Evolution of Circuits in Hardware *and* The Evolvability of Artificial Development

Tüze Kuyucu
Intelligent Systems Group
Department of Electronics
University of York, York, UK

Dissertation for the degree of Doctor of Philosophy in Electronics, June 2010.

Thesis Committee:

Prof. Andy Tyrrell, University of York
Dr. Julian Miller, University of York
Dr. Gianluca Tempesti, University of York
Prof. Yaochu Jin, University of Surrey

Anacığma,
Eğitimim için verdiğin emekleri asla unutmayacağım.

To Mom,
I will never forget your efforts for my education.

Abstract

Automatic design of digital electronic circuits via evolutionary algorithms is a promising area of research. When evolved intrinsically on real hardware, evolved circuits are guaranteed to work and the emergence of novel and unconventional circuits is likely. However, evolution of digital circuits on real hardware can cause various reliability issues. Thus, key mechanisms that produce reliable evolution of digital circuits on a hardware platform are developed and explained in the first part of this thesis.

On the other hand, the evolution of complex and scalable designs without any assistance is thwarted due to increasingly large genomes. Using traditional circuit design knowledge to assist evolutionary algorithms, the evolution of scalable circuits becomes feasible, but the results found in such experiments are neither novel anymore nor are they competitive with engineered designs.

A novel, biologically inspired gene regulatory network based multicellular artificial developmental model is introduced in this thesis. This developmental model is evolved to build digital circuits that can automatically scale to larger designs. However, the results achieved still remain inferior to engineered digital circuit designs.

Evolving a developmental system for the design of engineering systems or computational paradigms provides a variety of desirable properties, such as fault tolerance, adaptivity, and scalable designs automation. However, developmental systems in their role as computational networks are as yet poorly understood. Many mechanisms and parameters that a developmental system comprises are based on various assumptions, their biological counterparts, or educated guesses. There is a lack of understanding of the roles of these mechanisms and parameters in forming an evolvable platform for evolutionary computation.

Initially, various experiments are shown to demonstrate the evolvability of the new developmental system. A thorough investigation is then undertaken in order to obtain large amounts of empirical data that yields a better understanding of some of the crucial developmental mechanisms and parameters on the evolvability of multicellular developmental systems.

Acknowledgements

I would like to start by thanking my supervisors: Andy Tyrrell for his continuous encouragement and support, for proof reading, for his role in the funding of my PhD, and always being so punctual; Julian Miller for his help, advice, ideas and numerous discussions on evolution, development, and teaching.

I would like to thank Martin Trefzer, my research partner for the past 3 years; for the endless discussions, pair programming sessions, never ending ideas and enthusiasm, and above all for all the kit-kat and coffee sessions. Without him, it would have been impossible to finish this thesis within 3 years. It was a real blast working with you dude!

In addition to above, I would like to thank all my office colleagues for their support and constant supply of coffee and laughter: James Walker, for his help with genetic programming and university administration, and always convincing me that I have something worth to write about; Omer Qadir, for the beer, socials, and various discussions about the ultimate question of life, the universe, and everything; Yang Liu (Jerry), Cristina Santini and Mic Lones, for the discussions and advice on biological development, and research in general; Andy Greensted, Antonio, and James Hilder for their company and help with various geekery such as electronic hardware and Linux.

I would like to thank my undergraduate supervisors, Steve Cobb and James Hereford, for their valuable support and advice during and after my undergraduate years, who also inspired me to follow up a career in academia.

I would like to thank my parents, Dervişe and Sadi Kuyucu, whose passion for my education and ever growing support made this thesis possible. I would also like to thank my future wife, Jie Chen, for all her emotional support, understanding, valuable advice, and ability to cheer me up at the worst of times.

Finally, I would like to thank the Engineering and Physical Sciences Research Council (EPSRC) UK, for their financial support (project reference EP/E028381/1).

Contents

List of Tables

List of Figures

List of Algorithms

Hypothesis

Evolution of digital circuits in hardware can provide interesting and novel designs, but not complex and human competitive results. Multicellular development in biology can be simulated to create a scalable system for the evolutionary design of electronic systems. By understanding the evolution of circuits on real hardware and the behaviour of multicellular development in a computational environment, the key factors that determine the evolvability of an artificial developmental system can be determined.

Chapter 1

Introduction

Technology is an ever growing endeavour of human avidity, since the invention of simple tools like the wheel, spoon, or knife, the development of technology has never ceased. What has allowed us to progress and advance our lives into a global social structure today, has sometimes been labelled as the nemesis of nature [Kovel, 2002]. The invention of the steam engine, gun powder, combustion engine, paper, nuclear fission and fusion, are all outstanding sources of technology that have advanced against nature and its equilibrium. The degrees of advancement achieved in technology is awe-inspiring, and it provides us with prodigies that surpass the abilities of nature's organisms. The supercomputers that can carry out quadrillions (order of 10^{15}) of mathematical operations in a second [Bland et al., 2009], mobile phones that allow a person to talk to another from the other end of the world, the telescopes that can look into the depths of galaxies outside the Milky Way. Technology has created designs that operate beyond the scope of any biological organism inhabiting earth. Despite this, technology has so far failed to faithfully imitate even the simplest functions nature has long mastered for sustainability and survival.

Today as researchers, a group of us tries to analyse and understand nature and its organisms. We try hard to understand, mimic and compete with nature itself. Biological organisms exhibit behaviours that contrast markedly with engineering design. Adaptivity to unknown conditions, self-repair, self-replication, fault tolerance, intelligent and intuitive decision making, and learning are some of the properties biological organisms exhibit. Artificial Intelligence (AI) is a sub-field in engineering, where the latest technol-

ogy is used to build intelligent systems. However, the intelligent systems built in AI are far from achieving the intelligent behaviour and self control that biological organisms demonstrate.

Natural sciences study nature and the origin of the biological organisms and their environment. The study of natural sciences has partially revealed the complicated processes that have built and shaped the biological organisms to their current state. Hence, some researchers try to model what has been learnt from the biological way of building complex systems in the hope of achieving comparable behaviours in engineering technology.

Perhaps technology does not need to go against or at a different direction to nature. Lessons learnt from nature could be valuable in expanding technology to another dimension. This thesis attempts to model a small part of the knowledge attained from natural sciences for building artificial systems. More specifically, this thesis presents chapters on biologically inspired (bio-inspired) computational models. In an effort to learn from nature in advancing the technology present in computer science and electronics, this thesis attempts to develop and use bio-inspired techniques for the design of electronic circuits and computational systems. It aims to develop techniques that enable effective use of bio-inspired models of evolutionary design and multicellular development. The end result also provides a study for a better understanding of these bio-inspired techniques, which is progress towards understanding the evolution of biological organisms and the use of bio-inspired techniques for the design of computational technologies.

1.1 Thesis Layout

This thesis is organised into eight chapters.

Chapters 2 and 3 are focused on Evolvable HardWare (EHW). Chapter 2 introduces the field of EHW, where the process of evolution in nature is used as the inspiration for designing hardware models in engineering. Chapter 3 uses a new hardware platform for the evolution of real circuits on hardware, and develops a group of essential techniques for quick and effective evolution of digital circuits on real hardware.

Chapter 4 provides background on multicellular development and its existing artificial models in evolutionary computation. It also discusses the potential benefits of multicellular development to Evolutionary Computation (EC).

Chapter 5 introduces a new bio-inspired developmental model to assist the evolutionary design of artificial systems. The design of the developmental model is described in detail and compared with similar models in literature.

Chapter 6 investigates the performance of the new developmental system and validates its developmental properties. The general setup of the experimental environment used for the evolutionary developmental experiments in the rest of the thesis is provided in the algorithm configuration section (Section 6.1). Chapter 6 demonstrates; the responsive and dynamic nature of the Gene Regulatory Network (GRN), the ability of the individual cells in the multicellular Artificial Developmental System (ADS) to self-organise and differentiate, and the fault tolerant abilities of the ADS via simple experiments.

Chapter 7 investigates the design of digital circuits via the use of evolution and the proposed developmental system. For the first time, the design of a digital circuit via the evolution of a GRN based ADS includes the connectivity of the circuit components as well as the type of components. Chapter 7 demonstrates that the proposed ADS can be used for the scalable design of digital circuits.

Chapter 8 provides detailed investigations on the properties of various parameters and mechanisms that affect the performance of the artificial developmental system in order to better understand the use of multicellular development in evolutionary computation and improve the overall performance of the presented system.

The final conclusions on the work done and the future directions for the use of artificial development in evolutionary computation is presented in Chapter 9.

1.2 Contributions

The novel contributions of this thesis are:

1. The evolution of gate-level digital circuits on a novel hardware platform (Chapter 3).

2. The development of mechanisms that enable the effective evolution of valid circuits on real hardware (Chapter 3).

3. Providing experiments and discussions that supply evidence and support for the development of an effective evolutionary platform (Chapter 3 and Appendix B).

4. The design of a new artificial genetic regulatory network model and multicellular developmental system, which use inspirations from biology and engineering (Chapter 5).

5. Experiments, evidence, and discussions on improving the evolvability of a developmental system via identifying the suitable and the correct use of mechanisms, parameters, and constraints (Chapters 6, 7, and 8).

6. Demonstration of a scalable approach to the evolution of digital circuits via the evolution of a scaling circuit via the use of artificial development (Chapter 7).

7. An understanding of developmental mechanisms and their effects on the evolvability of a multicellular developmental system (Chapter 8).

8. Multiple demonstrations on the importance of an effective fitness function for the successful evolution of a system (Chapter 3, 6, and 7).

Chapter 2

Evolvable Hardware

Evolution, a mechanism of random alterations and intelligent selection that was first proposed by Darwin in 1859 [Darwin, 1859], is regarded as a key element in the emergence and advance of biological organisms. These biological organisms are complex and competent; they can survive harsh environmental conditions and are capable of accomplishing highly sophisticated tasks. Although the emergence of any interesting behaviour via evolution is a time consuming process due to the stochastic behaviour of evolution, the biological marvels that exist in nature today are all a result of evolutionary change. Thus evolution is and has been a fascinating and fundamental topic in biology. The resulting evolved biological systems, possess an important set of characteristics that the products of engineering can, at present, only aspire to. Adaptivity, fault tolerance and recovery, regeneration, and learning are a few of these characteristics, which engineering methods struggle to (or are unable to) capture.

Evolution has also attracted a large community of researchers outside biology who take inspiration from the principles of evolution to tackle problems in their fields. The early use of evolution in the field of computer science, electronics and engineering was with the works of Von Neumann in late 40s, Box and Friedberg in late 50s, and Bremermann in early 60s [Box, 1957; Bremermann, 1962; Friedberg, 1958; Neumann, 1966]. However, it was not until the introduction of Evolutionary Algorithm (EA) that the field of Evolutionary Computation (EC) started getting attention. EAs took inspiration from evolution in solving mathematical problems, and they were introduced in late 60s / early 70s [Back et al., 1997; Fogel et al., 1966; Holland, 1973; Rechenberg, 1973].

An EA uses random alterations and an intelligent selection mechanism to optimise an existing solution to a given problem. An EA creates multiple variations of the solution at hand and evaluates each of the new solutions for any improvements. The given problem defines the most important part of the selection mechanism: the **fitness function**. The fitness function is used for guiding the algorithm in its search for the optimal solution to a given problem. The fitness function assigns a fitness, a score, to each of the potential solutions, which reflects the quality of each solution. Once each candidate solution has a fitness, selection is undertaken and a new batch of candidate solutions are created via **mutation**[1] and **crossover**[2] for the next **generation**. The process of evaluation (using the fitness function), selection, mutation and crossover carries on over and over again until a satisfactory solution or a time (or more often generations) limit is reached. The representations of a candidate solution in an evolutionary algorithm is referred to as the **genotype** –inspired by the DeoxyriboNucleic Acid (DNA)[3] in biological organisms–, which is often a binary string.

Other than optimising an existing solution, EAs are also used for finding a previously unknown solution to an existing problem. In such a case the EA starts with a random set of candidate solutions and optimises these. Figure 2.1 depicts the process a generic EA goes through in finding a solution.

Three types of EAs that are widely used today are:

- *Genetic Algorithm (GA):* introduced by Holland [Holland, 1973], is the most common form of EA, and was designed to model adaptive processes. The original representation of the "genotype", which the algorithm worked on was in binary, and both mutation and crossover operations were used during the search process.

- *Evolution Strategy (ES):* introduced by Rechenberg and Schwefel is similar to GAs [Back et al., 1997; Rechenberg, 1973], and was designed for parameter optimisation. A floating point representation was used in the initial implementation of ES, and only the mutation operation was implemented during the search process. The key differences with the classic GAs are the lack of crossover operation, use of

[1] Random alterations in a candidate solution.
[2] Joining parts of two candidate solutions (parents) to produce a new candidate solution (child).
[3] A double helix structured nucleic acid that contains all the genetic information used in the development of all biological organisms.

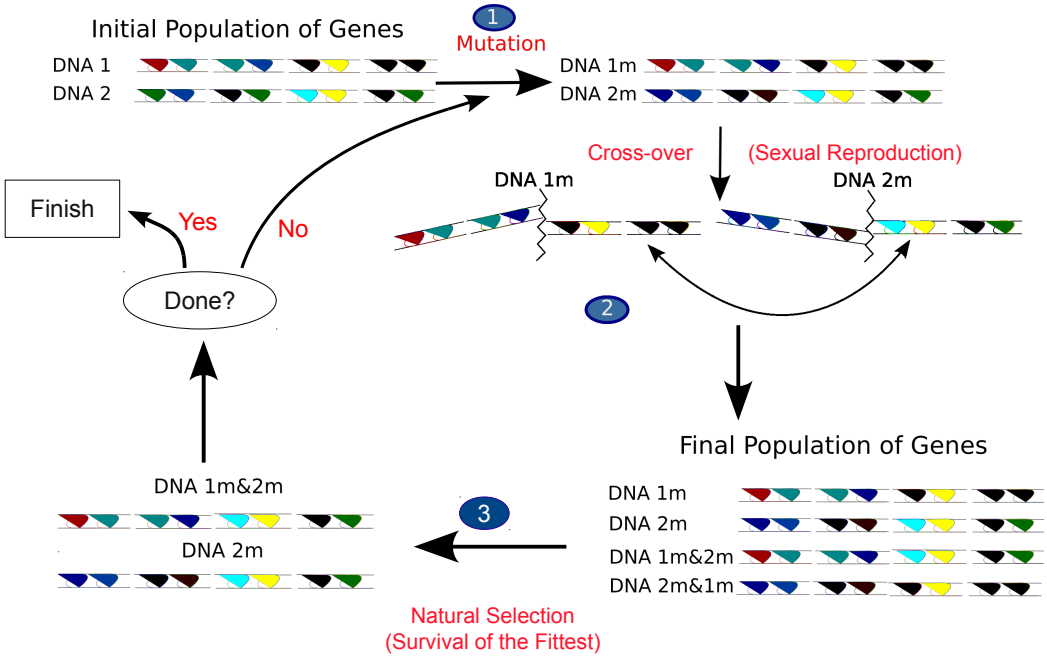


Figure 2.1: *The life cycle of an evolutionary algorithm.*

self-adaptation to adjust control parameters, and the strict definition of parent–offspring relationships in ES. The parent–offspring relationship in ES is defined by two versions of the ES as

$ES(\mu,\lambda)$ and $ES(\mu+\lambda)$.

Here μ is the parent size, and λ is the offspring size. The parents for the next iteration (referred to as generation in EAs) are deterministically selected only from the set of offspring of the current iteration (for the first case above) or from the set of parents of the current iteration and offspring (for the second case above).

- *Evolutionary Programming (EP):* introduced by Fogel [Fogel et al., 1966], was developed for the design of artificial intelligence through the evolution of finite state machines. EP is very similar to ES, and it also uses self-adaptation of algorithm parameters.

All these types of EAs share a similar process flow shown in Figure 2.1 with some differences such as the exclusion of crossover, different representations or selection mechanisms, and it is easy to create a different flavour of each of these EAs.

Initially, EAs were mostly used as optimisation algorithms to search for the global maximum or minimum in a mathematical function. They were successfully used for solving classical optimisation problems such as travelling salesman, or even tough Engineering Applications [Bramlette and Bouchard, 1991; Muller, 2002; Periaux et al., 1995]. Starting with the work of Fogel [Fogel et al., 1966], EAs were also used as a method to automatically create new designs. The popularity of the use of EAs as a method to automatically create new designs increased greatly by the start of 90s (more than 20 years after Fogels work). In 1990 Koza developed an evolutionary algorithm based automatic program design technique further and named it Genetic Programming (GP), which was first introduced by Cramer in 1985 [Cramer, 1985]. GP is an EA based methodology specifically developed for the evolution of computer programs that perform predefined tasks [Koza, 1992]. With the introduction of GP, more researchers started using EAs in the automatic design of computer programs and electronic circuits. GP has contributed to the EA community various novel computer program and circuit designs, and various researchers also worked on improving GP and developed their own versions [Banzhaf, 1993; Lones and Tyrrell, 2001; Miller and Thomson, 2000; Poli, 1996; Spector and Stoffel, 1996; Stoffel and Spector, 1996; Teller, 1993]. Traditionally GP uses an EA to evolve its genome, and the evolved genome is represented as a tree structure, which specifies the program being evolved. The early version of GP was designed to evolve LISP programs, and LISP is a programming language that favours tree structures. However GP and its many variations were later used for the design of various engineering problems, such as circuits and mathematical functions.

With the start of automatically generating designs using EAs, the field of EC expanded beyond combinatorial optimisation problems. EC involves the use of EAs on optimisation problems as well as design problems such as control systems and hardware design. The increasing interest in "evolving" hardware designs (particularly digital electronic circuits) in the research community, has lead to the emergence of the sub-field Evolvable HardWare (EHW). The use of EAs in the field of EC to create hardware designs is referred to as EHW. Evolving digital electronic circuits is the most popular and common design problem in EHW, and it is achieved on reconfigurable hardware platforms or more usually using computer simulations.

There can be various reasons for evolving electronic circuits rather than engineering them. Evolution is able to discover circuit topologies that an engineer would never consider as a design option. Therefore evolving circuits allows us to sample design spaces we have never sampled before. One of the most obvious advantages of discovering new circuit topologies is innovation.

2.1 Innovative Circuit Design

Evolution has a unique approach in designing electronic circuits and programs when compared to human designers. Human designers use the engineering approach of top-down, divide and conquer; dividing the problem into smaller sub-problems that are easily understood by the designer, then the designer combines the solutions to these sub-problems using conventional design techniques. Evolution on the other hand uses a bottom-up search, putting/removing components to find partial solutions to the problem, which are then further modified until the final solution is found. Unlike the engineering approach, the partial solutions found to a problem by evolution does not necessarily represent a sub-solution to the problem. The different nature of the design approach gives evolution the ability to sample a different design space that may not be within the reach of traditional design methodologies.

Part of the EHW community is interested in the creation of circuits for discovering innovative designs that are unattainable by conventional design methods. Thompson's work on evolving a tone discriminator in a Field Programmable Gate Array (FPGA) substrate via combinational logic only is a good example of evolution's ability to achieve innovative designs [Thompson, 1996]. Thompson discovered that evolution was able to design a circuit that could differentiate between a 1kHz and 10kHz wave by exploiting the analogue properties of the FPGA. Such a design is impossible when engineering design methods are used, since the components of an FPGA are considered to be strictly digital. Some examples of work towards finding unusual circuits using evolution are [Huelsbergen et al., 1999; Miller and Downing, 2002; Miller et al., 2000; Thompson, 1995a; Thompson et al., 1996; Trefzer, 2006]. A few years after Thompson's work, Linden was able to utilise evolution for the design of a complex antenna and obtain efficient and impressive designs that could not be achieved via traditional antenna design methodologies

[Linden and Altshuler, 1999]. This was a good demonstration of the importance of the choice of hardware design problem as an EHW application. The simple but innovative designs achieved via evolution were not only innovative but they were also good enough to perform as real-world antenna designs [Linden and Altshuler, 1999].

Although the early work on evolving digital circuits showed promise, in the recent years the interest in the evolution of digital circuits had diminished. This is because of the low complexity barrier that evolution encountered using digital components, which means that the digital circuits designed by evolution are too simple to be of any real use or interest.

In 2006 Harding demonstrated a proof of concept by evolving circuits on a liquid crystal substrate [Harding, 2006]; Harding used evolution purely for exploiting the characteristics of a liquid crystal, where there is no known circuit design methodology. Harding successfully evolved simple circuits on the liquid crystal, which is a good example for the ability of evolution in making use of the available substrate in its entirety.

The ability of evolution to create unusual designs is thus one of the main reasons why EHW is so attractive to the research community. As it was mentioned earlier, this ability of evolution is due to evolution's unusual approach to designing hardware. This unusual approach allows evolution to sample different design areas, which may yield to designs that are innovative and/or even fault tolerant.

One of the downsides of evolving circuits to create an unusual designs is that the resulting designs are not portable. Unusual designs such as Thompson's tone discriminator [Thompson, 1996] or Koza's "embryonic" analogue circuits [Koza et al., 1996] most often create configurations that are only valid for the specific piece of hardware used (in the first case) or cannot be implemented or guaranteed to work if implemented in real hardware. The evolution of designs that are not portable is a big disadvantage of evolution of unusual circuits.

2.2 Fault Tolerant Circuit Design

Fault tolerance has probably been the biggest area of research in evolvable and bio-inspired hardware in the last decade. With the increasing amount of hazardous, vital, and remote processes depending heavily on electronic hardware, it has become important that the hardware used is highly fault tolerant and maintenance free. Recent advances in nanotechnology have even brought more reliability issues into the fabrication processes [Jeng et al., 2007], which increased the need for effective fault tolerant designs.

The field of fault tolerance is a valued area with many diverse models. The variety of approaches in the field provide a good selection of methods that can be suited to meet the demands of specific cases. Redundancy is the key to achieving fault tolerance, whether it is via hardware, software, information, or time redundancy. In the special case of electronic hardware devices where failure of a device even for a moment is not acceptable, hardware redundancy is the solution [Lala, 2001].

There are various hardware redundancy techniques; each with major differences in their implementations. The three main hardware redundancy techniques are:

- **Static Redundancy**: Static redundancy works with multiple components that all contribute to the outcome, and the faulty ones are masked via the majority. Hence, in a simple case, there is a single common voter that receives the inputs from the redundant components and outputs the result of the majority vote. An example of this is N-Modular Redundancy (NMR) (Triple Modular Redundancy (TMR) being a special case) [Lala, 2001].

- **Dynamic Redundancy**: In dynamic redundancy only one module contributes to the outcome of the circuit, and when this module fails a fault detection system rules out the faulty module and replaces it with a working version. Hence, there is a single fault detection mechanism that monitors the working module and makes the decision about when to replace a working module (so long as there are spare ones). Dynamic redundancy systems are further divided into two classes: *cold-standby systems* and *hot-standby Systems* depending on the implementation [Lala, 2001].

- **Hybrid Redundancy**: As the name suggests this is when both dynamic and static redundancy is included in a system. An NMR system with spare standby modules to replace the faulty ones would be an example of this type [Lala, 2001].

Fault tolerance via circuit redundancy can be a costly and inefficient approach. Replicating the same circuit N times can be an expensive solution, and a fault may not always be caused by a component (or routing) failure; it could also be the changing environmental conditions which affect the normal behaviour of a healthy circuit. These environmental effects and the fabrication faults cannot be solved via circuit redundancy. Hence if a given design is rendered useless due to changing environmental conditions, all its implementations will fail in those conditions. Thus there is a need for more adaptable techniques that can tackle unforeseen circumstances.

Evolution has been used by researchers to evolve circuits that can sustain various faults [Canham and Tyrrell, 2002; Gwaltney and Ferguson, 2003; Hounsell and Arslan, 2001; Thompson, 1995b; Tyrrell et al., 2001], and it has been proven to be suitable and successful in creating fault tolerant circuits. Evolving circuits that are fault tolerant effectively creates circuits with redundant behaviour, but due to the bottom-up design approach taken by evolution the circuit redundancy can be kept to a much lower level than an engineered design [Gwaltney and Ferguson, 2003]. On top of hardware redundancy, time redundancy can also be achieved by evolution. Evolution can be used to reconfigure a faulty system once the fault is detected, e.g. [Teerakittikul et al., 2009] reconfigures a robot controller using evolution when a fault occurs in the controller. Thus using evolution, a fault tolerant system with hardware redundancy can be accompanied with a time redundancy element that can recover the system in case it is unable to sustain any more faults.

Further techniques have also been investigated to enhance evolution's ability to design fault tolerant circuits. Some of the implemented methods are borrowed from the non-evolutionary fault tolerant systems, and used in conjunction with evolution to obtain more effective fault tolerance mechanisms, e.g [Garvie and Thompson, 2004]. On the other hand, a lot of researchers try to use bio-inspired techniques to create fault tolerant systems. Examples of bio-inspired techniques used for the evolution of robust systems include the modelling of neural networks [Arad and El-Amawy, 1994; Here-

ford and Kuyucu, 2006], multicellular organisation via endocrinology based communication [Greensted and Tyrrell, 2003, 2004], immune systems and embryo development [Bradley et al., 2000; Bradley and Tyrrell, 2002; Canham and Tyrrell, 2003], and multicellular development [Liu et al., 2005; Miller, 2004; Tyrrell and Sun, 2006].

Although fault recovery is the final goal of fault tolerant circuits, fault detection is an important step towards recovery. Most of the fault tolerant systems achieve fault detection via Built-In Self Testing (BIST). BIST mechanisms are easy to implement and does not consume many resources [Davidson, 2005], but the mechanism itself is vulnerable to faults. On the other hand a fault tolerant system that has an emergent fault detection as part of the mechanism can provide a robust solution. In such a case, another link that can fail is eliminated from the chain.

A system may sometimes fail to work not because of existing faults but because of the changing environment. In a dynamic environment a system that works with the environment needs to be able to adapt to changes and respond appropriately. One of the research areas of EC is on real-time autonomous adaptation of systems, which is also applied to adaptive hardware systems in EHW.

2.3 Adaptive Design

Designing real-time systems that do not require human input to function in a changing environment is a challenging task, it requires the system designed to be interactive and adaptive with its environment. Engineering design methodologies in this area are not yet well developed, hence most of the time human intervention is required to adapt an existing system to the changing environment [Fahrmair et al., 2006]. EHW approaches to more adaptive hardware systems can be promising [Spector and Stoffel, 1996; Tufte and Haddow, 2000]. Continuous online evolution of already functioning systems may enable them to adjust to small unpredictable environmental changes quickly.

Evolution has been effectively used to create systems that are able to survive changing environments. Common examples of such systems are control, data compression, and signal processing systems. The most popular test case for control systems has been the wall avoidance in robots [Floreano and Mondada, 1994; Haddow and Tufte, 1999;

Krohling et al., 2003; Thompson, 1995a]. Metta et al. [Metta et al., 1999] demonstrate an adaptive evolutionary system for the control of visually guided reaching. A good example for successful adaptive data compression applications achieved via evolution is the adaptive image compression system evolved by Sakanashi et al [Sakanashi et al., 2001]. Adaptive digital filters [Sundaralingam and Sharman, 1998; Tufte and Haddow, 2000] and analogue filters [Zebulum et al., 2003] have also been successfully shown to evolve on hardware.

In EHW an EA is used to change the configuration of a hardware architecture and behaviour dynamically and autonomously by using the provided resources and environment to achieve the design goals. Two major methods of evolving hardware systems have been established in EHW: extrinsic and intrinsic.

2.4 Extrinsic Evolution

When a hardware design is extrinsically evolved, the complete process of evolution and evaluation is done in simulation. Typically the EA runs in a PC (or multiple PCs) and the generated hardware designs are simulated to determine how "fit" they are. An extrinsic EHW approach is flexible and generic, as the level of abstraction can be determined and conveniently altered by the experimenter, and it provides a portable EHW platform. Extrinsic EHW is also the cheaper and quicker way of setting up a basic EHW experiment, and its results are generally easier to analyse than those of an intrinsic approach. For these reasons extrinsic EHW is quite appealing to researchers as a first step of testing their EHW systems even if their final goal is to use intrinsic EHW. Examples of successful application of extrinsic EHW includes but not limited to digital circuit design by using synthesis tools [Araujo et al., 2003], simple gate level simulations [Koza, 1992; Miller et al., 2000] and function level simulations [Kalganova, 2000b]. Evolution of wire antennas [Linden and Altshuler, 1999], and synthesis of analogue circuits [Mattiussi and Floreano, 2006] are also successful examples of extrinsic hardware evolution.

2.5 Intrinsic Evolution

When the evolved designs are implemented and evaluated on real hardware rather than being simulated to test their fitness, the evolution is referred to as intrinsic. Intrinsic evolution brings many advantages such as, more reliable evolution of hardware systems and greater possibility of finding novel designs; since the substrate properties can be explored as well and the conventional engineering system design constraints can be relaxed. In some cases intrinsic EHW speeds up the evolution process as well since the evaluation time for hardware systems such as circuits is much shorter in hardware than it is in software simulations. Examples of Intrinsic Evolution in literature include most commonly the use of FPGAs: to evolve digital circuits [Hollingworth et al., 2000; Thompson, 1996; Thompson et al., 1996], and robot controllers [Krohling et al., 2003; Thompson, 1995a; Thompson et al., 1996]. The intrinsic evolution of antennas [Linden, 2001], transistor circuits [Trefzer, 2006], and analogue circuits [Zebulum et al., 1998] have also been explored. In the early applications of intrinsic evolution Thompson [Thompson, 1996] was able to evolve an unconventional circuit that made use of the physics of the FPGA substrate, which is not possible to achieve using conventional design techniques or extrinsic evolution of digital circuits.

2.5.1 Hardware for Intrinsic Evolution of Circuits

There are a number of different systems that could be evolved on hardware, such as robot controllers on real robots, filters on Digital Signal Processors, and circuits on a range of available hardware architectures.

EHW started with the evolution of digital circuits, which eventually became the most popular target application. Evolution of digital circuits could be done on various available reconfigurable digital circuit architectures the two current and most popular reconfigurable logic devices being Complex Programmable Logic Device (CPLD) and FPGA [Brown and Rose, 1996]. FPGAs provide a larger amount of logic and rich routing, thus they dominate most of the reconfigurable logic market. Most of the intrinsic EHW experiments target combinational logic designs, and even though CPLDs are meant to be the

preferred design platform for combinational digital circuits, FPGAs still dominate the EHW experiments.

FPGAs are reconfigurable devices that provide Configurable Logic Block (CLB) and configurable routing that connect the CLBs. Each CLB provides simple digital elements (usually a few look-up tables and flip-flops) that can be programmed to perform simple logic functions, which then can be connected together using the configurable routing to create large functional circuits.

Although intrinsic EHW started with the evolution of digital circuits, later the intrinsic evolution of analogue circuits started getting attention as well [Terry et al., 2006; Trefzer, 2006; Zebulum et al., 1998]. Intrinsic analogue hardware evolution is done either using Field Programmable Analogue Array (FPAA) [Hereford and Pruitt, 2004; Terry et al., 2006; Zebulum et al., 1998] or Field Programmable Transistor Array (FPTA) [Gwaltney and Ferguson, 2003; Stoica et al., 2001; Trefzer, 2006]. Both types of devices are similar to FPGAs in the way they work, but instead of having logic blocks, they have Configurable Analogue Block (CAB), at different levels of granularity; FPAAs being more coarse than FPTAs.

These reconfigurable devices are important in EHW, since they render intrinsic evolution of circuits possible. With the reliability issues in the recent advances in nanotechnology [Jeng et al., 2007], intrinsic evolution within reconfigurable devices that have been produced with latest fabrication processes could be used to build circuits that are tolerant to fabrication faults and variability. Evolution has been demonstrated to be effective in designing circuits tolerant to transistor variability [Hilder et al., 2009], but these experiments remain in simulation and can not be guaranteed to work once implemented.

2.6 Challenges of Evolving Hardware

Evolving hardware brings many promising properties; fault tolerance, adaptivity, autonomous reconfiguration. However, there exists some challenges that limit the practical applicability of the evolution of hardware systems, especially digital circuits. The two major issues in EHW that are regarded to be the main bottleneck in many EHW applications are *scalability* and *evolvability*.

2.6.1 Scalability

Over the years of research in EC, the complexity of the evolved designs has not increased significantly. The inability of evolution to find circuits at the desired level of complexity in a reasonable amount of time is a major problem. Scalability has been a problem for the evolution of digital circuits in FPGAs [Haddow and Tufte, 2001; Murakawa et al., 1996; Torresen, 1998], as well as in simulations [Kalganova, 2000a; Koza, 1994; Vassilev and Miller, 2000b; Walker and Miller, 2004]. The ability to achieve higher complexity systems from a smaller system in a reasonable amount of time is referred to as scalability. An example scalable technique would be to design a full adder, and create a 16-bit adder from the knowledge gained from the design of a full adder. In this way, the effort of designing the 16-bit adder is not greatly different from the effort of designing the full adder. If scalable engineering techniques did not exist, the design of many systems that would be considered simple today would have been impossible.

