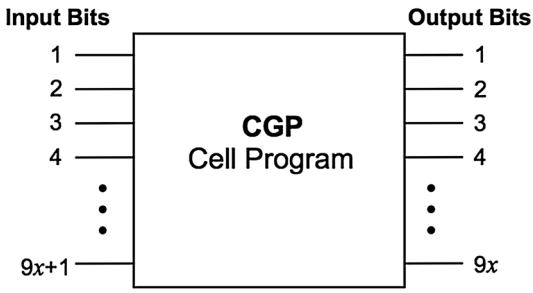
This section of the paper is split into two subsections. The first of these, Sect. 3.1 explains the DCM idea and gives details of how a DCM system is specified and executed. Section 3.2 details how the update rule in DCM systems can be evolved using the CGP method.

3.1 Specifying and running a DCM system

Each DCM system is based on an $n \times m$ grid of cells, each of which contains an x-bit state—these can be divided into x_{state} *state* bits that are relevant to the problem at hand and x_{spare} *spare* bits that are used during the assembly and repair phases but which are not part of the problem definition as such. These states can be read by the eight neighbouring cells in the grid (the *Moore neighbourhood*).

The states are changed over a number of discrete timesteps according to an update rule r, which is the same for each cell in the grid. The $9x + 1$ inputs to r are the x-bit states of the eight neighbours, the state of the cell itself, and a fixed bit (similar to the bias unit in neural networks) which supplies a constant *false* value. The aim of this additional bit is to allow the evolutionary value ready access to a constant where needed. Importantly, and in contrast to the classical CA model, the output is a $9x$-bit vector that changes the state of both the cell itself and its eight neighbouring cells (see Fig. 1). This is where we take inspiration from development, where local chemical interactions give rise to larger scale patterns. This is not, of course, the whole story of development—for example, larger scale chemical diffusion gradients play a role in overall structure [38]—but, similar processes have the potential to evolve in the spare bits of our system. As such, our system is somewhat different to the related system of Miller and Thomson [17] which uses an explicit representation of such gradients (see also [23]). The *capacity* for gradient-like structure to evolve in the spare bits remains, but the potential for other, richer structures to evolve is also available.

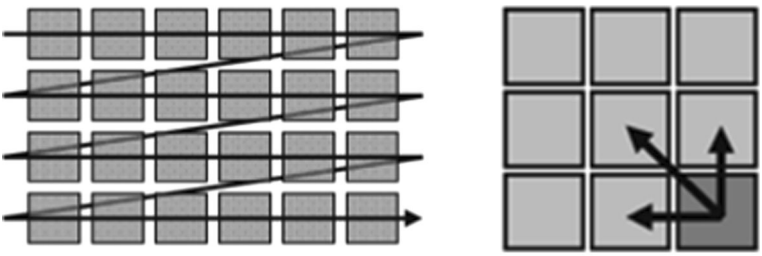
Classical CA use a so-called *democratic* method for updating the cells, where the state of all cells at time t is used to pre-calculate the new state at time $t + 1$, and then all cells are overwritten with this new set of states. In the DCM this is not possible, as states overwrite neighbouring states. As a result we use a serial "raster scan" model of update, where the state in the upper-left hand corner is calculated first, and the subsequent changes made to the neighbouring cells, before moving on to the next cell along in a horizontal direction until the end of the line, then along the next line, and so on, as illustrated in Fig. 2a. As a result there is a priority of update within the grid, with the lower-right hand cells in a local 2×2 area having the final influence on the neighbouring cells in a particular time-step (Fig. 2b). The aim of this is to provide the system with a fixed order of update, so that multi-step sequences can be evolved; the choice of the raster scan as the way of achieving this update is somewhat arbitrary. If an alternative method were to be used—for example, a random bit-by-bit update—then it would be *prima facie* harder for multi-step sequences of actions to evolve, as they could be disrupted by updates happening from other directions in the grid.



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

(a) **(b)**

Fig. 1 The structure of the update rule: **a** shows the structure of the update rule that is evolved using the CGP system, whilst **b** shows the mapping of the input/output bits to the x-bit state of the cell



(a) **(b)**

Fig. 2 The application of update rules. **a** Shows the update priority that is defined implicitly by the update method, whilst **b** shows the raster scan method by which the update rule is applied over the course of one timestep

Two different methods are available to process the cells at the edge of the grid. The first method is called the fixed-bordered model, where the grid is assumed to be surrounded by a set of *virtual cells* that retain a constant 0^x state, and which cannot be over-written (Fig. 3). The second variant is a *toroidal* model where the neighbours of an edge cell include the corresponding cells on the opposite edge of the grid, i.e. the edges of the grid have been identified to turn the rectangle into a torus (this is also known as *periodic boundary conditions*).

We have now described the key components of a DCM system. To summarize, we now formalise this. A DCM system is specified by a 5-tuple consisting of

- Whole numbers n and m specifying the size of the grid.
- A whole number x specifying the number of bits in each cell.
- The update rule r, which is a binary function from $9x + 1$ bits to $9x$ bits.
- A specification of whether the system is fixed bordered or toroidal at the edges.
- A limit t_{max} on the number of timesteps allowed for development.

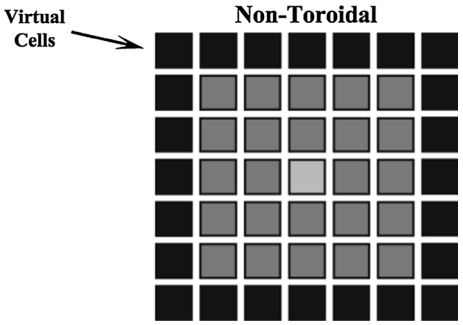


Fig. 3 In the fixed bordered model virtual cells are fixed edge cells that retain a 0^x state

To execute the DCM system, the following process is carried out

1. Initialise an $n \times m$ grid of cells, each of which has x binary states, all of which are initially set to 0.
2. Apply the raster scan method illustrated in Fig. 2 to update the states of the cells, using the update rule r.
3. Repeat step 2 until the pattern is stable or until it has been repeated t_{max} times.

In control experiments we will contrast the DCM with a *Classical CA* model. This is similar to the above, but uses the traditional democratic update method and an update rule that only updates the current cell rather than also affecting the neighbours.

3.2 Evolution of DCM systems using CGP

In the experiments below, CGP is used to evolve a population of DCM systems towards a system that self-assembles into a given target pattern. We refer to this as CGP/DCM. In practice, everything but the update rule r is fixed during a particular experiment; therefore, this could be seen as evolving a population of binary functions.

The CGP system creates individuals from the elementary binary functions. It has been shown [20] that CGP works well with a very small population and large numbers of generations. Therefore, in these experiments the population size is 5, and the population is allowed to evolve over 100,000 generations (Table 1). In each generation, only the best individual is selected and 4 mutants created from it to form the next population; this is commonly known as a $1 + \lambda$ evolution strategy [5]. A small mutation rate is used—this is common in CGP experiments because the influence of single mutations is often substantial. Recombination was not used. Full details can be found in our earlier paper [26] and experiments with differing population sizes have been given in [25].

4 Experimental methods

In a recent paper [26] we demonstrated that CGP/DCM is capable of evolving systems that *self-assemble*. That is, the final evolved update rule will, given a uniform $n \times m$ grid, develop into the desired pattern.

