Exploring Models of Development for Evolutionary Circuit Design

Timothy G.W. Gordon
Department of Computer Science
University College London
London WC1E 2BT
t.gordon@cs.ucl.ac.uk

Abstract- Traditional evolutionary circuit design does not scale well to large, complex problems. Nature solves the scalability problem by using a complex mapping implicit in the process of biological development. By modelling this process we aim to improve scalability in evolutionary circuit design. Here we extend our earlier work [1] by demonstrating that evolution can learn and encode useful circuit design abstractions in a developmental process. We go on to present enhanced models of development with improved intercellular communication and show how this improves their ability to generate circuits.

1 Introduction

In the past few years, interest in using evolutionary algorithms to automatically design circuits has increased dramatically [2,3]. However to date only small circuit design problems have been tackled successfully. The idea of aiding scalability by introducing a model of biological development is becoming increasingly popular [4,5].

Our earlier studies of developmental systems have shown they can allow iterative structures to be evolved that may be useful for circuit design [1]. In this paper we present new results that demonstrate how this kind of developmental system is able to encapsulate useful design abstractions and represent them succinctly. In [1] we were not able to evolve perfect solutions to our test problems. This paper goes on to explore why the system we used could not solve the problems and presents a new model of development that addresses these issues.

The rest of the paper is structured as follows: section two explains how development can aid scalability, and assesses previous models of development that have been used for circuit design. Section three reviews the exploratory developmental system presented in [1] and presents further analysis of the circuits evolved with the system. This analysis reveals design abstractions currently used by human circuit designers have been evolved, and can be represented in a concise form in the developmental rules. Section four describes changes made to the exploratory system to increase the computational power of the developmental model, and presents systems incorporating these improvements, along with results and analysis. Conclusions are given in section five.

2 Evolving Large Circuits using Development

A common approach to simplifying large evolutionary problems has been to reduce the search to a smaller space by using more complex primitives, an example being function level evolution [6]. Evolution is forced to use these primitives, and more evolvable or innovative primitives are unattainable. If evolution designed its own primitives, and the bias towards using them was soft, evolution may be free to search more interesting space.

2.1 How Developmental Systems Can Aid Scalability

Biological genomes use a complex process of regulated gene expression to map from genotype to phenotype. This process allows the formation of autocatalytic gene regulatory networks (GRNs) [7]. Such networks may be arranged as modules, controlled by a master control gene. When activated the master control gene causes a cascade of activity throughout a GRN module, and generates a complex feature in a phenotype. If the master control gene that activates the GRN is expressed multiple times, the module can be reused, thereby generating complex features that iterate throughout an organism [8].

Not only can development encode reusable modules, but as the process of generating and using the modules is under genetic control itself, evolution is not forced to make use of them. If we mimicked this process, applied to the construction of a circuit rather than a biological organism, we would have the ability to learn problem-specific circuit design abstractions in the form of modules generated by GRNs, with only a soft bias towards using them. Thus we could avoid the problems of approaches like function level evolution.

However we must decide on the features of biological development we need to model so the biases we impose through development capture useful design abstractions, for problems of sufficient complexity as to be of interest.

2.2 Previous Models of Development in Circuit Design

Generative genotype-phenotype mappings have been used in evolutionary circuit design for some years, often using genetic programming to evolve instructions to design circuits [9]. These approaches, which are dubbed ‘explicit’ [10], have had success in generating moderately complex structures, but the absence in the literature very large circuits evolved using explicit mappings suggests that a generative mapping is not the only key feature of development’s ability to scale to large problems.

Another category of developmental mappings that models the biological process more carefully is the ‘implicit’ approach, which may offer benefits of lower computational overhead and increased evolvability over explicit methods [1]. The idea behind these is that complex objects can be generated by successively rewriting a symbolic description of a simple object according to a set of rewriting rules. The map between genotype and

phenotype is specified by a fixed start symbol for the rule rewriting process, and the grammar is evolved. Hence the fixed program is implicitly evolved though alteration of its grammar. One class of implicit system, the D0L-System has been explored in the context of circuit design [4]. The D0L-System is again highly abstracted, and does not encapsulate a key concept we are interested in - context. Context sensitive grammars are more expressive than context-free grammars [11], so including context allows for a more expressive system. Additionally without context the ultimate fate of each symbol at any stage of development is predetermined. If the organism was to be damaged during development, it cannot make use of the unusual context to take action to repair the damage. This implicit fault tolerance could prove to be very useful.

