Evolving the Program for a Cell: From French Flags to Boolean Circuits

Julian F. Miller
School of Computer Science
The University of Birmingham, UK
j.miller@cs.bham.ac.uk

Wolfgang Banzhaf
Department of Computer Science
University of Dortmund, Germany
banzhaf@cs.uni-dortmund.de

1. Introduction

The development of an entire organism from a single cell is one of the most profound and awe inspiring phenomena in the whole of the natural world. The complexity of living systems itself dwarfs anything that man has produced. This is all the more the case for the processes that lead to these intricate systems. In each phase of the development of a multi-cellular being, this living system has to survive, whether stand-alone or supported by various structures and processes provided by other living systems. Organisms construct themselves, out of humble single-celled beginnings, riding waves of interaction between the information residing in their genomes – inherited from the evolutionary past of their species via their progenitors – and the resources of their environment.

Permanent renewal and self-repair are natural extrapolations of developmental recipes, as is adaptation to different environmental conditions. Multi-cellular organisms consist of a huge amount of cells, the atoms of life, modular structures used to perform all the functions of a body. It is estimated that there are of the order of 10^{13} cells in the human body. Some of them are dying and being grown again constantly, and it seems miraculous that an organism manages to remain stable. The developmental process supports an amazing variety of individual organisms of a species, yet these vast colonies of cells have the same basic body plan.

The contrast cannot be stronger to human-made systems and machines. The manner in which living systems are built is the antithesis of the way that we construct things. Functionally invalid until the very last part has been inserted, machines await construction from outside agents who follow plans the machines themselves have no clue about. Thrown into the world of action, machines are usually functioning in a very narrow band of environmental conditions, quickly wearing down if those conditions are not favourable. Being unable to repair themselves, they stop functioning once the weakest part breaks. This is true also for electronic systems and software. Transistors, the non-linear elements of our computer world, have a defined working point. Environmental conditions leading to a deviation from this working point will quickly destroy electronic systems. Similarly, input to software systems that has not been thought about beforehand and prepared for, will usually throw a software system out of order or cause an exception. Thus, many of the simple things that living organisms do naturally, are fiendishly difficult for a computer program to accomplish.

Nowadays it is commonplace for computer chips to be built on a sub-micron ($<10^{-6}$ metres in size) scale. As we try to miniaturise electronic circuits further we start to meet problems that arise purely for reasons of size. Firstly, the wires get in the way and their density becomes unmanageable, secondly, we find ourselves facing immense problems in verifying that the circuits work according to their specification. Imagine building circuits on a molecular scale, where we might have to specify the locations of the order of 10^{20} molecules. How will we get them to the correct position? The real

difficulty associated with extreme miniaturisation is the problem of getting information from the world we live into the tiny world of molecules.

Living systems have used an entirely different strategy: The information is in the smallest unit to begin with! Cells can be thought of as incredibly sophisticated robots that eat food, interact with the intra-cellular environment and build copies of themselves. The problem that researchers in developmental biology face is to try to understand how cells can accomplish these feats. If we want to learn from the natural example, an important question arises: How can we eliminate the many things that are incidental to the development process in biology from those that are fundamental? This is where computer scientists may be able to help. Already, a number of attempts have been made by computer scientists to create models inspired by biological developmental. There are two main directions to the effort in this field. Some researchers are trying to model (as accurate as possible) biological processes to help developmental biologists understand how development works. Others are exploring the degree to which developmental approaches may help us solve certain problems in computer science. There is no doubt that both fields of work will ultimately be of great benefit to humanity.

This chapter is about our attempts to explore the capabilities of a highly idealised developmental computer algorithm. Although we hope that the ideas discussed here may be of benefit to developmental biologists (at least on a conceptual level) we expect our work to be of more interest to computer scientists. We hope to ask some of the questions that concern developmental biologists, but ask them of developmental computer algorithms. Two of the questions we address particularly are the following: (i) How can we define programs that run inside cells that construct complex structures, when each cell runs an identical program? and (2) How can we obtain structures that are self-regulating (e.g. once mature, remain at a fixed size)?

The main original contributions of this chapter lie in sections 3-6. In section 2 we review briefly various attempts at modelling developmental phenomena on computers and at computer algorithms that are inspired by, or attempt to utilise, developmental ideas. Already these subjects are sufficiently advanced that we cannot hope, in the limited space of this chapter, to do justice to their breadth and subtlety. In section 3 we describe a form of Genetic Programming that has been recently created and applied to developmental algorithms. We make no claims as to its particular suitability to this problem, although it has been found to be effective on a number of problems. In section 4 we apply it to the problem of growing computer programs in the form of directed graphs. In section 5 we apply it in a very different way to the problem of growing two-dimensional arrays of 'cells'. Much of the work reported here is new and therefore by definition immature, however we feel that it has produced some surprising and even beautiful results.

2. Understanding and utilising ideas from biological development with computers

2.1 Pattern formation and morphogenesis

In 1952 Turing discovered that spatial concentration patterns can be formed if two substances with different diffusion rates react with each other [Turing52]. This was surprising as normally diffusion is thought of as a smoothing-out process. Since that time there has been considerable interest in types of reaction diffusion systems as a way of understanding pattern formation in living systems [Othmer, Maini and Murray92, Meinhardt98]. The idea is that morphogenesis happens as a result of cells reading and responding to changes of chemical gradients and establishing body plans and other

patterning accordingly. One of the primary motivations behind this idea is that one must account for the phenomenon of cells acquiring positional information so that they differentiate and move in the right way to construct the whole organism [Wolpert 1981, 1998]. The mechanochemical models of morphogenesis try to take into account the mechanical properties of cells and biological tissue into the process of morphogenesis, so that pattern formation and morphogenesis take place together [Murray 1989]. Other theories stress the importance of chemotaxis and have been very successful in describing the life cycle of organisms such as the slime mould Dictyostelium discoideum [Kessin 2001]. A very interesting and novel model of D. discoideum was developed by Savill and Hogeweg who used a three-dimensional hybrid cellular automaton together with a mathematical model of the cAMP signalling system [Savill and Hogeweg 1997]. They showed that with just three processes: (i) production of camp, (ii) chemotaxis to camp, and (iii) cellular adhesion they could obtain most of the behaviour of real D. discoideum amoebae (e.g., streaming, mound formation, sorting into two cell types, falling over and crawling away) without changing model parameters. Furusawa and Kaneko have also conducted some very interesting work on the emergence of multi-cellular organisms [Furusawa and Kaneko 1998].

Nowadays new ' morphogens' , i.e. substances that govern and direct the development of morphological features, are being discovered all the time. We are learning more about the genes that are responsible for the production of these morphogens and their receptors. Bonner likens the process to one of watching an ever-increasing telephone book enlarge and hopes for simple explanations that transcend the minute details (such as reaction-diffusion systems) [Bonner 2000].

2.2 Lindenmeyer systems and graph re-writing

Lindenmeyer systems (L-systems) were introduced by Aristid Lindenmeyer [Lindenmeyer68]. An L-system is a so-called rewriting system in which symbols are the basic elements. The action of an L-system begins with a start symbol (e.g. A), that is called an axiom, and employs rules that map ("re-write") the symbols already present to a finite set of symbols (e.g. A, B, C, D). At each iteration of the re-writing process every symbol is re-written according to the applicable rules in parallel. For example:

Axiom : A

Rules:

$A \rightarrow AB$
$B \rightarrow BC$
$C \rightarrow D$
$D \rightarrow A$

Iteration 0:	A
Iteration 1:	AB
Iteration 2:	ABBC
Iteration 3:	ABBCBCD
Iteration 4:	ABBCBCDBCDA

Figure 2.1: L-system with four iterations

L-systems may have context dependency so that symbols will only be replaced if specific other symbol precede (or succeed) a given symbol. If the symbols are interpreted as controlling a robot that leaves an ink trail it is possible to draw structures that look like plants with just a few iterations [Prunsinkiewicz and Lindenmeyer1990].

Boers and Kuiper have adapted L-systems to develop the architecture of artificial neural networks (ANNs), notably controlling the numbers of neurons and their connections [Boers and Kuiper 1992]. They used an evolutionary algorithm (see section 3) to evolve the rules of an L-system that, when

applied to an axiom, generated feed-forward neural networks. A method for training neural networks called backpropagation was used for learning and the accuracy of the neural networks on test data was used to evaluate the quality of the network. This quality was returned as a number to the evolutionary algorithm (the fitness measure). Boers and Kuiper claimed that although their results were preliminary their method was able to evolve more modular neural networks that performed better than networks with a predefined structure.

As noted by Boers and Kuiper, Kitano had developed another method for evolving the architecture of an artificial neural network [Kitano 1990] using a matrix re-writing system that manipulated adjacency matrices. Although Kitano claimed that his method produced superior results to direct methods (i.e. a fixed architecture, directly encoded and evolved), it was later shown in a more careful study that the two approaches were of equal quality [Siddiqi and Lucas 1998].

Gruau devised a graph re-writing method called cellular encoding [Gruau 94]. Cellular encoding is a language for local graph transformations that controls the division of cells which grow into artificial neural networks. The cells (which we can identify as nodes in the ANN) store connection strengths (weights) and a threshold value. The cells also store a grammar tree that defines the graph re-writing rules and a register that defines the start position in the grammar tree. When cells divide the daughter cells are identical to their parent. The grammar tree was evolved using an evolutionary algorithm. This method was shown to be effective at optimising both the architecture and weights at the same time, and achieving as good performance with direct encoding required the testing of many candidate architectures [Gruau, Whitley, and Pyeatt 1996]. The authors found that cellular encoding could find small architectures whose structure and complexity fit the specificity of the problem. Others have successfully employed this approach in the evolution of recurrent neural networks that control the behaviour of simulated insects [Kodjabachian, and Meyer 1998]. Koza has successfully employed a modified cellular encoding technique to allow the evolution of programs that produce human-competitive designs for electric circuits and other engineering problems [Koza 1999].

Hornby and Pollack have also evolved context free L-systems to define three dimensional objects (table designs) [Hornby and Pollack 2001]. They found that their generative system could produce designs with higher fitness and faster than direct methods.

2.3 Simulated proteins, regulatory networks, cell metabolisms and axon growth

Fleischer and Barr created a sophisticated multi-cellular developmental test bed and included realistic models of chemical diffusion, cell collision, adhesion and recognition [Fleischer and Barr 1992]. Their purpose was to investigate cell pattern generation. They noted that the design of an artificial genotype which develops into a specific pattern is very difficult. They also noted that size regulation is critical and non-trivial and that developmental models tend to be robust to perturbations.

Jacobi created an impressive artificial genomic regulatory network for the formation of an ANN, where genes code for proteins and proteins activate (or suppress) genes [Jacobi 95]. His simulated proteins were one of six classes: signal proteins (that diffuse from cells and turn genes in other cells on or off), movers (that respond to 'matched' signal proteins and are responsible for exerting mechanical forces on cells), dendritic proteins, both excitatory and inhibitory (when matched to signal proteins, they cause outgrowth of dendrites from a cell), splitters (responsible for cell replication), differentiators (responsible for cell differentiation) and threshold proteins (responsible for establishing the neuron thresholds in cells when the growth phase is over). Eventually, a recurrent ANN was grown, complete with neurons and dendritic connections. Activation was determined from

the directions of dendrite growth (excitatory or inhibitory) and the weights were interpreted from the strengths of signal proteins. This way, Jacobi was able to evolve a genotype which would set up this complex system of cells, proteins and dendrites that would eventually define a fully specified ANN that was used to control a simulated Khepera robot for obstacle avoidance and corridor following.

Nolfi and Parisi evolved encoded neuron position and branching properties of axonal trees that would spread out from the neurons and connect to other neurons [Nolfi and Parisi 1991] and in later work introduced cell division using a grammar [Cangelosi, Parisi and Nolfi 1993]. Unfortunately the number of genes in the genome was growing as the number of neurons, thus leading to poor scaling behaviour of their approach. Cangelosi [Cangelosi99] has later designed another system for development of a neural network that makes explicit use of heterochrony, a notion discussed intensively in the developmental biology community [Gould77].

In the mid 1990s Kitano created a strongly biologically motivated model of neurogenesis and cell differentiation [Kitano 1995]. He evolved an encoded cell metabolism involving chemicals and enzymes. He created a simulation involving diffusion and reaction rules, active transport of chemicals through the cell membrane, cell division, death and interaction, nerve growth factors and chemically driven axon growth. These were indirect consequences of the basic metabolic system, in other words division/ death etc. were not explicitly coded into the system. The fitness of cell genotypes was evaluated from amounts of "DNA" synthesized, number of cells and total length of axons. There was no cell movement. His primary aim was to create a tool that could improve the understanding of biological phenomena at a cellular level.

Astor and Adami have created a developmental model of the evolution of ANN that utilises an artificial chemistry [Astor and Adami 2000]. They evolve genetic regulatory networks on a hexagonal grid (so that all neighbours are equidistant). Kaneko and Yomo show, in another example of the use of an artificial chemistry, how differentiation into different cell types can be explained on the basis of a modelled interaction [Kaneko and Yomo 2000]. Other work [Hoile and Tateson 2000] use reaction-diffusion systems on the basis of ANNs to produce reaction output.