Table 1 Parameters for the CGP/DCM experiments

Parameter	Value
Population size	5
Number of generations	100,000
Selection strategy	$1 + \lambda$
Mutation rate	1 % per node
Number of nodes per CGP individual	100

In the experiments in this section we will revisit these experiments from the point of view of self-repair. In particular, we are interested to see whether the DCM systems evolved by CGP using a fitness function based solely on self-assembly. The *prima facie* reason for believing that this might be the case draws on the closeness of the development and wound-repair systems in biological systems. These two systems make use of related components and work in similar ways. It therefore seems reasonable to assume that there is some hope that system based on development and initial *assembly* of patterns might also contain the kind of robustness that would allow those patterns to *repair* themselves in the event of damage.

The self-assembly experiments are detailed in our earlier paper [26]. In short, the CGP/DCM algorithm was used with a fitness function that measures the Hamming distance between the current grid and target grid in the final two developmental timesteps. Perfect fitness is therefore present when the final pattern is identical to the target and stable. To ensure robustness of results, each experiment was run 100 times on three cases: fixed bordered, toroidal and the classical CA control case.

4.1 Fault model

In any study that wishes to assess the robustness and the self-repair qualities of a system, a precise fault model should be defined to make the scope of validity of the study clear. The fault model that is used in this work is simple and at the same time rather exhaustive. First, we only test here the self-repair abilities concerning the configuration process of the developmental cellular model. In fact, it could be said that we only test the self reconfigurability process. Therefore, we are not concerned with the faults that may occur in the function that maps the outputs of the evolved program to cellular states, and assume that this mapping is perfect. Within the scope of our study, the main limitation of our fault model is that it is a strike-once model, meaning, after a fault occurs, the reconfiguration of the pattern or the circuit, the repair phase, is done in a safe environment. Again, such a model is perfectly reasonable as long as this reconstruction process is fast and perfect. Fast here means that the model repairs itself within a few time steps, and the system is considered perfectly repaired when all configuration bits, including the spare bits are restored to their original states.

The fault model simply consists of choosing a certain number of cells to be affected by faults. Depending on the experiment, a cell can be affected by two types of faults:

Reset disruption: The state of the cell is reset to 0^x where x is the total number of bits each cell has.

Random disruption: The state of the cell is set to a random value.

These disruptions may seem over simplified; however, they encompass many different faults. Firstly, obviously they model a fault in the workings of the internal cell program that would entail writing the wrong state for itself and/or for its neighbours. Secondly, they also model a reading error. In effect if a cell misreads its environment, it makes a wrong decision (in interesting cases), and therefore at the next time step, the circuit has one or more cells in the wrong state. Finally, they also model writing errors.

It could be argued that the ability to carry out reset disruption is trivial, as the system is evolved to self-assemble from an initially blank state. However, this reset disruption is not applied to the entire cell grid, but only to a sample of points. Therefore, there is still interest in experiments using these disruptions, as they examine whether the system can recover from local resets as well as assembling from a complete reset of the whole cellular grid.

According to our fault model, the number of cells that are affected by the two types of disruptions can be randomly chosen. In this paper, each successfully evolved program can be separately subjected to five major kinds of disruptions: a one-cell reset, a five-cell reset, a five-bit random fault, a five-cell random fault, and a 2×2 block random fault. While the one-cell and the five-cell reset disruptions are applied exhaustively to all possible combinations, the random disruptions are applied 1,000 times. Therefore, while gentle time-wise; this error model can be considered as rather complete space-wise and working-wise. We also predict that since reset type faults are less disruptive to the patterns than random type faults, we expect that the evolved systems would be more resistant to them.

The fault is applied after a number of development timesteps; this will be referred to as the *disruption timestep* below. The self-repair is considered a success if the grid is restored (perfectly) back to the original state after a certain number of timesteps: this is referred to as *timesteps allowed for recovery* below. Experiments with different settings of these values can be found in [25].

4.2 Notes on experimental results

We should note that, as we have stressed while describing the fault model, in all self-repair experiments we are looking for perfect recovery. This means that an evolved system has to go through all error experiments, in some cases thousands of times and recover fully in each one without exception before it is declared perfect recovery. Although in some experiments (particularly those on random patterns) the number of tested systems is limited, even if a few of them manage to recover perfectly, it will still be an important indication to the model's abilities.

The confidence intervals have been calculated using the methods described by Agresti and Coull [1] and Newcombe [22].

Table 2 Setup for 125-bit random pattern experiment (experiment 1)

Grid size ($n \times m$)	$5 \times 5 = 25$ cells
Cell state bit-length (x)	5 bits
Total configuration bit-length	$25 \times 5 = 125$ bits
CGP node size	100 Nodes
Disruption timestep	8th timestep
Time steps allowed for recovery	20
One-cell reset disruption	Performed exhaustively
Three-cell reset disruption	Performed exhaustively
Five-bit random disruption	Performed 1,000 times
Five-cell random disruption	Performed 1,000 times
2×2 Block random disruption	Performed 1,000 times

Table 3 The table displays the number of perfectly recovered patterns for each disruption experiment in experiment 1

Disruption type	No. of successes Fixed bordered (6)
One-cell reset disruption	3 (13.9–86.0 %)
Three-cell reset disruption	3 (13.9–86.0 %)
Five-bit random disruption	1 (0.8–63.5 %)
Five-cell random disruption	1 (0.8–63.5 %)
2×2 Block random disruption	1 (0.8–63.5 %)

The numbers inside the brackets indicate the success rates with 95 % confidence. Except for the reset disruption experiments which are exhaustive, all experiments are repeated for 1,000 times to ensure that a success represents the the system being very likely to recover under any conditions

5 Experimental results and discussion

In this section we present the results of the self-repair experiments. Each section takes the successful evolved systems from our earlier work on self-assembly [26] and carries out self-repair experiments on them using the fault model above.

5.1 Experiment 1: self-repair experiments on 125-bit random patterns

The CGP/DCM was applied to evolving the self-assembly of 125-bit random patterns with the settings indicated in Tables 1 and 2. The results [26] showed that six of the fixed bordered experiments were successful in generating a self-assembling DCM system, whilst none of the toroidal or classical CA experiments were.

We applied the five different disruptions from our fault model to these six systems, with numbers of repeats given in Table 2. The results from these experiments are given in Table 3.

Table 4 Setup for 125-bit random pattern experiment with two spare bits (experiment 2)

Grid size ($n \times m$)	$5 \times 5 = 25$ cells
Cell state bit-length (x)	7 bits
Total configuration bit-length	$25 \times 7 = 175$ bits
CGP node size	100 Nodes
Disruption timestep	8th timestep
Time steps allowed for recovery	20
One-cell reset disruption	Performed exhaustively
Three-cell reset disruption	Performed exhaustively
Five-bit random disruption	Performed 1,000 times
Five-cell random disruption	Performed 1,000 times
2×2 Block random disruption	Performed 1,000 times

Overall, these results are inconclusive. Some repair is occurring, but there is such a small set of initial successful evolved systems that it is hard to conclude whether this is a strong effect or not.

5.2 Experiment 2: self-repair experiments on 125-bit random patterns with spare bits

This experiment uses exactly the same parameters as the earlier one except that for each cell there are two added spare bits. However, we should note that since we are looking for perfect recovery after errors, the added spare bits actually make the task harder. This is because according to our fault model, we are looking for perfect recovery meaning each bit of each cell including the spare bits has to recover its target state within the time given. In essence this task is equivalent to the self-repair experiment of 175-bit random patterns.

The settings are given in Tables 1 and 4. The results [26] showed that 22 of the fixed bordered experiments were successful in generating a self-assembling DCM system, whilst none of the toroidal or classical CA experiments were.

We applied the five different disruptions from our fault model to these 22 evolved systems, with numbers of repeats given in Table 4. The results from these experiments are given in Table 5. The evolved systems are more resistant to reset type faults. The reason for this result is due to the self-assembling nature of our model; it is specifically evolved to configure an empty cellular layer which is equivalent to resetting every cell in the grid. Even though the system is not accustomed to the random type errors, more than half of them still managed to self-repair perfectly in all 3,000 random disruption experiments.