3 An Exploratory Developmental System

We have argued that a number of features may be of use when designing a developmental system for evolutionary circuit design. These features are:

- The genotype-phenotype map should be generative
- The generative process should be implicit
- The generative process should be context-driven

3.1 A Cellular Structure

With the eventual goal of evolving complex, functioning circuits, an exploratory system based on these three principles was designed. It was decided that a rule-based system could satisfy all three criteria. But at this stage we have not determined what 'context' should mean, or even what a symbol might represent. Complex organisms segment their environment into a series of cells, with each cell maintaining an environment that differs from the surrounding cells and external environment. Additionally, cells receive chemical signals from neighbouring cells. Thus a cell's context consists primarily of chemicals generated by the cell and its neighbours. This cellular segmentation of chemical environments is a useful feature that has been incorporated into our model. But unlike biological organisms, the cells in our model are laid out on a two dimensional grid, mirroring the medium of electronic circuits rather than the three dimensional biological medium. This has the advantage of being easily mapped to a circuit design for a programmable logic device such as a Field Programmable Gate Array (FPGA), so is in keeping with our aim of developing a system with as little computational overhead as possible. To update the entire individual for a single developmental timestep, the set of rules that make up the chromosome is tested against the chemical environment in each of these cells. For each cell, only the rules that match that cell's environment are activated. If the environment differs between cells, it is possible for different rules to activate in each cell, which leads to their environments being altered in different ways. In this way, different chemical environments can be maintained between cells. At this stage we are left with the question of what chemicals generated by a cell and its neighbours we should model to provide a cell's context.

At the heart of biological development's generative ability is the process of DNA transcription. Transcription regulates the rate of gene expression through the presence of proteins called transcription factors, which either increase (activators) or decrease (inhibitors) the transcription rate of a particular gene. All transcription factors are proteins that are generated by the expression of other genes. Thus a dynamic, autocatalytic network of gene products specifies which genes are expressed. By modelling a cell's context with only transcription factors, and ignoring all other chemistry present in biological cells, we were able to keep our model as simple as possible yet encapsulate the key features that provide a generative, implicit, context-driven process. With this decision made, only the questions of how to model and regulate these gene products remained.

3.2 The Gene Model

To model an autocatalytic gene network, two basic features were needed. Firstly there must be proteins that serve as transcription factors. Secondly we needed a model of genes that when activated generate a transcription factor. The system must be able to regulate the activity of each of these genes depending on transcription factors that are present at any given time. In keeping with our philosophy of using the simplest model possible, transcription factor proteins were modelled as binary state variables. Each gene was modelled as a rule. The precondition of the rule specified which proteins must be present (activators), and which must be absent (inhibitors) in order for that particular gene to activate. The postcondition of the rule defines the protein that is generated if the rule is activated. An example rule with five transcription factors is shown in Figure 1.

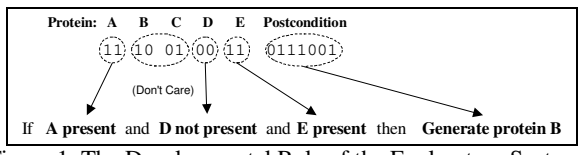

Figure 1: The Developmental Rule of the Exploratory System

For a gene like this to be activated, the proteins in the environment must match the pattern of proteins specified in the rule precondition. There are two bits in the rule precondition for each protein in the model (in this example proteins A-E) that specify whether it should be present, absent, or ignored for the gene to activate. A set of these rules make up the chromosome and defines how the proteins interact over time. At a given timestep in the developmental process the environment is inspected to determine which proteins are present, then each rule is inspected to determine whether the environment matches the rule, and if it does, the rule is activated, the protein defined in the rule's postcondition is generated, and goes on to make up part of the protein environment of the following timestep.

3.3 Modelling Intercellular Communication

A feature crucial to complex pattern formation has not

been discussed – communication between the cells. Context is a key feature of our model - cells must be able to affect their neighbour's environment. Biological development uses many forms of local intercellular communication. As modes of communication vary so wildly in their physical mechanisms, and as we require a simple model of development, we abstracted all details of communication, beyond the idea that it was local, and that communication involved the same gene products that are regulated by gene expression. In our model a cell inspects each of its neighbours to determine what proteins they are generating. (Recall that the generation of a given protein is a binary decision resulting from the activation of one or more rules.) If half or more of the cell's neighbours generate a particular protein, then the cell considers the protein as present in its environment at the next timestep.

To simulate this process, the cell model for our exploratory system contains a protein detector and a protein generator, in order to record the proteins that are present in the cell, and the proteins that are detected by the cell. A developmental timestep for a cell proceeded thus:

1. For each protein in the model the cell's protein detector sends a query to each of its Von Neumann neighbours (i.e. the four neighbours to the north, south, east and west on a 2D grid) to determine if they are generating that protein. If half or more of the cell's neighbours are generating a protein then the protein is considered to have been detected and the cell's protein detector is updated to reflect this.

2. The rule set is tested against the pattern of proteins detected by the detector in step 1. As each rule with a precondition matching the cell's current pattern of proteins is activated, the cell's protein generator is updated to represent the protein specified in the rule postcondition.