Dellaert, in his thesis "Towards a biologically defensible model of development", investigated models of genetic regulatory networks, multi-cellular development, and neural transport [Dellaert 1995]. In his model of multi-cellular development he began with a two-dimensional square, which was the zygote (fertilised egg) that was forced to divide. Subsequent divisions caused further subdivisions of the original square. The cells had state information provided by a random Boolean network model. Cell behaviour could be altered through neighbourhood interaction. He represented the cell states as different colours and devised performance metrics that could assign a fitness to patterns automatically. He showed that he could successfully evolve a range of morphologies within the divided square cell context.

Eggenberger has investigated a sort of genomic regulatory network, that he calls differential gene expression [Eggenberger 1997, Eggenberger and Dravid 1999]. He has evolved morphologies of organisms in three dimensions and also pattern formation mechanisms for patterns resembling lepidopteran wings. He suggests that the complex genotype-phenotype mappings typically employed in developmental models allow the reduction of genetic information without losing the complex behaviour. He stresses the importance of the fact that the genotype will not necessarily grow as the number of cells, thus he feels that developmental approaches will scale better on complex problems.

Bongard and Pfeifer have evolved genotypes that encode a gene expression method to develop the morphology and neural control of multi-articulated simulated agents [Bongard and Pfeifer 2001].

They found that they could evolve agents that perform non-trivial behaviours in a physically-realistic, virtual environment.

Bentley and Kumar examined a number of genotype-phenotype mappings (including developmental) on a problem of creating a tessellating tile pattern [Bentley and Kumar 1999]. They found that the indirect developmental mapping (that they referred to as an implicit embryogeny) could evolve the tiling patterns much quicker, and further that they could be subsequently grown (iterated) to much larger sized patterns. One drawback of their method, however, was that the implicit embryogeny tended to produce the same types of patterns.

3. Cartesian Genetic Programming

In this chapter we have used a form of genetic programming known as Cartesian Genetic Programming (CGP) [Miller and Thomson 2000]. CGP is a particular form of Genetic Programming, that allows the automatic evolution computer programs [Koza 1992, Koza 1994, Banzhaf et al. 1998, Koza et al. 1999].

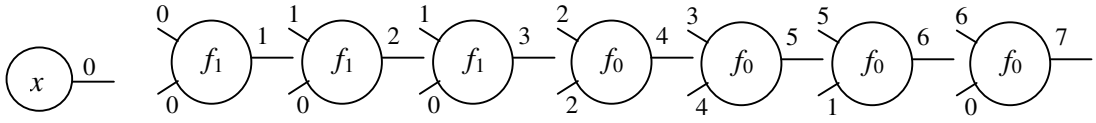
CGP encodes graph-like computer programs. The particular graphs used here are directed and acyclic graphs. Essentially the program is a mapping function, mapping input data to output data. In section 4 the programs map integers to integers, while in section 5, the programs map binary data to binary data. To clarify how this works we give a simple example of an encoded program that represents the mathematical function

$$g = x^4 + 2x^3 + x^2 + x$$

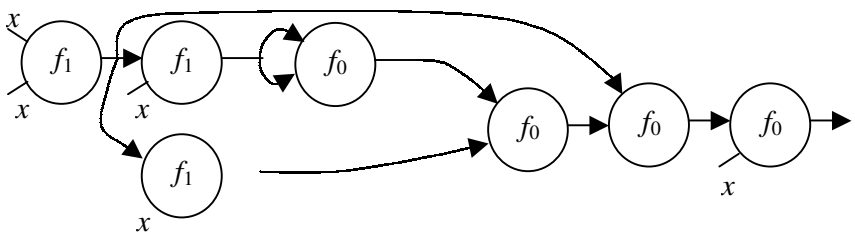
where the variable x stands for a numerical value. This can be represented using a set of primitive operations listed below:

$$f_0(a,b) = a + b$$
$$f_1(a,b) = a\,b$$

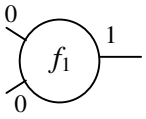
Now suppose that the value of x is made available to us. We will think of this value being supplied from a node labelled 0 (i.e. node 0 contains x). Function g can now be represented as an acyclic directed graph using the input values x and the functions f_0 and f_1.



In graph form this looks like



Now let' s look at the first function node



It is function 1 so it multiplies whatever is presented at its two inputs (labelled 0), these are both connected to the node that produces x, so this means that the value leaving the node is x^2. This value is labelled 1 (meaning the output of node 1). Node 2 is again a multiplication node and multiplies one input (x^2) by the other (x), thus producing x^3. The process continues in this way until we get to node 7 which outputs the function g. Note that the outputs of all nodes always connect to the input of a node on the right, this is called feedforward (hence directed, acyclic). In CGP graphs of this sort are represented by a list of integers. In this case each node requires three integers that describe what its inputs are connected to and the function of the node. The graph above would be represented by the following list of integers, which we call a chromosome, for reasons that will become apparent shortly (the node functions are represented by the function index, in boldface).

0 0 **1** 1 0 **1** 1 0 **1** 2 2 **0** 3 4 **0** 5 1 **0** 6 0 **0**

Now suppose that we altered one of these integers in the group 3 4 **0** to 3 2 **0**, something interesting happens. Node 4 is now not part of the graph and the function the graph represents has changed to g′

$$g' = x^4 + x^3 + x^2 + x$$

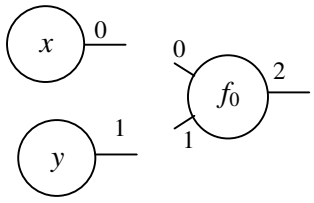
The $2x^3$ node (node 4) was disconnected and replaced the x^3 node (node 2). Now that node 4 is disconnected it could be changed without affecting the graph that was encoded by the chromosome. We will refer to node 4 as inactive. Changing a single integer in the chromosome is called a *mutation*. It is not difficult to guess now how we might go about *evolving* chromosomes of this sort. Next, suppose that somebody gave us a table of values for x and $h(x)$, where we are told that $h(x)$ is another polynomial, but we are not told its form. If we generate a chromosome at random (respecting, of course, that every third integer from the left must be a valid function label, and the two first integers in each triple, must be an integer less than the node label) we will always obtain some polynomial which we can evaluate to give a table of values for x. We could then compute the difference between the value for h predicted from our chromosome and the value for h given in the table. If we computed these differences for all the values in the table we could form a measure of how close our polynomial is to the unknown one. A measure of this sort is called a *fitness function*. We could proceed to solve this problem by evolving chromosomes of the form described. In all the experiments described in this paper we use the following algorithm:

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

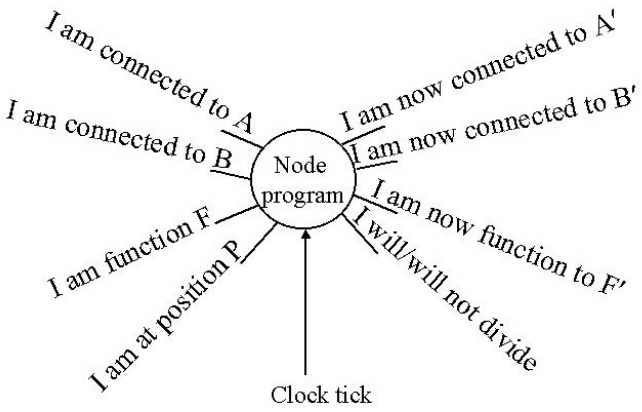
Step 3 is a crucial step in this algorithm: if more than one chromosome is equally good then always choose the chromosome that is not the *current_best* (i.e. equally fit but genetically different). In a number of studies this step has been proved to allow a genetic drift process which turns out be of great benefit [Vassilev and Miller 2000, Yu and Miller 2001, Yu and Miller 2002]. In later sections we may refer to the mutation rate, this is the percentage of each chromosome that is mutated in step 4. It should be noted that on some occasions (especially in section 5) the program that we are trying to evolve has many outputs. These outputs are taken consecutively from the rightmost node in the chromosome towards the left. The chromosome length is always made to be larger than the number of program outputs.

4. Developmental Cartesian Genetic Programming

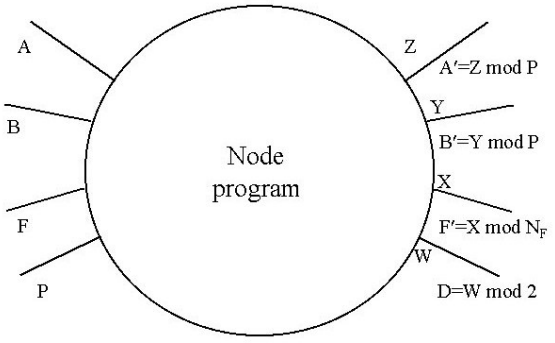
In this section we describe a new form of CGP which is developmental in character. The philosophical standpoint taken is that *the cell is the basic unit of biology*. We see evolution as a process of evolving a cell. The cell is a very clever piece of machinery that can, in co-operation with an environment, self-replicate and differentiate to form a whole organism. Thus in developmental Cartesian GP (DCGP) we attempt to evolve a cell that can construct a larger program by iteration of the cell's program in its environment. Since we are trying to construct graphs made of nodes it is clear that we need to identify a cell with a node and different nodes could be cells that have differentiated from each other. The environment of a node could be its position and its connections. Imagine that we start with a 'seed' node that can only be connected to the program inputs. The seed node will have a function also and a position. To clarify the idea let us suppose that the program we wish to build has two inputs, x and y (labelled 0 and 1). Let us also suppose that we have two node functions (or cell types) f_0 and f_1 each with two inputs, as in section 3, let f_0 add numerically the data presented to its inputs and f_1 multiply its inputs. Here is one possible seed node:



Now we want to run a program inside the node that uses information about the node to construct a new one, indeed, we also want it to be able to replicate itself, so that we can grow a larger program. The node needs four pieces of information, its two connections, its function and its position and the program inside the node needs to take this information and create a new node with connections and a function and whether the node should be duplicated. So the situation is now



What might the program inside the node look like? Well, we know that it must take in four integers and output four integers (the divide output could be just 0, for no replication, and 1, for replication). Thus we need to think up a way where a program (which we want to evolve) can do this. Well, if the program used the operations of addition and multiplication that would do fine. We have already discussed how to evolve programs of that form! However, we need to take care of one more point. We are using this program to construct a graph of nodes that is feed-forward so the integers that come out of the node program must take the right values (i.e. the connections must be to nodes on the left of our current position, the functions must be ones on the list of valid functions, and finally the divide must be 0 or 1). Unfortunately it is very difficult to construct programs that will automatically do that irrespective of the node position, so we need to resort to something that is a little inelegant. After we have run the node program we will need to carry out some sort of operation to bring the numbers into the correct ranges. A simple way to do this is to apply a modulo operation. Thus the new picture is shown below:



Let the outputs of the node program be W, X, Y, Z then if we divide Z and Y by the position P and take the remainder we will get two integers A′ and B′ which will be valid connections to nodes on the left of the current position P. If we have N_f possible node functions then the mod operation will once again map the output integer X to a valid function type F′. Finally we will need to take the integer, W, that decides whether the node is to be replicated, and find out whether it is even or odd using the mod 2 operation.

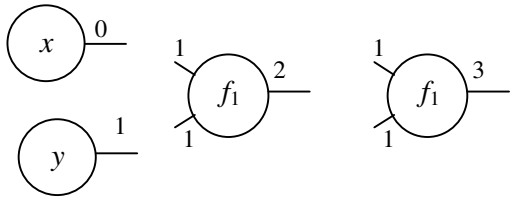
So far we have not defined a node program. Here will construct a simple one and apply it just to show that the whole idea actually works. Suppose the node program is defined by the following rules:

$Z = (2A+B)$, $A' = Z \bmod P$
$Y = (Z+F)$, $B' = Y \bmod P$
$X = Y + P$, $F' = X \bmod N_f$
$W = ZX$, $D = W \bmod 2$

Now we use the seed node defined earlier (A = 0, B = 1, F = 0, P = 2). Applying the above rules we get

$Z = (0+1) = 1$, $A' = 1 \bmod 2 = 1$
$Y = (1+0) = 1$, $B' = 1 \bmod 2 = 1$
$X = 1 + 2 = 3$, $F' = 3 \bmod 2 = 1$
$W = 3$, $D = 3 \bmod 2 = 1$

The divide signal is 1 so the new node is replicated and we obtain



The graph represents the function $g = y^2$. The *n* program outputs are assumed to be taken from the *n* nodes with the largest output labels (i.e. here we have one output taken from node 3).