After re-introducing GP in 1990, Koza realised the limited complexity GP could obtain in the evolved designs. Therefore, shortly after his introduction of GP, he suggested the use of Automatically Defined Function (ADF) [Koza, 1994]; a method of gene reuse during evolution. ADFs introduced modularity to GP aiming to speed up evolution and increase the achievable complexity. Koza demonstrated in his work that ADFs increase the evolutionary speed of GP [Koza, 1994]. However even the use of ADFs did not introduce scalability in the evolution of higher complexity systems. A similar modularity was introduced by Walker and Miller [Walker and Miller, 2004], for Cartesian Genetic Programming (CGP) [Miller and Thomson, 2000] to speed up the evolution of more complex problems with CGP, a different form of GP. It was shown that evolution of problems with modular CGP was much faster (20x in some cases), and scaled better for complex problems. The modularity in GP and CGP is done systematically, where a mechanism works in parallel with evolution to create modules from the already existing parts of the evolved system which can be reused by evolution.

Although achieving scalability in EHW is important, it is also important that the desired properties (such as innovation and fault tolerance) of evolution are not lost while doing so, e.g. [Shanthi et al., 2004] uses conventional circuit design knowledge to partition the desired problem before evolving, which constrains evolution to the traditional design

space. One of the obvious reasons for the scalability challenge evolution faces is the direct genotype[4]–phenotype[5] mapping that is present in most EHW systems; this causes the genotype to grow linearly with the phenotype, which creates an exponentially growing design space. This problem has already been addressed in detail by researchers, and biological development inspired approaches has been suggested as ways to introduce scalability to EC in general [Bentley and Kumar, 1999; Dellaert and Beer, 1994; Eggenberger, 1997] and specifically for evolvable hardware [Gordon, 2005; Haddow et al., 2001; Miller and Thomson, 2003; Roggen, 2005].

As listed above, there are many researchers that suggested the use of modularity [Haddow and Tufte, 2001; Kalganova, 2000a; Koza, 1994; Murakawa et al., 1996; Torresen, 1998; Vassilev and Miller, 2000b; Walker and Miller, 2004], and some of them have achieved some improvement by the use of modularity in EC. However, even with an explicitly defined mechanism that incorporates modularity into evolution, the scalability can not be fully achieved so long as there is a direct genotype-phenotype mapping. However, the extensive research and successful results on modular evolution suggests that a mechanism that provides modularity during evolution is desired and more likely to be successful than a one that does not.

Sekanina, in his paper in 2006, mentions that evolutionary algorithms are limited to a search space size of approximately 1000 bits [Sekanina, 2006]. He compares various encoding schemes used for the evolution of electronic circuits with respect to their scalability and innovativeness. Sekanina claims that all these methods have the potential to find innovative circuit designs, however their scaling properties vary greatly. He suggests that developmental approaches have the potential to be infinitely scalable [Sekanina, 2006].

Figure 2.2 shows the increase in the complexity of the circuits evolved vs the increase in the number of transistors per chip in Intel processors over a 14 year period. The biggest circuit that was evolved in year 1992 was a 5-bit parity that is composed of approximately 20 gates [Koza, 1992]. In year 2005 the biggest circuit evolved was a 6-bit multiplier that was formed of 500 gates [Stomeo et al., 2006, 2005], even though traditionally it would be much less. This is a 25 times increase in the number of gates used for the largest evolved circuit in 13 years, which is extremely low when compared to the

[4]Genetic information in a cell that is used to obtain a certain phenotype
[5]The physical form and characteristics of an organism; i.e the circuit.

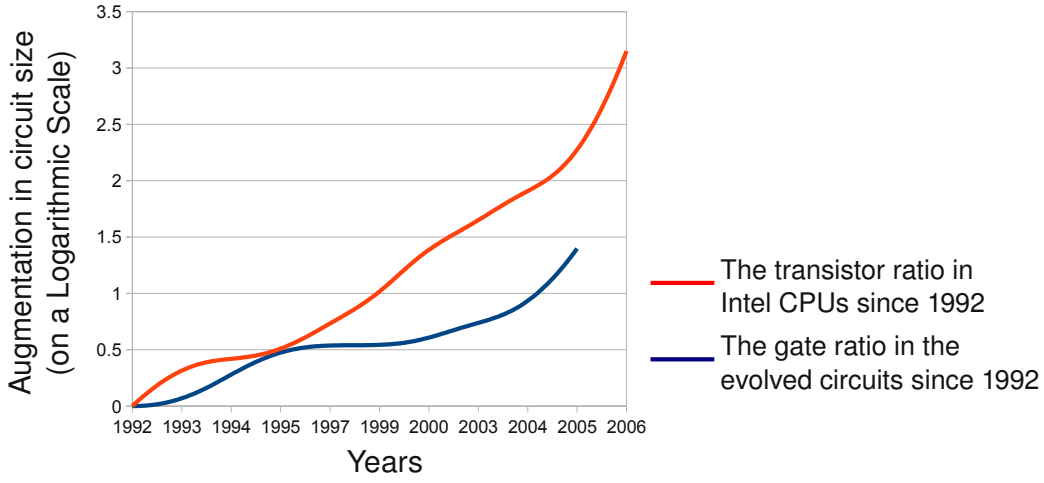


Figure 2.2: *The graph showing the increase in the number of transistors in Intel processors, and the number of gates in the circuits evolved in EHW from 1992 to 2006, on a logarithmic scale. Traditional design techniques advanced much quicker than the evolutionary circuit design in obtaining large circuits.*

increase in transistors per chip on the commercially available CPUs. In 1993 the number of transistors in an Intel Pentium chip was 3.1 million [Alpert and Avnon, 1993] and this number went up to 1.7 billion in the Intel Dual-Core Itanium 2 chips that were released in 2006 [Shiveley, 2006], a near 550 times increase in the number of transistors used per chip over 13 years. Even though this is not strictly a fair comparison, we can see that the increase in the complexity of the human designed circuits could scale up comfortably as the new fabrication processes allowed the scaling up, where as the evolved circuits remained at the similar level of complexity without demonstrating much scalability. On top of this, the graph in Figure 2.2 show that the size of the circuits evolved was not greatly affected by the advancements in CPU technology. Figure 2.2 shows the increase in the number of transistors and gates on the CPUs designed and digital circuits evolved as a ratio to the biggest circuit/CPU in 1992 on a logarithmic scale. A logarithmic scale is used in order to fit the data from both of the sources into one graph, since the increase in the number of transistors in the CPUs designed over the years is much larger than the increase in the number of gates in the evolved digital circuits.

As mentioned earlier there has been considerable amount of work done in evolving larger circuits. In 1994 Koza was able to double the number of gates in his parity circuits by evolving an 11-bit even parity (traditionally ~ 40 basic gates) using ADFs [Koza, 1994].

The number of gates was almost doubled by Thompson's innovative tone discriminator application, which was done in 1995 (approximately 70 gates) [Thompson, 1995a]. Then 8 years later a comparably larger circuit (a 5-bit multiplier) was evolved in 2003 by Torresen [Torresen, 2003]. 2-3 years after Torresen's evolved multiplier, Stomeo et al. managed to evolve a 6 bit multiplier [Stomeo et al., 2006, 2005]. The progress of evolving large circuits have been slow due to the evolvability and scalability issues. In the last few years the interest in the evolution of digital circuits has reduced and researchers have had more success and interest in other EHW applications such as analogue circuits [Gao et al., 2008; Hilder et al., 2009], antenna design [Hornby et al., 2007], printed circuit board tracing [Yasunaga et al., 2008] and micro-electromechanical devices [Hornby et al., 2008]. One of the reasons for the slow progress in the evolution of digital circuits is due to the evolvability of digital circuit problems. Evolvability will be discussed in Section 2.6.2.

The criteria for choosing the evolved circuits used to plot the graph in Figure 2.2 is listed below.

1. The building blocks of the circuits evolved had to be simple logic components, i.e. gates. However, some cases were discarded even though the evolved circuit was theoretically more complex at the time and only logic gates were used to evolve the circuit: the reason for this was the evolution of the popular problem even/odd parity generator circuits with XOR gates. Even though theoretically a 10 bit parity generator is more complex than a 5 bit parity generator, evolving a 10 bit parity using XOR gates is easier than evolving a 5 bit parity using AND/NAND, OR/NOR gates. The fact that the evolved circuit uses XOR gates does not make the evolved design less meaningful. These designs do not represent an increase in the achievable complexity by automated evolutionary mechanisms when compared with the earlier designs that used AND/NAND, OR/NOR gates, but a small improvement in the parity circuit size via the use of traditional engineering knowledge.
2. The circuit had to be evolved from scratch, i.e. not from an already existing design or building blocks.

One of the most impressive and largest circuits evolved up to date but not included in the graph is the prime number predicting circuit evolved by Walker and Miller, which was composed of approximately 400 multiplexers [Walker and Miller, 2007]. Due to the

design evolved by Walker and Miller being an unconventional circuit design with the use of multiplexers instead of basic gates their solution was not graphed in Figure 2.2.

2.6.2 Evolvability

In biology, evolvability has been termed to refer to "the capacity to generate heritable, selectable phenotypic variation" [Kirschner and Gerhart, 1998]. Kirschner and Gerhart explain such capacity as being able to reduce: the potential lethality of mutations and the number of mutations needed to produce phenotypically novel traits [Kirschner and Gerhart, 1998]. In evolutionary biology, evolvability is an important concept as it explains the possibility of meaningful evolutionary adaptation in biological organisms.

Evolvability is also important in evolutionary computation; it refers to the ability of generating fitter offspring from an individual via evolutionary operations [Turney, 1999]. The lack of evolvability may mean that the target design can never be found using evolutionary design. The problem at hand, the phenotypic substrate, and the genotypic representation all determine the evolvability of an evolutionary system.

The low evolvability of genotypic representation in the evolution of circuits is one of the reasons of the limited level of complexity that evolution can achieve when designing circuits. A genotype representation that can easily be manipulated by evolution and a genotype mapping that can provide a smooth design space in the face of operations like mutation and crossover are essential for an evolvable system. A representation and mapping system that is not easily evolvable severely limits the complexity achievable via evolutionary design.

Other important factors for the success of evolution in designing a system depend on the suitability of the design problem, the individual functions/components used to build the target system, and for intrinsic circuit design; the platform used. Evolving systems on well understood problems such as digital design may not return impressive results. Digital circuit design is well understood and has already been mastered by engineering techniques, thus evolving digital circuits that compete with the engineered designs would be a highly challenging task. Problems that are not well understood via the engineering techniques may provide an easier task for evolutionary design.

The building blocks used for the evolution of a system also determine the evolvability of the problem. Asking evolution to come up with designs that are not realistic even for engineers in a well understood design area would not be sensible. For example designing an 8-bit Multiplier from the most basic digital elements without resorting to reuse would not be a realistic task for either engineers or evolution. Therefore it is important to choose an evolvable application domain with an evolvable application in order to achieve success in evolutionary design.

Evolving circuits on hardware presents evolvability issues in terms of both the available components and the suitability of the achievable design problems. The commercially available hardware technologies are specifically designed for traditional top-down circuit design methods. Hence these hardware architectures provide the necessary components and routing for designs that have been mastered by the engineering design techniques. Therefore it is quite likely for evolution to struggle in finding solutions using a bottom-up design approach. Although there is no known definite programmable hardware structure for circuit evolution that favours the bottom up approach of evolution, there has been some work in the research community in developing hardware systems that favour evolutionary method of designing circuits. There have been various suggestions and implementations of evolvable hardware architectures specifically designed for digital circuit evolution [Greensted and Tyrrell, 2007b; Haddow and Tufte, 2000], controller evolution [Kajitani et al., 1999], analogue circuit evolution [Langeheine et al., 2001; Stoica et al., 2001], and evolution of biologically inspired systems [Samie et al., 2009; Tyrrell et al., 2003]. These hardware systems aim to provide a platform for modelling certain biologically inspired processes easily, and/or provide evolution with the freedom to discover designs in the unconventional design space.

Earlier work done in EHW used off the shelf programmable devices such as FPGAs (see Figure 2.3) for evolving circuits [Higuchi, 1994; Thompson, 1995a], since these devices were the only available digital hardware platforms most suitable for evolution. Commercially available FPGAs are still popular with EHW experiments [Lambert et al., 2006]; they are highly reconfigurable, and provide a large amount of programmable digital logic components (both combinational and sequential). Unfortunately, FPGAs are designed to be used as part of an intelligent design process; the complex routing and logic components on an FPGA are designed to provide the optimal implementation

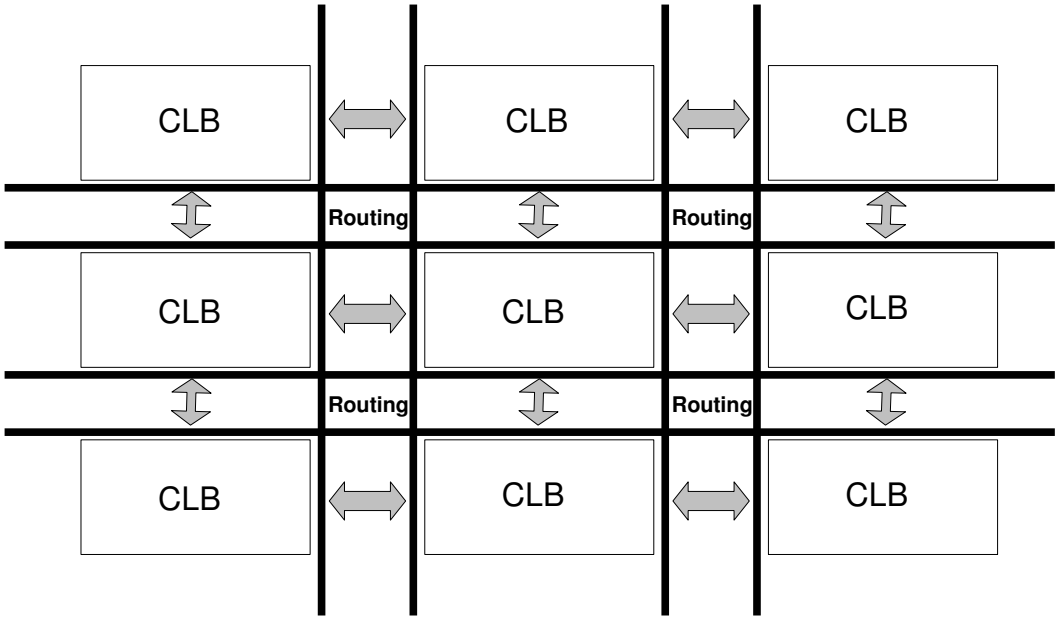


Figure 2.3: *A simplified FPGA architecture with CLBs and a routing layer. Generally, the CLBs are able to communicate with their nearest neighbours via local connections or via an extra routing layer (which usually consists of an array of switch blocks for long distance communication as well).*

for traditional designs, and if some of the constraints of the traditional digital design are ignored, it is possible to damage an FPGA. For example, if the bit string that programs an FPGA is altered in an uncontrolled manner it is possible to create invalid configurations that will damage the hardware. The only commercially available FPGA that allowed unconstrained configuration possibilities was the Xilinx XC6200 (no longer available in the market) [Xil, 1997]. XC6200 had a simpler routing scheme than the newer generation FPGAs, which limited the capabilities of the FPGA.

Haddow and Tufte discuss the shortcomings of the commercially available digital hardware devices and suggest a new design that would favour evolution and adaptive systems [Haddow and Tufte, 2000]. Their design is a simpler FPGA architecture with unlimited and unconstrained reconfigurability, simpler routing, finer grained logic blocks, and self and partial reconfiguration of the chip. These are some of the properties lacking in most of the commercially available reconfigurable devices for EHW applications, and many other researchers have tried to tackle these problems by developing their own

custom reconfigurable platforms. Some of the platforms developed by researchers are designed on existing FPGAs, which mask the underlying architecture to provide a more evolutionary friendly platform. Examples of this type of "virtual" platforms are [Haddow and Tufte, 2001; Sekanina, 2003]. Although this approach is easy to implement and it makes use of widely available commercial devices, it is not an efficient way of using the underlying substrate. Another approach to implementing an evolution friendly reconfigurable platform is to design and manufacture a custom Application Specific Integrated Circuit (ASIC). This is a less flexible, more costly and time consuming approach, however it gives the designer greater design freedom for a more evolvable platform and makes best use of the underlying silicon. The examples for this type of platform which were suggested as a replacement for commercial FPGAs include "RISA" [Greensted and Tyrrell, 2007b] and POEtic [Tyrrell et al., 2003] chips.

Considering the literature, the most powerful evolved digital circuits were evolved extrinsically in simulations [Stomeo et al., 2006; Walker and Miller, 2007]. Extrinsic evolution of circuits has the advantage of having a much higher variety of components and a much more flexible routing than intrinsic evolution. On top of that extrinsic evolution does not suffer from unwanted transient effects of unconstrained intrinsic evolution that may occur from looping connections and sequential components. But there is still an attraction to the intrinsic evolution of circuits; evolution of circuits on real hardware has the potential to use the properties of the underlying hardware in an unusual manner to its advantage, which may provide us with novel designs and provide further insight on the capabilities of evolution and the underlying substrate [Harding, 2006; Thompson, 1996].

In Chapter 3 an FPGA platform designed for EHW applications will be introduced and used for evolving digital circuits. This platform will be used to address the numerous issues of unconstrained evolution of digital circuits on hardware. A set of techniques will be introduced for quick and successful evolution of reliable digital circuits on hardware. Once an effective method of evolving circuits on hardware is accomplished, the complexity and characteristic of the evolved circuits will be discussed, and suggestions to evolving more scalable systems will be presented.

2.7 Summary

A brief introduction to evolutionary algorithms and their use in the design of electronic circuits was presented in this chapter. Evolution of circuits is an unconventional method of designing electronic circuits. Therefore, it provides a way to achieve designs that are novel and innovative in the electronics field. Evolution of circuits can be a way of achieving adaptive and fault tolerant circuits as well, hence making EHW an attractive research area.

The ability to evolve a design makes that an evolvable design. Lack of evolvability can be a serious set back in EHW. The target circuit, the medium of evolution and various other factors (such as the fitness function) determine the evolvability of the design for the problem at hand. Digital circuit domain is a tough area to evolve impressive designs in. This is because the digital circuit domain has already been mastered by engineering design techniques and available reconfigurable circuit platforms provide mediums that are not evolution friendly. Thus, evolving competitive digital circuits is a tough task. However the latter statement is not only due to evolvability issues, but also to scalability problems encountered in EC. Since evolution uses a bottom up design method, when there is one to one mapping of genotype to the phenotype, the growing phenotype (circuit) size correlates to an exponential increase in the evolutionary search space.

Publications I

The work presented in the following chapter (Chapter 3) is not solely the work of the author of this thesis, but a joint work with Dr. Martin A. Trefzer. The work presented in the following chapter has been published in a number of conferences as listed below:

Kuyucu, T.; Trefzer, M.; Greensted, A.; Miller, J. & Tyrrell, A. Fitness Functions for the Unconstrained Evolution of Digital Circuits. *9th IEEE Congress on Evolutionary Computation (CEC08)*, 2008, 2589-2596.0

Trefzer, M.; Kuyucu, T.; Greensted, A.; Miller, J. F. & Tyrrell, A. M. The Input Pattern Order Problem: Evolution of Combinatorial and Sequential Circuits in Hardware. *The 8th International Conference on Evolvable Systems: From Biology to Hardware*, 2008.

Trefzer, M.; Kuyucu, T.; Miller, J. F. & Tyrrell, A. M. The Input Pattern Order Problem II: Evolution of Multiple-Output Circuits in Hardware. *IEEE Symposium Series on Computational Intelligence - IEEE SSCI 2009*, 2009.

Kuyucu, T.; Trefzer, M.; Miller, J. F. & Tyrrell, A. M. Task Decomposition and Evolvability in Intrinsic Evolvable Hardware. *IEEE Congress on Evolutionary Computation*, 2009.

Chapter 3

Evolving Circuits in Hardware

This chapter presents a hardware evolution platform designed to allow the unconstrained evolution of digital circuits. An introduction to the evolution platform is followed by investigations into techniques that assist the evolvability of Evolvable HardWare (EHW). These investigations include evolutionary experiments that assess the capabilities of the hardware platform, and new techniques are subsequently developed to enhance the evolvability of hardware evolution systems in general. Section 3.1 describes the evolution platform used, and sections 3.2 and 3.4 discusses important issues that ensure circuit validity when evolving circuits on hardware. Section 3.5 presents simple methods to enhance evolution to achieve valid designs. Section 3.6 presents circuit designs achieved using the hardware evolution platform and investigates the progress made to ensure the evolvability of the platform. Finally, a summary on the progress made and conclusions are presented in Section 3.7.

3.1 Reconfigurable Integrated System Array (RISA)

RISA is a reconfigurable Field Programmable Gate Array (FPGA) device with an embedded on-chip microprocessor. RISA was specifically designed as a platform for intrinsic hardware evolution at the Department of Electronics, University of York [Greensted and Tyrrell, 2007a,b].

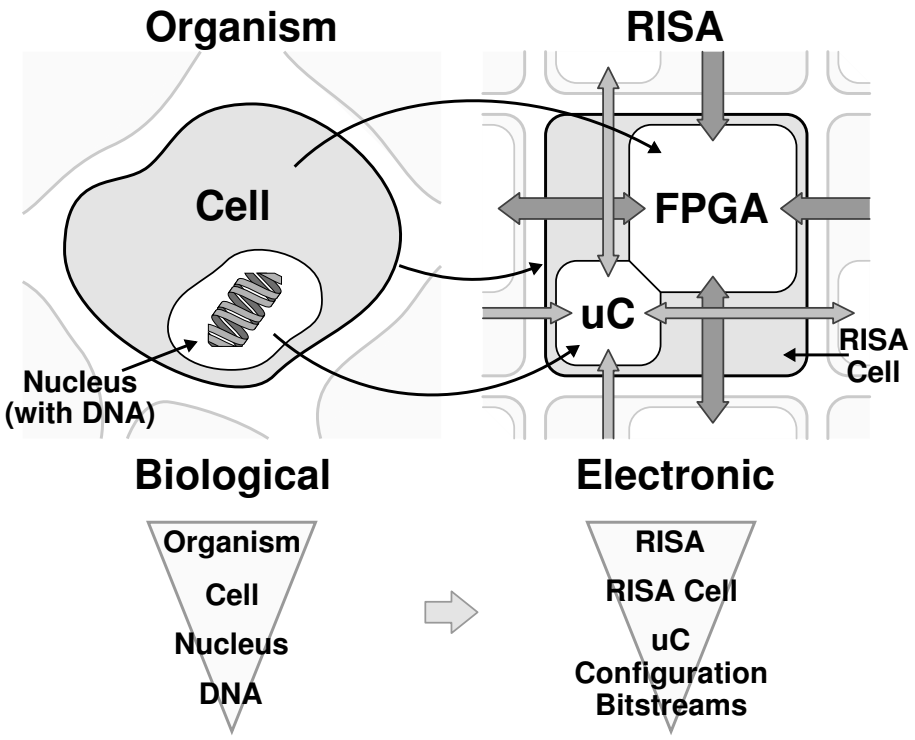


Figure 3.1: *The structure of the RISA cell is inspired by biological cells. The micro-controller operates as a centre for cell operations, controlling the cell functionality implemented in the FPGA fabric. FPGA fabric configuration bits may be stored and manipulated in the micro-controller [Greensted and Tyrrell, 2007a,b].*

One RISA chip provides both a programmable micro-controller and a configurable logic substrate, which are inspired by the main constituents of biological cells, namely the nucleus and the cell body, as shown in Figure 3.1. The custom designed micro-controller on RISA is called Simple Networked Application Processor (SNAP). Taking inspiration from the cell nucleus, SNAP stores and processes configuration data and is able to (re-)configure the FPGA substrate at runtime, i.e. without interfering with the currently running circuit configuration. SNAP is a Reduced Instruction Set Computer (RISC) and its instruction set is specifically tailored for Evolutionary Computation (EC). Furthermore, it provides communication interfaces to other RISA modules, as well as to the outside world.

The RISA chip is physically small in size, thus making it useful for various practical projects, e.g robotics and sensor networks. But the available microprocessor and FPGA fabric on the chip is considerably smaller in size when compared with commercial FPGAs with on-chip microprocessors.

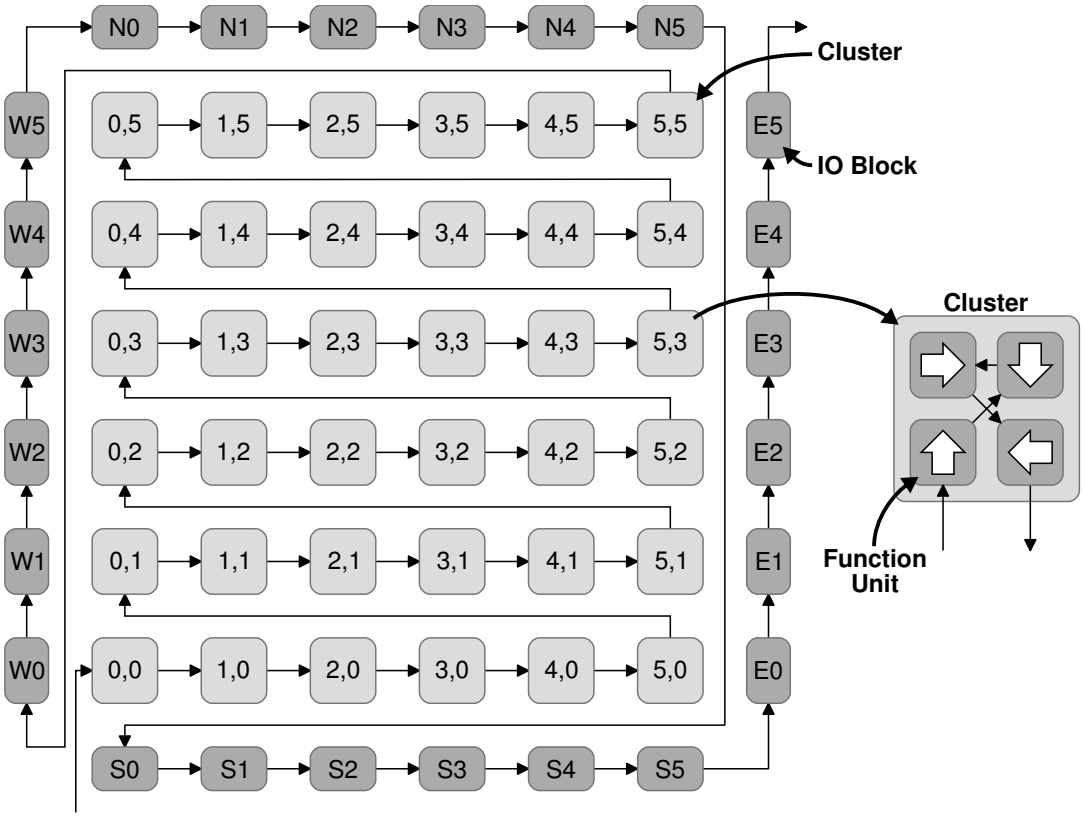


Figure 3.2: *The FPGA substrate of RISA consists of an array of 36 functional clusters surrounded by IO blocks. The configuration chain for the clusters and the IO blocks are connected serially, but each cluster and IO block can be configured individually, providing partial reconfiguration. To configure a single cluster 152 bits are required for the logic and 320 bits are required for the routing; resulting in a total of 16992 bits for the whole 36 cluster configuration bit-string. Each cluster features four directional function units (north, east, south, west), and each function unit provides a 16-bit LUT that can also be configured as a shift register or RAM. Each cluster also provides configurable routing among the four function units, and outside to the other clusters [Greensted and Tyrrell, 2007a,b]. See Appendix B for the schematics of cluster and IO routing in RISA.*

The configurable logic of RISA is designed in a similar fashion to generic FPGAs. As can be seen from Figure 3.2, the configurable fabric consists of an array of 6×6 Configurable Logic Block (CLB)s (referred to as clusters), surrounded by IO blocks. The IO blocks provide a total of 12 IO connections at each side of the RISA module (each block providing 2 IOs), which can be independently configured as either an input or an output of the FPGA, but not as both. Additionally, configurable logic of different RISA chips can be directly interconnected in order to build larger circuits, while the SNAP of different RISA modules can communicate to form a network of processors.

The FPGA substrate of RISA cannot be destroyed by random configuration bit strings, allowing unconstrained evolution. As discussed in Chapter 2, this feature is not present

in current commercial FPGAs: the access to the bit-string for configuring device is either constrained to be only modified by the manufacturer's synthesis tools in order to protect the device, or it is actually possible to damage the device by invalid configurations. The configuration of clusters in RISA FPGA can be changed independently from each other, hence offering partial reconfiguration. Partial reconfiguration can considerably accelerate hardware evolution [Hollingworth et al., 2000], since only the parts of the bit-string that have actually been changed by the Evolutionary Algorithm (EA) need to be reloaded into the device instead of reconfiguring the entire device.

The RISA was designed to provide an evolution friendly FPGA substrate, and it has a variety of properties that make intrinsic evolution of circuits on chip convenient. However RISA's FPGA substrate lacks a flexible routing scheme, and the number of available IOs for the FPGA substrate is very small. A maximum of 12 inputs can be used to access one section of the RISA chip, which only allows the addressing of a maximum of 3 16 bit LUT, when the number of available LUT is much higher (See Appendix B).

3.1.1 Experimental Setup

The essential components of the system used for the experiments reported in this chapter are a RISA chip, a Xilinx Spartan 3 FPGA, and a PC. The Spartan chip is interfaced to the RISA chip, and it runs the EA which evolves the circuits for the FPGA substrate on RISA. The Spartan chip is instantiated with a soft-core Microblaze processor, which runs the EA with RISA chip in the loop. The Spartan chip is also connected to a PC via a serial interface for storing the evolutionary progress during evolution on a PC as a text file.

The code for the EA is written in C programming language, and the evolution is done on the configuration bit-string of the RISA. With each evaluation step, a candidate solution is loaded into RISA and tested by a set of data in accordance with the fitness function.

The experiments presented in this chapter are evolution of circuits on the unconstrained FPGA substrate of the RISA platform. Experiments that are marked "constrained" are done on a constrained version of the FPGA substrate of the RISA platform. In the constrained RISA platform the size of configuration bits per cluster are reduced from 472 to 58 bits. Although the possible configuration options in the constrained version of the

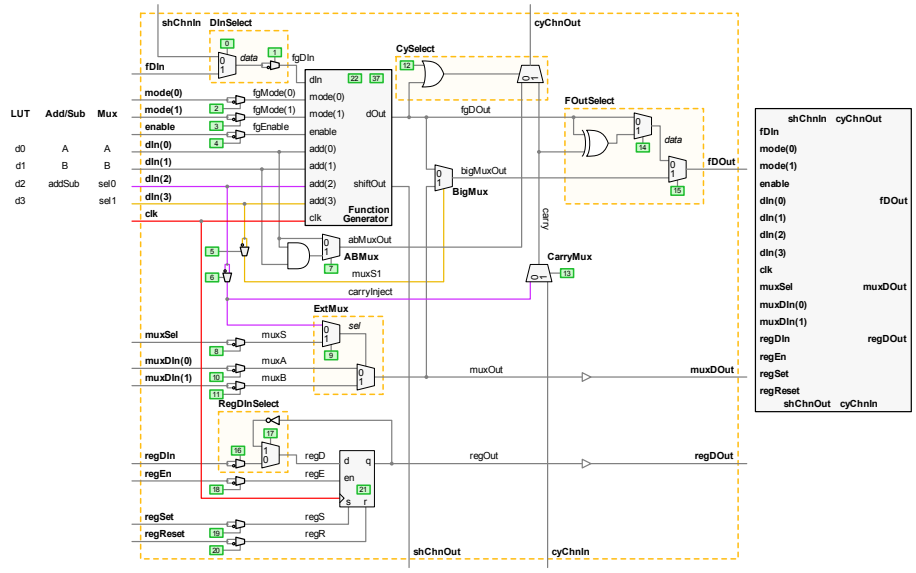


Figure 3.3: *The schematic of a function unit (four in each cluster) in the RISA FPGA.*

RISA platform are greatly limited, the local and inter-cluster feedback loops as well as delays are still possible. The large reduction in the configuration bits per cluster refer to the limited programmability of the routing, especially between the four functional units present in each cluster. Figure 3.3 illustrates the components available in a single RISA function unit. The constrained case allows the signals to be routed to the "Function Generator", which can only be used in a single mode (as a LUT), or the flip flop. Other components and routing lines are disabled in the constrained case.