These two are then repeated for a number of cycles, as shown in Figure 2, allowing the pattern of proteins formed across the global array of cells to change until a stable state or cycle of states is reached, or until development is halted after a predetermined number of timesteps.

3.4 Mapping Pattern Formation to Circuit Synthesis

The developmental system described so far models the process of forming patterns of gene products. What remains is for a mechanism to be introduced by which the patterns of gene products generate a circuit design.

Our aim was to achieve this with as little computational overhead as possible, so fitness could be measured quickly. We intended to measure fitness using a Xilinx Virtex FPGA[12], which contains an array of identical configurable logic blocks (CLBs). A simple solution was to map each cell in our cellular array directly to a CLB on the Virtex array, and to allow the activity of the genes in each cell to alter the functional components in the CLB in some way. So in addition to proteins, our cells also contained functional components that map directly to functional components in a CLB. In order to keep the model simple, we added as few components as possible to our cell model. Each cell would have four input wires that could be driven by its local neighbours, two 4 input lookup tables (LUTs), and an output wire from each LUT. The LUTs would map directly to two of the four LUTs in a Virtex CLB, and the input and output wires would map directly to manually selected single lines between the CLBs. For details of the Virtex architecture and how this mapping was made see [1].

Finally we must define how these components are altered by the activity of genes in each cell. At this stage the rules only generate proteins: if a rule is activated in a cell, the rule postcondition is used as a key to a lookup table of proteins, and the protein generator in the cell is updated for that protein. To allow the functional components to be altered by gene activity, we simply introduced new postconditions to the lookup table. These coded for an alteration to the logic in a CLB. Over the course of development, the activities of these circuit-altering postconditions were recorded by activity counters – one counter in each cell for each circuit-altering postcondition, and once development was complete the activity counters were inspected in order to determine what alterations should be made to a predefined template circuit on the Virtex. Details of this process are in [1].

3.5 Abstraction in Action

In [1] we presented attempts to evolve a two bit adder with carry using the system described above. We used 5 proteins, just as the example rule shown in Figure 1, and a 2x5 array of cells. Both protein and circuit-modifying rule postconditions were coded in a 7 bit lookup table. 80 rules were used, yielding 1360 bits for the rule section of the chromosome. Protein concentrations in the cells were also evolved, encoded in the chromosome as an additional 50 bits making the chromosome length 1410 bits. Each individual was allowed to develop for 30 steps. Fitness evaluation was carried out on a Xilinx Virtex XCV400 [12]. The five cells on the west edge of the evolved area were provided with five input signals of the two bit adder with carry problem, A0, A1, B0, B1 and Carry_in, as their eastbound signals. The task was to evolve a circuit where output of three predefined output cells matched the two bit adder with carry truth table. The genetic parameters are shown in Table 1. The experiments showed we could generate iterative patterns of proteins that automatically mapped to circuit designs, and partially solved the adder problem, but we did not discover fully functional adder circuits. Here we present new analysis that reveals some interesting features of the circuits that were evolved.

Operator	Type	Rate
Selection	2 Member tournament	80%
Crossover	One-point	100%
Mutation	Point	5 per chrom.
Other parameters: Generational GA with elitism, population size = 100, random initialisation.		

Table 1: Parameters for the evolutionary experiments

Figure 3 shows the best circuit evolved in [1], along with protein concentrations in the cells at the start and end of development. The circuit diagram shows the 2x5 array of CLBs, each containing two LUTs, labelled F and G. Input signals are shown on the left of the diagram, and the evolved routing wires are shown outputting the right hand

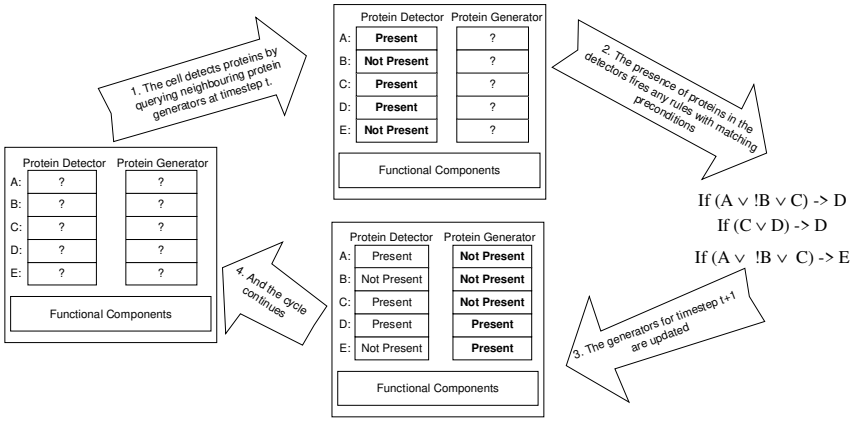

Figure 2: A developmental timestep highlighting the protein interaction model with a cell

side of each CLB, with the inputs appearing on the left-hand side of each CLB.