Now applying the node program in each node we obtain

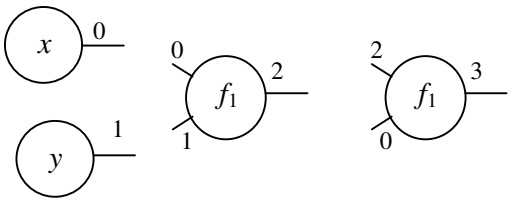
First node (P = 2)

$Z = (2+1) = 2$, $A' = 2 \bmod 2 = 0$
$Y = (2+1) = 3$, $B' = 3 \bmod 2 = 1$
$X = 3 + 2 = 5$, $F' = 5 \bmod 2 = 1$
$W = 2x5 = 10$, $D = 10 \bmod 2 = 0$

Second node (P = 3)

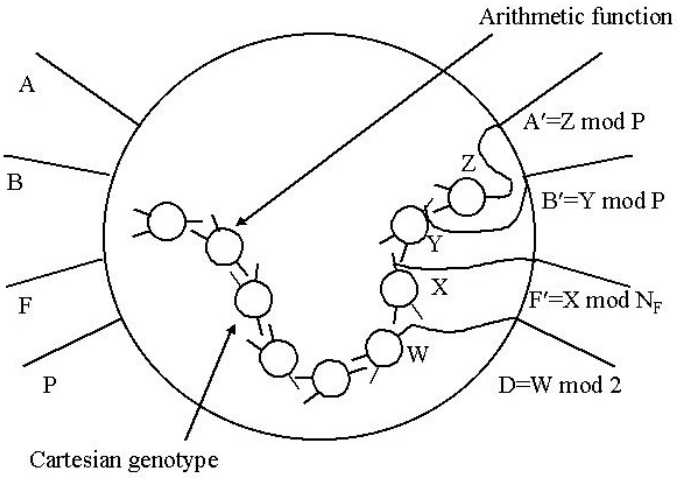
$Z = (2+1) = 2$, $A' = 2 \bmod 3 = 2$
$Y = (2+1) = 3$, $B' = 3 \bmod 3 = 0$
$X = 3 + 2 = 5$, $F' = 5 \bmod 2 = 1$
$W = 2x5 = 10$, $D = 10 \bmod 2 = 0$

So we now obtain



The nodes have connected themselves together and the graph represents the function $g = x^2y$.

How can we obtain the rules that define how nodes get transformed? Remember that the whole idea here was to evolve the rules inside a node, then apply those rules inside every node of the graph so that a new graph is built. But we already know how to represent rules that will map integers to new integers! We use a Cartesian program inside the nodes. So the picture now is:



Thus all the nodes of the developing graph have identical chromosomes inside them that represent the node program rules. For instance the rules we used in our example could be represented by a chromosome of the form:

0 0 **0** 4 1 **0** 5 2 **0** 6 3 **0** 7 5 **1**

Chromosomes of this form could themselves be subject to evolution using the algorithm defined in section 3.

Let us summarise what we have achieved here. We found described a way of building programs by running identical programs in nodes. We can iterate these programs and build enormous programs that would have required very large amounts of information to specify. We can evolve the node programs to try to obtain solutions to many problems. The way the method works is similar in some respects to real biology where cells replicate and differentiate to create enormous collections of cells that we call organisms. The hope is that it might prove easier to evolve relatively small programs and then iterate those programs to build very large programs rather than evolve the very large programs directly. Some early experiments have been performed using these ideas and it appears that for simple problems it is easier to evolve a chromosome that represents a program using a non-developmental approach (see section 3). Although, certainly, it would be inconvenient to have to evolve enormously long chromosomes it isn' t obvious that it would necessarily take more generations to evolve a good solution than evolving smaller developmental chromosomes. This is because as the size of a program grows there are more ways that it can solve a problem [Langdon 1999]. There are obvious reasons why nature chooses to build things in a developmental way. It is much easier for large organisms to carry little packets of information around that under the right circumstances can grow into a mature organism. Just think of a tree, its seeds can be blown or carried for miles by the wind or inside the gut of birds. Thus there are strong energetic reasons in favour of development.

The techniques described in this section have benefitted from a number of aspects of developmental biology, however it is not clear how the field of developmental biology can benefit. In the next section we will discuss another form of computer problem solving using developmental ideas that we think will hold some promise of contributing to developmental biology, inasmuch as it may allow the exploration of ideas that are relevant to that domain.

5. Evolving growing two-dimensional maps

In section 4 we described a method for evolving the program for a cell, where the cell was a node in a graph. As we continued to run the identical programs in the cells, graphs of increasing size could be constructed. After a pre-determined number of iterations of this process the graph (or organism) constructed was tested against a user specification of the problem that was to be solved. We found that is was necessary to impose modulo conditions on the outputs of the cells so that valid programs or graphs could result. Although the ideas behind the method were inspired heavily from biological development we wished to extend the ideas to an invented world that was in some sense, closer to the real biological world. In this way we hoped that our work would potentially shed some light at a conceptual level on real problems of biological development and, at the same time, allow us to address fundamental issues in the evolution of computer programs.

5.1 How cells and chemicals are represented

In this section we describe our attempts to create a two dimensional world of cells of different types, each carrying identical genotypes that may grow, die, change colour and receive and emit chemicals. As in section 4 we evolve the genotype of the cell so that after a period of time we obtain a two-dimensional map that meets the user's requirements. We emphasise that we have attempted to construct the simplest possible way of doing this. Our aim is to understand the importance of various factors in an embryological growth process and we feel that an overly complicated model would make it difficult to assess the relative importance of these factors. We wish to shed light on questions such as: How can collections of independent cells interact and coordinate their activity so that they build an organism? How can the cells build increasingly larger versions of that organism that are faithful to the same body plan? What is the relative importance of direct cell interaction and global chemical signalling? How can growth be regulated?

The world we envisage is binary, so that amounts of chemical, cell types and actions can be denoted as binary codes. The genotype is a representation that encodes a Boolean circuit (that implements the cell program) that maps input conditions to output conditions. A cell sees its own state and the states of its eight immediate neighbours. It also sees the amount of chemical (at present a single chemical) at the location of each of its eight neighbours. From this information the cell program decides on the amount of chemical that it will produce, whether it will live or die, whether it will change to a different cell type at the next time step, and how it will grow. Unlike real biology, when a cell replicates itself, it is allowed to grow in any or all of the eight neighbouring cells simultaneously (this is done to speed up growth, mainly for reasons of efficiency) . The situation is described in Fig 5.1

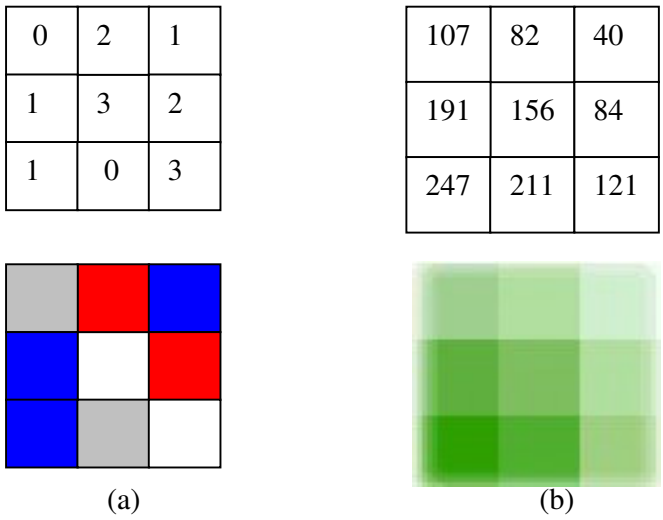


Figure 5.1: (a) shows a white cell surrounded by 6 neighbours (0 denotes an empty or dead cell, 1 a blue cell, 2 a red cell and 3 a white cell), (b) shows the chemical environment at the location of the cells.

Let us suppose that a chemical is represented by an eight bit binary code and that there are four cell types (dead, blue, red, white). The cell program reads simultaneously the eight bit codes for all the neighbouring locations, the two-bit code signifying the cell's own type, followed by the two-bit codes representing the types of all the neighbours in a clockwise fashion. The cell's program defined by its genotype (see below) then outputs the following bits: a four bit chemical code indicating the new strength of the chemical to be produced by the cell or daughter cells at the next time step, two new cell type bits, followed by an eight bit growth code that indicates where the cell will replicate itself, starting at NW and then denoting places to grow in an anticlockwise manner.

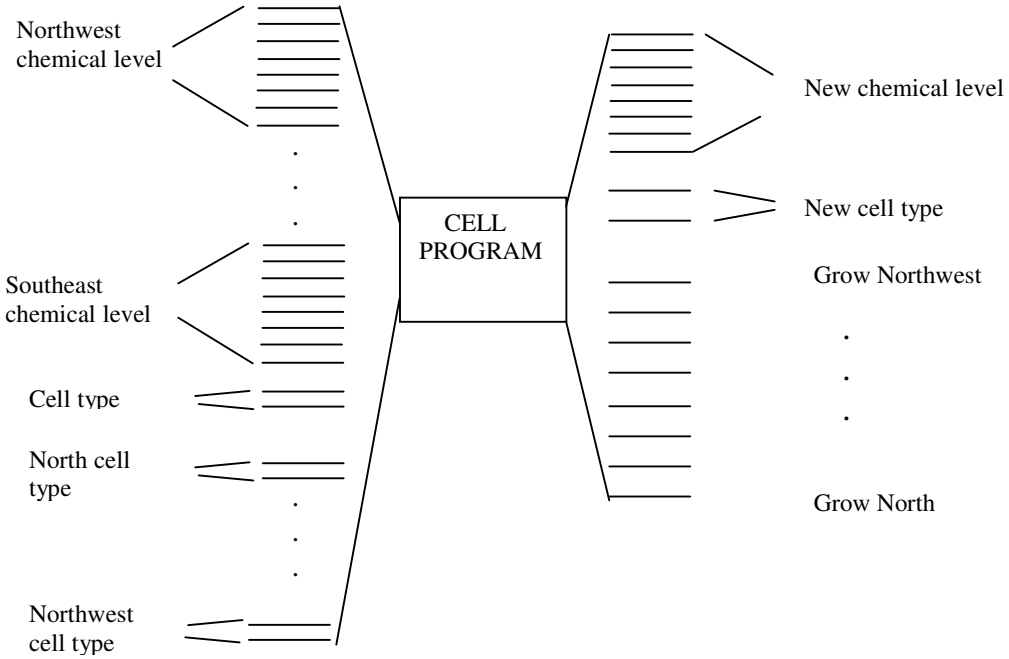


Figure 5.2: How data from the environment is read by the cell and results in change at the next time step

If the cell program dictates that the cell should grow but that its new type is 0 then the cell will still grow into its chosen locations, but it and its replicated offspring will then die. Also the chemical level dictated by the cell program only applies to locations where the cell grew. If the cell dies, the chemical level on that square will be the previous value.

Chemicals can diffuse according a number of user selected fixed diffusion rules. For all of the experiments reported in this chapter the diffusion rule is given below: Let *N* denote the neighborhood with neighbouring position *k,l*, the chemical at position *i,j* at the new time step is defined by the update rule:

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

In the absence of cells, this rules ensures conservation of the chemical. However cells have no "energy " constraints placed upon them and can pump out the maximum amount of chemical allowed (i.e. 255 if there are eight chemical bits). This diffusion rule is updated at the same time step (for the whole map) as the cells. Other diffusion rules have been implemented that allow chemicals to diffuse over a wider area but as yet we have no definitive results as to their efficacy. The way the chemical and cell maps are updated is as follows: The original maps are scanned cell by cell from the top left corner of the map. The updated cell growth, differentiation or death is recorded on the new map. Cells that are later in the scan sequence may overwrite an earlier cell. This introduces an unavoidable bias towards cells at the bottom right (since these are last to be scanned). The drawbacks of cell overwriting will be discussed later.

5.2 How a cell program is encoded into a genotype

In section 4, we discussed how one could represent a program or circuit in the form of a string of integers (the Cartesian genotype). In all the experiments reported here we used a genotype of 400 integers. Each group of four integers defines a type of 2 to 1 multiplexer and its three connections. A 2 to 1 multiplexer is a three input Boolean logic gate that expresses the following function:

$$f(A,B,C) = A.\overline{C} + B.C$$

where A, B, C are binary inputs and the dot represents the AND logic function, the + the inclusive OR function and the bar the NOT operation (inversion). We employed four types of multiplexers, corresponding to whether the inputs A and B were inverted. It is well known in digital electronics that any Boolean function can be constructed from multiplexers defined by *f* above (allowing the four types simultaneously increases the complexity of Boolean functions that can be built from a given number of gates).

5.3 Experimental parameters

The algorithm used to evolve the cell genotypes has been described in section 4. The particular experimental parameters used in the experiments reported in this chapter are the following (unless otherwise stated)

Experimental parameter	Value
Population size	5
Number of generations	30,000
Number of runs	10
Mutation rate	1%
Maximum number of multiplexers	200
Number of chemical bits	8
Number of cell types	4
Iterations at which fitness tested	variable

5.4 The tasks the cellular maps were supposed to accomplish

The French Flag model of Lewis Wolpert [Wolpert 1998] was the inspiration for the task the maps of cells were to achieve. In particular we looked at two problems. We wished to evolve a cell program that, over a given period of time, would construct a French flag. Moreover we wanted the flag to grow with time but always be recognisable as a French flag. For the second the problem we wanted to grow a French flag of a given size and have the flag remain that size subsequently. One can think of the French flag as an organism, a rectangular creature with three vertical stripes, blue, white and red. We were interested in whether it would prove possible to evolve the cells' program to construct such a creature. If such a creature were constructed, how would it behave when it was damaged, or particular cells were transplanted into other regions? In this toy world we reasoned what could conduct ' classic experiments' that have been demonstrated in the real world of biological development. In addition, by seeking to evolve programs that could be indefinitely iterated over time we wished to investigate methods of evolving programs that could grow structures without limit.