A total of 10 clusters of the RISA chip is used for the evolution of all the experiments presented in this chapter unless stated otherwise. The remaining 26 clusters are configured in a way that they pass incoming signals unchanged to their opposite side. This ensures that the circuit's output reaches the IO blocks and can be measured from outside the chip. Inputs are applied to the west side of the chip and the output is measured at the east side.

RISA is operated at a frequency of $4MHz$ and the inputs and outputs are sampled with a frequency of $0.5MHz$. The sampling frequency was chosen as a consequence of delays in input and output buffers of IO blocks and the expected delay of the candidate circuits. The input and output data consist of 512 samples for each measuring cycle. A $(2+5)$ Evolution Strategy (ES) is used (i.e. no cross-over), with a fitness proportional mutation strength of $1\%..10\%$ (never to be less than 1 bits per generation), that changes with respect

to the *rate of change* in the fitness, see Algorithm 1. 20 randomly initialised evolution runs have been carried out for all experiments and the generation limit is 5000.

The selection scheme used is a simple "best selection". Out of a population of 7 the two best candidates are chosen. Offspring are preferred over parents in case of candidates with best fitness, and in such cases with multiple offspring with same fitness, the selection is done at random. From the two best the rest of the population is created by mutating the two, 3 mutants are created from the first choice, and 2 mutants are created from the second choice.

The number of data samples used for each cycle, and the population size were chosen for the values stated above due to the memory limitations on the hardware architecture used in the experiments. The generation limit, lack of cross-over and the number of evolutionary runs were also chosen with the considerations of the speed of the hardware

Algorithm 1 The pseudo-code for the variable mutation rate. This is part of a function that is called once every evolutionary generation.

comment: At the start of evolution convergence is set to 1 and Mutation Rate is set to 0.5%
if one of the top two individuals from the previous evolutionary generation is now the best individual **then**
    $convergence--$;
else
    $convergence++$;
end if
if $convergence = 0$ **then**
    comment: Mutated individuals are all worse; decrease the mutation rate
    if $MutationRate > 0$ **then**
        $MutationRate--$;
    end if
    $convergence \leftarrow 1$;
else if $convergence > 0$ **then**
    comment: Mutation did not hurt; increase the mutation rate
    $MutationRate++$;
    $convergence \leftarrow 1$;
end if
if $MutationRate < 0.5\%$ **then**
    $MutationRate \leftarrow 0.5\%$;
else if $MutationRate > 10\%$ **then**
    $MutationRate \leftarrow 10\%$;
end if

platform used and the time required to evolve a design for the target circuit. The range of the mutation rate was manually determined, and the variable mutation rate function was specifically designed to be simple due to the limited computational resources available on the experimental hardware used.

3.2 Getting Acquainted with Evolution in Hardware

Hardware evolution has various aspects that are not addressed in software evolution. In particular, intrinsic hardware evolution demonstrates various types of behaviours that are not recognised by simple fitness evaluations. Transient effects are encountered in hardware, but do not exist in software evolution [Harding, 2006; Thompson et al., 1996]. It has been suggested that in order to get the most out of intrinsic hardware evolution and evolve complex and novel systems, an evolvable platform capable of implementing complex behaviours together with an unconstrained evolutionary approach is required [Huelsbergen et al., 1999; Thompson et al., 1996].

Unconstrained intrinsic hardware evolution is particularly good at exploiting the environmental conditions, and even though this is considered a big advantage for intrinsic evolution, it also brings extra challenges regarding the evolution of fully functioning circuits. When allowed to interact with its environment in a rich substrate, evolution is capable of finding various solutions that can satisfy its goal, and often these solutions are valid only for the exact environment presented during evolution, even though the desired result is aimed to satisfy a much broader problem.

The initial experiments with RISA included the evolution of a simple XOR gate. However due to several challenges, the evolution was unsuccessful. Since evolution was unconstrained, feedback loops could be generated by evolution on RISA. These feedback loops would create so called transient effects [Harding, 2006; Thompson et al., 1996]. When unconstrained, evolution explores any solution that is available in the medium, and it can sometimes find solutions that are only transient, this can trick the fitness function into believing that it has found the correct solution. In the initial experiments this was found to be the case, and the simple fitness function used in evaluating the resulting circuits was unable to distinguish the transient results. The fitness function that was

used would simply award/penalise the circuit evolved by making bitwise comparisons on the evolved circuit's current and desired outputs. Thus in most of the experiments performed, the evolution would either fail to find a solution under the maximum number of allowed generations, or find a solution that would behave like an XOR gate only in certain cases but not always.

To be able to guide evolution towards more robust and meaningful solutions without reducing its evolvability, effective yet computationally inexpensive mechanisms are required. These mechanisms are needed to ensure that the evolved circuits are fully functional, and they meet the minimum design requirements. In the rest of this chapter, mechanisms for unconstrained intrinsic hardware evolution that can help guide evolution better are explored.

3.3 Constrained vs Unconstrained Evolution in Hardware

Truly unconstrained evolution on an FPGA substrate is not strictly possible since an FPGA is already constrained by its design. A truly unconstrained hardware evolution would use the substrate to its fullest without any human constraints, i.e no gates or transistors. However building complex systems using such a model can be impractical for real applications, the use of such an approach is interesting for research purposes to investigate the behaviour of evolution [Harding, 2006], or may be to discover a new alternative to the current technology of transistors.

In the field of EHW, "unconstrained evolution" has been used in a few different ways one of which refers to the evolution of circuits without any design constraints. Thompson uses the term "unconstrained evolution" to refer to the evolution of digital circuits with relaxed design constraints, such as having no strict synchronisation of a sequential circuit to a global clock [Thompson et al., 1996]. Unconstrained evolution is used to refer to evolution of circuits that can use the properties of the digital substrate to its fullest while evolving circuits. Thus by "unconstrained evolution" Thompson refers to the lack of traditional design constraints for circuit evolution rather than meaning a substrate com-

pletely free of any constraints. However evolving circuits without any traditional design constraints on an FPGA substrate will enlarge the design space greatly with designs that are mostly unwanted and designs that can create local optima.

It is also worth explaining what is meant by "constrained evolution" in evolvable hardware in this thesis. The term refers to the application of domain knowledge to provide an evolvable hardware platform that has the essential components to aid the successful evolution of a design. Such an approach usually decreases the time required for evolutionary search in finding a functioning system, but it may limit the number of available designs and possibly prevent the emergence of novel designs. "Constrained evolution" however, should not be confused with Constrained Optimisation Problems (COP), which refer to the challenging task of locating a very specific solution of small feasibility in a very large search space. The "constrained evolution" here is more like "guided evolution", where the user provides a large amount of guidance to the evolutionary search using his/her engineering knowledge in order to speed up the evolutionary design.

Consequently one of the key questions when using intrinsic hardware evolution is whether unconstrained evolution is worth undertaking, or would it simply hamper the evolutionary design? In the rest of this chapter the intrinsic evolution of circuits on hardware is investigated, and the performance of evolution will be improved to allow the evolution of meaningful circuits via unconstrained evolution. The experiments involving unconstrained and constrained evolution of circuits will be presented for the purpose of exploring whether unconstrained evolution is worth undertaking.

3.4 Evolving Valid Circuits on Hardware

For the evolution of digital circuits correct operation of the evolved circuits is crucial. It is important that the evolved circuits provide a fully functional device that meets the design specifications. If a digital circuit does not function exactly the way it is supposed to, then most of the time that circuit is considered useless.

As stated earlier, evolution often finds specific solutions that are only valid for exactly the pattern of inputs and environment that are presented during evolution. Unfortunately, in the case of digital circuits the evolved circuits are likely to be only valid for exactly

the set of inputs and the environment that are presented during evolution. Even when certain parameters are varied, e.g. the location on the substrate where the candidates are tested or the order of the input vectors, evolution can produce circuits that only meet these minimal requirements. In the worst case, an evolved circuit can be just a pattern generator that always generates the same set of outputs irrespective of the applied inputs. As a consequence, the resulting circuits are not fully functional. This is due to the rich options hardware substrates provide in terms of components, and evolution may find it easier to find a "cheat" that appears to produce a valid circuit.

3.4.1 Hardware Sampling

Although this is a topic that does not receive much attention, if provided the opportunity, evolution can find a way to present solutions that only work at the output sampling rate used during evolution. In research on the evolution of combinational circuits, output sampling rate uses have received little, if any, attention; it is mostly chosen to be slower than the worst case settling time of the circuit. Using a sampling rate that is too slow may cause evolution to miss changing outputs and evaluate circuit incorrectly. Sampling at the rate of output change in the evaluated circuit ensures that every output is received and evaluated by the evolutionary system. This gives a more precise evaluation of the candidate circuits during evolution.

In the hardware experiments presented in this chapter, the sampling of the outputs of the evolved circuit is always set at a "cycle accurate" fashion (i.e. the outputs are always measured the instant they change without any delay) unless specified otherwise.

3.4.2 Randomness of the Input Pattern

There are only a few examples where input pattern problems are discussed, e.g. [Imamura et al., 2000]. Usually the work done is either related to validation and Built-In Self Testing (BIST) [Corno et al., 1996; Skobtsov et al., 2004] or sets the focus on the fitness function [Torresen, 2002] rather than the input pattern.

Since the structure and behaviour of an evolving circuit are unknown, EAs work on a black box problem [Imamura et al., 2000]. Thus it is necessary to include randomness in

both inputs and environment of the evolving circuit. The most effective test to assess whether a resulting circuit is sufficiently fit to cope with previously unknown input vectors is to measure its output multiple times while applying random test patterns. The success rate of the latter measurement provides a measure for the validity of the evolved circuit. When a circuit is being tested, randomising the input pattern avoids an incorrect assessment of the circuit by removing the chance of creating a valid test case for circuits that rely on a regular pattern of inputs.

Applying randomness in the order of applied input patterns is possibly one of the most crucial things that have to be ensured when setting up evolution experiments that depend on those patterns. If static or periodic input patterns are used in the assessment of solutions for circuits in hardware, it is highly likely that evolution finds circuits that produce the desired output, which might not be correlated to the input and will therefore generally fail for random test patterns. Thus creating a new random input pattern for each generation prevents evolution from exploiting regularities in the input pattern and drives evolution to find the acceptable solutions to a given task. This in turn should also help to keep the EA away from local optima caused by static input patterns.

In order to investigate the effect of including randomness in the input pattern, three different methods of organising the input pattern are used for the experiments in this section: first, the *ordered input pattern*, where the input pattern is fixed during evolution and the samples are ordered according to the truth table of the logic function. Second, the *static random input pattern*, where the input pattern contains all entries of the truth table of a logic function in random order. Third, *random input patterns*, which are newly created for each generation in order to prevent evolution from exploiting regularities in the input pattern. Therefore, evolution is driven to find more general solutions and is kept away from locally optimal solutions caused by static input patterns. The experiments carried out involve evolving circuits for a combinational 4 bit parity generator and a tone discriminator. The task for the tone discriminator circuit is to distinguish between a $31.25kHz$ and a $250kHz$ tone. As stated earlier, the input and output patterns consist of 512 samples for each measuring cycle. This allows the full truth table of 4-bit parity circuit to be applied 32 times, and the slowest tone discriminator circuit $31.25kHz$ can be applied for 4 frequency samples over each measuring cycle.

Table 3.1: *The results for the tone discriminator and 4 bit parity circuit experiments are shown. The number of runs where a perfect solution was found out of 20 runs is given in column entitled "Solution Found". The number of the circuits, which are successfully reloaded to the chip and tested with multiple (20) random input patterns is given in column entitled "Random Pattern". Solutions that also pass the test when measured at a different location on the chip are given in column entitled "Different Location".*

Target	Genome	Input Pattern	Solution Found	Random Pattern	Different Location
Tone Disc.	Constr.	Ordered	8	0	0
Tone Disc.	Constr.	Static Rand.	0	0	0
Tone Disc.	Constr.	Random	6	4	4
Tone Disc.	Unconstr.	Ordered	0	0	0
Tone Disc.	Unconstr.	Static Rand.	0	0	0
Tone Disc.	Unconstr.	Random	7	0	0
4 bit parity	Constr.	Ordered	10	0	0
4 bit parity	Constr.	Static Rand.	9	5	5
4 bit parity	Constr.	Random	7	6	6
4 bit parity	Unconstr.	Ordered	0	0	0
4 bit parity	Unconstr.	Static Rand.	1	0	0
4 bit parity	Unconstr.	Random	2	1	1

3.4.3 Testing the Evolved Circuits for Validity

Three different tests are carried out in order to assess the evolved circuits: first, the success rate for random test patterns is measured. Second, different sampling frequencies are tested and the frequency discrimination range of the tone discriminator is determined. Finally the candidate circuits are measured at different locations on the chip. A valid circuit is expected to be independent of the particular location of the chip it has been evolved on. Hence, the presented evolved circuits are tested at different locations of the chip, whenever it was possible.

In the constrained evolution experiments (see Section 3.1.1 for the difference in constrained vs. unconstrained experiment on RISA hardware platform), using *ordered input patterns* during evolution produced invalid solutions that failed the random inputs test as well as the testing of the circuit at a different part of the chip for both of the target circuits, see Table 3.1. Using *static random input patterns* was sufficient for the successful evolution

of fully working 4-bit parity circuits but they were still ineffective for the sequential circuit problem, the tone discriminator. In both constrained and unconstrained versions of the RISA chip using full randomised input patterns provided the best and most reliable evolutionary performance in evolving all the circuit problems. Furthermore, the ability of the resulting tone discriminator circuits to distinguish other pairs of frequencies than those demanded during evolution has been tested. The evolved circuits are able to correctly distinguish different frequencies with a typical range for the lower frequency at $0..62.5kHz$, and a typical range for the higher frequency at $125..250kHz$.

3.5 Tricks and Treats

One of the aims of EHW is to facilitate and automate the design process for increasingly complex applications. In order to achieve this, effective techniques are required to efficiently use the given substrate and solve the problem at hand as quickly as possible. Evolution of circuits on hardware is an arduous task with many obstacles, and if not taken care of, these obstacles could make the evolution of the simplest circuit impossible. Four simple "tricks" are introduced in this section with supporting empirical evidence showing their positive effect (the "treats"–reliable and quick evolution of circuits) on the evolution of digital circuits on hardware.

It is shown that the newly developed approaches can be used to cope with four important issues (some of which are commonly neglected) in the intrinsic evolution of digital circuits: computational ambiguities, transient effects, scalability, and rough fitness landscapes.

3.5.1 Fitness Functions

The design of the fitness function is one of the most important aspects of an EA whether it be for optimisation purposes, extrinsic or intrinsic hardware design. The fitness function is the specification of what is expected from the solution. In the case of extrinsic evolution most of the time the fitness function only needs to specify the desired outputs for all possible inputs. For a simple circuit this can be the truth table being applied to the test

circuit and the circuit output being compared to the desired output bit by bit. However for intrinsic evolution the circuit being evolved may require more information for an effective guidance.

New fitness functions are introduced to help evolution tackle transient effects in hardware evolution. Feedback loops in evolution of circuits are most often not allowed, even though they may yield interesting results. In this case, the term feedback refers to circuits that feature the ability to oscillate rather than being merely finite sequential circuits: examples for this can be found in [Huelsbergen et al., 1999; Thompson et al., 1996]. Obtaining feedback paths in circuits evolved on RISA is possible, and as a consequence of this the additional challenge for the EA is to find a static solution in a transient dynamic system. If a suitable fitness evaluation method is used that is able to accurately assess and control transient effects, it can speed up the evolution of circuits in hardware.

However, a further question is whether the fitness measuring method has an effect on the ability of an evolved circuit to cope with random, unknown input test vectors. In particular when random input patterns are applied during evolution, a desired property of the fitness function is not to immediately assign candidate circuits that perform poorly in a single case. At the same time, it is not supposed to promote bad solutions when they perform well by chance.

Three new fitness evaluation methods are listed, explained and further investigated in this subsection in addition to the classic bitwise fitness calculation. The first fitness evaluation method that was developed is a different version of the simple bitwise fitness calculation, it is intended to deal with the transient effects of hardware; it is called Bitwise Fitness Modified for Hardware (BMH). The second fitness evaluation method is the Hierarchical IF-and-only-iF (HIFF) method described in [Watson et al., 1998]. The final fitness evaluation method is a customised version of the HIFF. It is based on sampling the bit-string by evaluating blocks of bits of variable size and is therefore referred to as Hierarchical Bit-string Sampling (HBS).

- **Bitwise Fitness Calculation**: This is the classic way of measuring a candidate circuit's fitness. It's simply calculated as the Hamming distance between the measured output and the desired output. Thus every incorrect bit is penalised for one fitness score.

- **BMH**: BMH fitness evaluation method is specifically developed for combinational hardware evolution on unconstrained hardware where feedback loops are allowed. It is built around the simple bitwise comparison between measured and desired outputs, but it also undertakes a parity check and a check for transient faults: thus, the complete logic input pattern has to be iterated through the solution at least twice for every evaluation.

  To overcome the problem of being deceived by intermittent solutions, the BMH fitness evaluation method penalises transient behaviour by comparing the outputs of a given input combination at two different time steps for a change in the corresponding outputs: i.e. on the same circuit evolved, for test case X_N if the output of input A at time t_n is different to its output at time t_{n+1}, the fitness is penalised by an appropriate value. This is a simple yet effective way of directing evolution away from the solutions that provide intermittent results.

  One of the other common problems in hardware evolution is the unconnected or stuck-at output, that always provides a '0' or '1', no matter what the inputs are. In bitwise fitness calculation, this often gives a 50% (for an XOR gate or parity circuit) or higher success in the fitness value, thus possibly trapping evolution in local optima. To overcome this problem BMH fitness evaluation method awards bit variance to push evolution away from the solutions where outputs are all '0's or all '1's and push it towards better solutions; see Figure 3.4 for an example.

- **HIFF**: The HIFF method was proposed for hierarchical if-and-only-if problems in [Watson et al., 1998]. It extends the bitwise fitness calculation (Hamming distance) by introducing additional steps of fitness calculation, where blocks of bits of increasing size are compared, and the penalty is increased proportionally to the block size of the respective step. As a consequence of this, larger schema within the bitstring are recognised, and accordingly penalised (or rewarded) and are therefore preserved in case they match the desired output.

- **HBS**: In order to benefit from the context sensitivity of the HIFF method and at the same time further increase the granularity of the fitness value range, the HBS method is proposed. Rather than dividing the bit string into disjunct blocks of increasing size, the bit string is evaluated by sampling it with overlapping windows (blocks) of increasing size and adding the resulting penalties. It is also suggested

Table 3.2: *Ambiguities in fitness assignment for an XOR are shown. The goal is to minimise the fitness. There are two inputs, namely A:**0011** and B:**0101**. The desired output is the result of an XOR:**0110**. Four different fitness measures, Bitwise, BMH, HIFF and HBS are compared . The values in the parentheses used for fitness calculation refer to; for BMH: bitwise, parity, transient fault penalty, for HIFF: binary block sizes 1,2,4, and for HBS: binary block sizes 1,2,4.*

Case	Measuring at		Fitness			
	t_0	t_1	Bitwise	BMH	HIFF	HBS
A XOR B	0110	0110	0	0 (0+0+0)	0 (0+0+0)	0 (0+0+0)
X_0	0101	0101	4	4 (4+0+0)	16 (4+4+8)	48 (4+12+32)
X_1	0000	0000	4	8 (4+4+0)	20 (4+8+8)	50 (4+14+32)
X_2	0101	0111	3	12 (3+1+8)	15 (3+4+8)	45 (3+10+32)
X_3	0101	1111	4	22 (4+2+16)	18 (4+6+8)	48 (4+12+32)
X_4	0101	1010	4	36 (4+0+32)	16 (4+4+8)	42 (4+10+28)

that the increased context sensitivity—preceding and succeeding bits are now taken into account—can provide further benefit to the EA.

An example fitness calculation using all four methods are demonstrated in Figure 3.4.

3.5.1.1 Computational Ambiguities in Fitness Assignment

When evaluating the fitness of binary strings that result from measuring digital circuits, ambiguities in the fitness measure have to be considered. If the fitness values calculated from distinct solutions with different outputs are the same, then it is unclear which solution to promote when in reality one might be a better solution than the other. This is particularly true when the fitness is given as the Hamming distance between desired and measured output. As can be seen from Table 3.2, where bitwise fitness represents Hamming distance, four different outputs (X_0, X_1, X_3 and X_4) result in the same fitness value, although, X_0 is considered to be a better solution than X_1. This is because a measuring of X_1 could mean unconnected outputs and in such a case the probability of evolution to improve the candidate circuits becomes lower. The consequence of this can be either getting stuck in a local optimum or, even worse, misleading the EA towards worse solutions.

Furthermore, in the case of measuring circuits with feedback on hardware, transient effects and possible oscillations have to be considered as well. These effects cause further

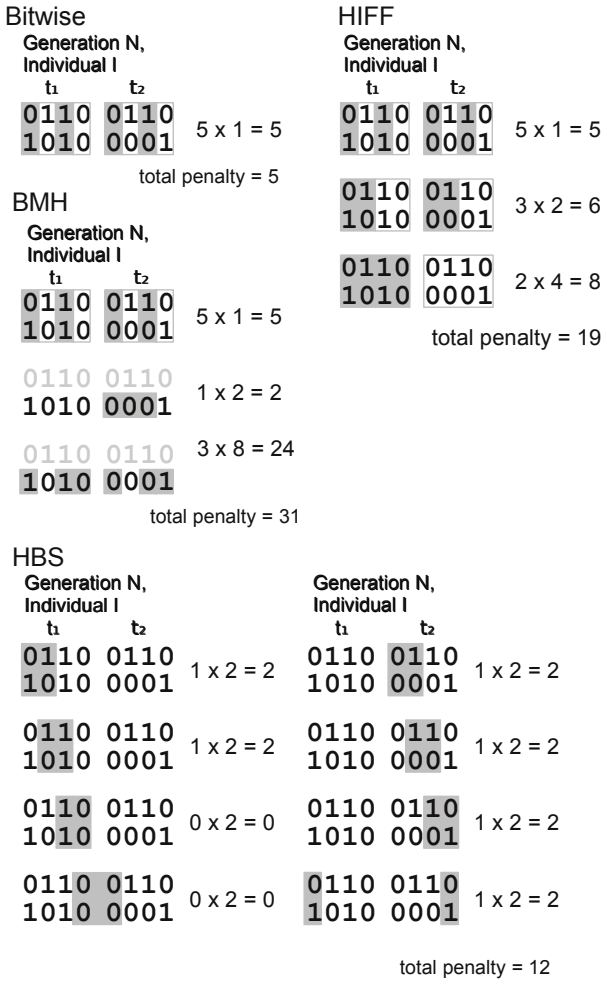


Figure 3.4: *Example fitness calculation for all four approaches are shown; first line of the two sets of outputs is always the desired output (XOR in this case), and the second line is the actual outputs of individual I at generation N. Bitwise: Only a single iteration is needed to calculate the fitness value, and it involves bit by bit comparison, for every wrong bit the fitness is increased by 1 (there are 5 in this case). BMH: For calculating the fitness using this method, 3 checks are done in a single iteration on the output, the first one is the same with the bitwise fitness calculation, the second check involves a bit variety check:* $constant \times (absolute\_value(No.\ of\ '1's\ in\ t_n - No.\ of\ '0's\ in\ t_n) + absolute\_value(No.\ of\ '1's\ in\ t_{n+1} - No.\ of\ '0's\ in\ t_{n+1}))$*; the constant is* 1 *in the example above and the result is* 2 *because of 3 '0's in* t_2*. The third check looks for a change in the outputs at different time steps (3 in the case above), and penalises each change by the number of output bits tested (8 in this case). HIFF: For HIFF 3 iterations are needed to calculate the fitness value: the first iteration is same with bitwise, the second iteration compares bits in sets of two and penalises each non-matching set by 2 (in this case there are 3 non-matching sets), and in the third iteration the bits are compared in the sets of 4 (a complete set of outputs), and each non-matching set is penalised by 4. HBS: In the example above only a binary block size of 2 bits is used, so only 1 iteration is needed. The only difference from HIFF block size 2 is that the binary block is moved every bit rather than every 2 bits. For the last output bit the binary block wraps around to the first output bit in the generation.*

ambiguities in the fitness assignment, thus misleading the EA. For instance, in the case of oscillations and unknown settling times, the same input can produce different outputs when measured at different points in time; as a consequence, good solutions cannot easily be recognised.

In Table 3.2 four different fitness measures are compared for a set of examples of possible solutions to evolve an XOR gate. It can be seen that the bitwise fitness assignment is not able to distinguish among the four different examples shown. Even though these solutions are not considered as equally good, the decision on which one to promote can only be based on random selection in the latter case. BMH, HIFF and the HBS approaches provide finer grained fitness measures, however, note that the distribution of the fitness values is different in each case.

Depending on the chosen measure, the fitness ranking is ambiguous and task dependent. Since the fitness landscape is not known before actually performing sufficient evolutionary runs, it is not always clear which solution is better. Consider for example solution X_0 or X_1 from Table 3.2, which cannot be distinguished by the bitwise fitness measure; BMH, HIFF and the HBS methods overcome this ambiguity and assign better (lower) fitness values to X_0.

The respective rank of X_1 and X_2 is different for BMH than for HIFF and HBS, due to the fact that in the case of BMH, transient faults are additionally penalised, independent of the actual position of the bit that causes the fault. Therefore, despite the fact that X_2 is featuring more correct bits than X_1, it obtains a worse fitness due to the extra penalty for bit 3, which delivers different results for measurements at different times.

The full consequence of using the different fitness measures can be seen in Figure 3.5, where the fitness values for all approaches and all possible results for t_0 and t_1 are plotted. The values are sorted in ascending order for BMH to provide a better overview over the fitness landscape.

Figure 3.5 demonstrates that bitwise comparison has the worst fitness landscape with lots of wide horizontal steps, which is a challenging problem for evolution to tackle. HIFF provides a better landscape than bitwise with smaller steps, whereas BMH and HBS methods provide even better sampling of the landscape with very small horizontal steps.

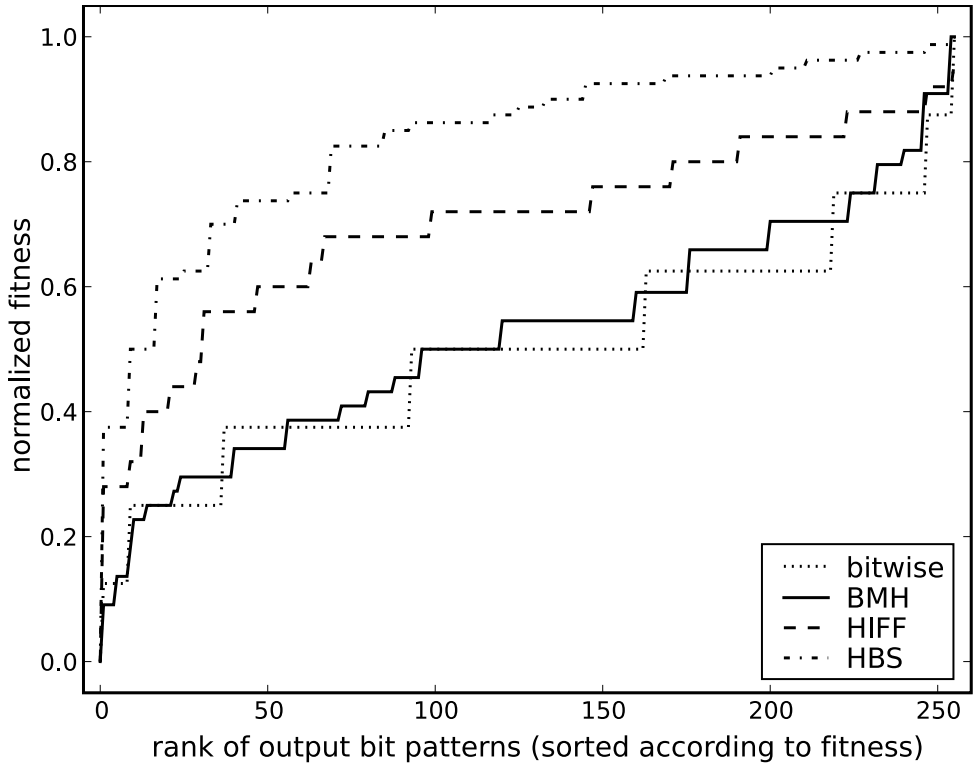


Figure 3.5: *Resulting fitness values for all approaches and all 256 possible logic input vectors for t_0 and t_1 (refer to table 3.2)are calculated and plotted. In all cases, the values are sorted in ascending order of fitness providing a better overview over the fitness landscape. Ambiguities in the fitness measures become visible in regions where the fitness remains constant for a wide range of different input patterns. In these cases, the EA is not able to decide which solution to promote.*

3.5.1.2 Experiments

The EA used in the experiments described operates directly on the configuration bit-string of RISA and evolution is completely free to use any resources available (routing and logic) and connect them freely. The 3×3 upper left RISA clusters are used, which are connected to two predefined inputs and one output of the chip.

Forty independent evolution runs with a target function of XOR are carried out using the four fitness evaluation methods explained earlier: bitwise, BMH, HIFF and HBS. For all experiments, an array of 64 input vectors is used for the evaluation of each candidate: for the first 32 entries, each test case I-IV (shown in Table 3.3) is repeated 8 times. The second half of the input vector contains 8 times the full input pattern I-IV, half of them in randomised order. Repeatedly testing the same test case and randomising the full input pattern makes it possible to measure transient effects like oscillations and delayed

Table 3.3: *Truth table used as input test pattern for XOR experiments.*

Test Case	XOR	A	B
I	0	0	0
II	1	1	0
III	1	0	1
IV	0	1	1

Table 3.4: *Evolutionary run results on evolving an XOR gate. Forty experiments for each fitness calculation method has been run, and the results of these runs have been analysed and put together in this table to give an idea of the performance and reliability of each method. Displaying: number of solutions found in 40 runs (Sol Found), number of successful solutions found (Success), average number of generations for each experiment (Avg Gens), standard deviation of average number of generations (Std Dev), average run time for each experiment (Avg RT), run time per generation (RT/Gen).*

Fitness	Results					
	Sol Found	Success	Avg Gens	Std Dev	Avg RT (mins)	RT/gen (secs)
Bitwise	16	10	3792	1684	206.03	3.26
BMH	34	26	2351	1720	127.65	3.26
HIFF	39	30	1600	1271	87	3.26
HBS	39	31	1364	1054	75.2	3.31

changes of the output. Therefore, through this method it is possible to detect and avoid intermittent results.

For the HIFF and HBS method, the block sizes used for evaluating the output—which is 64 bit wide due to the number of input test vectors—were 1, 2, 4, 8, 16 and 32 bits per block respectively. In the case of HBS there was no wrap around when the sampling window reached the end of the bit-string to conserve computational resources.

As can be seen from the results in Table 3.4, the problem caused by transients has been solved by using the new fitness functions. The type of fitness calculation method has little effect on the average speed of evolution, which is in each case ≈ 3.3 seconds per generation. Thus the higher complexities of BMH, HBS and HIFF fitness calculation methods do not appear to add considerable overhead to the evolutionary process.

The results show that bitwise fitness calculation method had the lowest evolutionary success rate where it found 16 solutions out of 40 runs and only 10 of these were success-

ful (i.e the other 6 were intermittent, and did not provide stable XOR gates). Whereas BMH method found 34 solutions out of 40 runs with 26 successful solutions, and HIFF and HBS found solutions in 39 of the 40 runs with 30 and 31 of them being successful respectively. If we look at the average runtime or number of generations as well as the standard deviation of the number of generations for each fitness calculation method, it can be seen that bitwise takes longer than any other method on average to finish a run, and HBS is the fastest and most dependable of all four. Thus in the results, HBS proves to be the most reliable, and fastest evaluation technique among the four tested methods.

Further experiments were carried out to intrinsically evolve a 4 bit parity and a tone discriminator on RISA. The task for the tone discriminator circuit is to distinguish between a $31.25kHz$ and a $250kHz$ tone (input as digital waves). Two different versions of the 4 bit parity problem are evolved; one combinational implementation, and one sequential implementation. The sequential implementation of the 4-bit parity problem and the tone discriminator circuits were devised to test whether the proposed fitness measure HBS works for sequential circuits or it is only effective for evaluating combinational circuits. In the sequential 4-bit parity experiments a stream of bits is applied to only one input of the circuit and the output is required to deliver the even parity result after the final input bit is applied. A constrained version of RISA, as well as the unconstrained version, is used for a set of the 4-bit parity experiments.