All but one of the routing wires for the circuit shown were generated by the seven rules also shown in Figure 3. The wires shown as thick black lines are generated by the three rules in Group 1, and transmit signals north. The wires shown as dotted lines are generated by the two rules in Group 2, and transmit signals east. We can see that these rules have general preconditions, and because of this, by the end-point of development they had been activated repeatedly in all cells. Hence these five general rules bias the phenotype to always pass signals north or east. A bias like this is useful for designing combinational logic because it avoids time-recurrent loops in the signal paths, which are unlikely to be useful. In other words the rules impose a useful design abstraction. We can now say that further to our work showing this system could generate simple iterative patterns of proteins, the protein patterns can be harnessed to represent useful design abstractions, encoded succinctly in a handful of general rules.

One of only two routing wires not generated by these two groups of rules is the wire shown as a dashed line. This wire was generated by the two rules in group 3. Whilst the first of these rules is general, and is actually activated in all cells, the second is very specific, and is only generated in a single cell. Additionally we can see that this rule breaks the feedforward design abstraction, passing a signal southward, and in fact forms a time-recurrent loop that would not have been considered by a human designer. This demonstrates that even though we can encapsulate useful design abstractions succinctly, under specific conditions a general abstraction can be overridden where evolution sees fit. So we have not prevented evolution finding a potentially useful innovation beyond the feedforward design abstraction that the first two groups of rules impose. This is precisely what we set out to achieve in section two.

4 Designing a New Model of Development

As it stands, the system is capable of learning and representing design abstractions, but as it is not capable of solving the relatively simple the two bit adder with carry problem that have previously shown can be solved by a component-based evolutionary search [1,2], it needs further development before it can be of engineering use.

To identify the shortcomings in the system, we considered how a design problem might be solved in two stages. The first stage is that of pattern generation. We must be able to generate a pattern of proteins that is of sufficient complexity that can eventually be mapped to a functioning circuit design. The second stage is how the

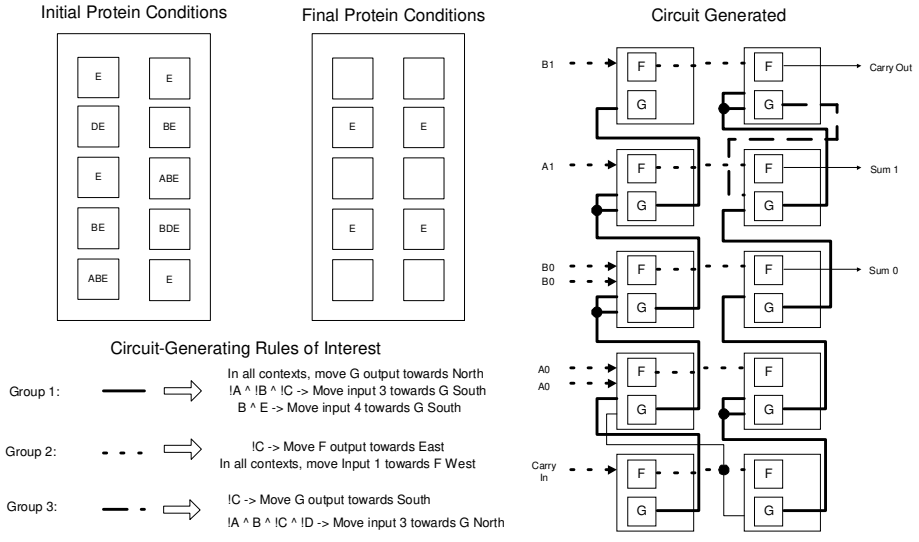

Figure 3: A Solution to the Two Bit Adder Found by the Exploratory System

pattern of proteins maps to a circuit. The rest of this paper is concerned with the first stage. We explore the system's ability to generate useful pattern of proteins, and describe the changes we have made to our system that allow us to generate and maintain these protein patterns.

4.1 Exploring the Initial Model's Computational Power

The first question that we needed to answer was what kind of protein pattern would a useful for generating adders? To answer this we hand-designed a two bit ripple-carry adder within the 2x5 array of cells we described above, and then deduced a pattern of proteins that could be mapped to our hand-designed circuit using the circuit-modifying postconditions we described earlier. So if the developmental system could generate this pattern of proteins, we could be sure that there was a fully functional adder that the system could learn to map to a circuit. The pattern of proteins we designed is shown in the Target Pattern column for Experiment 1A in Table 2. We then altered the developmental system so that the rule postconditions that had previously mapped to circuit alterations now had no effect, and attempted to evolve the protein pattern. This meant we still used an identical rule structure as shown in Figure 1, again with 80 rules to make a chromosome length of 1360 bits. Fitness was based on the Hamming distance of the candidate pattern of proteins from the target pattern of proteins. The longest distance possible was 50, which would result if every protein that was specified as required in the target pattern was absent in the candidate solution, in every cell of the 2x5 array, and vice versa. Fitness was set as the measured distance between the target and candidate solution subtracted from the longest distance of 50. This results in a maximum fitness of 50, and a minimum fitness of 0. The genetic parameters followed our earlier experiments, as shown in Table 1. Initial protein conditions and results are shown in Tables 2 and 3 respectively as Experiment 1A. The target pattern we tried to evolve consists of nine of the protein concentrations set high, and all other protein concentrations set low. In every run the best solution generated no proteins at all. This meant that all nine proteins set high in the target pattern were not generated, giving a fitness of 41. The system was unable to generate the pattern of proteins required, or any part of the pattern that required computation from the initial conditions.