Flags or maps were defined as a simple grid of cells that we wished the initial seed cell to grow into. Figure 5.3 shows how two French flags were defined:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 1 1 1 3 3 3 2 2 2 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 1 1 1 1 3 3 3 3 2 2 2 2 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.3: The specification for some French flags (medium, large)

Initially a single cell was placed at a position in the map and the task was defined so that after a given (user defined) number of iterations of the cell program in every position where the cell was alive we required the collection of cells to be as close as possible to the configuration defined in Fig. 5.3. Initially we could also define an initial chemical map, typically we would set the chemical to the maximum allowed value at the same location as the initial cell. To try to direct the evolution of cell programs to the desired end we could state a list of maps together with the iterations at which they were expected. At each test point the organism was compared with the desired map. A simple numerical sum was calculated of all locations in the map where the grown map agreed with that required (including empty or dead cells in the map). The sum was calculated for all the desired maps. Later we discuss various possible shortcomings of this definition of a fitness of a particular cell program.

Section 5.5 Experiments and results

In the first experiment we required that the cells develop into the medium sized flag at iteration 7 (iteration 0 means the starting map) and the large flag at iteration 9. The maximum score was 512. The best result we obtained in the ten runs had a fitness of 491 and the average fitness of the 10 runs was 456. In Figure 5.4 we show the history of development of the best solution up until iteration 9.

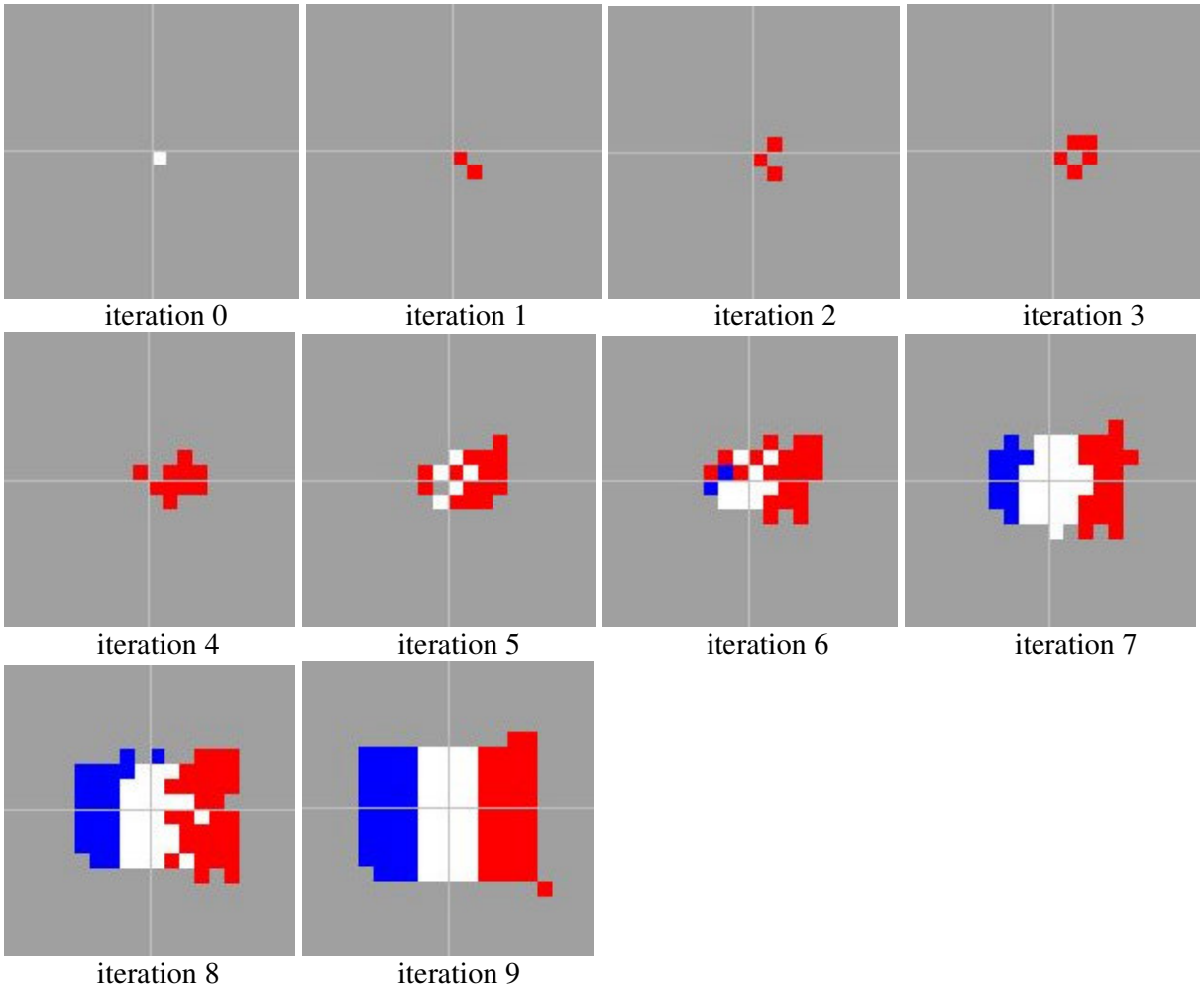


Figure 5.4: The developmental history of a cell program

It is curious how, even though the initial white cell is correctly placed with regard to the French flag, it nevertheless grows into two red cells. At iteration 5 white cells appear and blue cells make a first appearance at iteration 6, only one iteration away from the first test point! The corresponding developing chemical map is shown in Figure 5.5. At early iterations there appears to

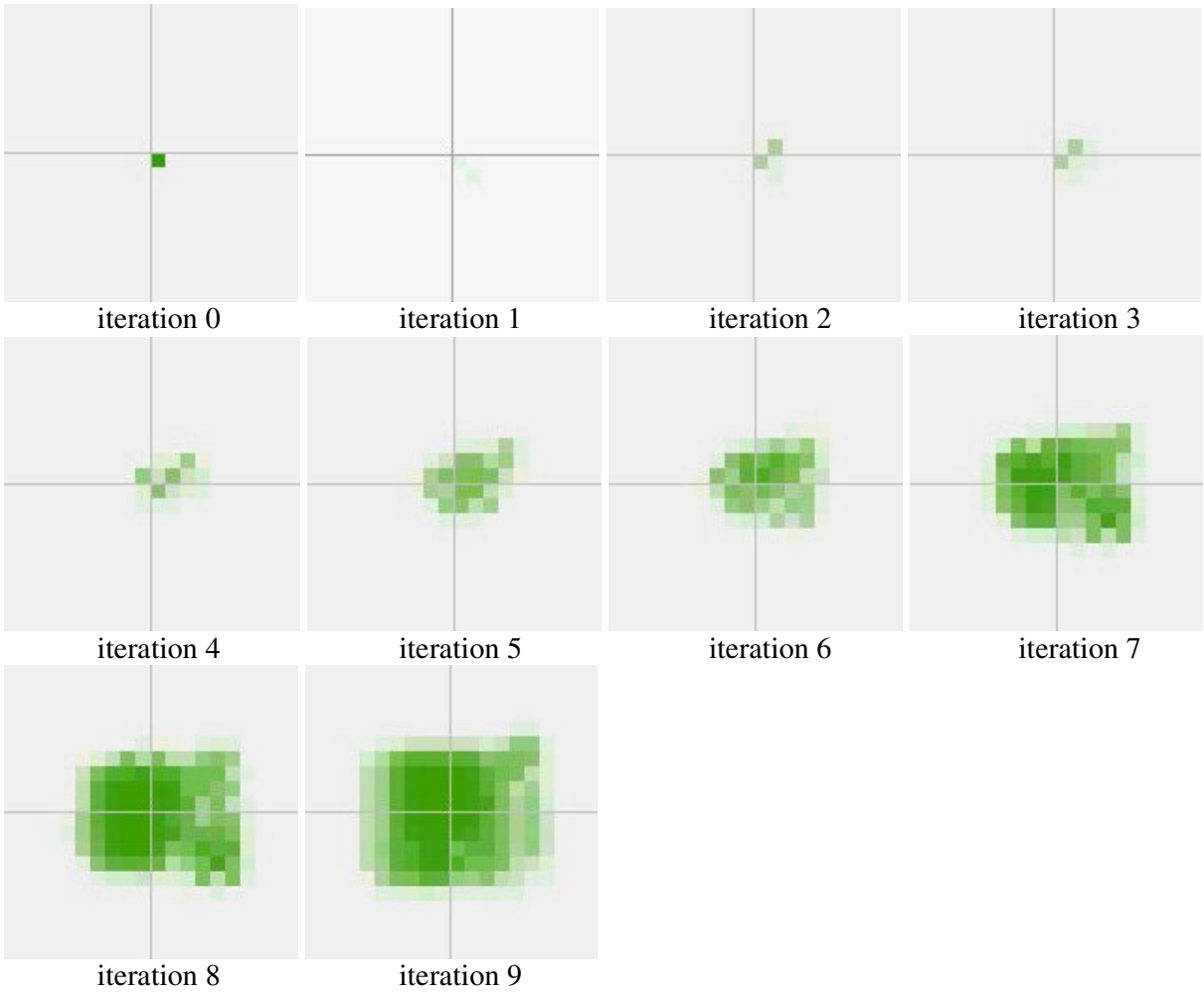


Figure 5.5: The chemical history of the growing cellular map

be virtually no chemical present, however levels are so low that they are barely perceptible in the grey background.

It should be stressed that during the evolution of the cell genotype the assessment of the cells phenotype (the cellular map) was only every made at iterations 7 and 9, so that different evolved genotypes are likely to display very different growth behaviours. In our next investigation we examined the longer-term behaviour of the growing cellular map. What would the embryo look like beyond nine iterations? Would the map become so distorted that it was no longer recognisable as a French flag? In Figure 5.6 we show how the cellular map develops over a further 11 iterations.

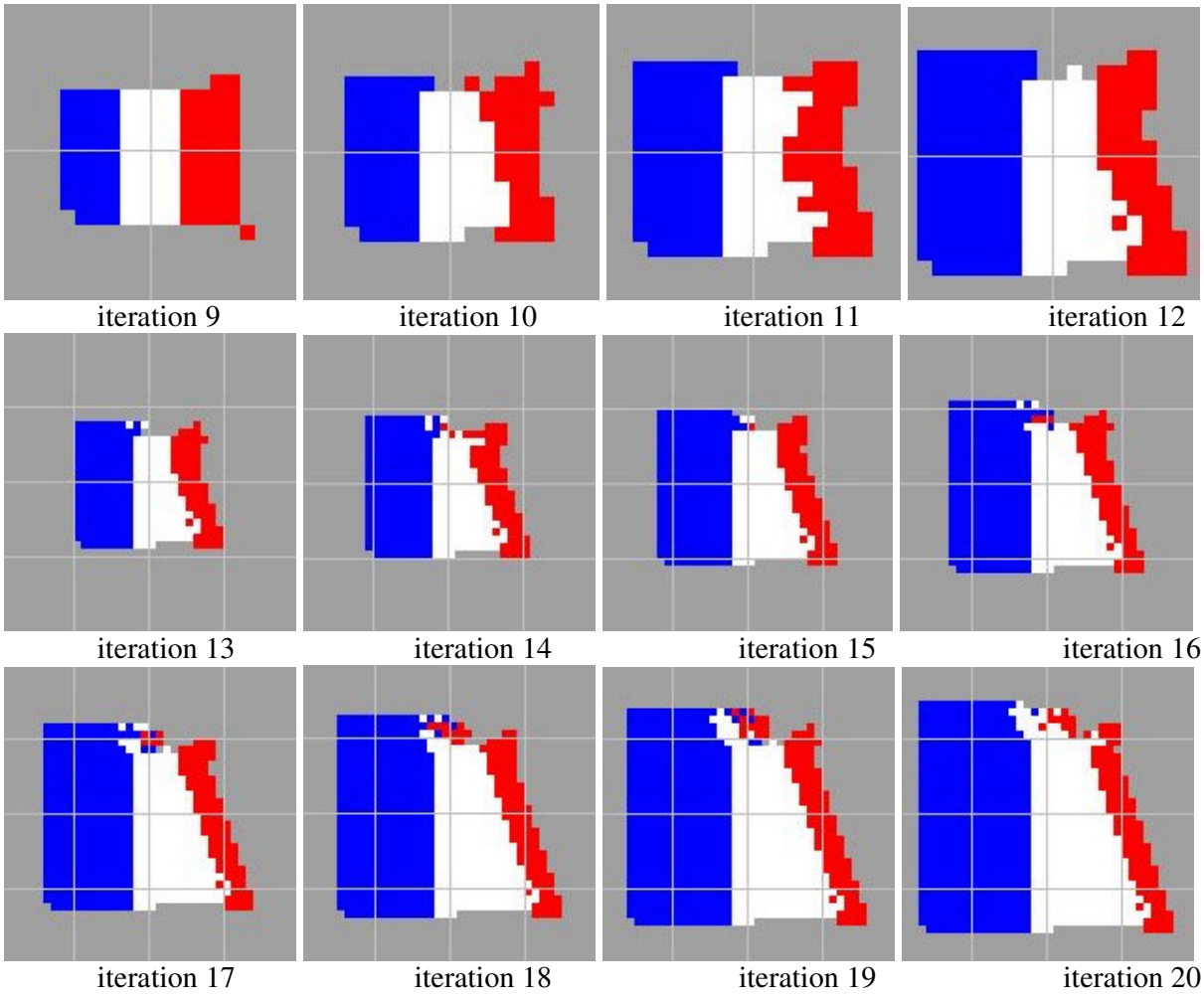
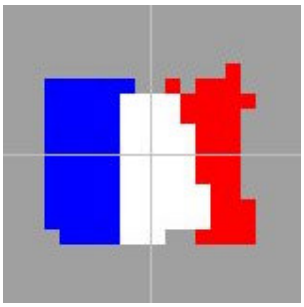
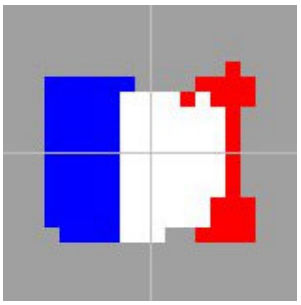


Figure 5.6: The developmental history of the cellular map beyond the last fitness testing point at iteration 9.