The same evolutionary settings were used as with the XOR experiments, but this time only 20 runs were carried out for each experiment. The reason for reducing the number of runs to 20 for each experiment is due to the increase in the evolutionary time for the experiments in this section, which involve circuits more complex than an XOR function. Table 3.5 details the results from the 4-bit parity and tone discriminator experiments comparing the HBS and bitwise fitness functions. The results displayed are the perfect solutions to the circuits being evolved, which worked correctly when they were tested with new random inputs and at different locations on the chip after they were evolved.

The HBS fitness method works equally well as the bitwise fitness calculation in the case of constrained evolution for the 4-bit parity problem but suffers when used for the tone-discriminator circuit. Using HBS fitness function evolution features a significantly better performance in the case of unconstrained evolution for the 4-bit parity circuit. When the 4bit parity task is implemented in a sequential way, the HBS fitness method again

Table 3.5: *The results for the tone discriminator and 4 bit parity circuits are shown. The number of runs where a perfect solution was found during evolution is given for each run.*

		Tone Discriminators	4 bit parity	
Genome	Fitness		Combinational	sequential
Constrained	Bitwise	4	6	-
Constrained	HBS	0	6	-
Unconstrained	Bitwise	0	1	3
Unconstrained	HBS	0	3	10

works better than the bitwise fitness calculation. This suggests that hierarchical fitness evaluation is not limited to combinational circuits but can also be applied to sequential problems. However, the fact that HBS performs worse than the bitwise fitness in the case of the tone discriminators indicates that its performance is task dependent.

It is important to choose the right fitness function for whatever the application may be. In the case with unconstrained intrinsic evolution it was shown that the conventionally used bitwise fitness measure fails to resolve ambiguities in the hardware measurements, which are caused by transient effects. These transient effects in many cases resulted in tricking the bitwise fitness measure with intermittent results that are not the true solutions to the target circuit being evolved. Due to its coarseness the Bitwise approach requires on average the largest number of generations and number of runs before finding a solution. However it was also shown that a fitness measure that works well for one problem may not do so for another. Fitness measures designed for a particular problem is likely to contain extra information for that problem, and when used for the evolution of a solution to another problem these fitness measures are likely to create unwanted biases. For example the hierarchical fitness measures (HIFF and HBS) favour circuits that implement the XOR function in them.

3.5.2 Multiplying Inputs

Applying the inputs of the target system multiple times to the evolutionary platform during evolution may provide higher chances of success by introducing richer environmental information to the hardware architecture.

Table 3.6: *The results of evolving the tone discriminator circuit on the constrained and unconstrained versions of the RISA platform are shown for two cases; the original case where the inputs of the circuit is applied from a single input port, and another case where the inputs of the circuit are applied at two different input ports.*

Genome	Inputs	Tone Discriminators
Constrained	Multiple Input	8
Constrained	Single Input	4
Unconstrained	Multiple Input	6
Unconstrained	Single Input	0

In the case of intrinsic hardware evolution where the substrate usually features a considerable amount of predefined interconnections of components and routing, it can be helpful to provide the input at different locations. This relieves the EA from having to distribute the input signals itself. Furthermore, in the case of unconstrained evolution it becomes less likely that the input is disconnected from the circuit with the flip of a single bit.

Applying the inputs of the target system multiple times might create scalability issues in evolving circuits with large number of inputs. Therefore this technique appeals to the evolution of circuits that require a small number of inputs (more often sequential circuits) that are applied serially. Therefore, the earlier used tone discriminator circuit is evolved in this subsection by applying the input of the circuit at two different locations of the RISA chip. The results are shown in Table 3.6.

The runs where the inputs were applied at multiple locations of the evolutionary hardware platform as opposed to the case where only one input is present indicate that the EA benefits from input signals that are more easily accessible from different locations. This suggests that for intrinsic hardware evolution experiments the task of routing and distributing signals on a given substrate is much more difficult than solving the computational task itself. This also hints that RISA's FPGA substrate's routing scheme, although simple, is not highly evolvable.

3.5.3 Decomposing Outputs

Many researchers have encountered the problem that the evolution of electronic circuits becomes exponentially more difficult when problems with an increasing number of outputs are tackled. Although this is an issue in both intrinsic and extrinsic evolution experiments, overcoming this problem in intrinsic evolution of circuits is particularly challenging. This is due to the highly constrained and limited logic and routing resources in a reconfigurable hardware architecture. Various approaches have been presented in the extrinsic evolution of circuits to solve the multiple output problem: partitioning the task with respect to the input or output space, incremental evolution of sub-tasks. In most cases the proposed methods are not applicable for intrinsic evolution of circuits, however there is still room for some of the techniques that emerged for extrinsic evolution to be used in speeding up the intrinsic evolution of circuits.

Evolving electronic circuits gets exponentially more difficult as the number of inputs and outputs of the circuit being evolved increases [Stomeo et al., 2005; Vassilev and Miller, 2000b; Yao and Higuchi, 1999]. There are numerous approaches in extrinsic EHW where automatic decomposition is used; there are examples where Automatically Defined Function (ADF) [Koza, 1994] and module acquisition [Walker and Miller, 2006] are automatically achieved in Genetic Programming (GP). There are some examples of Embedded Cartesian Genetic Programming (ECGP) where multiple chromosomes are utilised to divide the given task with respect to its outputs; each chromosome thereby represents an independent Cartesian Genetic Programming (CGP), which is required to solve the task for only one of the outputs [Walker and Miller, 2006]. Approaches where complex tasks are automatically partitioned by evolving modules that satisfy subsets of the demanded functionality can be found in [Hong and Cho, 2003; Kalganova, 2000a; Stomeo et al., 2006; Torresen, 2003]. In the case of [Kalganova, 2000a] the obtained modules are merged and optimised in terms of redundancy in a second stage (incremental evolution) of the EA.

Decomposing the outputs of a circuit to form separate sub-circuits for quicker evolution can also be achieved in hardware. This is easily achieved by using distinct circuitry for each output of the target circuit, and applying all the inputs to all these sub-circuits. This can be conveniently achieved on a RISA chip for up-to four partitions.

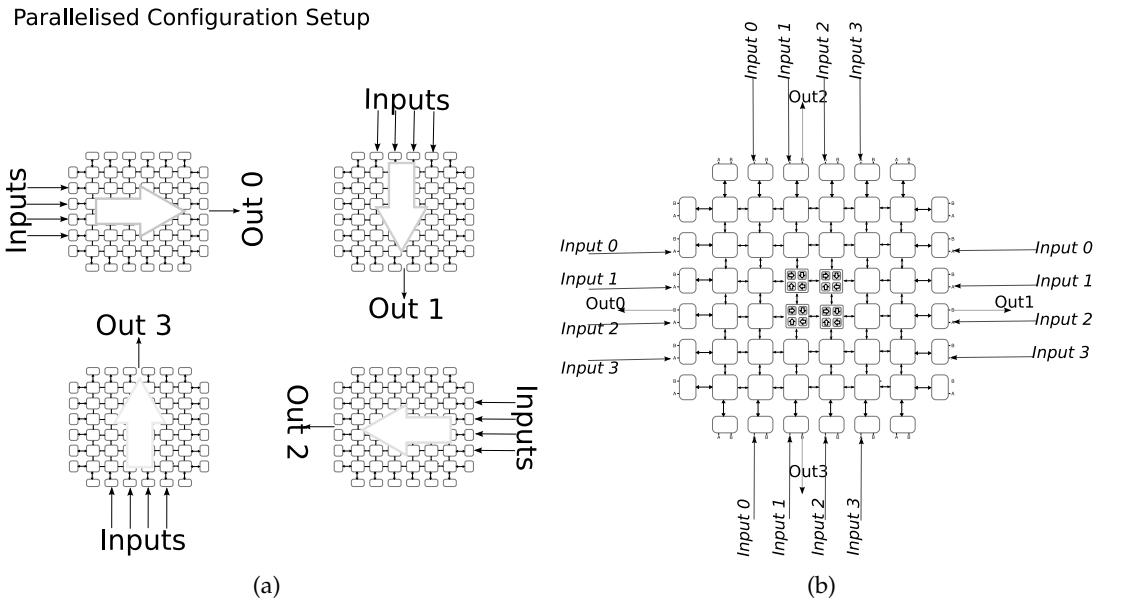


Figure 3.6: *To partition the problem outputs, the four directional function units of each cluster are constrained to direct the flow of the circuit, and then all inputs are applied from all four directions using the ports A of the I/O blocks. The individual outputs are then retrieved from a dedicated direction of the chip (using ports B of the I/O blocks). See Appendix B for a discussion and schematics of the cluster and IO routing scheme in RISA.*

Due to the four-fold architecture of the RISA chip, the four directional function units in each cluster are used as independent logic blocks. In principle, the routing configuration options allow the interconnection of four function units within each cluster. However in this case the architecture is constrained in a way that the four function units are kept separate and can only connect to the function units of the same kind of the neighbouring clusters.

The aim is to use the four separate blocks of circuitry obtained in RISA to evolve a multiple output circuit by partitioning it with respect to its outputs. Inputs are applied to all four circuits obtained, and each circuit is expected to evolve one of the outputs, rather than all of them, see Figure 3.6.

3.5.3.1 Experiments

Various experiments are carried out in order to determine how evolving circuits with multiple outputs in hardware is related to the amount of resources allocated to evolution. The problems examined in the experiments are, 4-bit even parity, 2-bit full adder, and 2-bit multiplier. Results are shown in Table 3.7.

Table 3.7: *Results obtained from 20 independent runs for 4-bit parity, 2-bit full adder and 2-bit multiplier. All experiments are carried out with two different setups, namely classic and parallelised. Classic case is where a single circuit is evolved with all the required outputs. Parallelised case is where the target circuit is partitioned into a number of circuits with each circuit providing only one of the outputs. Different series of experiments are performed with an increasing number of outputs in order to illustrate the increasing level of difficulty of the respective task. For instance for the runs where the task is to evolve a multiplier with only one output, three of its outputs were not considered in the fitness function.*

Logic circuit	Hardware setup	Outputs	Successes	Avg. no. generations
4-bit parity	classic	1	20	126 ± 123
2-bit full adder	classic	1	19	1214 ± 1709
		2	8	5156 ± 2086
		3	3	3536 ± 1534
	parallel	2	20	4903 ± 2816
		3	20	6491.5 ± 3115.3
2-bit multiplier	classic	1	20	140 ± 86
		2	19	2578 ± 1595
		3	16	3892 ± 3133
		4	10	5420 ± 2999
	parallel	2	20	1991 ± 1895
		3	20	2672 ± 1756
		4	19	3266 ± 2126

In order to investigate the effects of multiple outputs on the performance of evolution experiments are carried out where only a single output of a multiple output circuit is evolved. Subsequently, the number of outputs is increased in a series of experiments until all outputs of the desired circuit are included in the experiment. Not surprisingly, the evolutionary performance degrades in the "*classic*" case as the number of outputs expected to be evolved are increased, see Table 3.7. Partitioning the RISA chip in order to provide 4 different circuits to evolve, keeps the evolutionary performance stable when the number of outputs is increased.

It is observed that the "*parallelised*" configuration provided a major improvement and in almost all cases yielded a success rate of 100%, see Table 3.7. By partitioning the outputs of a target circuit and evolving a separate circuit for each of its outputs had dramatically reduced the evolutionary load and increased the evolvability of the problem.

3.5.4 Input Pattern Order Problem

It was shown in Section 3.4.2 that the design of the input pattern plays an important role in intrinsic EHW. The results show that randomising the input pattern during the course of evolution is mandatory in order to achieve valid circuits that work correctly under all conditions.

Input pattern ordering can also be used to enhance the evolutionary performance when evolving circuits with multiple outputs. Input pattern ordering can be designed intelligently with consideration of the problem at hand in order to balance the distribution of the output values of the target circuit. The aim is to equalise the emphasis on the type of outputs that occur by using weighted input vectors.

One of the most important steps in evolutionary experiments is the evaluation process: candidate solutions are tested and rated by a fitness value, which determines how successful they are in solving a given task. It is the evaluation process that provides the foundation for the selection process, and thereby greatly contributes to guiding evolutionary search through the solution space in order to construct the desired circuits. It was shown earlier that a fitness function, which is more suited for the particular problem and evolutionary platform at hand had a considerably more positive impact on the performance of evolution in designing the desired circuit. The importance of different selection schemes, being the second major driving force in EAs, have also been addressed by various researchers before [Deb, 2001; Xie et al., 2006]. Although evolution is guided to the same extent by both the fitness function and the selection process, the limiting factors for these processes often emerge from the information available from the evaluation stage: the inputs applied and the outputs measured during the testing process determine the shape and size of the design space evolution can sample.

Therefore, it is important that the evaluation process extracts as much and as accurate information as possible in order to provide precise and useful information to the selection process. In the evolution of circuits in hardware, the main (and most of the time the only) available inputs to the evolved systems are the inputs from the truth table that define the problem at hand. The number of times and the pattern of the inputs applied can therefore be used to shape the search space sampled by the EA. In fact the input pattern

ordering can be used to tackle various important challenges in EHW: avoiding local optima, achieving fully functional solutions and improving the performance of evolution.

3.5.5 Getting Stuck in Local Optima

The most likely reasons for getting stuck in local optima during the course of evolutionary search are:

1. Unsuitable fitness evaluation methods that are not capable of considering special cases within certain problem domains, see Section 3.5.1.2.
2. Intermittent solutions that can mislead the search process and cause the EA to stall.
3. Rugged fitness landscapes that can mislead and impede the EA in finding the optimal solution.

The shape of the fitness landscape is determined by the assessment of the output values and the genetic representation as well as the problem. As the given problem itself cannot be changed, the fitness evaluation method and the genetic representation remain to be optimised.

As mentioned before, including randomness in the input pattern order and using suitable fitness functions can alleviate the first two reasons for getting trapped in local optima. To a certain extent, randomising the input can also indirectly help in the case of rugged fitness landscapes (third reason). This is due to the fact that evolution is less likely to converge onto local optima that are only stable when a certain input pattern order is applied. This issue can further be tackled by using a well planned input pattern ordering during evolution that can balance the emphasis on the possible output values of the target circuit.

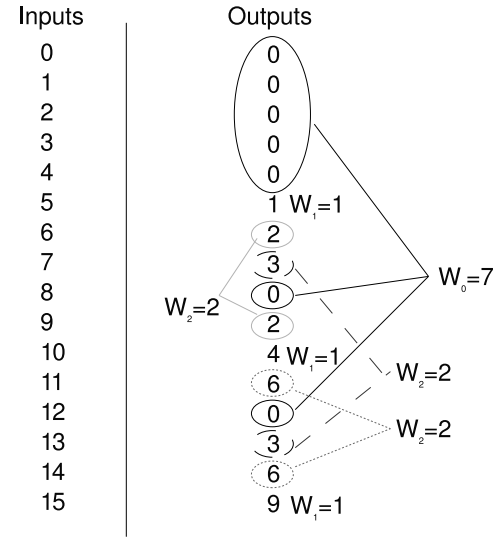
Unbalanced output value distribution of a problem description can create a rugged fitness landscape for the evolutionary design. An example of such a problem is a 2-bit multiplier, explained in Table 3.8. An example of a problem that inherently features balanced output distribution is a parity circuit where there are even numbers of 1s and 0s as outputs, see Table 3.8. Thus, the frequency at which the outputs occur is not in favour of a particular value of output.

Table 3.8: *Truth table for a 2-bit multiplier and a 4-bit Parity. For the multiplier, the number of input combinations that return an output of 0 are the most common, where the outputs 1, 4, 9 occur for only one input combination. When a 2 bit multiplier is evolved by testing the circuits using all input combinations the same number of times in every evaluation step, more emphasis is given in obtaining outputs that equal 0 than any other, and little emphasis is given in obtaining the correct outputs for the cases where the outputs are 1, 4 and 9; even though achieving the correct result for any input combination should be equally important for a fully functioning circuit. Unlike the multiplier, the parity problem has even number of input combinations that return the outputs 1 and 0. Thus there is equal emphasis in achieving either of the output values, and the search process is not biased to satisfy some outputs earlier (more likely) than others.*

A_0	A_1	B_0	B_1	Parity	Multiplier
0	0	0	0	0	0000 = 0
0	0	0	1	1	0000 = 0
0	0	1	0	1	0000 = 0
0	0	1	1	0	0000 = 0
0	1	0	0	1	0000 = 0
0	1	0	1	0	0001 = 1
0	1	1	0	0	0010 = 2
0	1	1	1	1	0011 = 3
1	0	0	0	1	0000 = 0
1	0	0	1	0	0010 = 2
1	0	1	0	0	0100 = 4
1	0	1	1	1	0110 = 6
1	1	0	0	0	0000 = 0
1	1	0	1	1	0011 = 3
1	1	1	0	1	0110 = 6
1	1	1	1	0	1001 = 9

In the case of a problem with unbalanced outputs, evaluating the circuit using the input pattern order unaltered from the truth table will create an uneven search space; with local optima around the outputs with high number of occurrences. Especially when done intrinsically in hardware, due to many limiting properties of hardware substrates like routing and fixed I/O locations, evolution can easily get stuck at these local optima thus failing to successfully evolve the desired circuit.

Balancing of the outputs can be achieved by assigning a weight to each output combination, which corresponds to the number of times it appears in the truth table; the more often it appears the greater is the weight. Using these weights the input pattern applied during the evaluation process can be shaped in a way that combinations of inputs that produce less frequently occurring outputs (outputs with smaller weights) are repeated



$$R_i = \frac{\sum W_k * W_{k+1}}{W_i}$$

$$R_{TOT} = n * \sum W_k * W_{k+1}$$

$$k = 0..n-2$$

e.g

$$R_{0_{multiplier}} = \frac{7*1*2}{7} = 2$$

$$R_{TOT_{multiplier}} = 7*(7*1*2) = 88$$

n: Number of different weights.
W_N: Weight *N*.
R_N: Repeat value *N*.

Figure 3.7: *An example process determining the weights and repetition parameters for the inputs and outputs for the 2-bit multiplier problem is demonstrated. Note that multiples of R_{TOT} are ideal input pattern sizes that need to be applied to achieve a perfectly balanced output pattern. The greater the weight (W) of an output value, the more biased evolution will be towards that output if output balancing input patterns are not used.*

more often. Thus, balancing the number of occurrence of each output combination is achieved.

An example of how the weights are determined in the case of the 2-bit multiplier is shown in Figure 3.7. In order to obtain the desired input pattern that yields balanced outputs, the weights can be used to obtain the number of repetitions for each input combination. Note that the number of input samples is set to 128 for the experiments in this subsection. Although for circuits like the full adder, a much larger number of input samples would be required to obtain a perfectly balancing input pattern, 128 is used due to memory limitations in the hardware system. This means that also for the multiplier example where a sample size of multiples of only 88 are required to represent a perfectly balancing input pattern, the number of samples used is set to 128 for simplicity purposes. As a consequence, in the experiments presented where balancing input patterns are used, the balancing of the outputs are actually not perfectly achieved. The corresponding number of occurrences of each output in the unbalanced and the balanced case respectively are shown in Figure 3.8.

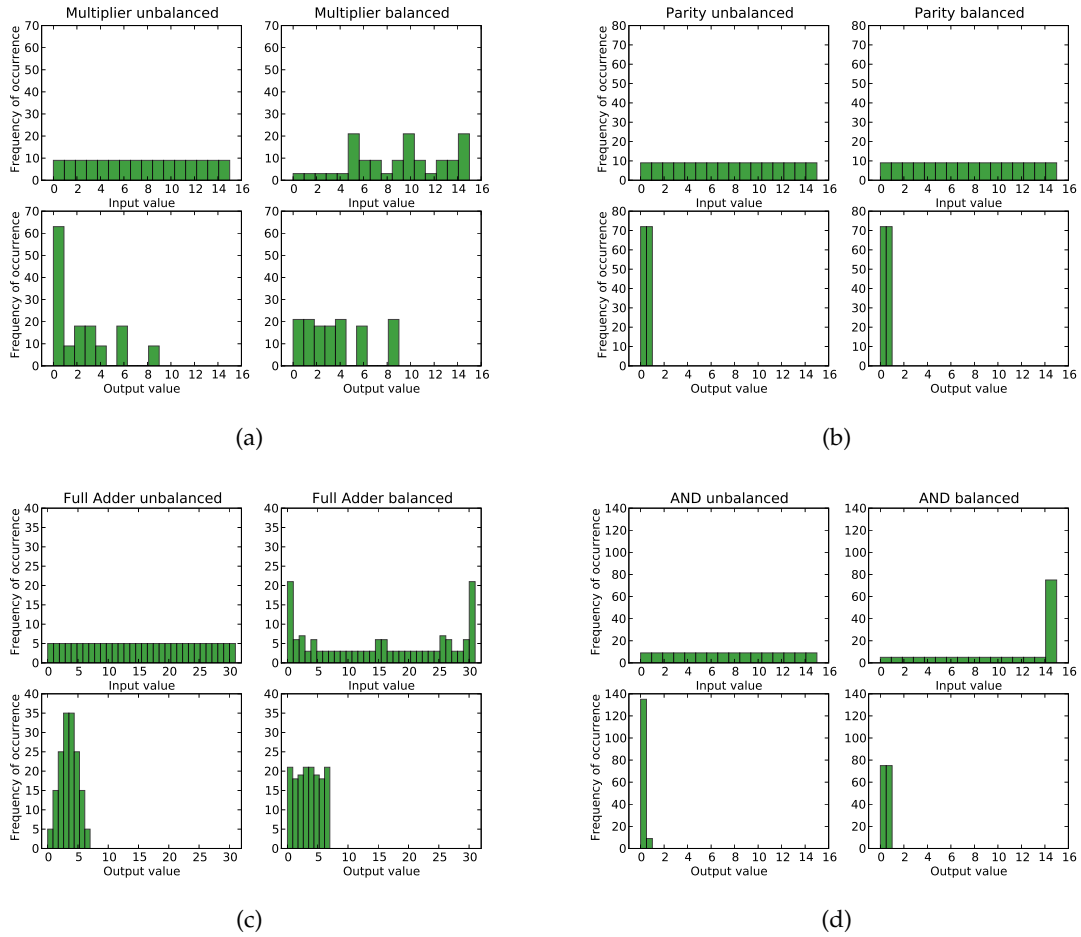


Figure 3.8: *The corresponding output occurrences are shown for the balanced and unbalanced output patterns used in the experiments presented in this section for all the circuits. In the case of the multiplier and the AND, it should be obvious that with unbalanced outputs, the output value 0 is favoured more than any other output, which would encourage evolution to prefer circuits that mainly produce 0s for their outputs: a highly undesirable bias for intrinsic evolution where a stuck at 0 could achieve a higher fitness score. On the other hand, for the parity circuit the non-balancing and the balancing input pattern orders are identical.*

Using an input pattern that yields balanced outputs during evolution to evaluate the candidate circuits provides a smoother design space for the EA, since there is no longer a bias towards solutions that only satisfy a more frequently sampled subset of the output value range. Hence, reducing the number of local optima.

3.5.6 Experiments

Evolution of 2-bit multiplier, 2-bit adder, 4-bit parity, and 4-bit AND circuits is done intrinsically and evolution of 2-bit multiplier also done extrinsically to demonstrate the

effects of balancing input patterns on both evolution platforms. For extrinsic evolution of circuits CGP is used.

The number of maximum evolutionary generations is increased to 15000 for the experiments presented in this subsection. A total of 20 randomly initialised evolution runs have been carried out for all hardware experiments and a total of 100 independent runs are performed in the case of evolution in software. A constrained version of RISA chip was used for these experiments, and the genotype for each cluster is reduced from 472 to 128 configuration bits. All of the clusters were used for the experiments presented in this subsection, resulting in a total genome size of $36 \times 128 = 4608$ configuration bits to configure the RISA FPGA.

The experiments that are undertaken to investigate the effectiveness of the balancing input pattern order are presented in Table 3.9. A variety of problems are tackled in the case of intrinsic evolution: 4-bit AND, 4-bit parity, 2-bit full adder and 2-bit multiplier. It can be seen from Table 3.9 that solutions to 4-bit parity and 4-bit AND are found quickly in the evolution runs. This is due to the fact that both problems are relatively simple circuits with only one output. However, the results show that even though evolution reliably finds solutions for simple problems like a 4-bit AND, a speed-up of factor two can still be observed when output balancing input patterns are used. The 4-bit parity is included as an example where the output value distribution is already balanced.

The intrinsic evolution of 2-bit full adders and 2-bit multipliers are unfortunately non-trivial tasks. As can be seen from the graphs in Figure 3.8, these functions feature not only multiple outputs but also a greatly unbalanced output value distribution. Thus, it is not surprising that the success rate of evolution is low in the cases where output balancing is not applied; for the 2-bit full adder only two and for the 2-bit multiplier only three out of twenty runs are successful. However, it can be observed that the success rate was increased by a factor of four for the 2-bit full adder, and a factor of three for the 2-bit multiplier in the runs where the balanced outputs method is applied.

In order to investigate whether the benefit that comes with balancing the output value distribution is restricted to intrinsic circuit evolution or it is also advantageous in the extrinsic case, additional experiments are carried out for the 2-bit multiplier in CGP. Since CGP is fairly quick in evolving 2-bit multipliers, compared to an embedded system,

Table 3.9: *Results with the unbalanced and balanced output patterns for 4-bit parity, 4-bit AND, 2-bit full adder, and 2-bit multiplier. The term Unbalanced output pattern refers to using each entry of the truth table inputs once when evaluating the circuits during evolution. The last results for the 2-bit multiplier presented in the table are the results obtained from software runs using CGP with balanced and unbalanced output patterns. In the case of the intrinsic experiments, it has been observed that using output balancing input patterns results in a real time speed up of 20-25%, in addition to the increase in the success rates.*

Problem	Input Pattern	Successful Runs	Avg. Gens for Successful Runs	Avg. Fit. for Unsuccessful Runs
4-bit parity	N/A	20	162 ± 123	N/A
4-bit AND	unbalanced	20	23 ± 23	N/A
4-bit AND	balanced	20	12 ± 8	N/A
2-bit adder	unbalanced	2	N/A	13.33 ± 7.77
2-bit adder	balanced	8	N/A	9.50 ± 5.02
2-bit multiplier	unbalanced	3	N/A	29.35 ± 33.66
2-bit multiplier	balanced	10	N/A	7.50 ± 6.26
2-bit mult.-CGP	unbalanced	100	4792 ± 7157	N/A
2-bit mult.-CGP	balanced	100	3776 ± 5257	N/A

it was feasible to perform 100 evolution runs. Although the benefit is not as significant in software as it is in hardware, a speed-up in the region of 20% can also be observed for extrinsic evolution experiments. The fact that extrinsic results are not as much affected by the unbalanced output distributions, i.e. the hardware substrate is not as capable of dealing with unbalanced outputs as CGP, suggests that the presented approach might be particularly useful in cases where representations with fixed or predefined topologies are imposed. Fixed topologies lack certain abilities of neutral search; altering the connections of the designs being evolved freely, which is part of effective neutral search, is not possible in fixed topologies. Thus, balancing the output could be both a means for assessing a hardware substrate's evolvability and a tool to improve the neutral search capabilities of a given substrate.

3.6 Circuits Evolved on RISA

Throughout this chapter various mechanisms have been introduced to aid the intrinsic evolution of digital circuits, which were demonstrated with experiments on the RISA

platform. RISA provides a configurable digital logic platform, which is specifically designed for the intrinsic evolution of digital circuits. RISA allows unconstrained evolution at bit-string level.

The successfully evolved circuits on the RISA platform are; 4-bit parity, 2-bit multiplier, 2-bit full adder, 4-bit AND, XOR, and tone discriminator circuits. Although these circuits have only been evolved to demonstrate the use and power of the suggested mechanisms for successful and reliable evolution of circuits, they were still challenging and time consuming to evolve on RISA. Evolution of all these aforementioned combinational circuits in simulation is quicker, and there is more room for scalability since the simulation substrate is not constrained by the available routing or number IOs. On the other hand, direct evolution of combinational circuits that are more complicated than a 3-bit multiplier becomes a merely impossible task even in simulation. Hence the achievable complexity of circuits in simulation (by using the similar techniques used here) when compared to RISA is not much higher. On top of this, evolution of sequential circuits in RISA is highly favoured and the results demonstrated with the tone discriminator and sequential 4-bit parity circuits were promising. In Appendix A an example analysis of two of the tone discriminator circuits evolved are shown, and the use of the underlying substrate by evolution is discussed. In the result presented, an important observation is made about the behaviour of evolution: evolution always explores every available resource during its search for the target design.

Most of the time evolving the circuits shown in the previous sections of this chapter had already been quite a challenge, and apart from the scalability issues of genotype–phenotype mapping, there exist several other limiting factors for evolution of circuits in RISA. The insufficient number and programmability of IOs, and limited routing resources are the main bottlenecks of the RISA platform. For a brief description and analysis of some these resources on RISA see Appendix B.

3.7 Summary

Evolving digital circuits in hardware is challenging. Evolution, as a stochastic algorithm, accepts all the information that is supplied by the fitness function and the evolutionary

environment in its entirety, and exploits every bit of available resource to meet the bare minimum of the requirements. This can create problems both in software and hardware evolution of circuits and systems, however, the unpredictable behaviour of evolutionary design becomes more abundant when used in hardware. A configurable hardware substrate like RISA contains components and factors (e.g. delays) that may be unaccounted for, and these may create a complex environment for evolution. In such cases, it is not an easy task to define fitness functions and selection mechanisms in order to guide evolution flawlessly in the search of a design.

In this chapter, a series of methods have been introduced and shown to greatly improve the success of evolution in designing valid digital circuits in hardware. In Section 3.4, the importance of input pattern ordering along with accurate output sampling rate and testing of circuits at different parts of a hardware substrate were emphasised in order to evolve valid circuits in hardware. It was shown that randomising the input pattern order is especially crucial to achieving valid circuits during evolutionary design.

In Section 3.5, ways of improving the performance of intrinsic hardware evolution were discussed. Once more, the importance of input pattern ordering was emphasised for intrinsic hardware evolution. It was shown that randomising the input pattern helps to keep evolution away from local optima. It was also shown that the input pattern could be shaped in order to balance the emphasis of every possible output value within the fitness landscape, and this considerably increased the success rate and the performance of evolutionary design of many circuits. The shortcomings of a simple bitwise comparison as a fitness function is discussed and demonstrated by its failure to distinguish a range of results, thus creating ambiguities. Bitwise comparison as a fitness function is especially ineffective in a hardware platform where transient effects are present. A range of fitness functions are introduced to address this issue, together with a discussion that highlighted the importance of using fitness functions that emphasise the target problem (e.g. HIFF fitness function for parity circuits). Using the correct fitness function is always important when using evolution to design circuits and systems, and it becomes more important as the evolution substrate becomes more intricate (i.e. moving from extrinsic to intrinsic evolution). It is later on shown in Section 3.5.1.2 that the best performing fitness function (HBS) can not perform well in the evolution of a tone discriminator. This demonstrates that each problem requires a well defined fitness function, where expert knowledge and

attention will be rewarding. Unfortunately there is no magic fitness function that can do all, and generic fitness functions like bitwise fitness can easily fail to guide evolution appropriately as demonstrated in Section 3.5.

Furthermore, two ways of speeding up intrinsic evolutionary design were suggested and demonstrated for their effectiveness: applying the inputs at more than one location during evolution, and decomposing the outputs to be evolved as separate circuits. It was shown that for sequential circuits (tone discriminator in the example provided), applying the inputs at more than one location really improved the evolutionary success in designing the correct circuit. This is due to the limited and inflexible routing present in a reconfigurable hardware platform. Also, a simple way of decomposing the circuit outputs for a quicker evolution of multiple output circuits was presented. It was demonstrated by experiments that using this decomposition method the evolution of more complex circuits was made possible, as well as achieving a speed up in the evolutionary performance.

All the mechanisms that have been developed thus far helps evolution to achieve circuit designs on hardware that are comparable to circuit designs achievable via extrinsic evolution, and this is done without imposing many constraints on the hardware architecture. Thus with the techniques presented in this chapter, the benefit of innovative evolutionary design is still available, e.g. see Appendix A. Now that the evolutionary properties of RISA are explored and an acceptable evolutionary performance for the design of circuits on RISA is achieved by laying out the essential techniques; evolution of higher complexity systems on the RISA platform can be the next step. As discussed in Section 2.6, scalability and evolvability are serious setbacks in achieving complex problems via evolutionary design. Therefore, it is necessary to study and develop a technique that can provide a scalable approach to the evolution of systems.