In addition to the need to form patterns from simple starting conditions, once a useful pattern had formed it would be useful if the system was able to maintain this pattern indefinitely. The target of our second experiment was purely to maintain the same pattern as we attempted to generate in experiment 1A, by setting the starting conditions to the same pattern as the target pattern, as shown in Table 2. The experimental results are shown in row 1B in Table 3. The solutions obtained were similar to those from experiment 1A – 18 of the 20 runs generated protein B in the two cells of the central row cell and no other proteins in any other cell. These results again suggest that the system could only generate or maintain only the simplest patterns from the given starting conditions.

Expt	Initial Conditions	Target Pattern	Example Rule Set
1A	B / B / B / B / A, B — A	D E / A C / B B / A C / C	N/A
1B	D E / A C / B B / A C / C	D E / A C / B B / A C / C	N/A
1C	ABD ACD / A D / B C / A D / ABD ACD	ABD ACD / A D / B C / A D / ABD ACD	A!B -> B !CD -> C B -> A C -> D
1D	AB B BC / A B C / A B C / A B C / AB B BC	AB B BC / A B C / A B C / A B C / AB B BC	A!B -> A B -> B !BC -> C
1E	(blank) / (blank) / (blank) / (blank) / A B C	AB B BC / A B C / A B C / A B C / AB B BC	N/A

Table 2: Results of evolving target patterns of proteins A-E on 2x5 and 3x5 cell arrays with the exploratory system

Expt.	Best / Max Fitness	% Perfect Runs	Mean (Best Fitnesses)	Std. Dev. (Best Fitnesses)
1A	41/50	0	41	0.00
1B	43/50	0	42.8	0.61
1C	50/50	10	45.1	2.43
1D	75/75	100	75.0	0.00
1E	61/75	0	61	0.00

Table 3: Results of evolving target patterns on 2x5 and 3x5 cell arrays with the exploratory system

A brief thought experiment revealed where some problems lay. We attempted to deduce a simple yet non-trivial pattern that the system would be capable of maintaining for an arbitrary number of timesteps if it were given as the starting condition on the 2x5 array. We set out to design a pattern and corresponding set of rules that maintained the right and left hand columns as different environments – no cell on the left should have an environment the same as one on the right. It turned out that hand designing such a pattern was not trivial, owing to the special conditions of intercellular communication present in each corner. Recall that each cell's protein environment consists of proteins generated by at least half of their neighbours, so any protein generated in a corner cell will always be present in the environment of its neighbours in the next timestep. The 2x5 array has adjacent corner cells so they will always detect any proteins generated by each other. However we managed to design a rather complex pattern that maintained different environments without relying on generation in corner cells to differentiate between the sides, which is shown as Experiment 1C in Table 2 and is maintained by the rule set also shown in Table 2. The system was set with this pattern as both starting and target pattern, with results shown in Table 3. When this experiment was rerun allowing 1000 generations rather than 100, the percentage of perfect runs rose to 80%. So we can see that in the case of the 2x5

array, selected patterns can be maintained, but the range of patterns that can be maintained is highly constrained, and finding such a pattern involves careful design. We also now know that a 2x5 array is a special case where corner cells are adjacent.

Ex. 1D in Table 3 shows the result of maintaining a simple pattern in a 3x5 array that differentiates between the environments in all three columns, with all other parameters as before. The pattern maintained is shown in Table 2, alongside results of the experiment, which show that every run results in a perfect score. This improvement is easily explained: by moving to a 3x5 array, we have a buffer cell between the previously adjacent corner cells, and remove the problem of how to prevent excessive communication between these cells. As arrays two cells wide are the only cases where this issue occurs, all further experiments were carried out using a 3x5 array, so more general conclusions could be made.

Having managed to show that a pattern that generates different environments in each column can be maintained by the system, we attempted to evolve a rule set that would not only maintain the pattern, but generate it from the simple starting conditions. The starting conditions chosen were that each column would initially be identified by a column-determining protein in the most southerly cell, as shown for Ex. 1E in Table 2. Results from the experiment are shown in Table 3. No perfect solutions were found, but on examination of the best solutions, every one generated protein B in every cell, protein A was only generated in the south-western corner cell and protein C was only generated in the south-eastern corner cell. From this we postulated that the system was able to make use of the starting conditions in the corner cells, and was able to generate general conditions across the entire array, but was not able to (easily) communicate information contained in the cells with special starting conditions (in this case the southerly row of cells) to the rest of the array, i.e. the cells could not *communicate* efficiently.