One of the striking observations about the maps after iteration 9 is the neat regularity of the blue section of the flag. Notice the single missing blue cell in the bottom left corner is present in all the maps, also that three sides of the blue region are straight as desired. At iteration 13 two white cells appear at the top right of the blue section. What has caused this behaviour? The cellular neighbourhoods in this region are fairly constant so it may be some underlying change in the chemical signature of the region. The diagonal red region may be caused by the asymmetry of the cell update rule, this bears further investigation. The chemical milieu is an important factor in the behaviour of the cells. This is easy to show by comparing the cellular map at iteration 10 when all the chemicals are removed from the chemical map at iteration 9 (Figure 5.8)



iteration 10 (chemical map untouched)



iteration 10 (chemical map altered)

Figure 5.8: Differences caused in growing cell map due to imposed chemical changes

The next series of experiments revealed some fascinating parallels with real developmental biology. If we remove some parts of the embryonic French flag organism, what will happen? We removed the white and red cells from the cell map at iteration 9 and then proceeded to continue to grow the map (leaving the chemical map at iteration 9 unchanged). Since the blue region of cells are growing in such a regular way it seemed natural to expect that we would obtain a growing blue region of cells. However the results shown below (Fig. 5.9) quickly revealed a much more interesting behaviour.

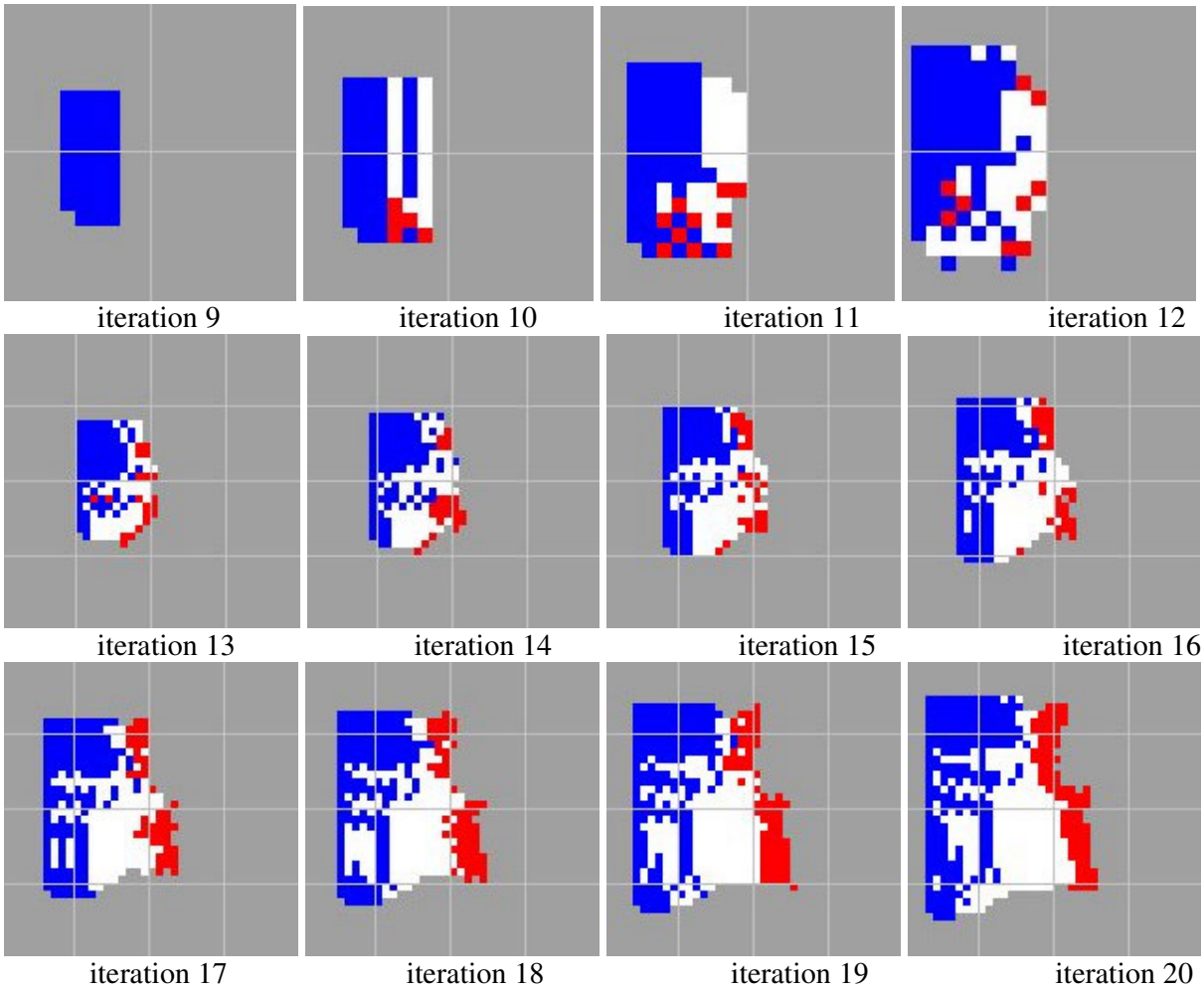


Figure 5.9: Development of corrupted French flag grown from blue cell region of original

The blue region tries to grow into the French flag again! Though it cannot quite manage it. The chemical map used for the blue region was the same as the chemical map that had been established for the whole complete map. It is possible that different behaviour may have occurred had we used an initial chemical map that was only local to the blue region. It is clear that the computer experiments have some parallels to the experimental problems that occur when developmental processes are altered in that alterations to the chemical conditions in altered embryos may lead to developmental effects (rather than solely being due to the alteration of the embryo alone).

In the first of two further experiments we cut a large hole in the centre region of the flag (at iteration 9) and iterated to see what would happen (Fig. 5.10) and also, we cut the flag diagonally (Fig 5.11).

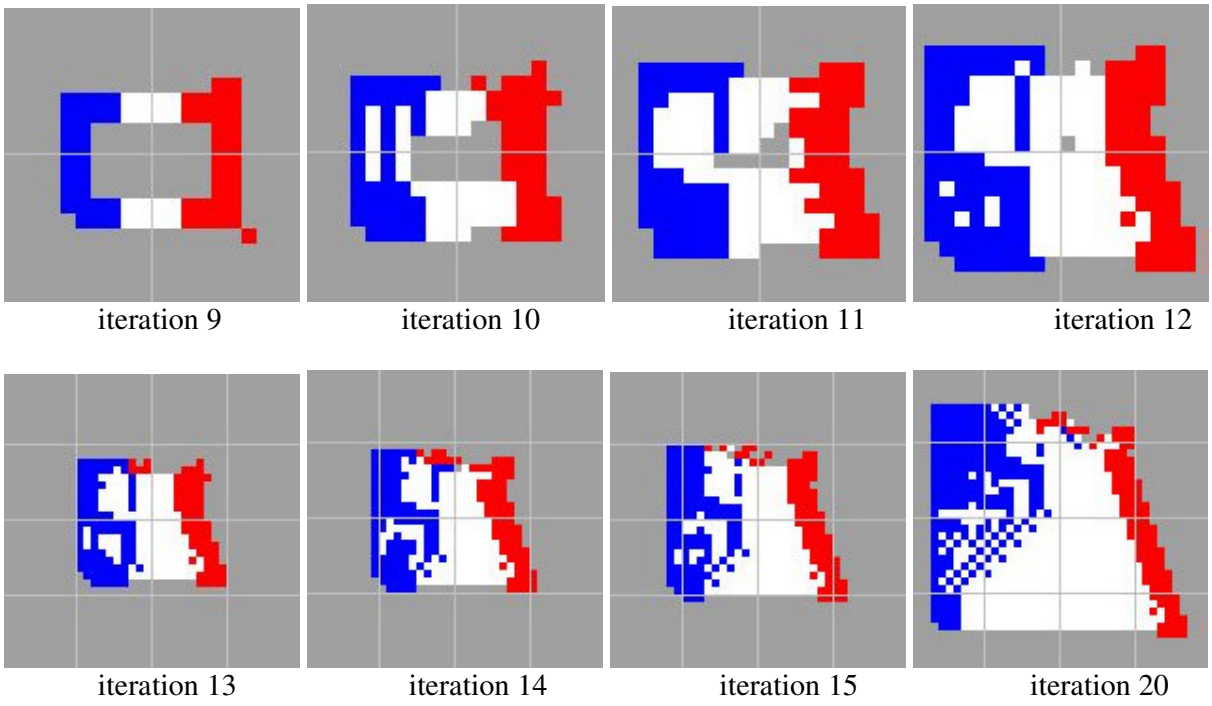


Figure 5.10: Development of cells when a rectangular section is removed from the embryonic map

Once again we see that the cells gradually rebuild the map. Interestingly the red region of the map at iteration 15 is identical to the red region on the original uncorrupted map at iteration 15. It seems that the section of red cells removed was of little importance.

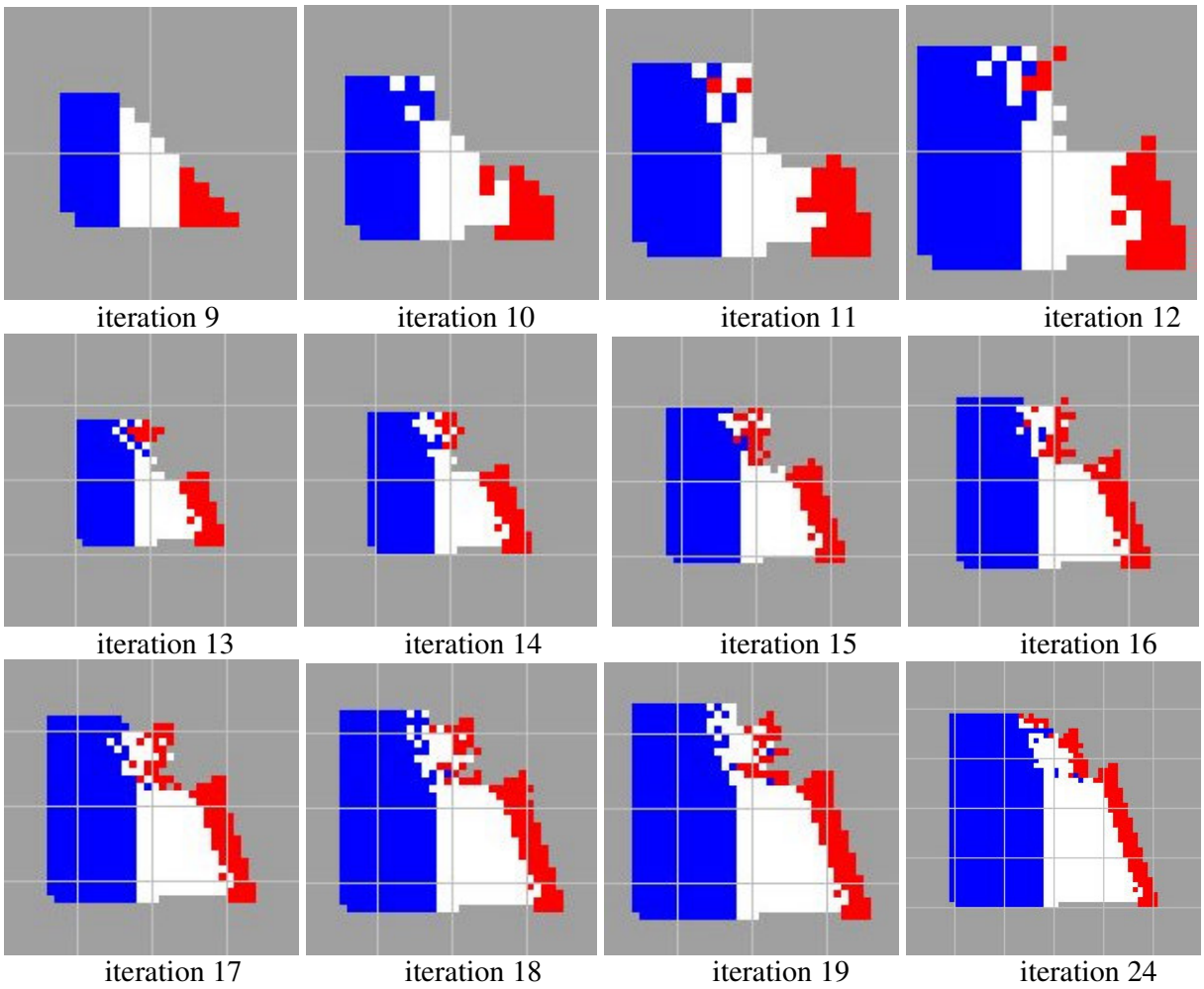


Figure 5.11: Development of cells when a diagonal section is removed from the embryonic map

After embryonic cells are removed (iteration 9) we see that the blue region remains untouched, once again we see the familiar growing rectangular blue shape (with the bottom left corner missing). Red cells begin to appear at the top right section of the map, while the remaining red region at the bottom right begins to grow upwards until by iteration 24, the regions are almost joined. The white cells initially invade the growing blue region but eventually (iteration 24) appear to retreat. As time progresses the embryonic array starts to recover completely from the initial damage and takes on an appearance similar to the cell maps of the undamaged embryo (Fig. 5.6).