Chapter 4

Development

"The development of multicellular organisms from a single cell–the fertilized egg–is a brilliant triumph of evolution" notes Wolpert in his book, Principles of Development [Wolpert et al., 2002]. The structure of a single cell may arguably be the most complex part of any organism, but a unicellular organism is vastly limited in the tasks it can achieve and is vulnerable to environmental threats. A multicellular organism is capable of multi-tasking using division of labour amongst the cells, and it is able to protect itself from environmental threats better than a unicellular organism would, since the loss of a cell or few cells does not necessarily harm the organism. Although multicellularity could have arisen through cell division failure or chance mutation in the evolutionary history of organisms, multicellularity is a key approach harnessed by biology to create complex and intelligent organisms capable of executing sophisticated behaviours and surviving harsh and changing environmental conditions [Bonner, 1998]. Multicellular organisms are a product of a process called development that builds these organisms from a single cell. Wolpert defines development as "The process of gene activity that directs a sequence of cellular events in an organism which brings about the profound changes that occur to the organism" [Wolpert et al., 2002]. In other words, when simply put, development is the step where all the genetic information gathered over the evolutionary history of an organism is put to use to create an adult organism from a single cell. However, development is also a mechanism that maintains the stability and functionality of an organism throughout its lifetime, and not merely the genotype-phenotype encoding mechanism of biology for creating complex multicellular organisms. Thanks to multicellular develop-

ment an organism is capable of surviving damage and loss of its physical parts, which otherwise would be lethal to the organism. This is illustrated in Figure 4.1.

The aim of this chapter is to present an introduction to the use of development in Evolutionary Computation (EC). A simple and brief overview of biological development with an emphasis on the inspirational and relevant aspects of development to EC is provided in Section 4.1. This is followed by Section 4.2, which explains and highlights the potential benefits of modelling biological development in EC. Section 4.3 provides a brief introduction to the existing models of artificial development, and introduces a new classification scheme to categorizing the artificial models of development. This new classification scheme distinguishes the artificial models of development in two clear and well defined categories. Finally, the chapter is summarized in the final section.

4.1 Biological Development

Development of a single cell into an adult organism is a marvellous but poorly understood process. As mentioned earlier, biological development is more than just the transformation of a single cell into an adult organism, but it is an ongoing process of building and maintaining a functional organism via the use of genetic information gathered over the evolutionary history of the organism. In biology, an **organism**–a living system capable of responding to outside stimuli, reproduction and homoeostasis–is formed of single or multiple **cells**. Hence, a cell is the most basic unit of a biological organism. All cells have some form of encapsulating structure, such as a cell wall or a **cytoskeleton** that holds the cell together and protects it from outside threats. Inside a cell is the **cytoplasm**, a medium where most of the cellular activities occur, and where the cell DeoxyriboNucleic Acid (DNA)–the genetic information or the genome–is kept.

Organisms that are multicellular, with the exclusion of colonies of cells, all have nuclei within the cytoplasm of their cells, which encapsulate the DNA of the cell. A **nucleus** is the control center, the kernel, of a biological cell. The nucleus, in cells where it is present, controls the activities of its cell via processing the DNA. The processing of the DNA creates a regulatory network among the genes of the DNA, where the interactions amongst these genes determine the development and faith of the cell.

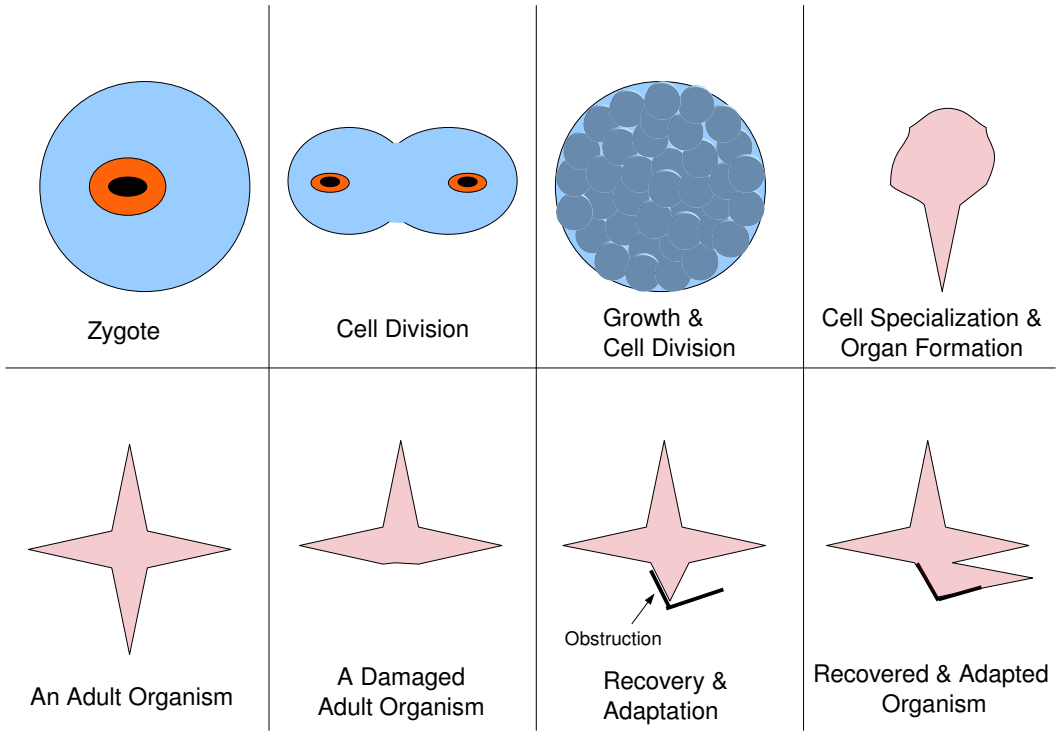


Figure 4.1: *A simplified depiction of development of a multicellular organism. A hypothetical organism's, shaped like a 4-pointed star, development is shown from a single celled stage; through the cell division and specialisation and up to the growth and formation of the adult organism. Then, an example of recovery and adaptation scenario for the organism is depicted via the last 3 frames. Although this figure simplifies the process of development, it depicts the high-level behaviour of development and the key mechanisms of development that make it attractive to the field of EC.*

Multiple or single celled, every organism has a functional GRN that can respond to the environment. In a multicellular organism all the cells have an identical genotype, but various cells appear different; this is due to different genes being expressed in different cell types. Genes interact with one another via proteins that control the activation of genes, and proteins are produced by active genes. This creates a self regulatory network of genes, which are also affected by proteins that may be produced due to a reaction to an environmental change or proteins that come from other cells. Proteins can cause a range of reactions in a cell that can directly or indirectly determine the cell behaviour. Figure 4.2 provides a simple diagram describing how a GRN works.

Figure 4.3 summarizes the process of **protein synthesis**, which is the process that takes place when a gene is activated. Biological protein synthesis involves the process of a protein transcribing a gene (binding the promoter sequence to activate the gene), where the

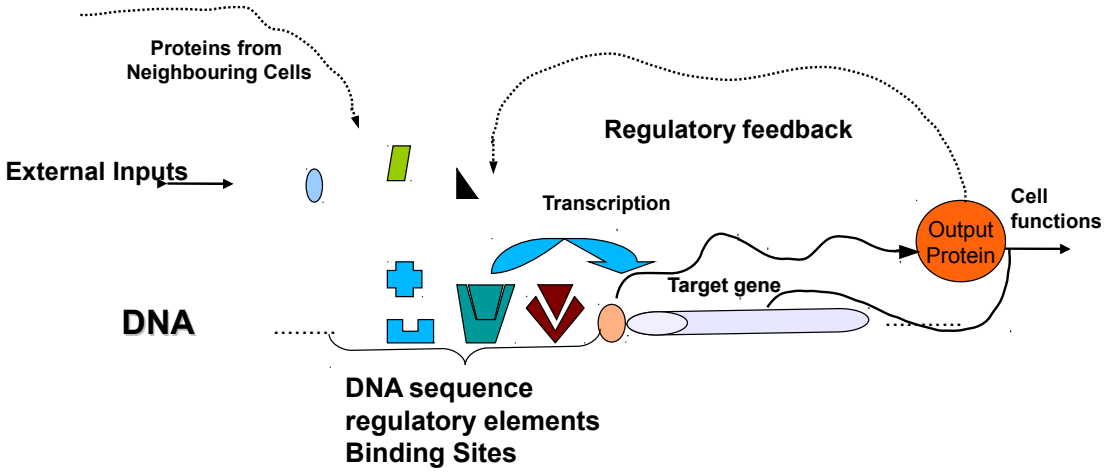


Figure 4.2: *A gene is activated by the correct matching of proteins that favour the transcription of the gene. The gene is transcribed when it becomes active producing a protein that may affect various functions of the cell it is produced in. Furthermore, the produced protein has the ability to bind genes within the cell to enhance or inhibit their activity or diffuse outside the cell to enter another cell. The resulting interactions of genes and proteins form a network of genes (GRN) creating a dynamical network. Hence, a gene's transcription is affected by the preconditional coding of the gene, activity of other genes, activity of genes in other cells, and environmental changes that may cause the production of further proteins.*

genetic information (postconditional region) is copied into a mRNA (messenger RNA[1]). This is followed by the mRNA strand moving out of the nucleus of a cell to meet a complimentary tRNA (transfer RNA). Each tRNA molecule compliments the opposing bases on the mRNA strand perfectly. These in turn have the amino acid sequence to successfully code for a particular amino acid which forms a protein. Hence the amino acids (aka peptides) from the tRNA and mRNA combine to form a polypeptide chain (proteins), and can be used in a variety of structures such as enzymes and hormones to carry out cell functions. Although protein synthesis in biology covers a series of complex processes, simply put it is the phase of building proteins. For an in depth description and discussion about GRNs, see [Davidson, 2006].

The interactions of genes that determine the profound changes to and the actions of an organism are the drive of biological development. Some cells have the ability to grow, multiply, differentiate and spatially organize themselves in order to form a multicellular organism. In a multicellular organism, various types of cells emerge, where similar types work together to achieve specific tasks and different types carry out separate tasks. Once

[1] A nucleotide chain that is transcribed from and similar to DNA, but has small structural differences.

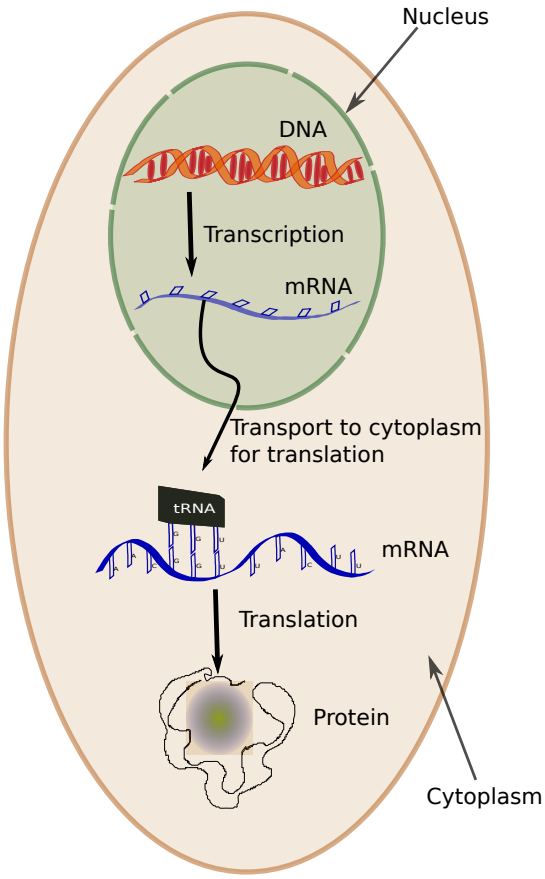


Figure 4.3: *A simple overview of protein synthesis in biological cells. In a cell, the DNA within the nucleus is transcribed into an RNA. The RNA may then move out of the nucleus into the cytoplasm of the cell in order to be translated to a protein. The produced protein is then used for controlling the cellular activities.*

specialized, two different types of cells are functionally different. However, all the cells (same or different type) still carry an identical genotype. Figure 4.4 illustrates the early development of a human embryo from a single cell. In this process a single cell is able to achieve a multicellular organism with distinct parts (organs) by cell division, growth, differentiation and morphogenesis[2].

Developing into multicellular organisms allows the emergence of robust creatures that benefit from scalable designs, which can carry out larger number of higher complexity tasks than a single celled organism, and tolerate larger number of environmental threats since the death of a few cells does not necessarily mean the death of the multicellular organism.

[2]Morphogenesis is the reorganization of cells in the development of a multicellular organism.

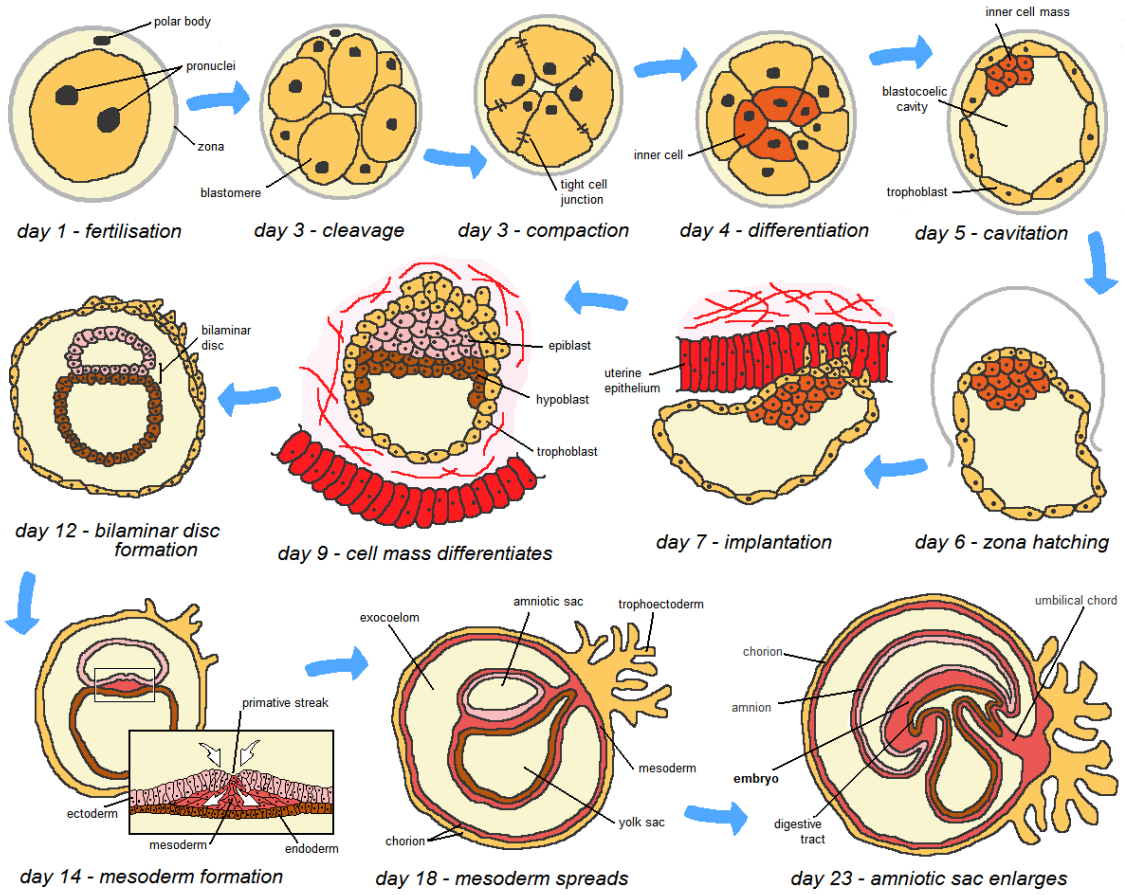


Figure 4.4: *The initial stages of human embryogenesis. This image is used with permission; it has been published in* ***http://en.wikipedia.org/wiki/Developmental_biology****, dated 31 May 2010.*

Multicellular biological organisms have been a topic of interest in the computer science and engineering fields as inspiration of models of intelligent systems. Their ability to be robust, adaptive, and scalable while they develop, make them interesting in computational intelligence as these properties are difficult to design using traditional approaches of computation or engineering. Biological organisms can grow from a single cell into a multicellular organism using the same genotype for all cells. These cells can then specialize to form different parts of an organism. Although the process of development in biology is clearly defined, its definition in EC can be different depending on the author [Chavoya, 2009; Devert et al., 2007; Eggenberger, 1997; Flann et al., 2005; Harding et al., 2007; Kumar and Bentley, 2003a; Miller, 2004; Stanley, 2007; Tufte, 2009]. While some artificial algorithms try to closely model the biological development, others are simply inspired by a mechanism of biological development. In the latter cases, artificial development is defined by the source of its main inspiration and the task it is used for. Roggen's diffusion based developmental model [Roggen, 2005], the self modifying

cartesian genetic programming by Harding et al. [Harding et al., 2007], and Stanley's pattern producing networks [Stanley, 2007] are examples of systems that use simple inspirations from few biological developmental mechanisms. Although most commonly referred to as artificial development, its name can also take many other forms; computational embryology, artificial embryology, artificial embryogeny, artificial ontogeny, computational development [Chavoya, 2009], and embryonics [Tempesti et al., 1999]. In the remainder of this thesis *artificial development* will be used to refer to a digital system that models the biological developmental process for the uses of understanding biology and/or aiding EC. Going back to Wolpert's definition of biological development [Wolpert et al., 2002], development will be used to refer to the formation and maintenance of multicellularity in an organism. After a review of the benefits of multicellularity in EC, already existing models of artificial development will be discussed further.

4.2 Benefits of Multicellular Development to EC

A multicellular design approach in EC benefiting from the decentralized organizational mechanism achieved in biological organisms could bring about scalability, fault tolerance and adaptivity to systems. These three possible benefits are discussed in further detail in this section.

4.2.1 Scalability

In Chapter 2, scalability and how it is lacking in EC, but is essential for evolving large complex designs was discussed. Use of modularity in evolutionary design was shown and suggested by many to speed up evolution and evolve higher complexity designs [Banzhaf et al., 2006; Haddow and Tufte, 2001; Kalganova, 2000a; Koza, 1994; Murakawa et al., 1996; Torresen, 1998; Vassilev and Miller, 2000b; Walker and Miller, 2004]. However, an explicitly defined mechanism that incorporates modularity into evolution can still not solve the scalability problem. A major limiting factor to achieving scalability is the direct genotype–phenotype mapping. A direct genotype–phenotype encoding causes the genotype to grow in proportion to the phenotype. This creates a large search space as the target system gets more complex, constricting evolution to small designs. [Banzhaf et al.,

2006; Bentley and Kumar, 1999; Gordon, 2005; Haddow et al., 2001; Miller and Thomson, 2003; Roggen, 2005; Vassilev and Miller, 2000b].

In nature, biological organisms achieve phenotypes specified by genes that are orders of magnitude smaller. An example of this is the human genome, which comprises approximately 30,000 genes, yet a human brain has roughly 10^{11} neurons [Braitenberg, 2001; Claverie, 2001]. It is noteworthy that the number of distinct cell types in human body is only 256, and this number is as low as 13–15 for the hydra, which is a predatory animal with regenerative ability [Kauffman, 1996]. The extremely complex structural and behavioural architecture of biological organisms is not through intelligent design, but emerges from the heavy reuse of cells and genes. Biology achieves a highly scalable mapping via multicellularity and gene reuse; each cell has the same copy of genotype, and each gene in a given genotype may have different effects depending on when and where they are expressed. Also, the same set of genes are used over and over again in building phenotypic structures of similar characteristics, e.g. limbs in animals. Taking inspiration from biology, the idea of multicellularity in achieving complex systems in EC has been implemented by many researchers. Although the ability of artificial development to be scalable has been demonstrated by simple experiments [Bentley and Kumar, 1999; Eggenberger, 1997; Gordon, 2005; Roggen, 2005], successful use of development in the design of systems at desired complexities that tackle real world problems–the solutions of which are competitive with already existing designs in engineering–is yet to be achieved.

4.2.2 Fault Tolerance

Biological organisms are robust creatures that can achieve very high level of fault tolerance. The hydra is an excellent example to organisms that display regenerative properties: it has the ability to regenerate even when cut in half, producing two hydrae [Bode, 2003]. The regenerative ability of plants is also an excellent example of fault tolerance and recovery in biological organisms. Plant cells are classified as totipotent[3], hence under the right conditions any plant cell would theoretically be able to grow into an adult plant [Leyser and Day, 2003]. One of the main reasons for the type of repair

[3]The ability of a cell to grow and generate all the specialized parts of an organism

and regeneration that happens in biological organisms is the lack of a central control mechanism. The ability of multiple cells to coordinate and organize themselves using various communication mechanisms provide an emergent adaptivity and fault tolerance in the whole organism.

Fault tolerant systems in electronics and computer science are important for remote, safety critical and hazardous applications. Almost all of the widely used techniques in achieving fault tolerance in electronics require a central control mechanism or a "golden" memory which is assumed to be failure-proof [Lala, 2001]. A de-centralized multicellular architecture can provide the system designed with redundancy, allowing the destruction of a number of cells before failure. In a multicellular design all cells are essentially identical to one another, hence a cell has the potential to change specialization and replace a damaged cell in order to recover from faults. This inherent multiple redundant behaviour in development can be used to create a system free of a single point of failure if each cellular structure is represented by an independent piece of hardware; hence removing the weakest link present in traditional redundancy designs.

In addition to cell redundancy, biological organisms also have functional redundancy in their genetic code. In biology this functional redundancy arising from different genetic codes is referred to as *degeneracy*. Edelman and Gally [Edelman and Gally, 2001] define degeneracy as "the ability of elements that are structurally different to perform the same function or yield the same output", and they also note that degeneracy "is a well known characteristic of the genetic code and immune systems." Edelman and Gally emphasize that degeneracy is a key mechanism for the robustness of complex mechanisms and that it is almost directly related with complexity. In biological organisms degeneracy is present at almost every functional level; from genes to high level behaviours like body movements and social behaviours [Edelman and Gally, 2001].

A developmental model can provide degeneracy both at the genotypic and phenotypic levels. Due to the indirect mapping of genes to the target phenotype, a developmental system can have multiple genes that perform the same function. Depending on their location in the organism each cell would have a different gene activity, but some of these cells would still have the same phenotypic functionality. Thus, degeneracy in developmental systems is a powerful fault tolerance mechanism, as it provides robustness

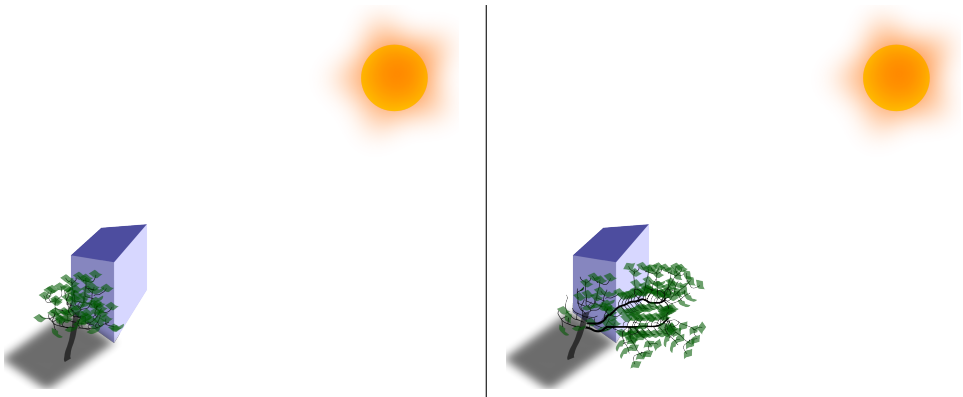
to genetic perturbations. An example of gene redundancy in biology is the control of platelet activation by collagen [Pearce et al., 2004].

Artificial development has been shown to provide a smoother degradation to perturbed genetic code [Bentley, 2005]; when the genetic code of an artificial organism is altered before mapping the genome to the respective phenotype, the damaged genome will provide a phenotype that shows a more "graceful" degradation for a developmental system in comparison to direct mapping of the genome. In fact in some cases it was shown that a small number of gene knock-outs did not affect the overall result of gene expression [Reil, 1999]. It has also been demonstrated that a developmental system may be able to recover from transient changes in the phenotype, despite sometimes not being explicitly trained to do so [Federici, 2004; Liu et al., 2005; Miller, 2004; Reil, 1999; Roggen, 2005]. However in order to benefit from the fault recovery properties of development, the developmental mechanism needs to be continuously running even when a fully functional phenotype is reached. In other words, the developmental system needs to have reached an *attractor*[4] that represents the desired phenotype. The ability of a developmental system to keep its phenotype unchanged after it has reached the target phenotypic form (i.e. represent the target phenotype as an attractor) is termed stability. To achieve a target phenotype at an attractor state of a developmental system (i.e. finding a stable system) is a harder task than simply achieving the target phenotype. Unfortunately, solutions that are not stable (i.e. occur as *transient* states) are of no use for fault tolerance or adaptivity.

4.2.3 Adaptivity

The adaptive behaviour of biological organisms is another attractive quality that is aimed to be captured in EC. Designing systems that change their structure to adapt to their environments is a very challenging task, especially when a lot of the environmental factors can vary unpredictably. Multicellular organisms achieve adaptivity smoothly, and they change their structures or behaviours to fit the given environment for maximum survival chances. An example of adaptivity in biology is the way the morphology of a plant depends on sunlight: if a plant "discovers" that there is an obstruction in the way

[4] An attractor is a state to which a dynamical system settles to after a time.



(a) A tree facing a wall that obstructs the sun. Only a very small portion of the tree from the right can see the sun.

(b) The tree leans and grows rightwards in order to receive more sunlight.

Figure 4.5: *The tree depicted in subfigure (a) uses the information from its leaves to maximize the received sunlight by changing its growth pattern as shown in subfigure (b).*

that blocks the sun, the plant will grow in a way to maximise sunlight exposure, see Figure 4.5.

It is intended that by modelling multicellular development an adaptive system will be achieved. However as mentioned in Section 4.2.2, in order to achieve an adaptive system the developmental system needs to be at a stable state when it achieves a functional phenotype. Once a stable state is achieved, the developmental process can run continuously in the background and adapt to environmental changes. A developmental system can have several attractors [Tufte, 2009]; in an ideal case a dramatic change in the environment making the current configuration ineffective will cause the developmental system move to another attractor that would suit the current environmental conditions better.

Again, degeneracy in a developmental system can also allow the system to be more adaptive, because of the existence of multiple implementations of the same function. For example a change in an environmental condition may affect the activation of some genes in a developmental system, however the functionality of the organism would still be protected due to the existence of other genes that serve the same purpose as the affected genes. Systems with high degree of degeneracy have been observed to be very adaptable in biology as well, and favoured by natural selection [Edelman and Gally, 2001].

4.3 Models of Artificial Development

In recent literature, developmental systems have sometimes been classified into two categories as *Grammatical* and *Cell Chemistry* developmental models [Chavoya, 2009; Federici, 2004; Flann et al., 2005; Stanley and Miikkulainen, 2003]. The reason for classifying developmental models into *Grammatical* and *Cell Chemistry* is because the grammatical models of development follow a high level abstraction of biology, whereas the models that involve cell chemistry follow a low-level abstraction of biological development, and these two models of development cover most of the artificial developmental models present in the literature. However, there are developmental models that do not fit either of these two classifications, e.g. in [Haddow and Hoye, 2007] the developmental system is designed in a way to function without the use of cell chemistry but the developmental model is far from a grammatical implementation. In order to make a similar but slightly clearer distinction amongst the present developmental models, the developmental systems are categorized into two different classes.

4.3.1 Macro-model Developmental Systems

A macro-model developmental system models the biological development at a high abstraction level, considering the overall behaviour of a biological organism or a developmental mechanism. A macro-model system's implementation is largely different to its biological inspiration, since the aim is to model the characteristic behaviour of the target developmental system/mechanism. Simply put a macro-model developmental system does not model individual cells in a multi-cellular organism, but provides a developmental behaviour in the system by the inclusion of time and ability to self modify over time. A widely known example of a macro-model developmental system is the Lindenmayer Systems (L-Systems) [Lindenmayer, 1968]. L-Systems, a parallel rewriting system, was introduced for modelling the growth processes of plant development using a set of rules via a grammar implementation, thus aiming to imitate biological development of plants using recursive functions. L-Systems have been applied to circuit design problems [Haddow et al., 2001; Kitano, 1998], neural networks [Boers and Kuiper, 1992],

and 3D morphology design [Hornby and Pollack, 2001; Sims, 1994]. Another example of a grammatical developmental system is Cellular Encoding (CE) [Gruau, 1994]. CE was designed to be used in the design of neural networks. Using CE, a neural net would learn recurrence and solve large parity problems such as a 51-bit parity problem [Gruau, 1994].

An example of a non-grammatical macro-model developmental system is self-modifying Cartesian Genetic Programming (CGP), which models a CGP system that could alter its own structure over time after the evolution phase is complete [Harding et al., 2007]. A macro-model developmental system should be computationally more efficient when compared to a micro-model developmental system in modelling developmental behaviour.

4.3.2 Micro-model Developmental Systems

A micro-model developmental system is a lower level implementation of the biological development that uses a bottom-up approach to modelling development. This category of developmental systems can also be seen as the more biologically plausible implementations, which imitate biological development at a cellular level. Hence a micro-model developmental system involves the modelling of individual cells and their interactions, which together make up a whole organism. Each cell in a micro-model developmental system has the same genotype and inter-cellular communication allows cells to specialize. All these cells together would form an organism which is the end product of development after each *developmental step*[5].

Although more biologically inspired, a micro-model developmental system does not necessarily model biological development perfectly. In fact there is much work in this type of artificial development with diverse design constraints, those that model biology closely [Eggenberger, 1997; Fleischer and Barr, 1993; Jakobi, 1995; Kitano, 1995; Kumar and Bentley, 2003b], those that aim to model biological development in a simplistic fashion [Roggen, 2005; Tempesti et al., 2003; Wolfram, 2002], and models that are "in-between" [Devert et al., 2007; Gordon, 2005; Haddow and Hoye, 2007; Miller, 2003]. Mimicking biology closely should provide a developmental system with high evolvabil-

[5]The time it takes to carry out all the developmental processes–such as the processing of the genome and cell signalling–once, is referred to as a developmental step.

ity [Bentley and Kumar, 1999; Dawkins, 2003], whereas a simplistic model would reduce the number of complicated processes that exist in biological development, reducing simulation times drastically. The first and one of the simplest examples to micro-model developmental system is Cellular Automata (CA), first developed in the 40s by Ulam and Neumann [N. and A., 1946; Neumann, 1966; Wolfram, 2002]. CAs model biological systems with a grid of cells that determine their states using the local information from their neighbours and a global rule; this way, CAs effectively model inter-cellular communication and cell specialization.

The process of designing an effective developmental model for EC reduces to implementing and sculpting the right biological mechanisms in an effective way. This aims to achieve an evolvable developmental system while maintaining a system that is not constrained or overwhelmed by undesirable biological processes. But how can we know which processes are useful and which are not? Implementations of artificial development in EC has been proposed since early 90's; e.g. [Dellaert and Beer, 1994; Fleischer and Barr, 1993]. But most of the developmental models designed still rely on educated guesses, and various assumptions on the suitability of the biological developmental processes for EC applications.

4.4 Summary

An introduction to the importance of multicellular development in biology and EC is accompanied by a discussion of developmental models in EC. The types of artificial developmental models that exist in literature are introduced and discussed in this chapter. Although development promises to provide an adaptive, fault tolerant, scalable and evolvable system for evolving computational models, the amount of information for the design of an ideal developmental system is very limited. The purpose of this chapter was to give the reader an idea of the potential benefits an Artificial Developmental System (ADS) might provide to EC, and introduce some of the related work that has been undertaken in the area. A new way of classifying the artificial implementations of biological development has been suggested. The new classification method aims to provide a clearer and more understandable distinction between different implementations of ADSs.

Chapter 5 will introduce a new artificial developmental system, and discuss the inspiration and reasoning behind many of the design decisions made for the developmental system. Chapter 5 will also discuss other artificial developmental models that exist in EC literature.

Publications II

The developmental model explained in the next chapter (Chapter 5) have been has been used in these published papers:

Kuyucu, T.; Trefzer, M.; Miller, J. F. & Tyrrell, A. M. On The Properties of Artificial Development and Its Use in Evolvable Hardware. *IEEE Symposium Series on Computational Intelligence - IEEE SSCI 2009*, 2009, 108-115

Trefzer, M.; Kuyucu, T.; Miller, J. F. & Tyrrell, A. M. A Model for Intrinsic Articial Development Featuring Structural Feedback and Emergent Growth. *IEEE Congress on Evolutionary Computation – IEEE CEC 2009*, 2009

Kuyucu, T.; Trefzer, M.; Miller, J. F. & Tyrrell, A. M. A Scalable Solution to N-bit Parity via Articial Development. *5th International Conference on Ph.D. Research in Microelectronics & Electronics – PRIME 2009*, 2009

Trefzer, M. A.; Kuyucu, T.; Miller, J. F. & Tyrrel, A. M. Image Compression of Natural Images Using Artificial Gene Regulatory Networks. *GECCO'10*, 2010.