5.3 Experiment 3: self-repair of a simple pattern

The remaining experiments work with regular patterns. Our recent paper [26] showed that self-assembly was much more effective on patterns with some degree of regularity. We would like to examine whether this follows through to self-repair.

Table 5 The table displays the number of perfectly recovered patterns for each disruption experiment in experiment 2

The numbers inside the brackets indicate the success rates with 95 % confidence

Disruption type	No. of successes Fixed bordered (22)
One-cell reset disruption	18 (58.9–94.0 %)
Three-cell reset disruption	18 (58.9–94.0 %)
Five-bit random disruption	12 (32.6–74.9 %)
Five-cell random disruption	12 (32.6–74.9 %)
2 × 2 Block random disruption	12 (32.6–74.9 %)

Table 6 Setup for simple pattern evolution experiment (experiment 3)

Grid size ($n \times m$)	$4 \times 4 = 16$ cells
Cell state bit-length (x)	1 bits
Total configuration bit-length	$16 \times 1 = 16$ bits
CGP node size	100 Nodes
Disruption timestep	8th timestep
Time steps allowed for recovery	20
One-cell reset disruption	Performed exhaustively
Three-cell reset disruption	Performed exhaustively
Five-cell random disruption	Performed 1,000 times
2 × 2 Block random disruption	Performed 1,000 times

The first target pattern used is the simple pattern on a 4×4 grid illustrated in Fig. 4. The settings are given in Tables 1 and 6. The results [26] showed that all 100 of the fixed bordered and toroidal experiments were successful in generating a self-assembling DCM system, whilst only 3 of the classical CA experiments were.

The results from this set of self-repair experiments are given in Table 7. This continues to show that that reset disruption faults are a lot gentler than random faults. The classical CA model unfortunately provided us with a very limited sample set and although none of the three systems managed to pass any self-repair experiment, numbers are too small to draw any conclusions here.

Apart from providing us with a larger sample size, the cellular developmental cycle of regular patterns are also easier to analyse visually compared to random patterns. Therefore at the end of this experiment we will now present an analysis of a sample evolved system that is subjected to self-repair tests. This was picked from one of the perfect evolved systems that successfully repaired itself under every self-repair test. For the visual analysis, we will subject the system to three different types of faults, a five-bit random fault, a five-cell random fault, and a five-cell reset fault which was found to have been gentler in our earlier experiments. Figure 5 shows the developmental cycle of the system on the fixed bordered model that is subjected to the three types of faults which are shown as a separate column each. The cells that are indicated by the red square and a cross at time step $t = 7$ are the ones that are affected by the introduced faults.

Figure 5 shows that while the reset disruption takes three time steps to heal, the random disruptions take four. The reasons for this effect are quite clear on the third column of the figure where only two cells are altered whereas on the random

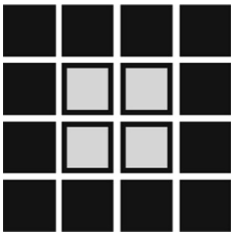


Fig. 4 The simple target pattern from experiment 3

Table 7 The table displays the number of perfectly recovered patterns for each disruption experiment in experiment 3

Disruption type	No. of successes		
	Fixed bordered (100)	Toroidal (100)	Classical CA (3)
One-cell reset disruption	84 (75.0–90.3 %)	80 (70.5–87.0 %)	0
Three-cell reset disruption	84 (75.0–90.3 %)	80 (70.5–87.0 %)	0
Five-cell random disruption	41 (31.4–51.3 %)	11 (5.8–19.2 %)	0
2×2 Block random disruption	41 (31.4–51.3 %)	14 (8.1–22.7 %)	0

The numbers inside the brackets indicate the success rates with 95 % confidence

disruption tests, three cells are altered. This is due to the random selection of cells; since some of the cells that are selected already have state 0, nothing happens to them under the reset fault. However, we cannot expect this effect to occur in more complex patterns where each cell has multiple-bit states. The single-bit state also results in the five-bit and five-cell random disruption tests to be equivalent in this case. Interestingly, in all three cases, we see that the pattern moves further away from the target pattern, before making a full recovery. This is probably due to the update priorities, and the propagation of the positional information from the borders (specifically the south eastern corner) of the grid. However, we will further examine this theory on a larger, more complex pattern before making any conclusions.

Another important issue that we should note before moving on to the next experiment is that the four time steps that it has taken to fully recover these patterns is not specific to these sample experiments. In fact the vast majority of the experiments conducted on perfect systems resulted with self-repair processes that concluded within four time steps. Similarly, the majority of the systems that were evolved for this pattern managed to stabilize within four time steps too. At this point we would like to point out that there might be a correlation between how quickly a pattern stabilizes during development to how quickly and more importantly how successfully it heals after faults are introduced. This also merits further investigations on larger cellular grids and more complex patterns. To that end, in the next experiment we will conduct self-repair tests on more complex patterns.

5.4 Experiment 4: self-repair experiments on diamond patterns

This experiment is similar to experiment 3, but the pattern being evolved is different, consisting of a diamond shape (see Fig. 6). The settings are given in

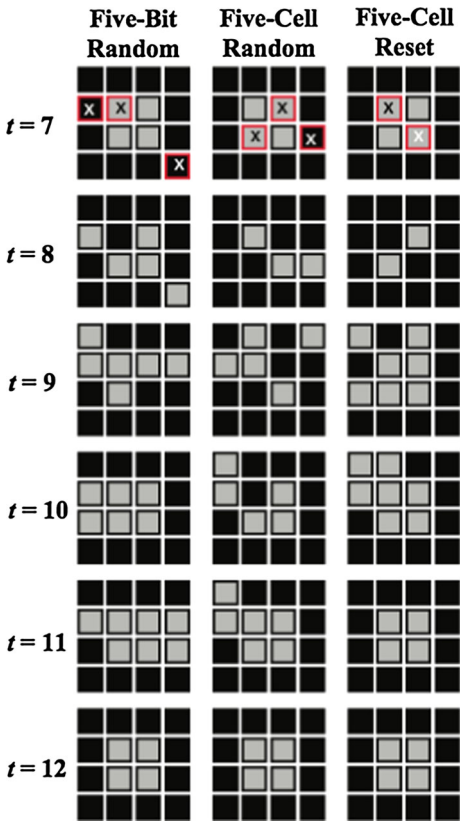


Fig. 5 The developmental cycle of an example system on the fixed bordered model that is separately subjected to three different types of faults at $t = 8$. The cells that are indicated by the *red square* and a cross are the ones that are affected by the introduced faults. The time steps are shown from *top to bottom* for each experiment (Color figure online)

Tables 1 and 8. The results [26] showed that all 100 of the fixed bordered and toroidal experiments were successful in generating a self-assembling DCM system, as were 74 of the classical CA experiments. Therefore, this gives a large sample size going into the self-repair experiments.

The results from this set of experiments are given in Table 9. The effects of the gentler reset type disruptions are more apparent in the results of this experiment compared to the previous one as can be seen. In this case only about 30, 3, and 18 % of the systems on the fixed bordered, toroidal, and the classical CA model that recovered fully for the reset disruptions perfectly healed after each random disruption experiment. Again, similarly all systems that fully recovered after the five-cell random disruption experiment fully recovered in all other disruption experiments, which clearly indicates that this specific disruption is the hardest to recover from. A very interesting and important result to name here is that while the toroidal model performed almost as well as the fixed bordered model on the reset type disruptions, it failed to show the same ability for the random type disruptions. Furthermore, unexpectedly the classical CA model also showed self-repair capabilities with two systems able to recover from all disruption experiments.