4.2 Improving Intercellular Communication

In order to improve the communication, we devised a mechanism by which information could be transmitted from a cell at the south of the array to a cell at the north. using a wavefront of information moving from the bottom to the top of the array, in effect 'growing' a pattern behind it. With our model, this would be very difficult, if not impossible, to achieve. If we alter our model so a cell detects *every* protein generated in it's neighbourhood rather than only those generated by at least half of its neighbours, the task becomes much simpler. An example of wavefront growth from a generative step is shown in Table 4 with the first column showing an initial generative step, with a row of protein N being generated. Above this is a row of protein I acting as an interface, and above that, the protein to be replaced, protein O. The second column shows the proteins detected by each cell in the old model and the third shows what would be detected by a model where every protein generated in a cell's neighbourhood is detected. The fourth column contains a rule set that with the new model would generate a moving wavefront of I that travels up the array, replacing cells behind it with N. The final column describes the function of each rule. This set of rules is easily modified so that the pattern generated by the wavefront alternates. However, there is still difficulty in maintaining some patterns after the wavefront has moved on. This is the case, for instance, if we required the wavefront to leave behind horizontal stripes of cells that continue to generate one of two alternating proteins, as the symmetry of the pattern is such that each cell generating a protein has two neighbours generating the other and vice versa, so every cell would experience the same environment, containing both proteins.

One solution is to introduce the concept of concentration to the detection model. For example consider a model where proteins are generated by cells at unit concentration. If a cell is part of a row that is generating a stripe of protein A, and the cell were to sum the concentrations generated by itself and its neighbours, it would experience A at a concentration of 3. If the rows below and above generated protein B, it would only experience B at a concentration of 2. Hence cells in each stripe would experience different enviroments. So by using a model like this we can open up a range of useful mechanisms for communicating information between cells. This approach is closely related to a two dimensional totalistic cellular automata, as described in [13].

4.3 The Totalistic Developmental System

We developed a new model to include these changes. To take advantage of the richer neighbourhood conditions the syntax of the rule precondition must be altered. In addition to the inequality operators used earlier, precedence operators were introduced. Each protein condition is now specified by five bits. The final three bits define the protein concentration that the operator will act upon. Hence a rule can specify concentration values to range from 0 to 7. The first two bits of the protein condition specify the operator – not equal to (00), less than or equal to (01) greater than or equal to (10) or equal to (11). Again the protein to be tested is determined by the locus of these bits. We reduced the postcondition size to two bits, and every postcondition in this new system mapped directly to one of four proteins, A-D. This change also reduced the precondition size to 20 bits. An example rule is shown in figure 4. Finally the number of rules was reduced from 80 to 15. This resulted in a chromosome length of 330 bits.

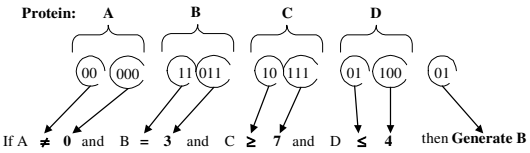

Figure 4. The Totalistic Developmental Rule

Several experiments were then conducted to verify that the patterns we have discussed could be evolved from simple starting conditions. The first experiment evolved a target pattern where the concentration of protein A was highest

States Generated	States Detected (Old)	States Detected (New)	Rule Set	Rule Description
O O O O O O O O O I I I N N N	O O O O O O O O O I I I NI N NI	O O O O O O IO IO IO INO INO INO NI NI NI	N->N O->O O+I->I N+I+O->N N+I->N	Maintains cells already set to N Maintains cells still set to the O Creates a new row of I above the old row Grows a row of N, replacing the old I Maintains the row of N below I

Table 4: An example rule set that generates a travelling wavefront with the new model, but not the old

in the western column, B was highest in the central column, and C highest in the eastern. Initial conditions were set as if the southernmost three cells from west to east had just generated A, B and C respectively.