The next chapter does not share content with the above publications where Dr. Trefzer is the first author.

Chapter 5

Modelling Multicellular Development

The development of an organism is an ongoing process throughout the whole lifetime of the organism, even when a perfectly functional adult organism is present. The developmental system in an organism is supported by various processes; these processes keep the system stable in an ever changing and noisy environment. When modelling biological development as an algorithm, there are various decisions that can drastically affect the performance to achieve these desired properties of the model. An important step before designing an artificial model of development is to decide whether a macro or a micro model approach will be taken. As a more biologically plausible approach, a micro-model Artificial Developmental System (ADS) should be more evolvable [Bentley and Kumar, 1999; Dawkins, 2003] and provide a more effective way of modelling multicellularity than a macro-model, therefore the model presented in this thesis is chosen to be a micro-model. A micro-model multicellular developmental system has various mechanisms, such as the "cell controller", cell signalling, and growth, which can be implemented in a variety of ways. The design of these developmental mechanisms will also have a large effect on the evolvability of the ADS achieved.

In Chapter 4, biological development was introduced and discussed as a source of inspiration for the evolution of "developmental organisms" in Evolutionary Computation (EC). The different artificial implementations of biological development were discussed and categorised under two classes as; Macro-model and Micro-model developmental systems. In this chapter, the design of a new ADS is described. Before describing the model and the design approach for the model used in this thesis, micro-model developmental systems are described in further detail in Section 5.1. A selective list of ADSs is compiled in Table 5.1 detailing some of the design choices for each ADS.

5.1 Micro-model Developmental Systems

The core component of a biological developmental system is the *Gene Regulatory Network (GRN)*, Section 5.1.1. GRN provides the control of a single cell, and the single cell is integrated into a multicellular environment via *cell signalling*, Section 5.1.2. The emergence of multicellular organisms from a single cell is possible through the *growth and cell division* processes, Section 5.1.3. These mechanisms, which are important part of micro-model developmental systems, are described and their uses in existing ADSs are discussed along with methods of *genotype–phenotype mapping* (Section 5.1.4) under this section.

5.1.1 Gene Regulatory Network

As explained earlier in Section 4.1, a Gene Regulatory Network (GRN) is the heart of a biological developmental system. Similar to the different artificial implementations of biological development, artificial implementations of biological GRNs can have various levels of detail with or without certain biological GRN mechanisms; GRNs have been modelled using simple boolean rules [Dellaert and Beer, 1994; Kauffman, 1969], as well as detailed simulations [Kumar and Bentley, 2003b; Reil, 1999].

When the aim of modelling a GRN is accurate simulation of a biological network, as much detail as possible in the model needs to be included. Often differential equations describing the dynamic system formed by the interactions of chemicals and genes are used for the latter case. Some researchers have also looked at using differential equations for modelling GRNs as part of an ADS in EC [Kitano, 1995; Steiner et al., 2007]. These detailed and accurate models provide interesting insights into the benefits of continuous models, but because such models are computationally expensive, simpler models may be more successful in solving computational problems.

Simpler and more abstract models of GRNs are more appropriate for evolutionary optimisation and design where the intention is to acquire prevalent properties of GRNs rather than model them in detail. Rule based GRN models are the most common implementations in EC; this is true for most of the models listed in Table 5.1. This is due to the highly

flexible implementation of the rule based GRN models that allow easy modification after the initial design phase.

GRNs can be seen as the "*controller*" of the cell, hence in artificial development it is possible to use any processing mechanism to replace the GRN core of a biological cell in the simulated environment. There are different cell controllers in the developmental systems present in literature ranging from combinational circuits to bio-inspired GRNs, see Table 5.1. With current knowledge it is not possible to tell what type of cell controller is best suited for an ADS designed for EC. The advantages of evolving a combinational circuit controller is its ability to be implemented in hardware, and also the well established successful training mechanisms for small combinational circuits such as [Miller and Thomson, 2000; Stomeo et al., 2005; Walker et al., 2006]. Evolution of Artificial Neutral Networks (ANN) has also been well developed, and there exist many tools and mechanisms for the evolution of complex ANNs, such as [Arad and El-Amawy, 1994; Stanley and Miikkulainen, 2002]. Although both combinational circuits and ANNs are more widely used and established in EC, GRNs also have their advantages. GRNs (or GRN inspired computational models) are a common way of modelling a cell control mechanism within a developmental system. GRNs are highly dynamic computational networks that provide complex dynamics. A system with complex dynamics may benefit from the ability to change its state with the changing environmental conditions without the need to retrain, which can create an adaptive mechanism within the system. The highly dynamic nature of a GRN can provide multiple attractors for the same genotype allowing adaptation, differentiation, and robustness against disturbances and loss of functionality in genes [Reil, 1999]. Another advantage of a GRN based controller is that a GRN can be highly evolvable; a biologically inspired computational model may be more evolvable than a traditional computation model such as circuits. This is due to biological processes being a product of evolution itself [Dawkins, 2003].

In summary, different implementations of cell controllers aim to achieve similar objectives, and the choice of controller type is up to the researcher. In most of the models listed in Table 5.1, the cell controller is a model of the biological GRNs. Although each of the GRN models use various implementation approaches, such as Differential Equations (DE) or rule-based formalisms, they all aim to create a dynamic network of auto-regulatory genes. A numerical implementation of GRNs has advantages such as

better precision in predictions (if modelled correctly). Whereas, a qualitative model like rule-based GRNs can incorporate more detailed and varied biological knowledge into the system. For a more detailed information on various ways of modelling GRNs see [de Jong, 2002].

A developmental model that was not listed in Table 5.1, but has been shown to provide successful results is the model designed by Roggen [Roggen, 2005]. The system designed by Roggen lacks the use of any cell controller, and relies completely on the effects of growth and cell signalling (diffusion). In this model it was assumed that the cell controller is not needed for the developmental system designed. The system would always start with a group of pre-placed diffusers in a cellular grid, and the chemical diffusing from these diffusers would activate cells on contact. The state of each cell is determined by the concentration of the chemical present in the cell. Hence, there is a lack of gene processing unit in each cell, and the genotype may only represent the location of diffusers in the cell grid and the contents of a chemical level to function block translation table.

5.1.2 Cell Signalling

Cell signalling is an important part of development in multicellular organisms. It controls cellular activities and coordinates the cells in an organism. In biology, the cellular communication is established via the use of chemicals, which carry information depending on their type and intensity (concentration). In biology there are three main modes of signalling:

1. **Direct Contact:**

   This type of cell signalling only involves the cells in direct contact of each other. The signalling chemical does not diffuse from the cell producing it, but travels to the cell in direct contact of the host cell; hence no other cells would be affected by the signals. Fagotto in [Fagotto and Gumbiner, 1996] refers to contact signalling in cells as "emerging as a major mechanism of communication in developing tissues". Thus contact signalling is a crucial part of the initial stages of multicellular development in biology.

2. **Short Distance:**

   Short distance signalling involves the diffusion of chemicals from a cell to its neighbours. The diffusing chemicals degrade quickly limiting their effective area to a small neighbourhood of cells. Short distance signalling complements contact signalling mechanism in embryonic patterning [Fagotto and Gumbiner, 1996].

3. **Long Distance:**

   Long distance signalling can also be important for a multicellular organism in regulating the development and coordination of the overall organism. In animals, the endocrine system is the mechanism that establishes a long distance chemical signalling for cells. In plants auxins are thought to be inter-cellular messengers for long distance signalling. Through auxin signalling, cellular patterning, and meristem and vascular development are thought to be mediated [Berleth and Sachs, 2001].

   Coordination of large number of cells via simple chemical diffusion is not possible, and a more complex system like the endocrine or auxin is needed. It is noted in [Fagotto and Gumbiner, 1996] that; "unrestricted diffusion is often undesirable in embryos in which small ensembles of pluripotent cells are required to respond only to local signals for proper patterning, even if they express a large number of different surface receptors." However long distance communication of cells is a more complex mechanism than the short distance and direct contact signalling mechanisms, and its role in development is less understood in biology.

Direct contact signalling is the more common signalling mechanism in ADSs that model multicellular development: it can be observed in Table 5.1 that the listed models prefer direct contact signalling more often than short distance (diffusion) signalling. Models that use direct contact signalling only are: [Dellaert and Beer, 1994; Devert et al., 2007; Gordon, 2005; Haddow and Hoye, 2007; Tufte and Haddow, 2003]. The majority of these models, [Dellaert and Beer, 1994; Haddow and Hoye, 2007; Tufte and Haddow, 2003], that only use direct contact signalling are developmental models with similar characteristics to cellular automata, which specifically model the developmental effects of neighbourhood communication. Models that only implement direct contact signalling, also do not need to model graded chemicals in their system. The lack of simulating chemicals in an artificial model creates a simpler design.

An advantage of using boolean interactions instead of simulating graded chemicals in a developmental system is the decrease in the processing time and complexity of the developmental model [Gordon, 2005; Haddow and Hoye, 2007]. Other examples of ADSs using boolean interactions include [Bentley and Kumar, 1999; Dellaert and Beer, 1994; Tufte and Haddow, 2003]. In fact [Bentley and Kumar, 1999] and [Tufte and Haddow, 2003] only use the local phenotype state to model regulation rather than using explicit communication strategies. On the other hand modelling chemicals in a developmental system provides the ability to have more precise interactions amongst genes and cells. All the other researchers listed in Table 5.1 made use of the graded chemicals in their systems. In the earlier models, Kitano and Jakobi used a complex model of response strategies that may have over-complicated the system [Jakobi, 1995; Kitano, 1995]. Eggenberger's work that followed Kitano and Jakobi succeeded in providing impressive results, and this model had provided a much simpler response strategy [Eggenberger, 1997].

Modelling chemicals in an ADS allows the use of communication via chemical diffusion in the system. Chemical diffusion has been used successfully as an organizing mechanism on its own [Roggen, 2005], thus it is a mechanism worth implementing as part of a developmental system. Most of the developmental models presented in Table 5.1 also use graded chemicals as part of their system, and achieve promising results. Steiner et al. [Steiner et al., 2007], Jakobi [Jakobi, 1995], and Zhan et al. [Zhan et al., 2008] only use diffusion based communication mechanisms to achieve multicellular coordination and differentiation in their models. Kitano, [Kitano, 1995], mentions that diffusion in correct proportions is beneficial to achieving useful developmental behaviour. However, developmental systems that do not use chemicals have also been shown to demonstrate scalability and stability in EC [Gordon, 2005; Haddow and Hoye, 2007; Tufte and Haddow, 2003].

In the experiments provided by Flann [Flann et al., 2005] direct contact signalling was the more effective signalling method when compared with diffusion in achieving various patterns. All the pattern types; mosaic, patch, and border patterns were successfully achieved using contact signalling only. Whereas diffusion signalling was only effective for some patch patterns and the border patterns [Flann et al., 2005]. However, diffusion may be useful for controlling the growth of an organism. A gradient of chemicals can be created via diffusion amongst the cells that can provide the necessary information for

stopping growth of a developmental organism before the entire organism space is filled. Miller had noted that without the presence of graded chemicals, it would be "unlikely, if not impossible, to achieve solutions that grow and then stop growing that meet the target objective" [Miller, 2004].

The need for graded chemicals have been partially investigated by Haddow and Hoye [Haddow and Hoye, 2007] after an earlier attempt by Miller [Miller, 2004]. Miller later realised that his experiments used boolean regulation for chemicals rather than graded regulation, i.e. the developmental model under test did not make use of the graded chemicals. Haddow and Hoye [Haddow and Hoye, 2007] concluded that increasing the number of chemicals in the developmental model inversely affected the performance of the system in achieving target patterns.

The GRN model presented by Haddow and Hoye uses a unique approach to modelling GRNs: only the cell state of the neighbouring cells are used for inter-cellular communication, and the chemicals that are produced inside a cell as a result of gene transcription are only used within the cell for self-regulation of genes. Unlike most other GRN models that use graded chemicals, Haddow and Hoye's model does not represent proteins as chemicals but as genetic actions, and there is no chemical diffusion or any other form of communication involving chemicals. Hence graded chemicals are only used for representing information within a cell. Due to the minimalistic role of chemicals, they increase the complexity of the search space rather than providing finer tuned information processing in the GRN and the developmental system [Haddow and Hoye, 2007].

5.1.3 Growth/Cell Division

Growth and cell division in biological development is a key process that enables a single cell organism to transform into a multicellular organism. Cell division creates a copy of a single cell with both the parent and daughter cells sharing the exact copy of the genome. The number of cells in an organism grows due to cell division. Growth is the change in the overall size of the organism due to multiplying numbers of cells as well as physical growth in the size of individual cells. An organism may grow to a predefined size that is mostly determined by the genotype e.g. animals, or its size may be highly dependent on

Table 5.1: *List of some of the micro-model artificial developmental systems specifically designed for computational problems with a list of the common developmental mechanisms used by each model. Evidently this list is not a comprehensive cover of all the major artificial developmental systems. There were several criteria in determining which developmental systems to include: chronological spread of the designs and a spread of researchers involved in the design of the developmental system were part of the criteria. Another criterion was to include systems specifically designed to solve computational problems using multicellular development. Space was also a limiting factor as well as the listed criteria. The developmental systems listed on this table will be referenced throughout the rest of the paper.*

Developmental Model	Cell Controller	Communication Mechanisms	Growth/ Division	Cell Structure	Graded Chemical Use	Target Phenotype	Stable	Robustness
Dellaert and Beer 1994 [Dellaert and Beer, 1994]	RBN	direct	YES	emergent	none	2D patterns (control)	none	none
Jakobi 1995 [Jakobi, 1995]	GRN	diffusion	YES	functonal proteins	present (robot control)	ANN	none	none
Kitano 1995 [Kitano, 1995]	GRN	diffusion & direct	YES	protein concentration	present	ANN	none	none
Eggenberger 1997 [Eggenberger, 1997]	GRN	diffusion & direct	YES	functonal proteins	present	3D patterns	none	none
Bentley and Kumar 1999 [Bentley and Kumar, 1999]	rules (CA)	direct & routed	YES	emergent	none	2D patterns	none	none
Miller 2003 [Miller, 2003]	CGP (Circuits)	diffusion & direct	YES	circuit output (emergent)	present	2D patterns	upto 10 steps	transient fault recovery
Tufte and Haddow 2003 [Tufte and Haddow, 2003]	rules (CA)	direct	YES	emergent	none	2D patterns on hardware	none	none
Federici 2004 [Federici, 2004]	ANN	diffusion & direct	YES	NN output (emergent)	present	2D patterns	upto 2 steps	transient fault recovery
Gordon 2005 [Gordon, 2005]	GRN	direct	YES	protein concentration	none	circuits	none	none
Haddow and Hoye 2007 [Haddow and Hoye, 2007]	GRN	direct	YES	functional proteins	optional	3D patterns	upto 10 steps	none
Devert et al. 2007 [Devert et al., 2007]	ANN	direct	NO	NN output (emergent)	present	2D patterns	stability checked	transient fault recovery
Steiner et al. [Steiner et al., 2007] 2007 [Steiner et al., 2008]	GRN	directable diffusion	YES	functional proteins	present	3D pattern	stability checked	resistance to unwanted mutations
Zhan et al. [Zhan et al., 2008] 2008 [Zhan et al., 2009]	GRN	controlled diffusion	YES	protein concentrations	present	Circuits	stable protein levels	transient recovery

the environmental conditions e.g. ivy. Hence depending on the nature of the organism and the environment, the growth and cell division processes may never stop.

All the ADSs listed in Table 5.1 feature some form of growth and cell division except the model designed by Devert et al [Devert et al., 2007]. Cell division and growth in ADSs is a way of controlling the number of cells active in the organism, and it provides the opportunity for a system to expand when extra resources are provided, hence adapting to its new environment. However, growth and cell division processes add extra complexity to the developmental system and they increase the simulation time and extend the developmental steps required to reach a mature organism. If the number of cells required for the target organism's size is known and is equal to the maximum number of cells available then eliminating the growth and cell division mechanisms from the developmental system may be beneficial.

5.1.4 Genotype-Phenotype Mappings

There are various implementations of mapping the developmental interactions to phenotype in the literature. Some systems (especially ones that do not use GRNs), interface the regulatory elements of the developmental system with the phenotype by using the phenotypic information as a regulatory element. In these models information from the phenotype is retrieved to be used as a regulatory input to the developmental system, and an output of the developmental system is used as a feedback to alter the phenotype directly. Dellaert and Beer use this approach in their Random Boolean Network (RBN) based developmental model [Dellaert and Beer, 1994], as well as Bentley and Kumar, Tufte and Haddow who use CAs in their developmental models [Bentley and Kumar, 1999; Tufte and Haddow, 2003]. Miller's circuit based, and Federici's ANN based developmental models also interface phenotypic information directly to cell regulation [Federici, 2004; Miller, 2003]. Haddow and Hoye also use this mechanism in their GRN based model [Haddow and Hoye, 2007].

In the remaining of the developmental systems listed in Table 5.1, only artificial proteins/chemicals are used to define developmental environment, and only this information is used by the genes. Some of these systems use predefined proteins/chemicals as

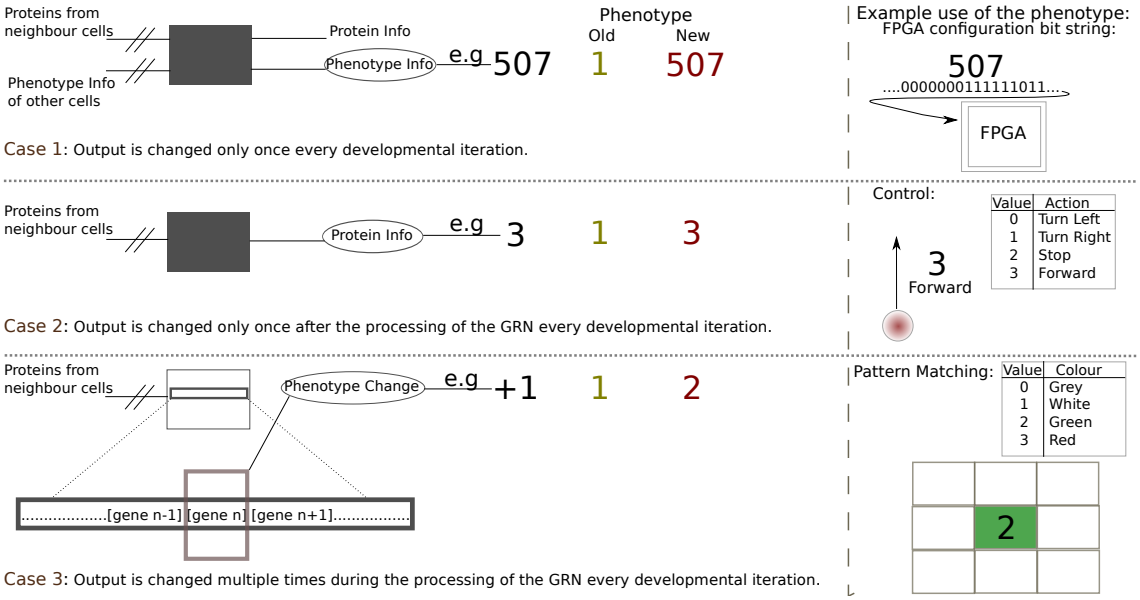


Figure 5.1: *Three common ways of obtaining an output from an artificial developmental cell, and using the output to build/change the organism phenotype. The examples shown are for clarity and do not directly correspond to a real system. The output from the first two cases are obtained at the end of of developmental step, and the system in case 3 does not have an output, but directly alters the organism phenotype during the processing of its genotype (when the structural proteins are transcribed). An good example to a external inputs are sensors.*

either structural or regulatory proteins, and others allow all proteins to perform both structural and regulatory functions. The system of Devert et al. is an exception to this, since it uses the chemicals from neighbours as inputs to an ANN in order to determine the phenotype and output chemicals of a cell [Devert et al., 2007]. In the rest of the approaches the structural proteins are interfaced to the phenotype in two different ways. They are either used as functions that are part of the transcription process that make use of the genetic information in a gene to alter the phenotype [Eggenberger, 1997; Jakobi, 1995; Steiner et al., 2007] or they are used as phenotypic outputs of the developmental system and mapped directly (or after some post processing, e.g. via a look up table) to the phenotype at the end of a developmental phase [Gordon, 2005; Kitano, 1995; Zhan et al., 2008].

Figure 5.1, provides an illustration of the three methods of mapping the developmental interactions of a cell to the organism phenotype.

5.2 The Artificial Developmental System

The ADS that is used in this thesis will be described in this section. As noted earlier, the ADS used in this thesis is a micro-model that uses an artificial GRN as the cell controller. The GRN is implemented as a rule based formalism. Each gene is treated as a rule with a condition and a result (i.e. pre and post-conditions). A gene will be activated when the precondition is met, and when a gene is active its postcondition will be processed. The precondition of a gene specifies the proteins (and other chemicals) that need to be present or absent in order to activate the gene. Thus a protein (further detailed in Section 5.2.2) can inhibit, enhance or have no effect on the activation of a gene. The postconditional part of a gene specifies which protein to produce, and the action to be taken by the produced protein. In the model presented here some of the proteins have functional tasks, but all of the proteins participate in regulating the activation of genes. Hence, there is no difference in the regulatory or behavioural proteins as all proteins are regulatory. A short pseudo-code description of the simulation process of the GRN is detailed in Algorithm 2.

The developmental system presented here features cell division and growth as these are inspirational parts of biological development. The process of initiating cell division is done as a cellular actions, which is further detailed in Section 5.2.3. The developmental system provides time dependent development where each developmental step the GRN of each cell is processed. Every time step the developmental system progresses a step further, making it *older*. Therefore, a developmental organism is always assigned an age, which is incremented every developmental step.

The model presented here makes use of graded chemical regulation of genes; all the proteins are treated as chemicals. Only direct contact and short distance signalling are used as cell signalling mechanisms in the ADS. Since there is still little understanding for the capabilities of ADSs in solving engineering and computer science applications, the described ADS is not expected to be capable of designing organisms complex enough to benefit from long distance signalling. As stated in Section 5.1.2, long distance signalling is the least understood cell signalling mechanism in biology, and the least modelled communications in ADSs. Hence it is also convenient not to implement it as part of the ADS, which will result in a simpler design.

The direct contact mechanism implemented in the model here was inspired by the Plasmodesmata in plants. Plasmodesmata are microscopic channels that breach the cell walls of plant cells forming a tunnel between two cells, hence enabling the passage of chemicals between them [Leyser and Day, 2003]. There are two ways of creating plasmodesmata, they are either formed during cell division or between two mature cells. As described in Section 5.2.2 the direct contact signalling is implemented as a result of the synthesis of a plasmodesma protein, which can create a tunnel connecting the two cells. When a tunnel between two cells is established a free flow of chemicals between the two cells is allowed, hence the two cells share the same concentration of chemicals. It is possible that a cascading tunnels emerge in an organism connecting a large neighbourhood of cells via these direct tunnels. In such a situation two cells that are physically far apart can be considered connected indirectly via these cascade of tunnels. Therefore, the direct contact mechanism implemented in the model here can turn into a distant signalling mechanism as well.

The other signalling mechanism implemented is the short (or medium) distance signalling, which is modelled as diffusion. The diffusion process is carried out for all chemicals available in every cell; half of an available chemical diffuses out of a cell equally to the four nearest neighbours, i.e. each neighbour obtains $\frac{1}{8}$ of the cell's chemicals. This diffusion model used is unrestricted diffusion, and although this model has been commonly used in developmental models before [Liu et al., 2005; Miller, 2003; Tyrrell and Sun, 2006; Zhan et al., 2008], it may be undesirable in artificial development as well for similar reasons to biology. As mentioned earlier, in biology it is noted that an unrestricted constant diffusion is often undesirable [Fagotto and Gumbiner, 1996].

Change in chemical levels due to diffusion and direct contact signalling are adjusted at the end of a developmental step for all cells. Thus, the GRN always works with the chemical levels that are determined at the start of a developmental step; this aims to reduce the bias in the course of development due to the order of cell updates.

5.2.1 Gene Representation and Processing

The genotype is represented as a chain of binary strings, and each binary string represents an individual gene in the genome. A fixed length binary implementation of the genes

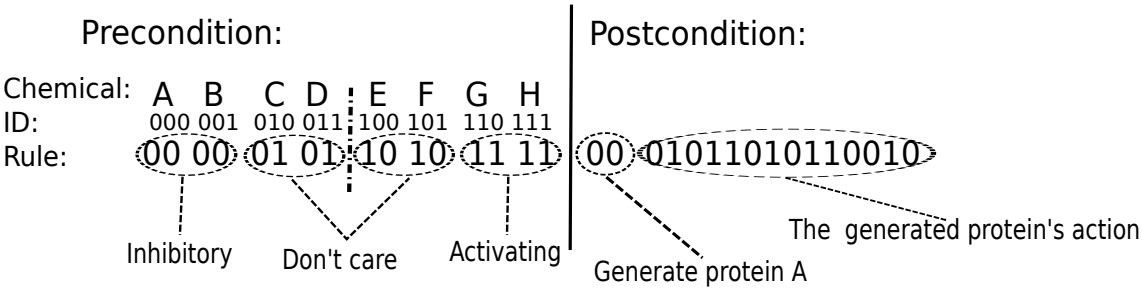


Figure 5.2: *An example gene of 32 bits is shown. The first 16 bits are reserved for the preconditional part, which specifies the rules to activate the gene. There are 8 chemicals defined in this figure; the first 4 being reserved for proteins, while the last 4 are messenger molecules, see Section 5.2.2. Each chemical's required presence or absence is specified by a 2 bit number, which provide two don't care states. In the event of a don't care state, the presence or absence of a chemical has no effect on the activation of the particular gene. The second 16 bits of the gene is reserved for the postconditional part, which provides the ID of the chemical produced as a 2 bit number (this means that only the first four chemicals [proteins] can be produced, i.e. the messenger molecules can not be produced via the activation of a gene), which is then followed by a 14 bit number. The last 14 bits in the gene define the action of the chemical produced if it has one (further explained in Subsection 5.2.2), if not the last 14 bits are treated as junk.*

was chosen to make the application of the GRN model straightforward on an embedded processor. Similar to the gene representation used by Gordon [Gordon, 2005], each gene is modelled as a rule with two sections; pre and postconditional sections. The preconditional part of a rule specifies whether the activation of the rule could be enhanced, inhibited or unaffected by the presence of known chemicals. For a gene to be activated all the activating chemicals must be present and all the inhibiting chemicals must be absent. In other words a logic *AND* is used as the precondition function. Each chemical has a concentration, and it is considered to be present if its concentration is at or above a predetermined threshold level, otherwise it is considered absent. Chemical concentrations are represented as 8-bit integer values, hence the possible values are between 0 and 255. All of the processing in the ADS is done using binary and integer operations. The postcondition of a gene defines the chemical that is produced, and depending on the type of chemical produced, the information for the action it is going to take is also included within the postcondition. Encoding of an example gene is illustrated in Figure 5.2. More detail on the types of chemicals is provided in Section 5.2.2.

The GRN is processed synchronously, from the start of the genotype to the end. The GRN of each cell is also processed synchronously, starting from the top left cell and moving from one cell to another (left to right and top to bottom). This means that the results of a processed ADS are dependent on the order of genes in the genome as well as the order

of cells in the organism. Although this diverts from biology, the processing of the ADS becomes straightforward. The GRN of each cell is processed every developmental time step which synchronises the organism. The changes to the chemical concentrations due to gene activations are done in real time, but the changes caused via cell signalling are recorded in a temporary buffer and made at the end of each developmental step. Whether a chemical will bind a gene is dependent on the concentration level of the chemical and the consumption rate of the chemical. In a cell (i), the concentration level (Z_{igtc}) of a chemical (c) by the time it reaches a gene (g) depends on its initial concentration (X_{itc}) at the start of the developmental step (t), the amount of the chemical used by the previous genes in the genome (B_{igtc}), and the concentration of chemical produced by the previous genes in the genome (O_{igtc}). At age 0 the concentration levels of all chemicals are 0, and the consumption rate (R_{gc}) of a chemical is a constant (either hard-coded or evolved) that specifies the amount of a chemical consumed if the chemical binds a gene.

$$Z_{igtc} = X_{itc} + O_{igtc} - B_{igtc}, \tag{5.1}$$

Whether a chemical will bind a gene is a boolean operation (Y_{igtc}), and it depends on available chemical concentration, and chemical consumption rate.

$$Y_{igtc} = \begin{cases} true & \frac{Z_{igtc}}{R_{gc}} \geqslant 1 \\ false & \frac{Z_{igtc}}{R_{gc}} < 1 \end{cases}. \tag{5.2}$$

5.2.2 Protein Synthesis

Biological protein synthesis involves the process of a protein transcribing a gene (binding the promoter sequence to activate the gene), where the genetic information (postconditional region) is copied into a mRNA (messenger RNA[1]). This is followed by the mRNA strand moving out of the nucleus of a cell to meet a complimentary tRNA (transfer RNA). Each tRNA molecule compliments the opposing bases on the mRNA strand perfectly. These in turn have the amino acid sequence to successfully code for a particular amino

[1] Ribonucleic acid: a nucleotide chain that is transcribed from and similar to DNA, but has small structural differences.

acid which forms a protein. Hence the amino acids (aka peptides) from the tRNA and mRNA combine to form a polypeptide chain (proteins), and can be used in a variety of structures such as enzymes and hormones to carry out a cell function. Although protein synthesis in biology covers a series of complex processes, simply put it is the phase of building proteins.

Algorithm 2 The pseudo-code for the simulation of Gene Regulatory Network for one time step.

```
1: ORGANISM stores all the cells available for development up to MAXCELL;
2: CELL is an individual entity that models a biological cell
3: for each CELL cell in ORGANISM do
4:   if cell is ALIVE then
5:     for each GENE g in GENOME do
6:       if (promoter chemicals for g) > PROMOTE-THRESHOLD then
7:         promote ← true
8:       else
9:         promote ← false
10:      end if
11:      if (inhibitor chemicals for g) > INHIBITTHRESHOLD then
12:        promote ← false
13:      end if
14:      if promote then
15:        comment: Process the postcondition of g
16:        proteinProduced ← protein type specified by postcondition of g
17:        increase proteinProduced in cell by PROTEINPRODUCTIONRATE
18:        comment: Check if proteinProduced has any functions to carry out
19:        if proteinProduced = PLASMODESMA then
20:          tunnelDirection ←information from g on the direction of tunnel
21:          tunnelOpen ←information from g whether to block or form a tunnel
22:          if cell's neighbour in direction tunnelDirection is not ALIVE then
23:            if tunnelOpen then
24:              grow a new cell in direction tunnelDirection and
25:              form a plasmodesmata between cell and the new cell
26:            end if
27:          end if
28:        else if proteinProduced = STRUCTURING then
29:          extract information from g to modify cell structure
30:        else if proteinProduced = SENSOR then
31:          extract information from g for monitoring the cell surroundings
32:        end if
33:      end if
34:    end for
35:  end if
36: end for
```

In the ADS described here, protein synthesis does not involve the simulation of RNA molecules. Once a gene is activated with the correct binding of proteins the postconditional part of the gene is processed, i.e. the gene is transcribed. The first part of the postconditional part of a gene specifies the ID of the protein to be produced. The protein produced is supplied into the cell at a predetermined protein production rate. Once the protein produced is added to the cell protein repository, the rest of the gene postcondition is processed depending on the type of protein produced. The production (synthesis) of a protein triggers a protein action in the cell. For this action, the protein produced uses the remaining information provided by the gene it was produced by to coordinate its actions. The chemical types (including proteins) and their actions are detailed within Section 5.2.3.

5.2.3 Chemicals

The GRN system implementation used in the ADS described here makes use of various chemicals that are used for gene regulation; affecting cellular functions, and communication. Chemicals are classified in two groups as *proteins* and *molecules*. The proteins build the organism, create a regulatory network, and are used for multicellular communication. The molecules are used as part of the regulatory system as well, and they help monitor the environmental changes for regulatory adaptation.