In all the experiments described so far we have seen continued growth of the embryonic cellular map. Of course, in biological development growth ultimately ceases. The next experiment attempts to create an embryonic cell that matures to a French flag appearance of a given size and then subsequently remains at that size irrespective of the number of further iterations of the cells' program. It is very hard to see how such a task could be accomplished without chemical signalling, as there are two phases: growth, followed by stasis. Something would have to change in the environment to make this possible. It was hoped that the cells might be able to use some characteristics of the chemical map to make this possible.

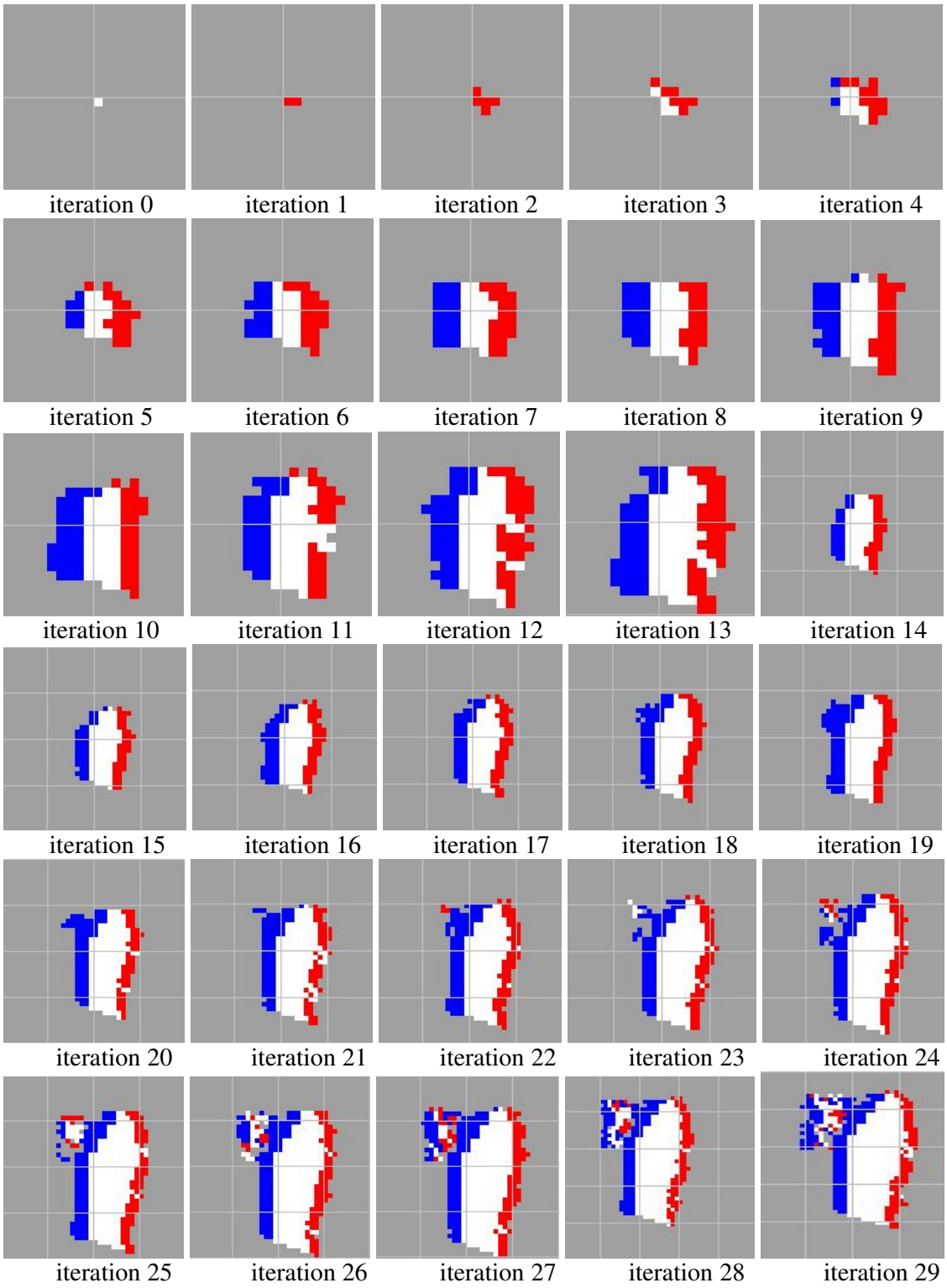


Figure 5.12: Development of cells required to grow to a fixed size map (at iterations 5,6,7,8) and then iterated over a much longer period.

The results shown in Fig 5.12 are very encouraging. Firstly, although the growth of the embryonic array of cells does not halt, it is considerably slowed (this is evident when one compares iteration 20

with the same iteration in Fig. 5.6). Additionally the appearance of the cellular map is much closer to the French flag, in that the coloured bands are more regular and are approximately vertical. The longer-term behaviour of the cellular map is more complex. Fig. 5.13 shows the situation at iteration 46.

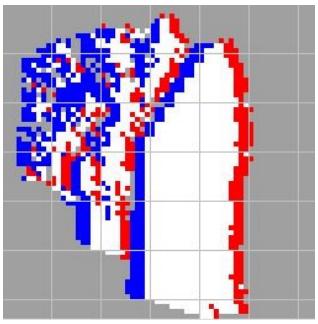


Figure 5.13: Cellular map at iteration 46 of developing cell program shown in Fig. 5.12

In the next experiment the same cell program was run but with all chemical levels set to zero. In this way we could investigate the sensitivity of the cell program to the chemical environment during growth. We could also investigate the importance of the chemical to slowing down growth after maturity. The results are shown in Fig. 5.14.

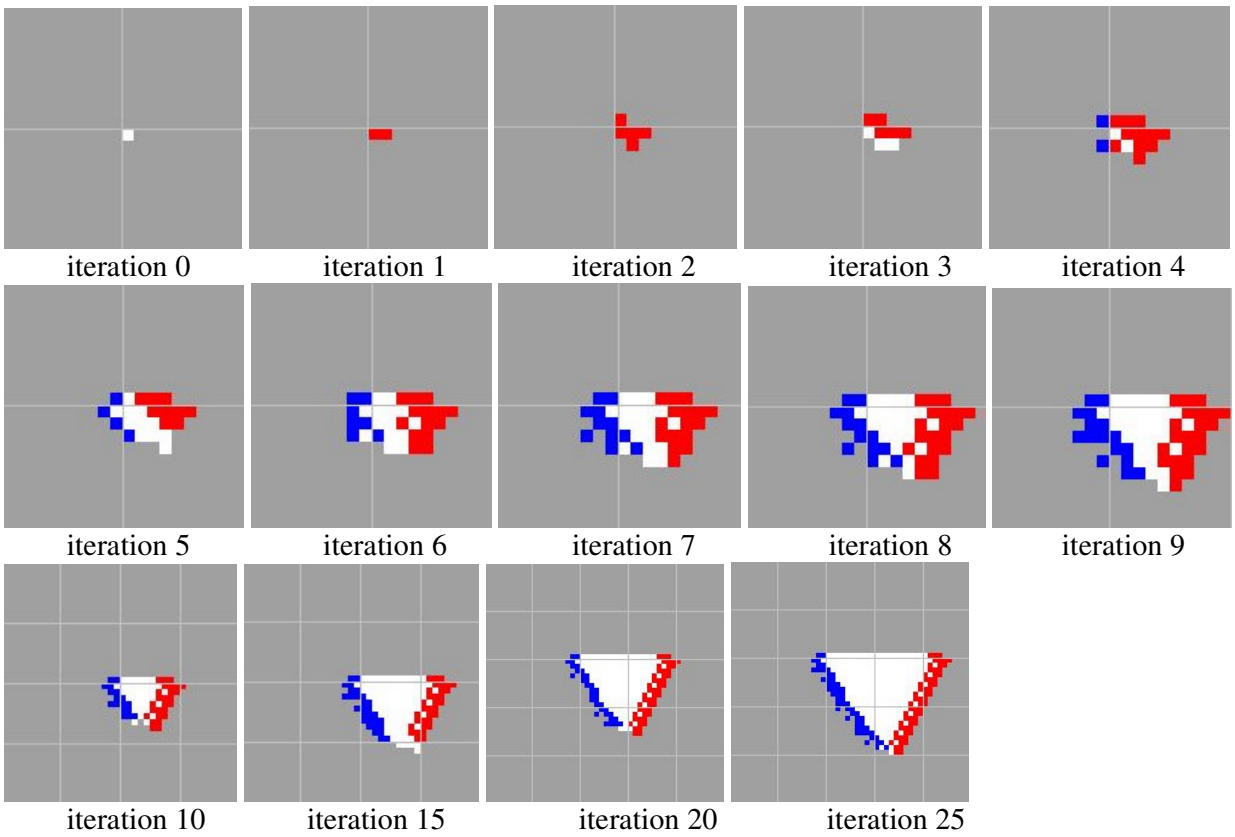


Figure 5.14: Development of cells grown from the program in Fig. 5.12 but without any chemical environment

At first the embryonic cell map develops in the same way (until iteration 2), but then gradually departs from the same behaviour as its chemical-rich counterpart, eventually assuming a triangular shape. Note that the cell mass appears to be growing at approximately the same rate. It is not possible to conclude from these data that chemical sensitivity of cells is essential to prevent growth. However, further experiments revealed that the presence of chemicals is an important factor in obtaining a cell program with a high fitness. Two sets of ten runs of the evolutionary algorithm were performed under identical conditions except that in one there were no chemicals. The runs with chemicals had an average fitness of 438 (standard deviation = 26.04), while the runs without chemicals had an average fitness of 400 (standard deviation 14.06).

In the final experiment we wanted to see whether we could obtain a regular series of spots. We chose to require that the embryonic array of cells assume the pattern below (with perfect fitness 196) after the fifth iteration (the initial being 0)

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 3 3 3 3 3 3 3 3 3 0 0
0 0 3 3 3 3 3 3 3 3 3 3 0 0
0 0 3 3 1 1 3 3 1 1 3 3 0 0
0 0 3 3 1 1 3 3 1 1 3 3 0 0
0 0 3 3 3 3 3 3 3 3 3 3 0 0
0 0 3 3 3 3 3 3 3 3 3 3 0 0
0 0 3 3 1 1 3 3 1 1 3 3 0 0
0 0 3 3 1 1 3 3 1 1 3 3 0 0
0 0 3 3 3 3 3 3 3 3 3 3 0 0
0 0 3 3 3 3 3 3 3 3 3 3 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.15: Required ' spots' pattern using white cells and blue spot cells

We still allowed red cells. The experimental parameters were as before (section 5.3). We obtained three runs out of ten that achieved the perfect score. The three runs produce very different solutions to the problem! The runs are shown in Figs 5.16 and 5.17.

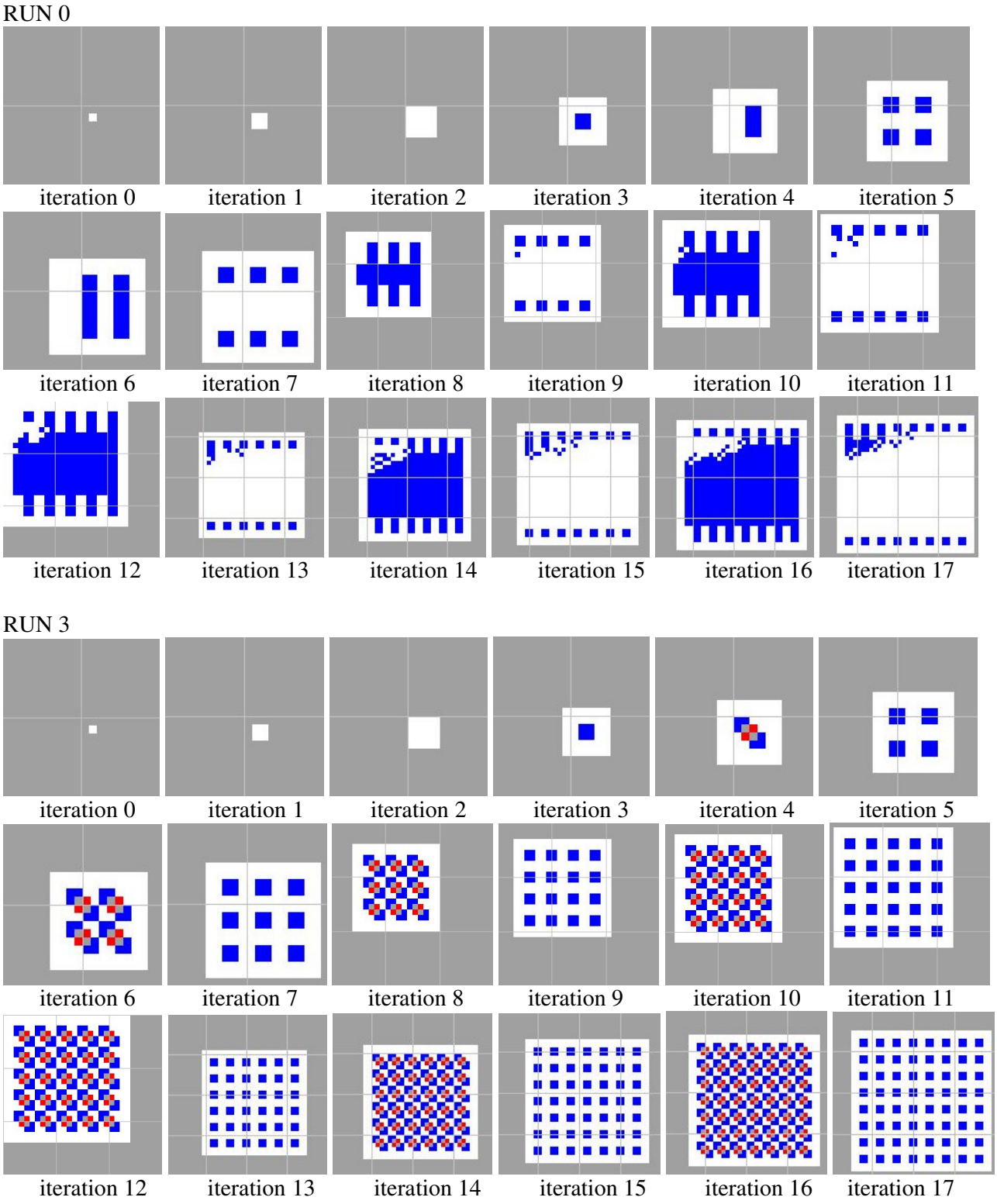


Figure 5.16: Development of cells that solve perfectly the required task of 4 blue spots at iteration 5.