Since we have established from the results of this and the previous experiment that the five-cell random disruption is the hardest to recover from, we have decided

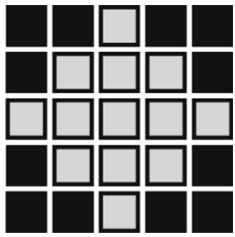


Fig. 6 The diamond target pattern from experiment 4

Table 8 Setup for diamond pattern evolution experiment (experiment 4)

Grid size ($n \times m$)	$5 \times 5 = 25$ cells
Cell state bit-length (x)	2 bits
Total configuration bit-length	$25 \times 2 = 50$ bits
CGP node size	100 Nodes
Disruption timestep	8th timestep
Time steps allowed for recovery	20
One-cell reset disruption	Performed exhaustively
Three-cell reset disruption	Performed exhaustively
Five-cell random disruption	Performed 1,000 times
2×2 Block random disruption	Performed 1,000 times

to conduct our visual analysis using only this type of disruption. The large sample set provided by the successful evolution experiment for each developmental model allowed us to conduct self-repair experiments that produced at least a couple of fully perfect evolved systems for each model. Therefore, the visual analysis shown in Fig. 7, shows the developmental cycle of three evolved systems that are subjected to the five-cell random disruption at $t = 8$, on the fixed bordered, toroidal and classical CA models respectively. The cells that are indicated by the red square and a cross at time step $t = 7$ are the ones that are affected by the introduced faults.

Similar to the simple pattern experiment, the patterns generated by the sample systems shown in Fig. 7 also move further away from the target patterns before making a full recovery. In the case of the fixed bordered model, the sample indicates the pattern starts to reorganize itself based on the positional information gathered from the borders and the correct pattern starts to form at the south eastern corner of the grid. However, this observation still needs more supporting evidence and it will be further investigated in the next experiment.

We also see from the figure that the classical CA model takes longer to recover from this fault. It does not mean that it is finding it more difficult, since this system has passed all self-repair experiments without a flaw. It just means that the stabilization and recovery process is more complicated for this model.

Another result that is worth noting here before moving on to the next experiment is that similar to the simple pattern experiment, the four time steps that it takes to fully configure the pattern in the fixed bordered model is not specific to this sample experiment. Similarly the vast majority of the experiments conducted on the perfect evolved systems resulted with self-repair processes that concluded within four time

Table 9 The table displays the number of perfectly recovered patterns for each disruption experiment in experiment 4

Disruption type	No. of successes		
	Fixed bordered (100)	Toroidal (100)	Classical CA (74)
One-cell reset disruption	69 (58.8–77.6 %)	65 (54.7–74.1 %)	13 (10.1–28.5 %)
Three-cell reset disruption	68 (57.8–76.7 %)	63 (52.7–72.2 %)	11 (8.0–25.5 %)
Five-cell random disruption	21 (13.7–30.5 %)	2 (0.3–7.7 %)	2 (0.4–10.3 %)
2×2 Block random disruption	22 (14.5–31.6 %)	4 (1.2–10.5 %)	2 (0.4–10.3 %)

The numbers inside the brackets indicate the success rates with 95 % confidence

steps. Looking at the formations of this pattern in the evolution experiment conducted in our earlier paper [26], we have found that majority of them managed to stabilize within four time steps too. This finding provides further evidence of the correlation between how quickly a pattern stabilizes during development to how quickly and more importantly how successfully it heals after faults are introduced. However, we cannot see the same correlation with our toroidal model which leads to the tentative conclusion that our fixed bordered model in general is more resilient to faults. The next and final experiment of this subsection will now investigate the validity of these conclusions in a more complex, and larger developmental cellular grid.

5.5 Experiment 5: self-repair experiments on French flag patterns

The biological developmental process and the differentiation of cells and thus the formation of tissue layers have been likened to the formation of striped patterns such as those found in French flags, and since then they were used as a benchmark experiment for both evolvability and self-repair in a number of previous studies [19, 23]. In our experiment here, we will investigate the self-repairability of the systems that were evolved in the French flag pattern evolution experiment in our earlier paper [26].

The bit-length for the French flags is 4 bits per cell. The first two are used to represent the different colours of the cells; the remaining two are left free for the developmental process during evolution. However, as with the previous experiments, the spare bits are considered part of the target pattern, and the system has to recover their correct values as well to achieve a perfect fitness. That essentially makes this experiment the recovery of 256-bit patterns.

The settings are indicated in Tables 1 and 10. The target image is shown in Fig. 8.

The evolution experiments resulted in a lower number of successfully evolved systems than the earlier experiments of this subsection. With 77 systems for the fixed bordered model, and 83 systems for the toroidal model, it still allows us to draw strong conclusions for these two models. For the classical CA model there were only 5 systems, not enough to present us with any meaningful results.