5.2.3.1 Plasmodesma Protein

Plasmodesma proteins are inspired by the *Plasmodesmata* in plants (similar to the gap junction in animals). When a plasmodesma protein is synthesised, it forms a tunnel in one of the four cardinal directions (North, South, East, West), and if the neighbouring cell also has formed a tunnel in the corresponding direction, the two tunnels join together allowing the passage of proteins between the two cells (the joint tunnels are termed plasmodesmata in this thesis). After the two cells are connected by plasmodesmata (tunnels), they share the same protein distribution. If the neighbouring cell space (of the cell where the plasmodesma is produced) is not occupied, a cell division is initiated. During the cell division plasmodesmata is created connecting the two daughter cells (one of which now fills the empty cell space). Although in plants the plasmodesmata size is known

to shrink as the cells mature (thus filtering larger proteins) [Leyser and Day, 2003], this is omitted in our developmental system for simplicity. However, the callose deposition is simulated. The callose deposition causes the blockage of the plasmodesmata, which prevents the transport of all proteins through the plasmodesmata. In our system this is initiated by a specifically encoded plasmodesma protein.

5.2.3.2 Structuring Protein

Structuring proteins are the type of proteins that build and change the physical structure of the cell. When a structuring protein is produced, it uses the information provided by the gene to alter the physical structure within the cell. The physical structure of a cell refers to the cell's role in the application domain, e.g. a component part of a digital circuit, a control system or an image pixel. When a structuring protein is produced, the cell phenotype is altered. For example, if the cell phenotype is represented as a 32-bit integer, a transcribed structuring protein may make bitwise changes in the 32-bit number using the postconditional information provided by the gene producing the structuring protein, see Figure 5.2. This provides an emergent structure for the cells in the developmental system, aiming to create an effective mapping of developmental dynamics to organism structure.

There is still little understanding on how to map the dynamics of a developmental system to engineering applications. Hence, it is not possible to distinguish between the different methods of processing and mapping the developmental information to a phenotype. Using specific proteins that can alter the phenotypic structure of the cell during the ongoing developmental process has the potential to extract more information from the dynamics of the ADS rather than using the protein concentration values at the end of a developmental stage.

We reviewed various genotype-phenotype mappings used in micro-model developmental systems in Section 5.1.4. However, in such developmental approaches it is still not known how best to map the dynamics of a developmental system to engineering applications. Hence, it is not possible to distinguish between the different methods of processing and mapping the developmental information to a phenotype.

5.2.3.3 Sensor Protein

The sensor proteins are produced by the GRN to act as sensors around the cell by monitoring the external environmental changes in the phenotype domain. The sensor proteins produce different kinds of messenger molecules for different outside activity. This enables environmental factors to affect the GRN activity, possibly creating a more interactive developmental system.

5.2.3.4 Regulatory Protein

Although every protein is a regulatory protein, there are proteins that only exist for regulatory purposes. These proteins, just like every other protein, control the GRN activity by their presence or absence, but they do not have any other purpose. Thus when a regulatory protein is produced by a gene, the information provided by the post-condition of the gene is discarded. The unused parts of the gene provide a neutral search space, which can have positive effects on evolvability. It has been shown that neutrality provides stability via preventing deleterious mutations, and helps evolution avoid local optima, thus creating a smoother search space [Harvey and Thompson, 1997; Vassilev and Miller, 2000a].

5.2.3.5 Messenger Molecule

Apart from the four proteins introduced, there is another type of chemical called a messenger molecule; molecules have different regulatory properties compared to proteins. The messenger molecules are produced by the sensor proteins that monitor the phenotype domain when present. The messenger molecules can not be produced directly as a result of gene activity (protein synthesis) since their purpose is to regulate the gene activity via the environmental response, but they can still bind to activate or deactivate genes.

It is intended that these chemical types will give the cells the ability to form an interactive multicellular organism that can achieve scalability, adaptivity, and fault tolerance. Figure 5.3 portrays the role of each protein in a simple multicellular organism.

Pseudo code summarizing and describing the implementation of the ADS is presented in Algorithm 3. This algorithm only details the functions that are executed during each developmental step. It refers to the gene regulatory activity in each cell as a function call to **GRN()**. The pseudo-code description of the gene regulatory network simulation in the organism is detailed in Algorithm 2. Lines 8–13 of Algorithm 3 detail the cell signalling procedures. When two cells are connected via tunnels (plasmodesmata) the concentrations of their chemicals are combined and shared equally to the two cells. This

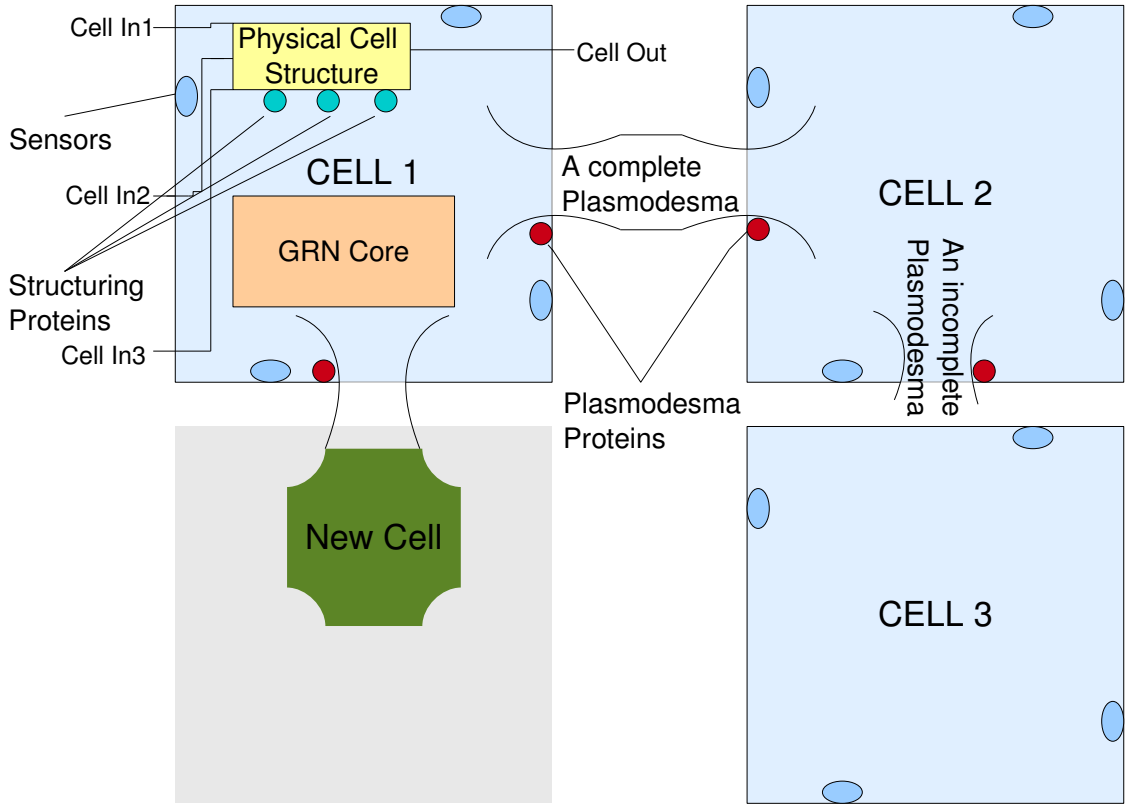


Figure 5.3: *In a multicellular environment using the 4 basic protein types a cell is able to: interact with its environment, grow, structure itself, and form a multicellular organism. The basic functions of the listed proteins are demonstrated in this figure. Only cell 1 is drawn completely, certain components are omitted in other cells. In the actual implementation of the organism there are no spaces between cells, they are only separated by their borders. In the example above, cells 1 and 2 both have active plasmodesma proteins, which cause the formation of a channel on both cells towards the other, creating a plasmodesmata to allow free movement of proteins from one cell to other. Cells 1 and 2 both also have active plasmodesma proteins on their southern sides. Cell 1's southern neighbour does not exist, so the active plasmodesma protein initiates a growth process in that direction. However, cell 2's southern neighbour is an alive cell with no active plasmodesma protein, thus cell 2 forms an unconnected channel on its southern wall. The 4 sensors drawn monitor the outside activity on 4 sides of each cell and produce sensor proteins with the changing environment. The Structuring proteins are produced by the GRN to change the physical structure of the cell, which is connected to the physical inputs and outputs of the cell.*

process is carried out by individual cells, and each cell only controls the tunnels to its east and south neighbours. Since the processing is done from top left most to the bottom right most cell, there is no need for each cell to check all four directions for the contact cell signalling process. Lines 15–18 detail the diffusion process where each chemical is constantly diffused out to four nearest neighbours. During the ongoing diffusion process, the changes to the chemical concentrations are not recorded on the actual values but stored in temporary variables. These variables are then later used at line 23 to update the actual chemical concentrations for each cell. This way the order of update of cells does

Algorithm 3 The pseudo-code for the simulation of artificial development for one time step.

```
1: ORGANISM stores all the cells available for development up to MAXCELL
2: CELL is an individual entity that models a biological cell
3: call function GRN(); comment: see Algorithm 2
4: for each CELL cell in ORGANISM do
5:    if cell is ALIVE then
6:       comment: Direct Contact signalling.
7:       comment: Only need to check EAST and SOUTH as the previous cells were to
         the NORTH and WEST
8:       if cell has a PLASMODESMATA to EAST then
9:          share chemicals with the neighbouring cell on the EAST
10:      end if
11:      if cell has a PLASMODESMATA to SOUTH then
12:         share chemicals with the neighbouring cell on the SOUTH
13:      end if
14:      comment: Diffusion
15:      for each CHEMICAL c in CHEMICALS do
16:         divide 1/2 of c to each neighbouring cell
17:         store changes to chemicals in a buffer available for each cell
18:      end for
19:   end if
20: end for
21: for each CELL cell in ORGANISM do
22:   comment: Update Cell States
23:   update the actual chemical values from the buffers
24:   reset the buffers to 0
25:   adjust the chemical values to be within 0 to MAX_CHEMICAL_LEVEL
26:   if cell was newly created then
27:      set cell state to ALIVE
28:   end if
29:   comment: Build Target Structure - problem dependent
30:   translate cell's structure for the target system
31: end for
```

not bias the diffusion process. By the end of the Algorithm 3 at line 30 the cell structure, which is a single or an array of integers, is translated for the target system. If the target system is a reconfigurable hardware device, the integer values are converted to a binary string to configure a part of the reconfigurable device. If the target system is an image, and the cell represents a pixel, then the integer value[2] is used to represent a pixel colour for the cell.

5.3 Summary

Following on from Chapter 4, a review of literature on micro-model developmental systems designed for EC has been provided in the first section of this chapter. The literature discussed forms part of the inspiration and the source of developmental information used for the design of the ADS described in the following section (Section 5.2). Several mechanisms involved with ADSs are discussed for their role in biology, EC, and their use by several existing ADSs. An extensive table of relevant ADSs in literature is compiled and used in the discussions. After the literature review, the ADS used in this thesis is introduced with all its mechanisms. The main purpose of the model described is to provide a mechanism for the evolution of stable, scalable, fault tolerant and adaptive systems. The following chapter will demonstrate the evolvability of the ADS in fulfilling a series of simple fitness functions, as well as briefly demonstrating the fault tolerant properties of the ADS.

[2]The integer value representing the cell function needs to be kept within 0 to max number of possible colours for the target image. A modulo operation is used to ensure this.

Chapter 6

Validating the Artificial Developmental System (ADS)

An initial set of experiments demonstrating the developmental and computational properties of the ADS are presented in this chapter. These include single cell experiments that test the environmental responsiveness of the Gene Regulatory Network (GRN) and its ability to maintain a stable but dynamic system. The single cell experiments should provide insight into better understanding the evolvability of the GRNs.

Pattern formation experiments that test the multicellular coordination, cell differentiation, ability to maintain cells with stable states, modularity in multicellular development, and fault tolerance follow up on the single cell experiments. Although the cells in these experiments use the same GRN model from the single cell experiments, it will be shown that the evolvability of the ADS is highly dependent on the physical constraints imposed on the ADS. The evolved developmental organisms will be tested with different types of faults and shown that in many cases, the developmental organisms can partially or fully tolerate various types of faults even when not evolved to do so.

At this point it is worth to state the extent of the meaning of the term "evolvability" when used in this and the rest of the chapters of this thesis. In biology, the ability to survive natural selection and produce meaningful evolutionary adaptation is termed evolvability [Kirschner and Gerhart, 1998]. The evolvability of the presented ADS will be measured by the degree of its ability to improve its fitness over evolutionary change.

6.1 Algorithm Configuration

The evolution of a multicellular developmental system can be layered as three algorithms working together; the Evolutionary Algorithm (EA), the GRN and the multicellular development.

6.1.1 The Evolutionary Algorithm

In the experiments involving the developmental system (i.e. rest of the whole thesis), the resulting genotypes for the developmental organisms are always evolved using an evolutionary algorithm. Evolution is used to find the genetic information (genome) that can represent the desired system after being fully developed. The evolutionary parameters were kept same (or with very minor changes) with the earlier experiments presented in Chapter 3 for consistency.

A customised version of the Evolution Strategy (ES) (see Chapter 2) with a population size of 7 and an elite size of 2 is used, i.e. ES(2+5). There is no crossover implemented but an adaptable mutation rate (mutation strength) in the 0.5-10% range that changes with respect to the *rate of change* in the fitness is used, as shown earlier in Algorithm 1. The mutation rate is increased if the previous mutations result in individuals that have equal or better fitness than their parents, but it is decreased otherwise. The effects of previous mutations are tracked by the *convergence* variable used in Algorithm 1. The use of adaptive mutation rate in ES has been previously shown to improve the algorithm performance in Evolutionary Computation (EC), especially for combinational problems [Kramer, 2008]. More information and various investigations into self-adaptive evolutionary parameters such as mutation and crossover can be found in [Kramer, 2008].

6.1.1.1 Selection

A refined fitness function is used for all the experiments discussed in this thesis; further detailed in each experimental section. All fitness functions used in the experiments work by penalising the bad solutions. Hence, the lower the penalty given to an individual the better it is considered to be. Each generation is formed by carrying over the two best

Table 6.1

Parameter	Value
Max. No. Genes	5–50 (dependent on the example)
No. Chemicals	4
Max. Chemical Concentration	255
Chemical to Gene Binding Threshold	127
Protein Production Rate	10
Chemical Consumption Rate	5

performing individuals, and creating 3 offspring from the best and 2 offspring from the second best individuals. The best parent is chosen to contribute to the population via a larger number of mutant offsprings because of its higher fitness, but at the same time the second best parent is always ensured to supply its own offsprings in order to provide diversity. An equally performing offspring is always favoured over a parent in the next generation's selection process.

6.1.2 The GRN Settings

The default settings for the GRN is given in Table 6.1. Most of these settings remain the same for all experiments, but it is noted in the corresponding section when they differ from the default values. Number of chemicals is set to 4, since in the experiments that will be presented the messenger molecules will not be used. This is because the problems tackled will be stable, such as patterns and simple digital circuits. The maximum chemical concentration is limited to 255 (8-bits); a decision that was made to allow convenient processing of the GRN in an embedded processor. Chemical to gene binding threshold is set to half of maximum chemical concentration. The chemical consumption and protein production rates are kept low in order to provide the GRN the opportunity to benefit from a wide range of protein concentrations. This is hoped to give the GRN a more responsive and precise behaviour with smooth changes in protein concentrations.

6.1.3 The ADS Settings

The two main variable parameters for the ADS at the multicellular level are the organism size and maximum organism age (see Section 5.2 for more detail on organism age). The organism size is generally $n \times n$, and n is dependent on the experiment being undertaken. The maximum organism age is also dependent on the experimental problem, but it is generally between 10–30. The maximum organism age specifies how the maximum number of iterations the ADS will be allowed to run before the organism is considered mature.

6.2 Single Cell Experiments

Before conducting experiments involving the whole of the developmental system and evolving multicellular organisms that try to solve computation problems, single cell experiments are carried out to demonstrate and understand the GRN model's responsiveness to environmental stimuli and its ability to achieve stable solutions. An artificial GRN system is a dynamical system; a mathematical system that describes a state with time dependency. A dynamical system's current state will always be one of the determinants of the next state. In a dynamical system there exists at least one *attractor* which the system settles into after a time. Attractors can be represented by single states or multiple states, see Figure 6.1. An attractor in a dynamical system has a basin, which represents all the points that will eventually converge to the attractor after a certain period of time. Hence, in a dynamical system the final state of the system will always be one of the attractor states. In order to have a stable GRN that maintains its desired state over time, the desired state needs to be an attractor in the dynamical system represented by the GRN. These states might be a single point in the state space representing a fixed behaviour or a set of points representing a more complicated behaviour. When a dynamical system is in an attractor state, it will always return to that same attractor state even when slightly disturbed; hence protecting its functionality. On the other hand, a big enough disturbance that can change the basin of attraction can produce differentiated behaviour [Strogatz, 1994].

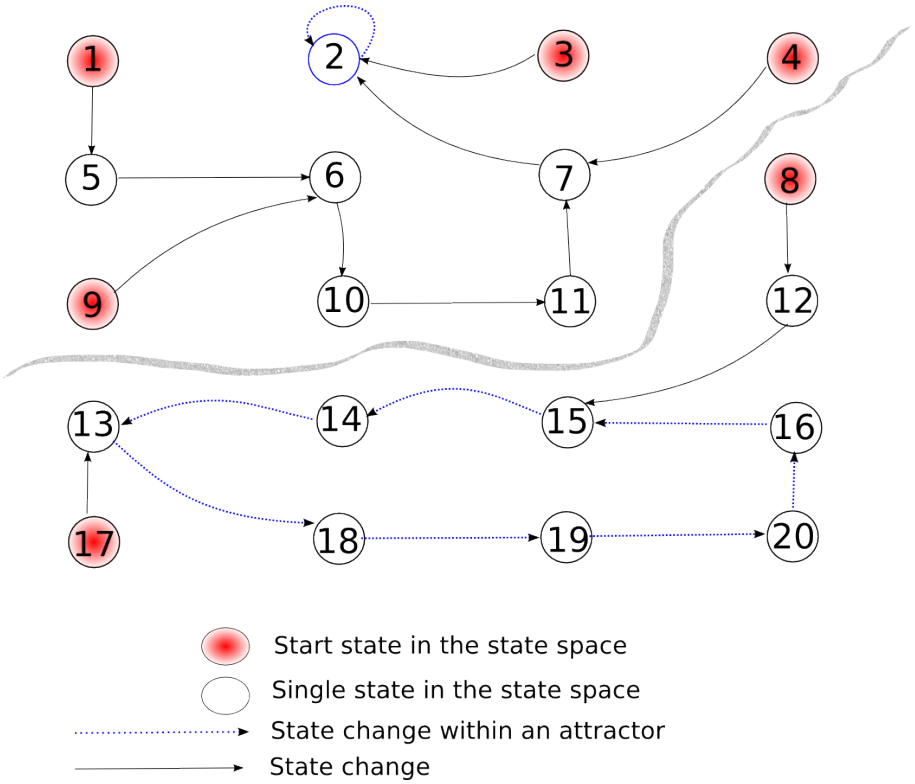


Figure 6.1: *A hypothetical state space represented by possible states and state transitions. Two possible attractors are present in this example: a point attractor at state 2 and a cyclic attractor with states 13-16 and 18-20. The two basins of attraction are divided via the grey line in between. This example system will always end up either at state 2 or cycling states 13-16 and 18-20 in the order shown in this figure. There are more different types of attractors which are not demonstrated here, for a more comprehensive explanation of dynamical systems and attractors see [Strogatz, 1994]*

The GRN system is tested to see whether its dynamics can create stable, changing, and responsive conditions within a single cell. These simple tests demonstrate the evolvability of the GRN and its ability to create simple functions that can be part of a larger computational network. In the experiments the concentration levels of proteins are monitored throughout the lifetime of a single cell. Protein concentrations of an artificial GRN have already been used as outputs to investigate the evolvability and dynamics of the model, as well as for validation the model by previous researchers in EC [Banzhaf, 2003; Knabe et al., 2006]. Knabe evolved a GRN to behave as an active control system that could respond to periodic environmental stimuli with appropriate behaviours. A similar approach to the latter will be taken in this section to investigate the evolutionary properties of the proposed GRN model.

The experiments presented in this section use the EA and GRN settings described in Section 6.1 with a maximum number of evolutionary generations set to 10 *million* generations. The number of genes used for each experiment is marked on the corresponding plot of each experiment.

The first experiment tests the responsiveness of the GRN model to simple outside stimulus. This is done by trying to evolve a single cell that will control the production of a protein (maintaining a constant level) when another one is supplied as an input. Three variations of this control model are evolved: 'ON' switch, 'BOOSTER' and 'ON-OFF' switch. For the case of an 'ON' switch, the production of the output protein will not start and its concentration level will be kept at 0 until the input protein is supplied. Once the input protein is supplied the output protein will be constantly produced reaching the *maximum* concentration level. Once the production of the output protein is started the cell does not need to consider any other conditions to meet the behaviour described by an 'ON' switch, hence it provides a simple task to achieve. However, it is an important behaviour nonetheless, since it is a simple demonstration of interactivity, and ability to switch to a different attractor state when there is a drastic change in the system.

In the second case the input protein is expected to behave as a 'BOOSTER' and increase the concentration value of the output protein every time it is supplied. When the input protein is at concentration level 0, the output protein is expected to remain at its last concentration value without any change. This is a simple model of a time-dependent control mechanism with memory, where the output protein reflects the time the cell is exposed to the input protein. In addition to a responsive mechanism to an input protein, in this example the cell is also expected to maintain the value of the protein concentration **below** the *maximum* concentration level through gene interactions and before *saturation*. When a chemical reaches the predefined *maximum* concentration level and is still produced, an artificial mechanism always makes sure that the concentration value of the chemical is always set back to the maximum value which is 255 in our case: this process is called *saturation*. In the example of 'ON' switch, the output protein concentration level is kept at the value 255 via *saturation*, therefore the dynamics of the GRN only creates a trigger mechanism that indefinitely produces the output protein once a high concentration of input protein is detected. In both the 'BOOSTER' and 'ON-OFF' switch experiments, the GRN is evolved to have dynamics that keep the output protein concentration at or

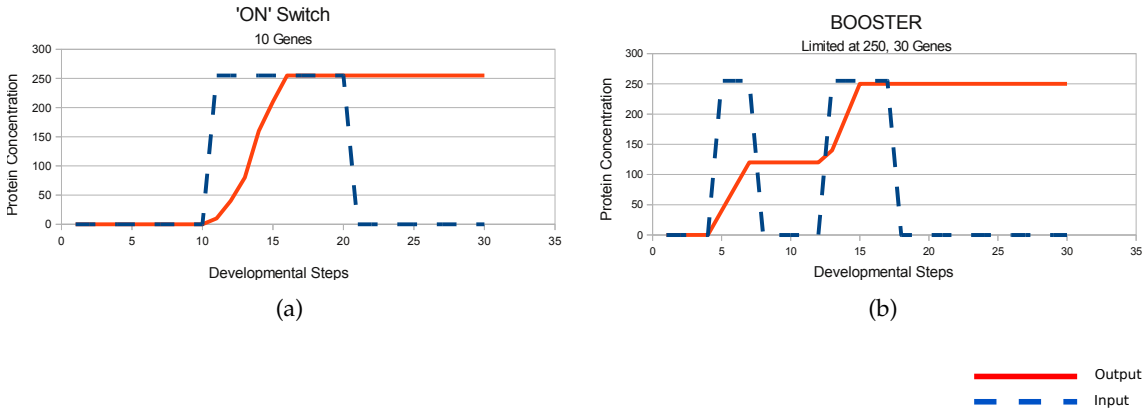


Figure 6.2: *Desired GRN output protein to input stimulus is shown for the 'ON' switch and 'BOOSTER' cases. In (a) the output protein remains* 0 *until the input protein is supplied at a high concentration which then triggers the production of the output protein. This is a simple demonstration of a trigger mechanism implemented by the GRN. In (b) the output protein level is initially set to zero, until the input protein is supplied at a high concentration level, but in this example unlike (a), the concentration level of output protein stops increasing when the input protein concentration level is dropped back to* 0*. The GRN in (b) is also trained to keep the concentration level of the output protein at or below* 250 *even when the input protein is supplied at maximum concentration.*

below 250. Such a precise control of a signal can be highly desirable in some applications. Controlling the level of a specific output is a function that exists both in biology and engineering. In biology, an example of this is the control of hormone supply, which is usually kept below a certain level otherwise high levels of hormone secretion can cause disorders, e.g. acromegaly due to growth hormone excess [Giustina and Manelli, 2001]. Similarly in engineering, some outputs are expected to be kept below a certain level in order to keep the system safe and reliable. The 'ON' switch and the 'BOOSTER' experiments are further detailed with evolved examples for each in Figure 6.2.

The third case models the input protein as an 'ON-OFF' switch, so when the input protein is present (i.e. the input protein is $\geq$ protein to gene binding concentration threshold) the output protein's concentration starts increasing, and when the input protein is absent, the output protein's concentration starts dropping. This creates an 'ON - OFF' behaviour that is synchronized with an outside stimulus, see Figure 6.3. Two versions of this case are defined in the experiments; one with a minimum output protein concentration level at 70 and maximum output protein concentration level at 150, and the other with a minimum output protein concentration level at 0 and maximum output protein concentration level at 250. Evolving an 'ON-OFF' switch behaviour is more difficult than the first two

examples as it models a more complex correlation between the input and the output proteins.

An example of similar behaviour to this in biology is the change in behaviour of organisms with respect to the change in day and night. The input protein can be seen as an indication on the presence of the Sun, and the output protein is a response from the organism (e.g. sleeping). The output protein could also be analogous to a short term behaviour like the production of the chemical adrenalin, which is produced when a threat to an organism is present nearby (signalled by the input protein), and when the threat disappears the adrenalin levels drop. Evolving the example with chemical levels bound within $70-150$ required less number of genes than having chemical level bounds within $0-250$. In order to achieve the 'OFF' state at 0 when the input protein was at 0, the protein binding thresholds had to be evolved for each gene in order to allow the output protein use of the whole concentration range. This will be further discussed later on in this section.

The plots of example runs shown in Figure 6.3 demonstrate that it is possible to change the behaviour of the GRN drastically when there is a notable change in the environment, as well as demonstrating that it is possible to set lower and upper limits for the protein concentrations via gene interactions. As shown in Figure 6.3(a) the output protein is maintained at concentration level; 70 when the input protein is at concentration level 0, and 150 when the input protein is at concentration level 255. This is an example demonstration of setting upper and lower limits to the production of a protein via gene interactions. In Figure 6.3(b) the output protein is maintained at concentration level 0 when the input protein is at concentration level 0, and although the GRN was evolved to maintain the output protein at concentration level 250 when the input protein is at concentration level 255, the GRN can not achieve this so well and the output protein oscillates around 235. At the third peak of the plot in Figure 6.3(b), the maximum concentration level of the output protein drops down to oscillate around 180 at developmental step 330 and onwards. The 'ON-OFF' switching is still preserved, but maintaining an absolute value for the 'ON' state is shown to be a challenge. In Figure 6.3(c) the GRN is evolved with the same conditions as Figure 6.3(b), but this time the result fits the fitness constraints well. Increasing the number of genes to 30 for this problem made it possible to create the dynamics to keep the 'ON' state constant at 250. This hints that evolving

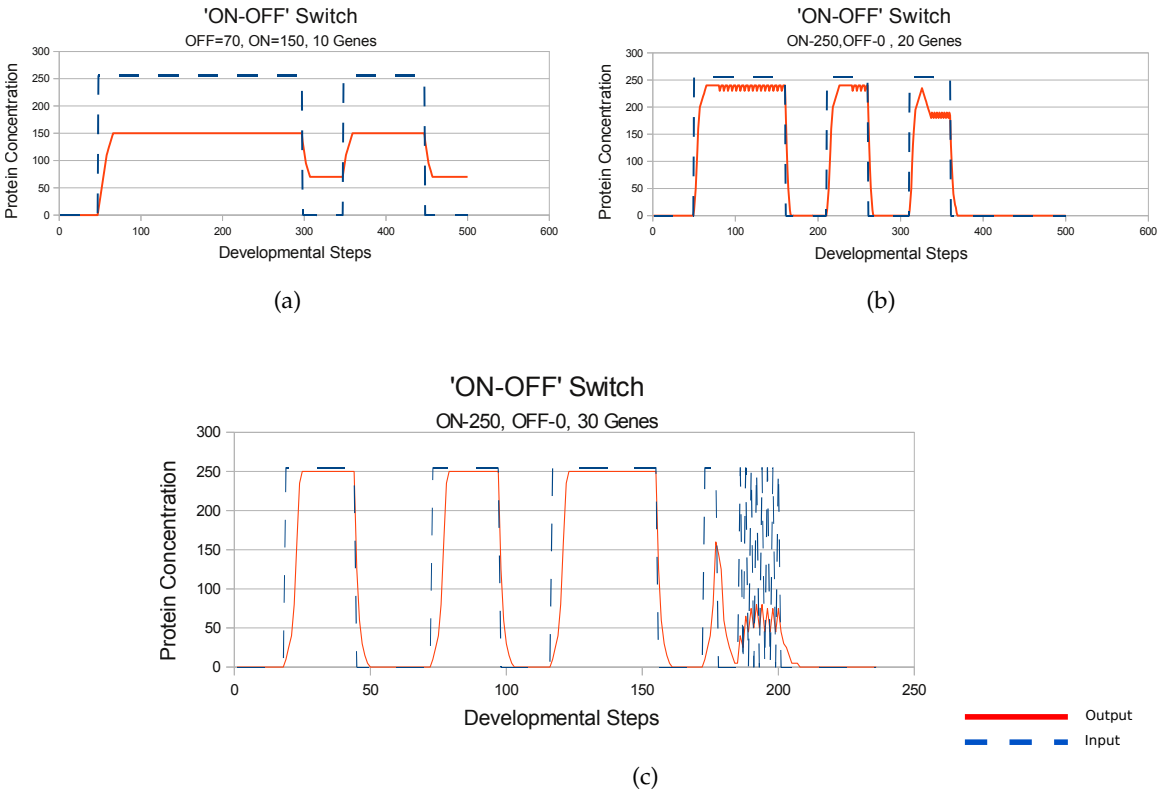


Figure 6.3: *Example runs from the 'ON-OFF' switch case. For run (a) the protein to gene binding threshold was set to* 127*, whereas for the other two runs the protein to gene binding thresholds were evolved per gene.*

the behaviour for the GRN was not a problem, but maintaining the boundaries at a fixed value was the real challenge. At the end of the plot displayed in Figure 6.3(c), the input concentration is raised to 250 for short periods of time. Expectedly, the GRN responds to this by creating short peaks with the output protein concentrations.

For the 'ON-OFF' switching experiments the fitness function had to be designed carefully otherwise evolution would always find it easier to "memorize" (i.e. over-fit) the exact tested conditions and create a GRN system that would only meet those conditions. This could happen in the following example. Assuming the candidates were tested over 30 developmental steps every evolutionary generation, and the input protein was at *maximum* concentration from developmental step 12-15 and 20-25 and at 0 concentration for all the other developmental steps. The resulting GRN could either always produce the output protein at steps 12-15 and 20-25, and consume it for the other steps without considering the input protein, or only work for the case when the input protein is applied at steps 12-15 and 20-25, producing the output protein erratically for all the other cases

of inputs. To overcome this a fitness function considering the input pattern ordering was designed to tackle these "transient" effects, as discussed earlier in Chapter 3.

The fitness function defines the behaviour of an evolving system. Since the structure of the evolving system can not be tested, the behaviour defined by the fitness function alone shapes the fitness landscape for the EA. When applying the test inputs to an evolving system, the ordering of the possible inputs may have an important role in shaping the fitness landscape. This is especially true when evolving dynamic systems that can create feedback loops within its computational network. Hence, it is necessary to include randomness in the test inputs of the evolving system. The fitness function used here uses a new randomly generated input sequence during the evaluation phase of every evolutionary generation as described in Section 3.4.

Algorithm 4 details the application of the input protein to the evolving GRN for one evaluation step. The GRN is tested for five times every evaluation step, and each time the input protein is set high and low at pseudo-random time intervals (except the first test, which is the same for each evaluation). Although the switching for the input protein from 0 to 255 and 255 to 0 is done immediately, for the target output the switching is spread over six developmental steps (see lines 21–36). This is done in order to encourage a more analogue behaviour (smoother and more gradual changes) being reflected on the output protein of the GRN.

One can argue that achieving an 'ON-OFF' switching behaviour should be a much simpler task that can be fulfilled by a smaller number (~ 5) of Genes and evolutionary generations. This is correct to a certain extent and can be done by adjusting some of the GRN parameters such as protein production and consumption rates per gene, and altering the fitness function to accept a digital behaviour (i.e. a big change from concentration 0 to 255 and vice a versa for the output protein when the input protein changes likewise). However, these adjustments to the GRN would change the application domain, which removes the meaning of these experiments that is the fine control of proteins (via complex gene interactions) with respect to changing environmental conditions.