This is analogous to Ex. 1E, where the aim was to generate and maintain a different protein environment in each column of the array. The initial protein conditions, target protein pattern and the best pattern evolved are shown in Table 5 as Ex. 2A. For a given cell, the fitness contribution of each protein state was calculated as the difference between the final and target concentrations, subtracted from the maximum concentration, which was 7. This gave a maximum fitness of 420 and a minimum fitness of 0. Experimental results are shown in Table 6 as Ex. 2A. The experiment was repeated for 10 runs of 1000 generations, with all other parameters as before. Of the ten runs, two evolved perfect solutions. The first perfect run used only three rules to generate the pattern:

(1) A!=0, B!=4, C==0, D<=1->C
(2) A<=2, B<=7, C==1, D<=3 ->B
(3) A!=1, B!=5, C!=0, D<=6 ->A

We can interpret these rules quite easily. Rule 1 and Rule 3 work in concert. In any given timestep, Rule 1 will generate C in all cells down one side of the array that currently contain A. Rule 3 will generate A in all cells of the opposite side of the array that contain C. So A and C generation swaps from side to side at each timestep. At the same time, the pattern grows one cell northwards at each timestep as the cell to the north detects a low concentration of that protein. Finally Rule 2 states that if A is present and C is present, then B should be generated. This only occurs where concentrations of A and C meet, in the central column. So we see that B generation, being dependent on A and C generation grows upward, following a cell behind the A and C generation front until the array is filled. At this point B generation is constant, and the only change between timesteps is that the presence of A and C alternates from side to side. The example has developed almost as we had envisaged – with a wavefront travelling up the array until the array is filled.

Experiment	2A	2B	2C	2D	2E
Best Fitness	420	420	412	420	410
Max Fitness	420	420	420	420	420
% Perfect Runs	20	20	0	80	0
Mean (Best Fitnesses)	408.5	408.3	409.0	416.2	405
Std. Dev. (Best Fitnesses)	8.40	7.42	0.99	8.35	9.10

Table 6: Results of evolving target patterns on and 3x5 cell arrays with the totalistic system.

The initial conditions had been chosen to mimic what would have been present if A, B and C had been generated in the southernmost cells at timestep –1. This has the effect of reducing the workload on the learning algorithm – all evolution had to do was to continue from the last step of protein generation. This time the starting conditions were simplified. Only the southernmost cell of each column was set with the protein it should eventually fill with. These conditions are shown as Ex. 2B in Table 5. The results were gathered over 10 runs of 1000 generations, and are shown in Table 6. The performance was almost identical to those of Ex. 2A. So it seemed that, at least in this case, it was not necessary to take great care over selecting initial conditions, for instance by emulating a hypothetical earlier generative step.

4.4 A Diffusion-based Model of Protein Concentrations

The next experiment set out to generate horizontal stripes of protein A alternating between low and high concentration from row to row, and horizontal stripes of protein B alternating between high and low concentrations. It was envisaged that this would be the result of horizontal stripes of A and B protein generation. The results for Ex. 2C were gathered over 10 runs of 1000 generations, and are shown in Table 3. A perfect solution could not be found, but the vast majority of the 10 runs achieved a fitness of 410 or above. On closer examination of the patterns, it was clear that the system as it stood could not produce a striped pattern of protein A and B *environments* from a stripes of protein A and B *generation*. Each cell on the east and west sides of the array has three neighbours. So each side-bound cell has an environment of two cells from a stripe generating one protein (itself and one cell horizontal to it) and two cells from stripes generating the other (one above and one below) As both of these groups make equal contributions to the cell's environment, the cell would have no means of determining which type of row it was located in.

So it seemed that the level of communication between generating and non-generating cells again was not high enough for them to be able to organise themselves into the pattern required, because there is no mechanism for a cell to deduce if it is generating a protein. One solution is for cells to detect proteins they generated themselves at higher concentrations than if generated only by a neighbour cell. Thus, when a cell detects a high concentration the likelihood that that the cell itself is generating the protein would be higher than in the previous model, making the task of separating local signals from neighbouring signals easier. This models the physical process of diffusion, where concentration reduces as molecules diffuse away from their source. We set our model so that when a protein was generated, the cell in which it was generated would gain a contribution of 3 towards its final detected concentration, as opposed to 1 from each neighbour that generated the protein. The experiment was repeated having updated the starting conditions and target pattern to reflect these changes, and the results are shown as Ex. 2D in

Expt	Initial Conds	Target Pattern	Best Evolved
2A	00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00 00 10 11 11 01 00 10 10	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20
2B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40 04 00 00 00 40	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20
2C	00 00 00 00 00 00 00 00 00 00 00 00 32 32 32 00 00 00 23 23 23 00 00 00	23 23 23 00 00 00 32 32 32 00 00 00 23 23 23 00 00 00 32 32 32 00 00 00 23 23 23 00 00 00	22 33 22 00 00 00 32 32 32 00 00 00 22 23 22 00 00 00 32 32 32 00 00 00 22 33 22 00 00 00
2D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 10 10 00 00 00	40 50 40 00 00 00 24 25 25 00 00 00 42 52 42 00 00 00 24 25 24 00 00 00 42 52 42 00 00 00	40 50 40 00 00 00 24 25 25 00 00 00 42 52 42 00 00 00 24 25 24 00 00 00 42 52 42 00 00 00
2E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 10 10 00 00 00	41 40 10 00 20 31 23 21 10 10 31 23 42 50 40 00 20 02 23 21 10 10 31 23 41 40 10 00 20 31	41 40 10 00 10 01 23 21 10 10 31 13 42 50 40 00 20 02 23 21 10 10 31 13 41 40 10 00 10 01
Where the concentration values shown are for the following proteins:			AB AB AB CD CD CD AB AB AB CD CD CD AB AB AB CD CD CD AB AB AB CD CD CD AB AB AB CD CD CD

Table 5: Initial protein concentrations, target protein patterns and best patterns evolved with the totalistic system. A key to protein concentrations is shown in the lower right hand side of the table.