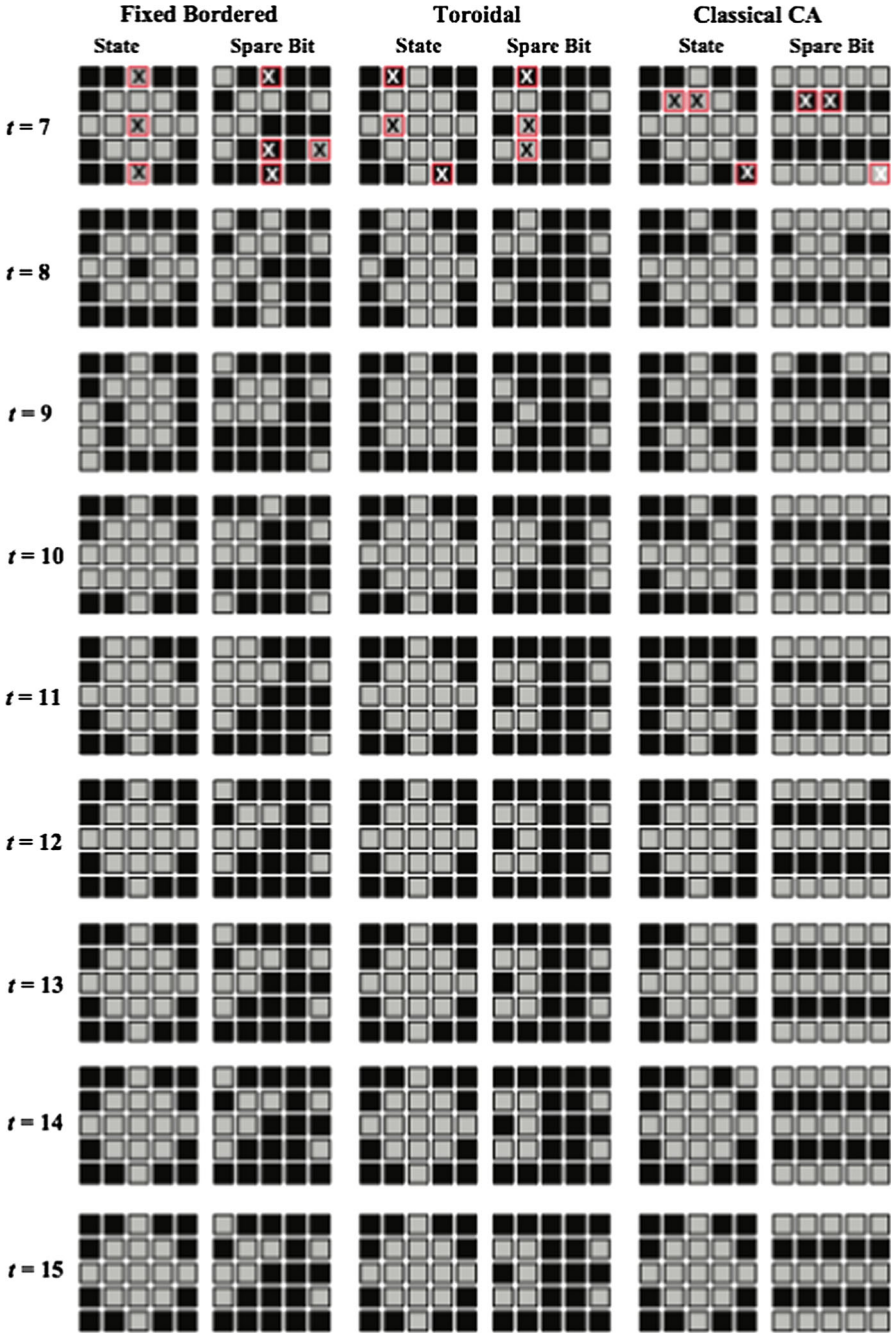


Fig. 7 The developmental cycle of three evolved systems that are subjected to the five-cell random disruption at $t = 8$, on the fixed bordered, toroidal and classical CA models respectively. The cells that are indicated by the *red square* and a cross are the ones that are affected by the introduced faults. The time steps are shown from *top to bottom*, and the spare bits of each model is shown on the *right columns* (Color figure online)

Table 10 Setup for French flag pattern evolution experiment (experiment 5)

Grid size ($n \times m$)	$8 \times 8 = 64$ cells
Cell state bit-length (x)	4 bits
Total configuration bit-length	$64 \times 4 = 256$ bits
CGP node size	100 Nodes
Disruption timestep	8th timestep
Time steps allowed for recovery	20
One-cell reset disruption	Performed exhaustively
Three-cell reset disruption	Performed exhaustively
Five-cell random disruption	Performed 1,000 times
2×2 Block random disruption	Performed 1,000 times

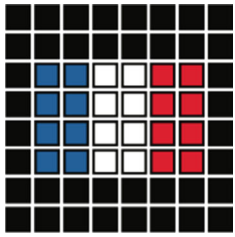


Fig. 8 The French flag target pattern from experiment 5 (Color figure online)

Similarly, as with the previous self-repair experiments, they were all tested with the four different types of fault experiments, as given in Table 10

The results are given in Table 11. We can see that as in the previous experiments, the reset type disruptions were a lot gentler than the random disruptions. We can also easily see that while it has shown similar self-repair ability with the reset disruptions, the toroidal model was a lot less successful compared to the fixed bordered model in random disruptions. Although the classical CA model had no systems that perfectly self-repaired in all tests, it is hard to generalize this result as the sample size is only five. Furthermore, since this model has shown that it is indeed capable of self-repair (however limited) in the previous experiment, and since when we take a closer look at the self-repair results, we see that on 30.5 % of the five-cell random disruption tests the systems were able to fully recover, we can still conclude that the classical CA model, being a stabilizing, developmental model, is capable of emergent self-repair.

Figures 9, 10 and 11 display the self-repair processes for a sample evolved system for each three developmental models. For all three figures, the left column displays the combined and colour coded two state bits of the cell, and the middle and right columns display the rest of the bits of the system. The colour-to-state mappings are as follows: 00: black, 01: red, 10: blue and 11: white. Similar to the previous visual examples of self-repair, they all start with a fully developed and stabilized pattern, in this case the French flag pattern at time step $t = 7$, and the five-cell random fault strikes at the 8th time step. The cells that are indicated by the red square and a cross at time step $t = 7$ are the ones that are affected by the introduced faults.

Table 11 The table displays the number of perfectly recovered patterns for each disruption experiment in experiment 5

Disruption type	No. of successes		
	Fixed bordered	Toroidal	Classical CA
One-cell reset disruption	59 (65.3–85.2 %)	52 (51.3–72.8 %)	0
Three-cell reset disruption	59 (65.3–85.2 %)	50 (48.9–70.6 %)	0
Five-cell random disruption	27 (24.8–46.9 %)	1 (0.1–7.5 %)	0
2×2 Block random disruption	28 (25.9–48.2 %)	1 (0.1–7.5 %)	0
Perfect patterns	27 (24.8–46.9 %)	1 (0.1–7.5 %)	0

The numbers inside the brackets indicate the success rates with 95 % confidence

The sample self-repair process for the fixed bordered model can be seen in Fig. 9. The five-cell random disruption strikes at $t = 8$ and affects 4 cells on the flag pattern, and several other cells on the spare bits. As we recall from our earlier discussions, we have predicted that for the fixed bordered model, the south east corner usually provides a strong anchor to the pattern where positional information is gathered and propagated towards the north east of the grid. When we look at this sample figure, it can be clearly seen that this process works as predicted. At $t = 9$ most of the errors on the cells that make up the pattern are not just fixed, but rather pushed towards the north east of the grid. The process continues at $t = 10$ and the pattern stabilizes successfully at $t = 11$. The same process can also be observed in the spare bits.

In Fig. 10, we can see the self-repair process after the fault strikes at $t = 8$ on the toroidal model. Indicated by the red squares and crosses, the affected bits are mostly the spare bits. Although the French flag pattern is affected in a limited way, the pattern takes a lot longer to recover compared to the fixed bordered model. This is because the spare bits are as important as the rest when it comes to stabilizing the pattern. The alterations in the spare bits immediately cause further, more extensive disruptions in the flag pattern in the following time steps. The French flag pattern starts to heal only after the spare bits recover their original state, and become useful in providing positional information. Since there are no border cells in this model, and the flag pattern does not wrap around and touch from the sides, establishing positional information is harder, and in this case mostly taken care of by the spare bits which are free to do just that.

In Fig. 11, we can see another sample self-repair experiment, this time performed on the classical CA model. However, as we recall from the experimental results, none of the systems managed to fully recover under every self-repair experiment. In this sample, we have picked the most resilient one of the five systems that were successfully evolved. Even so, this system did not manage to achieve a perfect recovery. As can be seen in the figure, although the general shape is more or less preserved through many time steps, the pattern never stabilizes.

Finally, from the sample experiment with the fixed bordered model, we see that the system manages to stabilize within three time steps. However after taking a closer look into the experimental data gathered from the 1,000 five-cell random