RUN 7

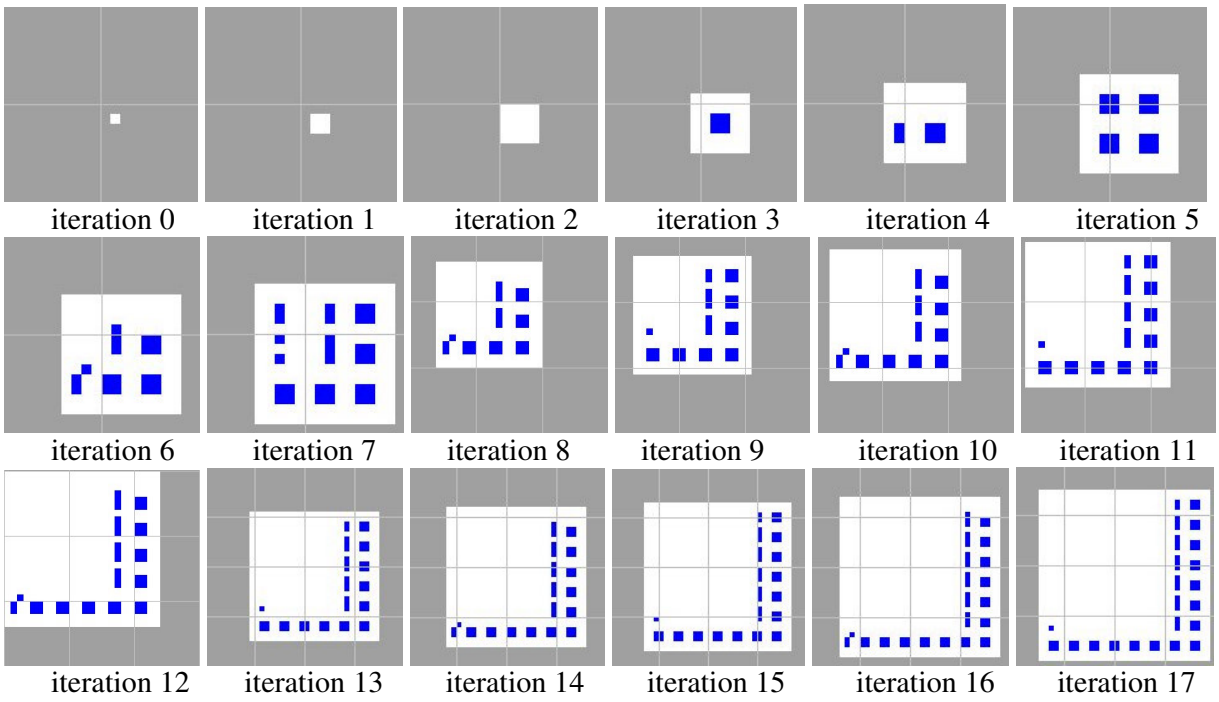


Figure 5.17: Third run that solves perfectly the required task of 4 blue spots at iteration 5.

In Figs. 5.16 and 5.17 we see an extraordinary variety of ways of accomplishing the given task. The embryological development is identical for the first three iterations. They all show an alternating pattern type on odd versus even iterations. The cellular maps corresponding to run 3 are quite extraordinary in their beauty and consistency. At even iterations there are repeating units of the pattern seen in iteration 4. A magnified section in the centre is shown in Fig. 5.18.

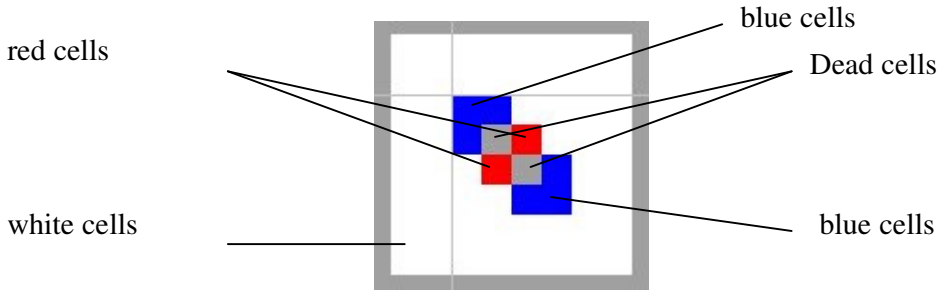


Figure 5.18: Magnified section of cellular map for run 3 iteration 4 .

How important are the chemical emissions to this precise process? We investigated the cellular program for run 3 without any allowed chemicals (i.e. the program was identical, however all chemical outputs produced by the cells were set to zero). The results were dramatically different (Fig 5.19).

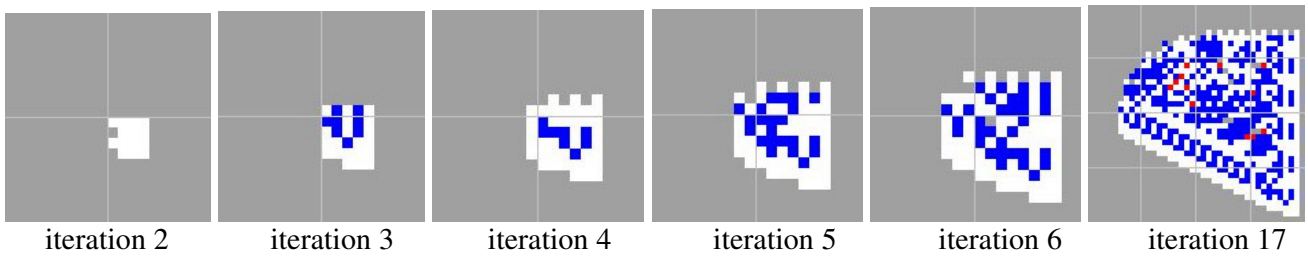


Figure 5.19: Cell program of run 3 with no chemical environment

This is an amazing result. The cellular map for run 3 is extremely precise and regular yet it depends on a chemical map with its inevitable fluctuations from iteration to iteration. Figure 5.20 shows the chemical maps corresponding to the cellular maps for run 3 (Fig. 5.16). It is important to note there are small, apparently random fluctuations of the chemical level that are not visible in the figures, it appears that the extremely precise cell behaviour is insensitive to this minor variation.

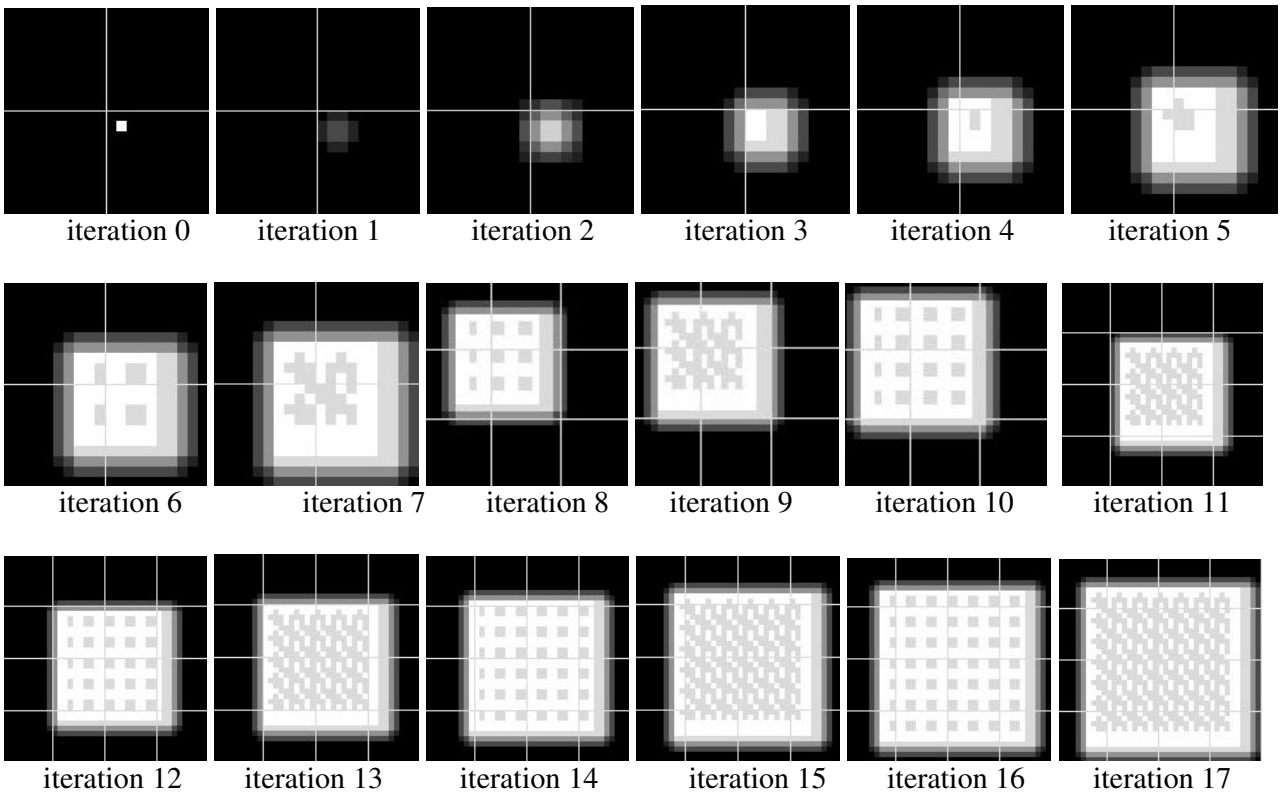


Figure 5.20: Chemical maps corresponding to the cellular maps for run 3 (Fig. 5.16) - black indicates an absence of chemical. The almost white regions correspond to a chemical level of 228 and the light grey regions to a chemical level of 221.

A magnified picture of the cellular and corresponding chemical map at iteration 17 are shown below:

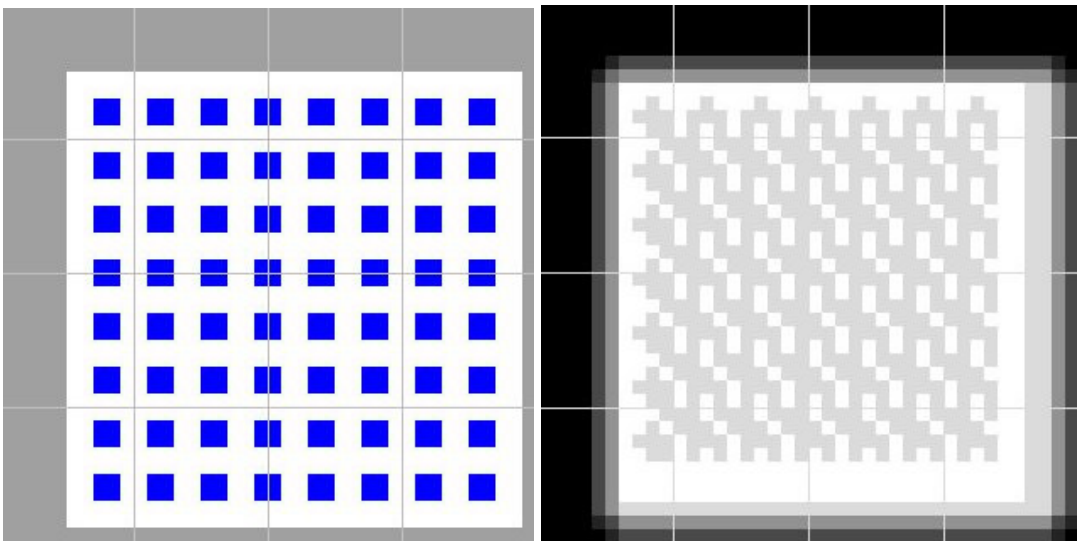


Figure 5.21: Cellular and corresponding chemical map at iteration 17.