Tables 5 and 6. The results show that 8 of the 10 runs succeeded in evolving the pattern perfectly.

4.5 Learning an Adder Pattern Successfully

We again hand-designed a two bit ripple-carry adder, this time within the 3x5 array of cells, and then deduced a pattern of proteins that could be mapped to the hand-designed circuit using the circuit-modifying postconditions we described earlier. Thus we could be sure that there was a fully functional adder that the system could learn to map to a circuit, as before. The pattern we designed is shown as Ex. 2E in Table 5. The experiment was run for 5000 generations, with all other genetic parameters unchanged from the previous experiments. The results for 20 runs of the experiment, shown in Table 6, showed no perfect solutions, the best solution found yielding a fitness of 410/420. As can be seen in Table 5, this solution solved the problem correctly for three proteins, but protein C was only generated in two of the fours sites required.

In [5] Miller had some success using extremely small populations. With this in mind the experiment was repeated using a hillclimbing algorithm with point mutation set at 5 mutations per chromosome. If a new solution had equal fitness to the current best, the new solution was selected with 50% probability. The hillclimber was run for 500,000 evaluations. Over the course of 20 runs, 2 perfect solutions were successfully found, showing that the system was capable of generating a pattern that can be mapped to an adder using the mechanisms that we had previously developed.

5 Conclusions

The problem of scalability is one of great importance if evolutionary circuit design is to achieve much impact in the real world. We have demonstrated an important benefit of using a developmental mapping, namely that development softly biases evolution towards learning useful design abstractions and encodes them succinctly, thereby aiding scalability.

The rest of the work presented here focused on changes we made to our model of development in order to improve the computational power of the pattern formation stage of development. These improvements showed the importance of good intercellular communication to development's ability to generate and maintain a range of patterns. Finally we showed that the computational power of the pattern formation model is now able to learn patterns that can successfully be mapped to fully functional adder circuits, where previously we could not, which is an important step towards tackling real-world problems with development.

Acknowledgements

The author would like to thank Peter Bentley for his invaluable insights and advice on this work.

References

[1] T. G. W. Gordon and P. J. Bentley, "Towards Development in Evolvable Hardware" 2002 NASA/DoD Conf. on Evolvable Hardware, Washington D.C., U..S.A, 2002.

[2] T. G. W. Gordon and P. J. Bentley, "On Evolvable Hardware" in *Soft Computing in Industrial Electronics*, S. Ovaska and L. Sztandera Eds,.Physica-Verlag, Heidelberg 2002, pp 279-323.

[3] A. Tyrell, P. C. Haddow, and J. Torresen, "Evolvable Systems: From Biology to Hardware" *LNCS* vol. 2606, Springer, Berlin, 2003, pp. 465.

[4] G. Tufte and P. C. Haddow, "Building Knowledge into Developmental Rules for Circuit Design" 5th Int. Conf. on Evolvable Systems, Trondheim, Norway, 2003.

[5] J. F. Miller and P. Thomson, "A Developmental Method for Growing Graphs and Circuits," 5th Int. Conf. on Evolvable Systems, Trondheim, Norway, 2003.

[6] W. X. Liu, M. Murakawa, and T. Higuchi, "ATM cell scheduling by function level evolvable hardware" in *Evolvable Systems: From Biology to Hardware*, LNCS vol. 1259, 1997, pp. 180-192.

[7] J. M. W. Slack, *From Egg to Embryo*, 2nd ed. Cambridge: Cambridge University Press, 1991.

[8] G. Wagner and L. Altenberg, "Perspective-complex adaptations and the evolution of evolvability," *Evolution*, vol. 50, pp. 967-976, 1996

[9] J. Koza, F. H. I. Bennett, D. Andre, and M. A. Keane, *Genetic Programming III*. San Francisco, Morgan-Kauffmann, California, U.S.A., 1999.

[10] P. J. Bentley and S. Kumar, "Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem.",in Proceedings of GECCO'99, Orlando FL, USA, 1999

[11] N. Chomsky, *Syntactic Structures* Moutin & Co., The Hague, 1957.

[12] Xilinx_Inc., *Virtex 2.5 V Field Programmable Gate Arrays Data Sheet*: http://direct.xilinx.com/partinfo/ds003.pdf, 2001.

[13] S. Wolfram, *A New Kind of Science*. Champaign, IL, U.S.A.: Wolfram Media, 2002.