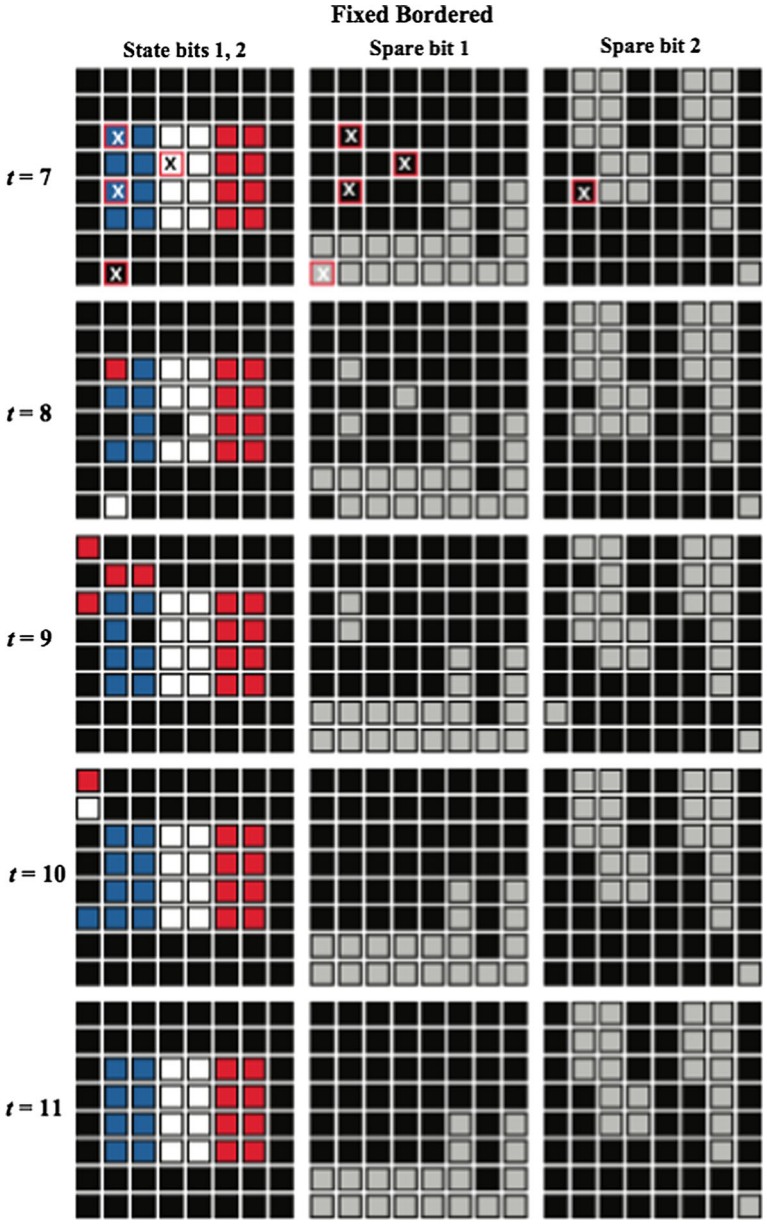


Fig. 9 The developmental cycle of an evolved system that is subjected to the five-cell random disruption at $t = 8$, on the fixed bordered model. The first two bits of the cells are mapped to a colour and displayed on the *left column*. The time steps are shown from *top to bottom*, and the spare bits are shown on the *middle* and *right columns*. The cells that are indicated by the *red square* and a cross are the ones that are affected by the introduced faults (Color figure online)

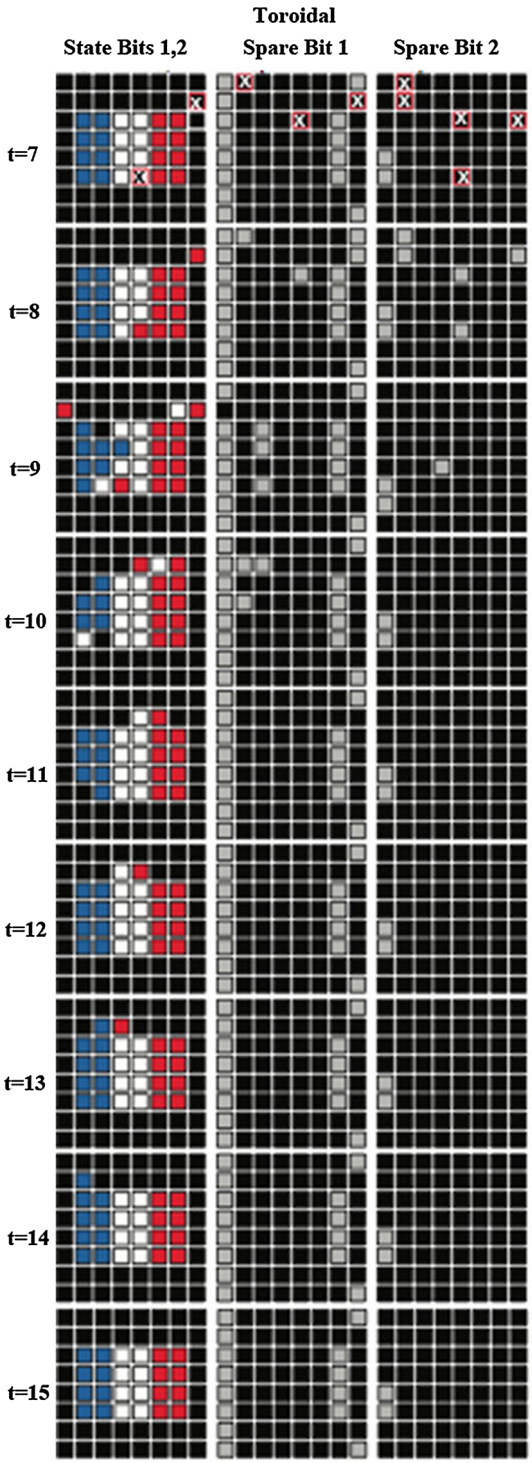


Fig. 10 The developmental cycle of an evolved system that is subjected to the five-cell random disruption at $t = 8$, on the toroidal model. The first two bits of the cells are mapped to a colour and displayed on the *left column* and the spare bits are shown on the *middle* and *right columns*. The cells that are indicated by the *red square* and a cross are the ones that are affected by the introduced faults (Color figure online)

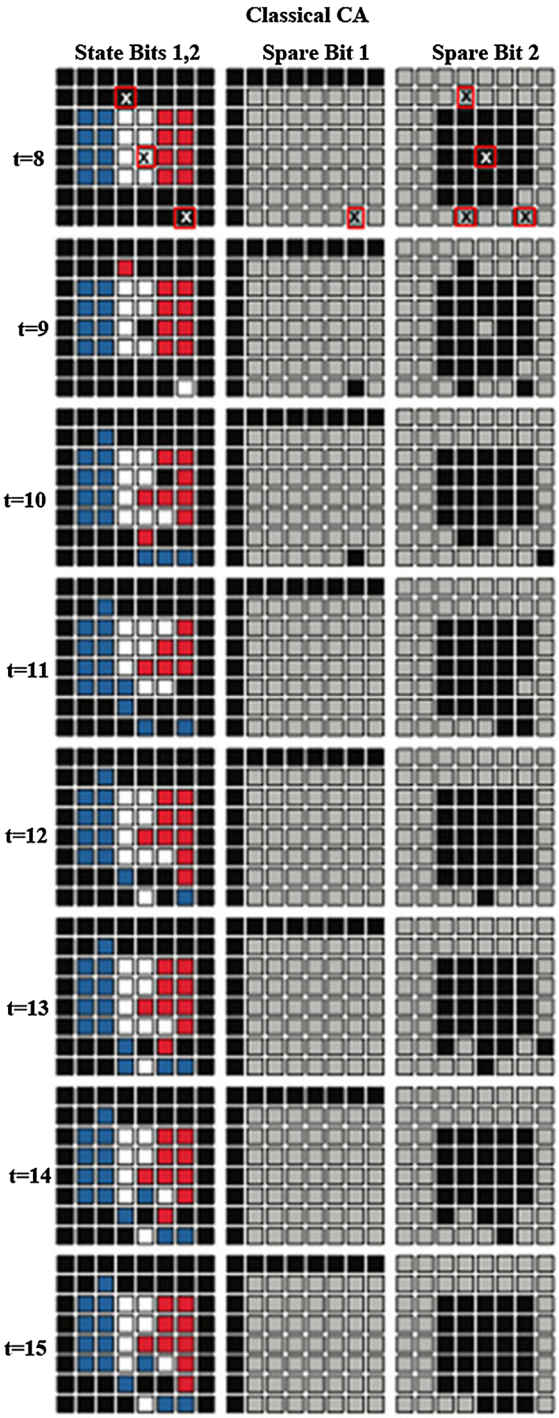


Fig. 11 Figure shows the developmental cycle of an evolved system that is subjected to the five-cell random disruption at $t = 8$, on the classical CA model. The first two bits of the cells are mapped to a colour and displayed at the *left column*. The time steps are shown from *top to bottom*, and the spare bits are shown on the *middle* and *right columns*. The cells that are indicated by the *red square* and a cross are the ones that are affected by the introduced faults (Color figure online)

disruption experiments that were repeated for each of the 27 fully perfect patterns, we have seen that for this model the system were able to recover within five to six time steps in majority of the cases. This provides further evidence in the case of the fixed bordered model to the correlation between how quickly a pattern stabilizes during development to how quickly and more importantly how *successfully* it heals after faults are introduced. However, it is now evident that the same correlation cannot be found with the toroidal case.