When the cell is evolved with no chemicals at the outset it achieves an average fitness of 181.7 and the best solution reached has a fitness of 192 (out of 196), while under the original scenario the average fitness is 190.6 (with the three perfect solutions described).

5.6 Analysis of the genotype

One potential advantage of the techniques presented in this chapter is that the genotype is explicitly inspectable (i.e. a system of rules). The genotypes have had 200 nodes (strings of 400 integers). Typically only half of the nodes are active. At present the programs that produce the interesting cell growth and behaviour discussed have not been analysed. We hope that we might learn a lot from such an analysis. How can a program produce the elegant behaviour seen in Fig. 5.16? We hope to report our findings in due course.

5.7 Discussion

There are many aspects of this work that warrant further investigation. Firstly the basic technique described could possibly be improved. It is likely that the cell update procedure being used may be making it more difficult to evolve a cell program that exhibits the desired behaviour. At present the cell map is scanned in a fixed order row by row starting at the top left position and finishing in the bottom right position. Cells may overwrite each other. This is illustrated below:

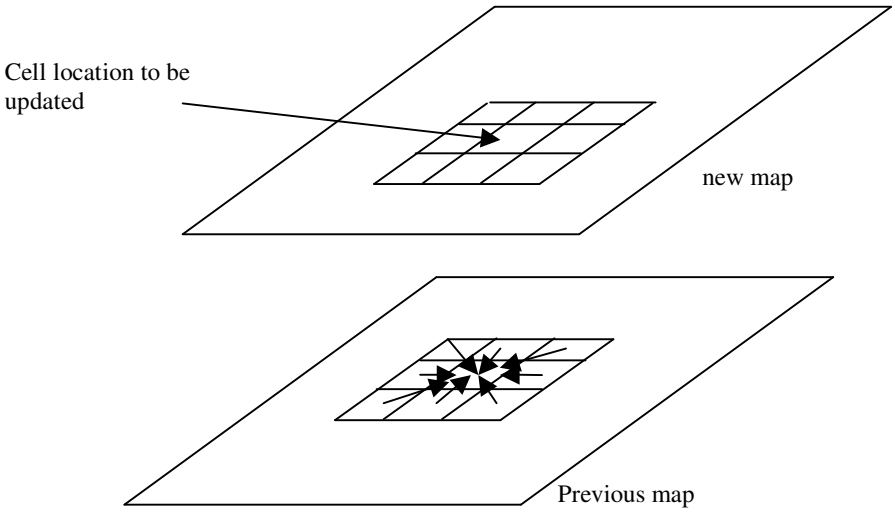


Figure 5.22: The cell update problem

In the real world cells, of course, occupy space and cannot be located at the same place, or overwrite each other. Therefore, one idea would be to include, within the cell's own genotype, an adjudication function that decided how the cell would grow into the new location depending on the demands of the other cells that could possibly grow into the same location. Ideally, one would implement some sort of cell shuffling procedure so that overwriting was not possible, however this would be difficult and time consuming in practice.

In real biological development there are stem cells that gradually differentiate into more specialised types of cells. In the simulations reported here all cells are stem cells. One could, of course, create restrictions so that cells could only replicate cells of the same type and introduce a special stem cell type that could replicate and then specialise. The idea of allowing simulated cells to gradually differentiate would be difficult without dictating exactly how they should do so. One of the motivations for the work reported here is to give evolution as much freedom as possible.

Another line for future work will be to allow a number of different chemicals, perhaps emitted by particular cells (i.e. blue, red and white chemicals). One could perhaps observe the emergence of morphogens.

One of the real difficulties of the approach described here is the imposition of the fitness function. It would be nice to see if a fitness function could be defined that tested the growing cellular map in an orientation independent way. This would be likely to speed up the rate at which complex maps could be evolved. Further work is already underway on adapting the methods described here to the problem of growing electronic circuits (in simulation). In this scenario one would be testing the *function* of a particular map, this would be less dependent on orientation.

We saw how these embryonic arrays could repair themselves after damage. This observation suggests a further investigation of such behavior. One could impose damage sporadically and try to evolve cellular programs that naturally repaired themselves. Currently, in a field known as Evolvable Hardware there is strong interest in self-repairing circuits.

In the work described in section 5 we have used digital multiplexers as our basic computational units. More complex functions could be employed (such as digital adders, comparators, multipliers), this would probably increase the speed of evolution and the complexity of resulting cellular maps.

In preliminary work we included cell movement as part of the cell program output, but this deserves further investigation. Chemotaxis could be evolved and investigated by placing a chemical source at a point in the map and defining the cells' task as one of moving (and/or growing) toward the source.

There is a bias toward growth in the representation currently used, as there are eight growth bits, all of which must be zero for zero growth to occur.

In the experiments reported here we have begun with a single cell, however it would also be interesting to investigate multiple seed cells.

8 Concluding Remarks

Most researchers in computer science who are using developmental processes are trying to define genotypes which they can evolve that lead to solving enormously complex problem. At present there is still only tentative evidence that direct encodings are less effective than developmental ones. Part of the problem is that researchers have still not reached a consensus about what are complex problems. Almost by definition one is imagining problems whose smallest reasonable solution is of enormous size, these problems are inevitably time consuming and so are relatively unexplored. There is a great need within the community to define clearer milestones for achievement. The new researcher is faced with a difficult question at the outset: Should I try to create as realistic simulation of real biology as possible or should I try to build the *simplest* system that can generate complexity? How can I define complexity? It is our conviction that if we are to make any contribution to thinking about biological development we need to look for the simplest and most general principles. We should be trying to define much more clearly how a form of simulated development that is evolvable can contribute to computer science. It is clear that real biological development is making use of enormous physical complexity. Some researchers are trying to enrich their developmental systems so that novel exploitable features may emerge [Bentley 2002], indeed some are thinking even further and trying to use a computer to control and evolve developmental processes that are embedded in the physical world.

8 References

Astor J. C., and Adami C. (2000), "A Development Model for the Evolution of Artificial Neural Networks", Artificial Life, Vol. 6, pp. 189-218.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998), Genetic Programming. An Introduction, Morgan Kaufmann.

Bentley P., and Kumar S. (1999), "Three ways to grow designs: A comparison of embryogenies for an Evolutionary Design Problem", in Banzhaf W. et al. (eds.), Proceedings of the Congress on Evolutionary Computation, IEEE Press, pp. 35-43.

Bentley P. (2002), "Evolving Fractal Proteins", Late breaking papers, Genetic and Evolutionary Computation Conference, New York.

Boers, E. J. W., and Kuiper, H. (1992), "Biological metaphors and the design of modular neural networks", Masters thesis, Department of Computer Science and Department of Experimental and Theoretical Psychology, Leiden University.

Bongard J. C. and Pfeifer R. (2001), "Repeated Structure and Dissociation of Genotypic and Phenotypic Complexity in Artificial Ontogeny", in Spector L. et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, Morgan-Kaufmann, pp. 829-836.

Bonner, J. T. (2000), First Signals. The Evolution of Multicellular Development. Princeton University Press.

Cangelosi, A. (1999), "", in Banzhaf W. et al., Proceedings of 1st Conference on Genetic and Evolutionary Computation, GECCO-99, Morgan Kaufmann, San Francisco, pp.

Cangelosi, A., Parisi, D., and Nolfi, S. (1993), "Cell Division and Migration in a ' Genotype' for Neural Networks", Technical report PCIA-93, Institute of Psychology, CNR, Rome.

Dellaert, F. (1995), "Toward a Biologically Defensible Model of Development", Masters thesis, Department of Computer Engineering and Science, Case Western Reserve University.

Eggenberger, P. (1997), "Evolving morphologies of simulated 3D organisms based on differential gene expression", in Husbands P. and Harvey I. (eds.) Proceedings of the 4th European Conference on Artificial Life, ECAL 1997, Springer, Heidelberg, pp. 205-213.

Eggenberger, P. and Dravid R. (1999), "An evolutionary approach to pattern formation mechanisms on lepidopteran wings", In Proceedings of the 1999 Congress on Evolutionary Computation, Volume 1, IEEE Press, pp. 470-473.

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, Addison-Wesley, pp. 389-416.

Furusawa C., and Kaneko, K. (1998), "Emergence of Multicellular Organisms with Dynamic Differentiation and Spatial Pattern", in Adami C. et al. (eds.) Proceedings of the 6th International Conference on Artificial Life, MIT Press.

Gruau, F. (1994), "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm", PhD thesis, Ecole Normale Supérieure de Lyon.

Gruau, F., Whitley, D., and Pyeatt, L. (1996) "A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks", in Proceedings of the 1st Annual Conference on Genetic Programming, Stanford.

Gould, S.J. (1977), "Phylogeny and Ontogeny", Harvard University Press, Cambridge, MA.

Hoile, C. and Tateson, R. (2000), "Design by morphogenesis", in Bedau M., McCaskill J., Packard N. and Rasmussen S. (eds.), Proceedings of the 7th international Conference on Artificial Life, MIT Pres, pp. 141-145.

Hornby G. S., and Pollack J. B. (2001), "The Advantages of Generative Grammatical Encodings for Physical Design", in Proceedings of the Congress on Evolutionary Computation, IEEE Press, pp. 600-607.

Jacobi, N. (1995), "Harnessing Morphogenesis", Cognitive Science Research Paper 423, COGS, University of Sussex.

Kaneko, K. and Yomo, T. (2000), "Sympatric speciation from interaction -induced phenotype differentiation", in Bedau M., McCaskill J., Packard N. and Rasmussen S. (eds.), Proceedings of the 7th international Conference on Artificial Life, MIT Pres, pp. 113-121.

Kessin, R. H. (2001), Dictyostelium. Evolution, Cell Biology, and the Development of Multicellularity, Cambridge University Press.

Kitano, H. (1990), "Designing neural networks using genetic algorithms with graph generation system", Complex Systems, Vol. 4, pp. 461-476.

Kitano, H. (1995), "A Simple model of Neurogenesis and Cell Differentiation Based on Evolutionary Large-Scale Chaos", Artificial Life, Vol. 2, pp. 79-99.

Kodjabachian, J. and Meyer, J-A. (1998), "Evolution and Development of Neural Controllers for Locomotion, Gradient-Following and Obstacle-Avoidance in Artificial Insects", IEEE Transactions on Neural Networks, Vol. 9, pp. 796-812.

Koza, J. R. (1992) Genetic Programming: On the programming of computers by means of natural selection. MIT Press.

Koza, J. R. (1994) Genetic Programming II: Automatic Discovery of Reusable Subprograms. MIT Press.

Koza, J., Bennett III, F. H., Andre, D., and Keane, M. A. (1999). Genetic Programming III. Darwinian Invention and Problem Solving. Morgan Kaufmann.
Langdon, W. B. (1999) "Scaling of Program Fitness Spaces", Evolutionary Computation, Vol. 7, pp. 399-428.

Lindenmeyer, A. (1968), "Mathematical models for cellular interaction in development, parts I and II", Journal of Theoretical Biology, Vol. 18, pp. 280-315.

Meinhardt, H. (1998), "The Algorithmic Beauty of Sea Shells", Springer, Heidelberg

Miller, J. F. and Thomson, P. (2000), "Cartesian genetic programming", in *Proceedings of the Third European Conference on Genetic Programming. LNCS*, Vol. 1802, pp.121-132.

Murray, J. D. (1989), Mathematical biology, Springer, Heidelberg.

Nolfi, S., and Parisi, D. (1991) "Growing neural networks", Technical report PCIA-91-15, Institute of Psychology, CNR, Rome.

Othmer, H. G., Maini, P. K., and Murray J. D. (1993), Experimental and Theoretical Advances in Biological Pattern Formation, Plenum Press.

Prunsinkiewicz P., and Lindenmeyer A. (1990), The Algorithmic Beauty of Plants, Springer-Verlag.

Savill, N. J., and Hogeweg P. (1997), "Modelling Morphogenesis: From Single Cells to Crawling Slugs", Journal of Theoretical Biology, Vol. 184, pp. 229-235.

Siddiqi, A. A., and Lucas S. M. (1998), "A comparison of matrix rewriting versus direct encoding for evolving neural networks", in Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 392-397.

Turing, A (1952), "The chemical basis of morphogenesis", Philosophical Transactions of the Royal Society of London B, Vol. 237, pp. 37-72.

Vassilev V., and Miller J. F. (2000), "The Advantages of Landscape Neutrality in Digital Circuit Evolution", Third International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, Vol. 1801, Springer-Verlag, pp. 252-263

Wolpert, L. (1981), "Positional information and pattern formation", Philosophical Transactions of the Royal Society of London B, Vol. 295, pp. 441-450.

Wolpert, L. (1998), "Principles of Development", Oxford University Press.

Yu, T. and Miller, J. (2001), "Neutrality and the evolvability of Boolean function landscape", in *Proceedings of the Fourth European Conference on Genetic Programming*, Springer-Verlag, pp. 204-217.

Yu, T. and Miller, J. (2002), "Finding needles in haystacks is not hard with neutrality", in *Proceedings of the Fifth European Conference on Genetic Programming*, Springer-Verlag, pp. 13-25.