5.6 Overall discussion of experiments

To investigate the emergence of the self-repair ability with our model, we have set up and ran a number of error experiments. Knowing that in any study that wishes to assess the robustness and the self repair qualities of a system, a precise fault model should be defined to make the scope of validity of the study clear, we have described our fault model in detail prior to presenting our experiments. The fault model that we have used in the experiments that we have seen so far is simple and at the same time rather exhaustive. It consisted of two types of disruptions that we have named, reset and random type disruptions. We have first started our experiments on the successfully evolved systems of the random pattern evolution experiments. Since these experiments produced only a limited number of systems and although the indications were good, we have refrained from drawing conclusions and continued the self-repair experiments with systems evolved during regular pattern evolution experiments. With the large sample sets provided by these experiments, we were able to make a number of observations.

Based on the experiments that were run in this section, there are a few important outcomes. Firstly, to some degree, our developmental cellular system always exhibits a self-repair ability. Within a few evolutionary runs, it is possible to find, with a high probability with our fixed bordered developmental model, a fully perfect system that is stable even under the worst conditions. This is even true to some extent, for the less successful toroidal and classical CA developmental models.

Secondly, it is clear from the self-repair experiments that our fixed bordered model should be adopted within a framework that has resilience as an aim. While the results for the toroidal model are not bad in themselves, they do not bear comparison with the fixed bordered case, especially when it comes to the five-cell random disruption which is shown to be the hardest type of fault to recover from. The reasons for this difference are hard to spot at first. Obviously, the first idea that springs to mind is that borders favour stability, thereby favouring self-repair. However, the results in terms of evolvability seen in our previous paper [26], somehow contradict this hypothesis as the toroidal model's performance in general was only slightly worse than the fixed bordered model. In effect, evolution is considered successful if the system develops into the required target pattern, but only if it does so in a stable manner. Here, unlike in many other works, there is no explicit growth phase with a stopping time. The process stops itself by reaching a configuration that is stable. If the toroidal model was truly more unstable than its fixed bordered counterpart, similarly its evolution should also prove a lot harder,

which is not the case. Besides, becoming stable fast yet not achieving the target pattern is not better either.

In our recent paper [26] we have seen that larger cellular grids such as the one used with the French flag experiment make the signal flow and the gathering of the positional information harder. In the case of the French flag evolution experiment, we have found that the toroidal model was more evolvable, and concluded that the increased cellular connectivity introduced by the toroidal model greatly reduces the maximum distance between any two cells thereby making the propagation of the signals quicker. However, the toroidal model did not show any increased self-repair ability on the French flag pattern regardless of its evolvability performance thereby confirming the superiority of the fixed bordered model in both evolvability and self-repair ability.

Upon closer investigation of the developmental system both with the fixed bordered and the toroidal models, we have found that it was indeed easier for the fixed bordered model to gather positional information. In almost all cases the border cells determined their outputs from the virtual cells, thereby guaranteeing that they will always configure the grid into the correct state after a perturbation. Although the evolvability of the toroidal and the fixed bordered models are similar, and on average they each stabilize in the correct configuration within the same number of time steps; because of the high likelihood of the cells in the fixed bordered model to deliver correct outputs even after faults occur, this model is shown to be the most resilient in all self-repair experiments. Furthermore, this investigation has also revealed that for the fixed bordered model, there is a correlation between how quickly a pattern stabilizes to how resilient it is to the introduced faults.

Apart from these more general conclusions, we can also make an observation specific to our framework. During our investigations of the developmental process after the introduction of errors, we have seen that in most cases the pattern first moved further away from the target pattern before restabilizing into the final, correct configuration. This peculiarity is tied to the overwriting property of our model which results in cell state update priorities. As the cell at the south eastern corner of the grid has the highest update priority, the fault recovery is most likely to start from there which also results in the disruptions to be pushed northeast of the grid as we have seen with the sample French flag self-repair experiment on the fixed bordered model.

6 Conclusions and future work

In this paper we have demonstrated that a DCM with an update rule learned by Genetic Programming for self-assembly can produce self-repair behaviour without any additional learning steps aimed specifically at self-repair. This has been demonstrated on a number of problem types within the broad area of cellular pattern formation.

There are a number of areas for future work. Firstly, it would be of interest to see if this could be applied to other evolved patterns, for example self-assembled circuits in FPGAs. In particular, it would be useful to explore self-reconfiguration

around cells in the system that were permanently fixed in one configuration or not functioning, which would more closely model hardware faults.

There is clearly a lot of work to be done to bridge the gap between the self-assembly and self-repair of simple patterns on a grid and the repair of complex circuits. However, a rough analogy can be drawn between the different colours in the patterns with individual component types or with input-output behaviours of components.

One contrast between these simple pattern-assembly systems and circuits is that the fitness measure used to learn the assembly of circuits is indirect—the fitness of a circuit is based on its behaviour, whereas that of the pattern is direct. It is possible that new methods based on so-called "semantic operators"[21, 37]—which facilitate the exploration of spaces of program/circuit behaviour—may be of help here.

Another problem is to tease out how much of the *potential* self-repair ability is obtained "for free" from the self-assembly evolution. For example, it might be of interest to repeat the experiments above with an explicit self-repair based fitness function and measure the marginal benefits of this over the results already obtained.

Another area to explore is changes to the fault model. In particular, it would be interesting to explore the consequences of faults that can change the update rule as well as the cells in the system. Another variant on this system would be to train the rules so that they are learned from random patterns rather than all-zero patterns—we might expect such rules to be more robust still.

The DCM is a sequential model—the updates are carried out in a fixed sequence. It would be interesting to explore how important this is to the model, how effective a developmental model based e.g. on random updates (which would be more similar to the biological developmental process that this work takes its inspiration from) would be.

Another kind of alternative model is the Data-and-Signals cellular automaton [35], which is designed to provide a CA-type structure but with more similarity to traditional digital electronic components. In such an automaton, rather than it transmitting its state to its neighbours, a pair of computational units within each cell—the *processing unit* and *control unit*—process respectively *data* (the information that is relevant to the problem at hand) and *signals* (information that aids with the processing of the data and the growth of the activity in the automaton). It would be interesting to apply learning techniques of the kind outlined in this paper to such models and see if the robustness still obtains. This is particularly interesting as this would be taking us closer to the long-term aim of evolving robust circuits and related executable structures.

A different kind of comparative experiment would be to compare the self-repair ability of evolved systems with those systems that are written by hand. This would be hard to achieve with the current system because of the challenge of crafting the update rules by hand—it is difficult for a human programmer to grasp the implications of a change to the update rule to the emergent behaviour of the system as a whole. However, in other domains this might be possible. This is of particular importance because it would allow us to investigate the cause of the self-repair ability. Is the free self-repair ability a consequence of the *developmental* system or

is it a consequence of the fact that those programs have been *evolved* (or a combination of the two)? These are interesting questions with important consequences both for self-repairing systems and also for an abstract understanding of developmental and evolutionary biology.

References

1. A. Agresti, B. Coull, Approximation is better than 'exact' for interval estimation of binomial proportions. Am. Stat. **52**, 119–126 (1998)
2. M.S. Capcarrere, An evolving ontogenetic cellular system for better adaptiveness. BioSystems **76**, 177–189 (2004)
3. P.A. Cariani, The homeostat as embodiment of adaptive control. Int. J. Gen. Syst. **38**(2), 139–154 (2009)
4. H. de Garis, Artificial embryology and cellular differentiation. In: P.J. Bentley (eds) *Evolutionary Design by Computers*., (Morgan Kauffmann, Los Altos, 1999), pp. 281–295
5. A. Eiben, J. Smith, *Introduction to Evolutionary Computing*. (Springer, Berlin, 2003)
6. S. Franchi, Life, death and resurrection of the homeostat. In: S. Franchi, F. Bianchini (eds) *The Search for a Theory of Cognition: Early Mechanisms and New Ideas*., (Editions Rodopi, Amsterdam, 2011), pp. 3–52
7. M. Garvie, A. Thompson, Evolution of self-diagnosing hardware. In: *Evolvable Systems: From Biology to Hardware: Proceedings of the 5th International Conference*, vol. 2606, Lecture Notes in Computer Science, (2003), pp. 238–248
8. J. Gray, D.P. Siewiorek, High-availability computer systems. Computer **24**(9), 39–48 (1991)
9. F. Gruau, *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. Ph.D. thesis, Ecole Normale Superieure de Lyon, 1994
10. S. Kumar, P.J. Bentley (eds), *On Growth, Form, and Computers*. (Academic Press, New York, 2003)
11. H. Liu, J.F. Miller, A.M. Tyrell, An intrinsic robust transient fault-tolerant developmental model for digital systems. In: *Workshop on Regeneration and Learning in Developmental Systems at the 2004 Genetic and Evolutionary Computation Conference (GECCO 2004)*, 2004
12. R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability. IBM J. **6**, 200–209 (1962)
13. N.J. Macias, L.J.K. Durbeck, Self-assembling circuits with autonomous fault handling. In: A. Stoica, J. Lohn, R. Katz, D. Keymeulen, R.S. Zebulum (eds) *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, (IEEE Computer Society, Los Alamitos, 2002), pp. 46–55
14. D. Mange, M. Sipper, A. Stauffer, G. Tempesti, Towards robust integrated circuits: the embryonics approach. Proc. IEEE **88**(4), 516–541 (2000)
15. D. Mange, A. Stauffer, Introduction to embryonics: towards new self-repairing and self-reproducing hardware based on biological-like properties. In: N. Mangenat, D. Thalmann (eds) *Artificial Life and Virtual Reality*, (Wiley, London, 1994), pp. 61–72
16. J.F. Miller, W. Banzhaf. Evolving the program for a cell: from French flags to boolean circuits. In: S. Kumar, P.J. Bentley (eds) *On Growth, Form, and Computers*, chap. 15. (Academic Press, New York, 2003)
17. J.F. Miller, P. Thomson, Beyond the complexity ceiling: evolution, emergence, and regeneration. In: *Workshop on Regeneration and Learning in Developmental Systems at the 2004 Genetic and Evolutionary Computation Conference (GECCO 2004)*, 2004
18. J.F. Miller, Evolving developmental programs for adaptation, morphogenesis and self-repair. In: W. Banzhaf, T. Christaller, P. Dittrich, J.T. Kim, J. Ziegler (eds) *Advances in Artificial Life: 7th European Conference* (Springer, 2003), pp. 256–265
19. J.F. Miller, Evolving a self-repairing, self-regulating French flag organism. In: K. Deb, R. Poli, W. Banzhaf, H.G. Beyer (eds) *Proceedings of the 2004 Genetic and Evolutionary Computation Conference (GECCO 2004)* vol. 3102. Lecture Notes in Computer Science, (Springer, 2004), pp. 129–139
20. J.F. Miller (eds), *Cartesian Genetic Programming*. (Springer, Berlin, 2011)

21. A. Moraglio, K. Krawiec, C.G. Johnson. Geometric semantic genetic programming. In: *Parallel Problem Solving from Nature: PPSN XII—Lecture Notes in Computer Science*, vol. 7491, (Springer, 2012), pp. 21–31
22. R.G. Newcombe, Two-sided confidence intervals for the single proportion: comparison of three methods. Stat. Med. **17**, 857–872 (1998)
23. C. Öztürkeri, M.S. Capcarrere, Emergent robustness and self-repair through developmental cellular systems. In: J. Pollack, M. Bedau, P. Husbands, T. Ikegami, R. A. Watson (eds), *Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, (MIT Press, Cambridge, 2004), pp. 26–31
24. C. Öztürkeri, M.S. Capcarrere, Self repair ability of a toroidal and non-toroidal cellular developmental model. In: M. Capcarrere, A.A. Freitas, P.J. Bentley, C.G. Johnson, (eds) *Advances in Artificial Life: 8th European Conference on Artificial Life*, vol. 3630, Lecture Notes in Artificial Intelligence, (Springer, 2005), pp. 138–148
25. C. Öztürkeri, *Investigation of Developmental Methods for Growing Self-repairing Programs*. Ph.D. thesis, University of Kent, Canterbury, 2008
26. C. Öztürkeri, C.G. Johnson, Evolution of self-assembling patterns in cellular automata using development. J. Cell. Autom. **6**(4), 257–300 (2011)
27. G. Pask, *An Approach to Cybernetics*. (Harper and Brothers, New York, 1961)
28. R. Poli, W. B. Langdon, N.F. McPhee, *A Field Guide to Genetic Programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk , 2008. (With contributions by J. R. Koza)
29. W. Powers, *Behavior: The Control of Perception*. (Aldine, New York, 1973)
30. L. Prodan, G. Tempesti, D. Mange, A. Stauffer, Embryonics: Electronic stem cells. In: R.K. Standish, M.A. Bedau, H.A. Abbass, (eds) *Artificial Life VIII: Proceedings of the Eight International Conference on the Simulation and Synthesis of Living Systems*, (2002), pp. 101–105
31. P. Prusinkiewicz, *A. Lindenmayer*. (Springer, Berlin, 1990)
32. B. Randell, P. Lee, P.C. Treleaven, Reliability issues in computing system design. ACM Comput. Surv. **10**(2), 123–165 (1978)
33. D. Roggen, D. Federici, Multi-cellular development: is there scalability and robustness to gain? In: X. Yao, E. Burke, J.A. Lozano, J. Smith (eds), *Proceedings of PPSN VIII 2004: The 8th International Conference on Parallel Problem Solving from Nature*, vol. 3242, Lecture Notes in Computer Science, (Springer, 2004) , pp. 391–400
34. D. Roggen, D. Floreano, C. Mattiussi. A morphogenetic evolutionary system: phylogenesis of the POEtic circuit. In: A.M. Tyrell, P.C. Haddow, J. Torresen, (eds) *Evolvable Systems: From Biology to Hardware: 5th International Conference, ICES 2003*, (Springer, 2003), pp. 153–164
35. A. Stauffer, M. Sipper, The data-and-signals cellular automaton and its application to growing structures. Artif. Life **10**, 463–477 (2004)
36. F. Streichert, C. Spieth, H. Ulmer, A. Zell, Evolving the ability of limited growth and self-repair for artificial embryos. In: W. Banzhaf, T. Christaller, P. Dittrich, J.T. Kim, J. Ziegler (eds) *Advances in Artificial Life: 7th European Conference*, (Springer, 2003), pp. 289–298
37. L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: *16th European Conference on Genetic Programming*, (Springer, 2013), pp. 205–216
38. L. Wolpert, *Principles of Development*. (Oxford University Press, Oxford, 1998)