The second set of single cell experiments test the GRN's ability to reach a state where a protein (the output protein) is produced and consumed in cycles, i.e. forming a sinusoidal wave altering a protein concentration. This experiment is done with and without the

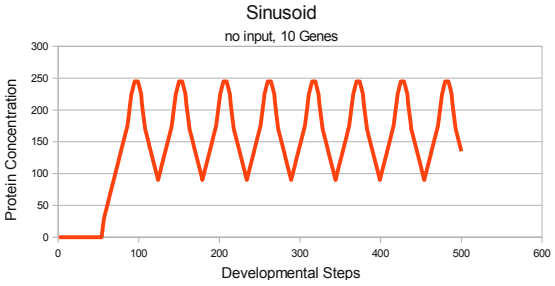
Algorithm 4 The fitness function pseudo-code of the 'ON-OFF' switch experiment.

```
1: inputHighStartAge = 5
2: inputLowStartAge = 30
3: inputHighStartAge2 = 60
4: inputLowStartAge2 = 95
5: noTests = 5
6: MAXAGE = 150
7: for testCount in range 0 to noTests do
8:    incrVal=255
9:    target=0
10:   reset and initialize GRN;
11:   for age in range 0 to MAXAGE do
12:      if age = inputHighStartAge or age = inputLowStartAge or age = inputHigh-
         StartAge2 or age = inputLowStartAge2 then
13:         inputValue = inputValue + incrVal
14:         if inputValue=255 then
15:            incrVal = −255
16:         else if inputValue=0 then
17:            incrVal = 255
18:         end if
19:      end if
20:      if age > 1 then
21:         if inputValue=255 then
22:            if target < 128 then
23:               if target=0 then
24:                  target = target + 10
25:               else
26:                  target = target ∗ 2
27:               end if
28:            else
29:               target = 250
30:            end if
31:         else
32:            if target ≤ 10 then
33:               target=0
34:            else
35:               target = target/2
36:            end if
37:            organismOutput = run the organism using inputValue as input
38:            fitness = fitness+ |target − organismOutput|
39:         end if
40:      end if
41:   end for
42:   inputHighStartAge = rand(2 to 20);
43:   inputLowStartAge = rand(15 to 35) + inputHighStartAge;
44:   inputHighStartAge2 = rand(inputLowStartAge+30 to inputLowStartAge+45);
45:   inputLowStartAge2 = rand(15 to 35) + inputHighStartAge2;
46: end for
```

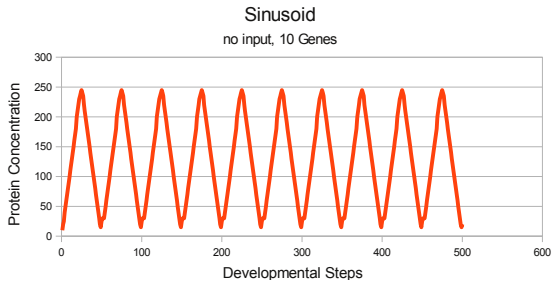
presence of an environmental stimulus. The experiment investigates whether an internal clock can be evolved with the GRN, which can respond to environmental stimuli via starting or stopping the internal clock. Evolving an internal clock with a GRN was successfully achieved by [Knabe et al., 2006]. The ability to create a clock enables the GRN to create synchronous conditions, and implement behaviours that depend on cycles. Synchronous behaviour is an important part of both biological (e.g. biochemical oscillations, circadian rhythms) and electronic systems (e.g. sequential circuits). Figure 6.4 shows the result of some example runs, and details the experiments further.

In Figure 6.4(a) the protein to gene binding threshold is a constant at 128, which means that the output protein can not bind to the genes once it is below 128, hence the limited range displayed on subfigure (a). The same problem was encountered with the 'ON-OFF' switch experiments, and this problem can not be overcome by simply setting the protein to gene binding threshold to a very low value. Evolving the protein to gene binding threshold *for each gene* is the solution. The evolution of protein to gene binding thresholds for each gene created one or more genes that would still consume the output protein even when its concentration is near zero.

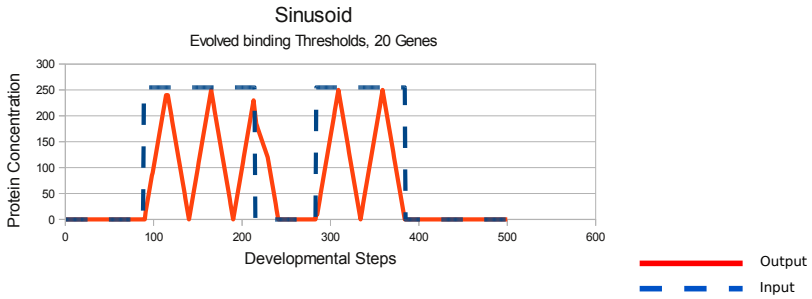
The GRN was able to create a system that can produce the required behaviour shown in figures 6.2, 6.3, and 6.4. The results demonstrate that the GRN is able to respond to the environmental stimuli in both a stable and dynamic way. In the experiments discussed in figures 6.2 and 6.3, it was shown that the GRN is able to respond to a change in the value of a protein (referred to as environmental stimulus) using a different type of protein (the output protein), and regulate the concentration of the output protein using the information learnt from the evolutionary constraints and the change in the environmental stimulus. It was observed that evolving the GRN to achieve a behaviour with minimal constraints required a smaller number of genes than evolving the same behaviour with constraints that are tough to achieve. The 'ON-OFF' switch experiments showed that the concentration of the input protein could be mimicked by the output protein, Figure 6.3(c). If a periodic input protein was applied the GRN would synchronize and create an internal periodic signal. This demonstrates a cell synchronizing with its environment, which is an important behaviour both in biology (e.g coordination in plant development [Leyser and Day, 2003] and chemical interactions [Goldbeter, 1996]) and computing.



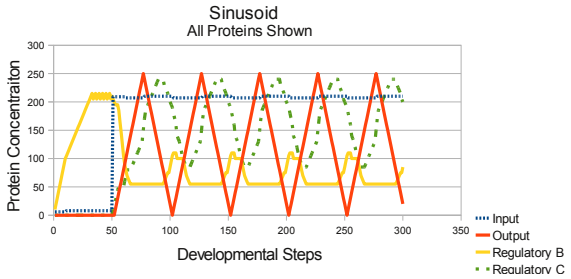
(a) Constant cyclic production/consumption of output protein when the protein to gene binding threshold is set to 128.



(b) Constant cyclic production/consumption of output protein.



(c) Cyclic production/consumption of output protein when an environmental signal is provided.



(d) The concentration of all the proteins at the end of a developmental step (including the input protein) of example (c).

Figure 6.4: *For the examples above, cells with 4 proteins types are evolved. One of these proteins is the input protein, another protein is the output, and the other two proteins are internal proteins (not shown in the figures except subfigure (d). The graphs show the input protein as dashed lines and the output protein as a solid line. The ideal 'ON-OFF' values are displayed under the title of each graph along with the maximum number of genes the GRN was allowed to use to achieve the displayed behaviour. In subfigures (a) and (b) the cells produce the displayed behaviour without any input proteins. In (c) a clock-like behaviour is achieved that is directly influenced by an outside stimulus. The actual output protein level oscillates between its maximum concentration value and 0 when the input protein is at a high concentration level. When the input protein is at concentration 0, the actual output protein concentration gradually drops down to 0 and stays 0 until the input protein concentration is increased. This matches the desired behaviour almost perfectly. In (d) the concentration of all the proteins, including the internal proteins, are displayed at the end of each developmental step for the example shown in (c).*

In Figure 6.4, the oscillating production of a protein also demonstrates an ability to form a dynamical network that can set a regular rhythm amongst the genes. The ability to produce cyclic behaviours and respond to outside stimuli is important for a GRN, both as a dynamical system and a biological model. The results suggest that the GRN model is evolvable for synchronized signal processing as long as it is tuned correctly for its application domain. Figure 6.4(d) graphs all the proteins involved in the GRN evolved to produce an oscillating output when a constant high concentration of input protein is supplied. The GRN achieves this by asynchronised oscillations of the internal regulatory proteins when the input protein is high. When the input protein is low, the production of all the proteins (except regulatory protein B) is suppressed by a high concentration production of regulatory protein B (see Figure 6.4(d)).

6.3 Multi-Cellular Experiments

Patterns are an easy and convenient way of representing the cell states of a multicellular organism. Patterns can be used to represent a solution to various computational problems (including circuits), at its simplest form a pattern can be treated as a look up table that provides the information for the type of component to be used in a given location. More importantly, spatial patterns already exist in biology and are a fundamental part of biological development. Developing patterns is very useful for examining and comparing biologically inspired developmental approaches. In biology, various types of patterns are observed and they are sometimes used as guidelines in determining what to expect an ADS to achieve [Flann et al., 2005]. The patterns and shapes observed in biology such as patches in the segmental divisions along the insect anterior-posterior axis and borders forming boundaries between the anterior-posterior compartments of the Drosophila wing [Lawrence, 1992] are a result of development. All these shapes present an ordered pattern often with a symmetry. These patterns and shapes also show a high degree of modularity via their repetitive structures. These qualities are what is expected to appear in the systems built using artificial multicellular development. Thus, it is expected that the multicellular developmental system presented here would be able to achieve symmetrical patterns and display modularity[1] in the final solutions.

[1] The ability to build modular structures. In other words structures that are made out of smaller segments, which are repeatedly used over the whole of the structure.

Another important aspect of biological development is that it does not stop when the organism has matured. When an organism is mature the developmental system adopts a steady state and maintains a stable pattern throughout the rest of the life-span of the organism. In the artificial multicellular developmental system presented here it is also one of the major goals to achieve the solutions in such attractor states and be able to keep the system stable while the developmental system continually runs.

In the multicellular experiments presented, the concentration of each protein is represented by an 8-bit unsigned number; thus the maximum concentration is 255. The protein to gene binding threshold level is kept at a constant of half the maximum concentration level (128) for all cells, genes, and chemicals. The organism is initialised with one live cell placed in the middle of the virtual cell space. The structural part of the cells are represented as 64-bit integer values, which are then treated as a colour value for a pixel at the cell's location. All the organisms are developed for a set number of developmental steps and evaluated once when they are fully developed, unless stated otherwise.

6.3.1 Simple Motifs and Dynamics

The developmental system is evolved to match three different versions of a simple pattern, which is a 6x6 image that has an even number of white pixels covering one half of the image and black pixels covering the other half. Three versions of this motif are used as the target patterns in the experiments: white pixels on the left half of the image (blank-fill), white pixels on the right half of the image (fill-blank), and an oscillation between the two. The patterns are illustrated in Figure 6.5.

The patterns shown in Figure 6.5 were evolved to test the organisational and differentiation abilities of the developmental system, and understand how these functions work and whether they have any biases. In addition, the oscillatory state was also included in the experiments to see whether the developmental system would be able to achieve a stable oscillatory state. In all cases, the developmental system was able to achieve both of the static patterns in relatively few generations. However, there is a clear bias in achieving the "Fill Blank" pattern as it was evolved more quickly in comparison to the "Blank Fill" pattern every time.

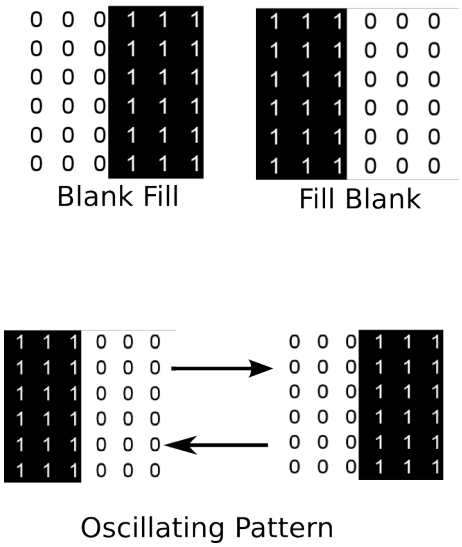


Figure 6.5: *Three target states for the first multicellular experiments. The first two being an inversion of each other, the third state is an oscillation between the two.*

This is due to the initialisation of the developmental system. The developmental system was initialised with one live cell in the centre of the organism–fourth cell from the left, and fourth cell from the top of the 6*x*6 organism–which should be a white pixel in the "Fill Blank" case. The initialised cell always has a structural value of 0 by default, and this seemed to create a notable bias on the evolvability of the developmental system to achieve certain patterns. Figure 6.6 details the performance of the evolved experiments. The oscillatory state was successfully found only 50% of the time, the other half of the time a perfect switch between the two patterns was not achieved.

The original experiments (marked as 'A' and 'E') were evolved to mature after a set number of developmental steps, in this case 30, and the organism was only evaluated once at developmental step 30. The fitness function that was used to evaluate the matured organism carries out pixel by pixel matching to the target pattern, and assigns a fitness reflecting the number of pixels that failed to match the target pattern.

Two possible improvements to the evaluation stage during evolution were devised and tested on the simple two colour pattern. The first possible improvement involved the use of a non deterministic maturing for the candidate organisms. This means that the organism would not always be expected to mature at developmental step 30, but it would be allowed to mature at any developmental step between a minimum (3 in the

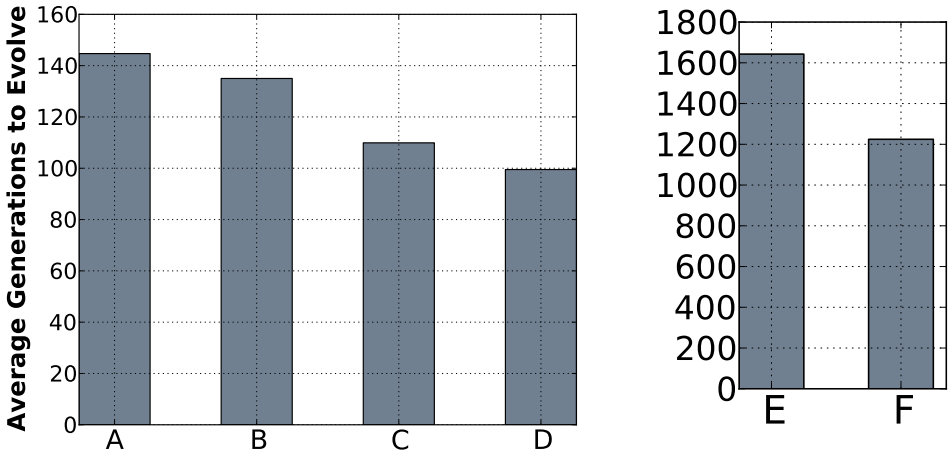


Figure 6.6: *The average number of evolutionary generations it takes to find the perfect solution for the two patterns is shown in the figure above with other comparisons. Each experiment was run 10 times and every run found the target pattern successfully. The "Blank Fill" examples (E-F) are shown on different scales because of the large magnitude of difference in the average number of generations in finding each of the example patterns. The aliases used in the x-axis are explained in Table 6.2.*

Table 6.2: *The explanations of aliases for experiment results in Figure 6.6.*

Reference in Plots	Experiment
A	Fill Blank pattern
B	Fill Blank pattern with non deterministic maturing
C	Oscillating pattern with non deterministic maturing
D	All four permutations of a 2 patch pattern with deterministic maturing
E	Blank Fill pattern
F	Blank Fill pattern with non deterministic maturing

experiments presented in this section) and a maximum (30 in the experiments presented in this section).

This was done by testing the organism against the target pattern every developmental step between 3-30, and use the best evaluation age as the maturing age. This approach was further developed to promote stable organisms (i.e. organisms that achieve the target pattern and keep the target pattern even when they are developed for a longer time) by stopping the development during evolution if the organism obtained a worse fitness score than the previous developmental step. As well as promoting more stable results, this also reduced the evolution time. The results of non deterministic maturing are shown in Figure 6.6, which display improvement in number of evolutionary generations (and

time for the "Blank Fill" example). Using a non deterministic maturing also improved the success rates for the oscillating pattern from 50% to 100%. Therefore only the results of oscillating pattern with non deterministic maturing are shown in Figure 6.6 (labelled as 'C'). Non deterministic maturing has been suggested before by Devert et al. as an implicitly adaptive method of evaluating a developmental organism [Devert et al., 2007]. Their results had reported robustness toward perturbations during the growth process due to a global attractor being favoured by non deterministic maturing.

A second possible improvement to the evolutionary approach to find the target pattern is to "relax" the constraints by not asking perfect matching of a pattern. As discussed in Section 6.2, evolving the developmental system for a behaviour is much easier than evolving it to exactly fit that behaviour to a constrained framework. Although this was the case for a single cell, it may still apply for a multicellular developmental system. In the case of two colour patterns two more experiments were tried out. In one of these cases, the pattern matching constraints were relaxed by allowing the development of an organism that matches either versions of the pattern shown in Figure 6.5 without any preference to one or the other. In the second case these constraints were further relaxed by allowing the developmental system to develop an organism that matches either versions of the patterns described earlier plus their 90° tilted versions, creating 4 possible patterns to develop into. Although in essence all these patterns are the same, relaxing the global location requirements for the cells during evolution has sped up the evolution. The performance gain from relaxing the constraints for this example was around 25-30% when compared with the best case of the perfect pattern matching (see Figure 6.6), which is not a visible gain in this case but for more complicated problems this performance gain might make a large difference. The improvement obtained by a more fuzzy fitness function suggests that evolution of a developmental organisms would be more successful in the design of systems that may benefit from such behaviour. This will be further investigated in the rest of this chapter.

6.3.2 Higher Complexity Patterns

The developmental system is now evolved to match three different patterns of higher complexity than the two colour patch patterns used in the previous experiments. The

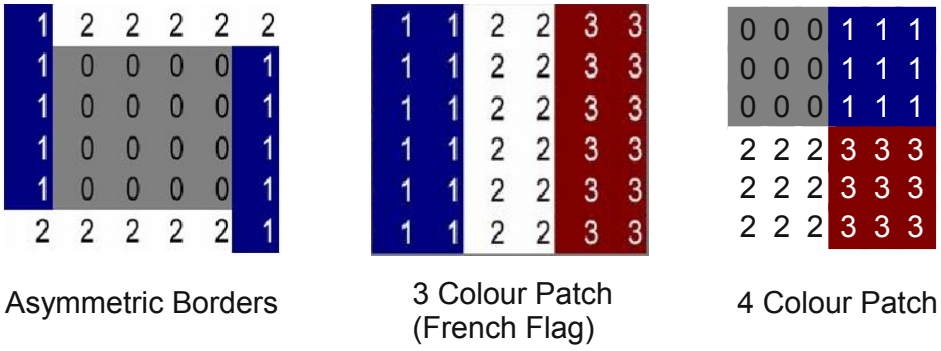
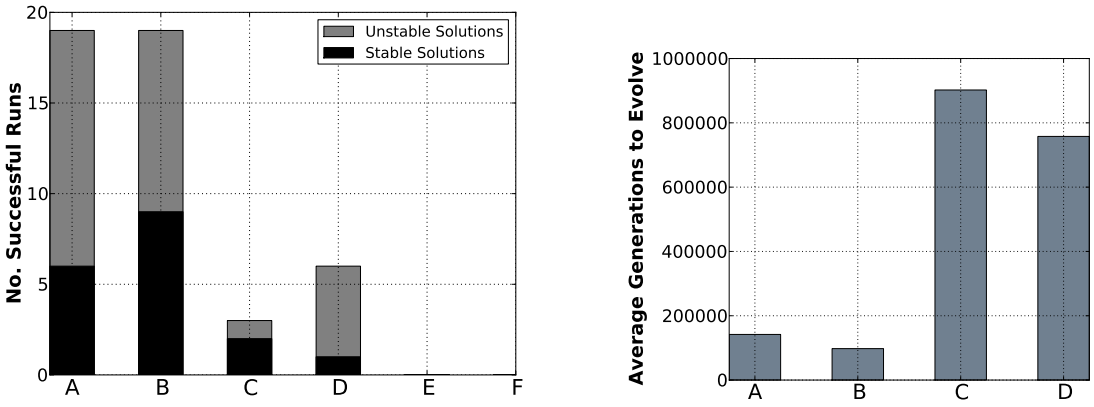


Figure 6.7: *The developmental system is evolved to develop three patterns on different experimental runs. The developmental organism has 6x6 cells with 4 possible colours to use as their structural representation: grey, blue, white and red. For the first two patterns the developed organism needs to be evolved not to use all the colours, whereas the four colour patch pattern needs the developed organism to use all four colours.*

aim is to investigate the organisation abilities of the developmental system in more detail, and understand the abilities and limits of the system for solving computational problems. The patterns evolved are listed in Figure 6.7. Each of these patterns have different characteristics and complexities. The first pattern is a border pattern, which is formed by a block of same type cells surrounded by a strip of different type of cells. In the case with the first pattern the frame is asymmetrical and composed of two different types of cells in order to make the task more challenging. The French flag pattern, the second experimental pattern listed in Figure 6.7, is an example of segmentation of cells, and it is a symmetrical structure. French flag has been described in developmental biology [Wolpert et al., 2002], and artificial development, e.g. [Gordon, 2005; Miller, 2003], as a popular example for demonstrating multicellular organisation. The third example is another patch pattern, however this time there is no symmetry in the pattern.

Again the initial objective was to evolve an organism to achieve maturity after a given number of developmental steps. In the experiments in this section the maturing age was set to 20 developmental steps, and the organism was evaluated for 10 more developmental steps for stability. The maturing age of 20 was chosen to be a good value after an initial set of experiments were done to manually determine this value. As long as this age is not too low, the performance of the developmental system when being evolved didn't seem to change considerably. However, these can only be taken as the gut feelings as there is no solid data to support this. The fitness function that was used to evaluate the matured organism would do pixel by pixel matching to the target pattern, and assign a fitness



(a) (b)

Figure 6.8: *Subfigure (a) shows the number of successful runs for each experiment with the number of stable runs shown as black bars. The aliases used in the x-axis are explained in Table 6.3.*

Table 6.3: *The explanations of aliases for experiment results in Figure 6.8.*

Reference in Plots	Experiment
A	Asymmetric borders pattern
B	Asymmetric borders pattern with non deterministic maturing
C	French flag
D	French flag with non deterministic maturing
E	Four patches pattern
F	Four patches pattern with non deterministic maturing

reflecting the number of pixels that failed to match the target pattern. Another set of experiments were also done where the maturing age was non deterministic as described in Section 6.3.1. In this case, the organism was evaluated for 10 developmental steps for stability after it reaches maturity, not fixed at age 20 but maximum limit of 20. This time each experiment was repeated 20 times and the maximum number of evolutionary generations set to 1 million. The organisms were evolved to match the patterns illustrated in Figure 6.7, and two aforementioned fitness functions were used. Table 6.3 lists all the experiments undertaken.

The results for these experiments are shown in Figure 6.8. By allowing the organism mature at a non pre-determined age, the number of stable solutions improve in the asymmetrical borders pattern experiments. However, even though the overall number of successful runs improve for the French flag, the number of stable solutions for the French flag case drops from 2 to 1 when the non-deterministic maturing is used.

Even so, using a non-deterministic maturing improves the overall performance (displayed in Figure 6.8(b)) and success rate of the developmental system. Neither of the experimental cases were able to find any perfect matches for the four patches pattern.

Once more the experiments suggest that having a non-deterministic maturing age improves the overall performance of the developmental system. Unfortunately the developmental system fails to find any perfect matching solution to the four patches pattern; the best solution found was a pattern with 1 mismatched pixel. The four patches pattern presents a challenge for the developmental system because of its asymmetrical nature and the existence of the strict pixel ordering. The failure to find a perfect match to the four patches pattern suggests that the multicellular developmental system used needs to be tuned or some of the mechanisms used need to be redesigned. Although using a multicellular developmental system to perfectly match a pattern is not the most efficient way of employing a developmental system, a 6x6 4 colour pattern should not be a major challenge.

Evolving a developmental system to produce general behaviour is likely to be an easier task than asking evolution to find a solution to a highly constrained problem. In fact, if we look at biological organisms there is no equivalent of pixel by pixel matching; it is acceptable to have certain amount of deformations in the target pattern's appearance and size, hence every individual of a species have visible differences in their appearance. A similar fitness function to that described in Section 6.3.1, were devised. In this, different permutations of a pattern are all accepted as the target.

A pattern of four patches can be permuted to form $4! = 24$ acceptable variations. During the evaluation stage the organism is tested to see if it matches any of these possible patterns and awarded a fitness value for the best matching result. A lower fitness means a better match. The code for evaluation stage is detailed in Algorithm 5. Even though the evaluation stage becomes more computationally intensive by using Algorithm 5 rather than matching with one target pattern, the increase in the overall evolution computation time is negligible. This is due to the comparably large computing time required for processing the developmental system. Figure 6.9 provides a visual explanation for Algorithm 5, which describe a generic fitness function for evaluating a patch pattern for its organisational properties.

Algorithm 5 The pseudo-code for fitness function of patch patterns.

```
1: comment: define the window size
2: define countXlimit
3: define countYlimit
4: define pixelsPerColour
5: initialize groupingScore[NOofCOLOURS]
6: for x in range 0 to SIZEORGANISMX do
7:    for y in range 0 to SIZEORGANISMY do
8:       pixelValue ← phenotype[x][y]
9:       for countX in range 0 to countXlimit do
10:         areaStartX ← x − countX
11:         areaEndX ← areaStartX + countXlimit − 1
12:         if areaStartX < 0 or areaEndX ≥ SIZEORGANISMX then
13:            break
14:         end if
15:         for countY in range 0 to countYlimit do
16:            areaStartY ← y − countY
17:            areaEndY ← areaStartY + countYlimit − 1
18:            if areaStartY < 0 or areaEndX ≥ SIZEORGANISMY then
19:               break
20:            end if
21:            groupingScore[pixelValue] ← 0
22:            for i in range areaStartX to areaEndX do
23:               for j in range areaStartY to areaEndY do
24:                  if phenotype[i][j] = pixelValue then
25:                     groupingScore[pixelValue] ← groupingScore[pixelValue] + 1
26:                  end if
27:               end for
28:            end for
29:            if groupingScore[pixelValue] > bestGroupingScore[pixelValue] then
30:               bestGroupingScore[pixelValue] ← groupingScore[pixelValue]
31:            end if
32:         end for
33:      end for
34:   end for
35: end for
36: fitness ← 0
37: for colourCount in range 0 to NOofCOLOURS do
38:    fitness ← fitness + pixelsPerColour − bestGroupingScore[colourCount]
39: end for
40: return fitness
```

The results of the experiments evolving three and four patches patterns with the consideration of all possible variations is displayed in Figure 6.10. Evolving the four patch pattern for all the possible combinations increased the number of successful runs from 0 to 5, and evolving the French flag pattern for all the possible combinations increased the

number of successful runs from 6 to 10. Hence in both cases, evolving the developmental system to match one of the all possible versions of a pattern rather than a single version improved the performance and the chances of success.

6.4 Fault Tolerance and Recovery

Two of the most stable solutions from the evolution of asymmetric borders and French flag patterns presented in the Section 6.3 are used to investigate their fault tolerance and recovery properties. The development of these patterns are illustrated in figures 6.11

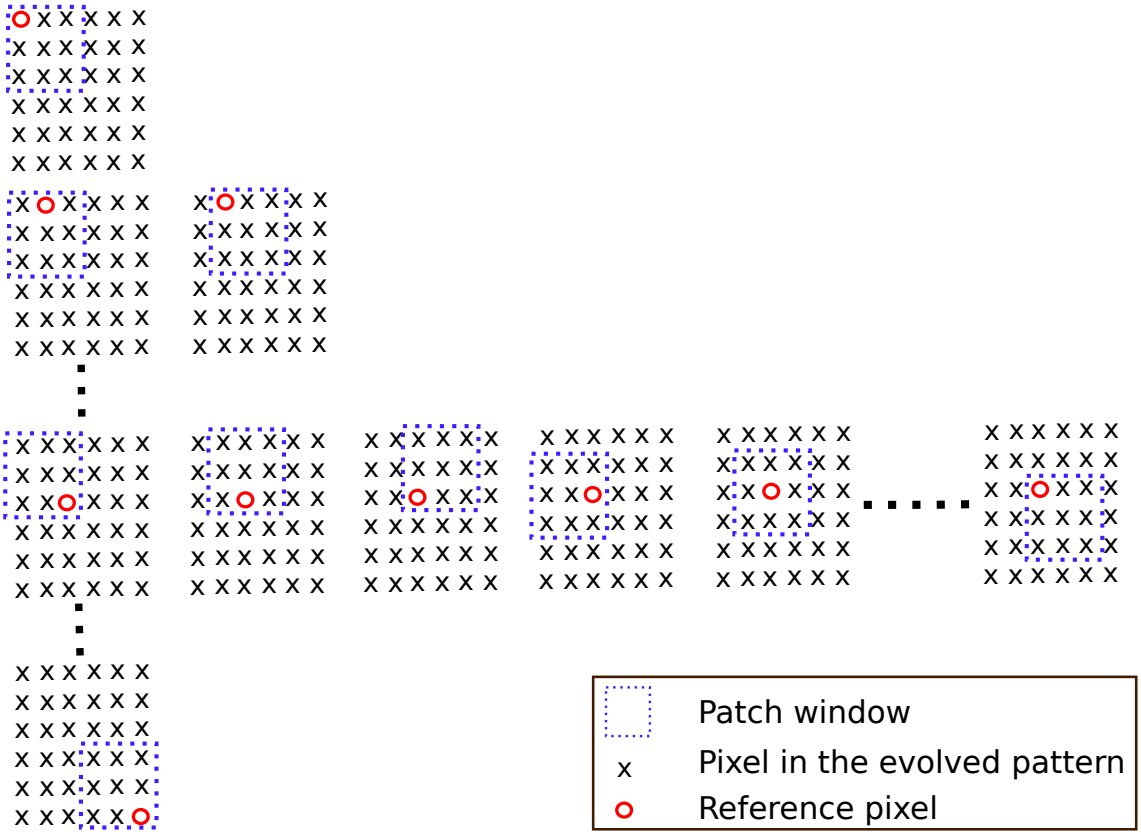


Figure 6.9: *The method of evaluating a patch pattern only for its organisational properties is shown. The specific example shown here is for a four patch pattern. A window is created depending on the type of patch pattern (in this case it is 3x3). The surrounding area of a pivot pixel (all pixels are taken as a pivot one by one) is scanned using the window, and depending on the colour of pixel, a fitness value is assigned for each scan. The best fitness value for each colour is recorded and the rest of the calculations are discarded. This encourages all the same colour pixels to group together. Since there are four colours present in a four patch pattern, the compromise for all the colours is to pick a corner and group together in that corner. Thus for a four patch pattern the best fitness is obtained by achieving a four patch pattern similar to the one displayed in Figure 6.7, but this time the corner occupied by each colour is not part of the fitness function.*

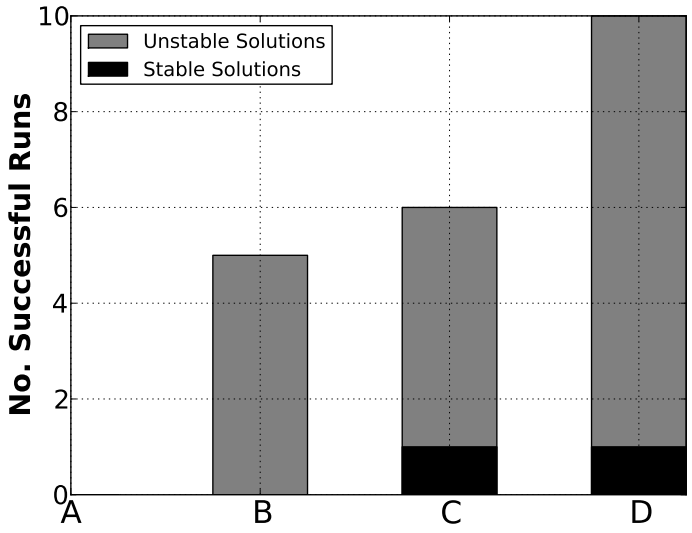


Figure 6.10: *Number of successful runs out of 20 for each experiment. A clear improvement in the number of successful runs for all runs can be observed when the pattern matching constraints are eased. The aliases used in the x-axis are explained in Table 6.4.*

Table 6.4: *The explanations of aliases for experiment results in Figure 6.10.*

Reference in Plots	Experiment
A	Four patches pattern evolved with fixed patch locations
B	Four patches pattern evolved with all possible combinations
C	Three patches pattern matching French Flag
D	Three patches pattern evolved with all possible combinations

and 6.12. These figures show snapshots of the developmental organism as it matures to organise itself to the desired pattern. Note that not every developmental step is shown in these figures.

None of the experiments presented in Section 6.3 were evolved with fault tolerance as an objective function, hence this section explores the emergent fault tolerance and recovery properties of the developmental organisms. Furthermore, by investigating the effects of introducing various faults into the system, it is expected to improve the understanding of multicellular ordering properties of the developmental system.

In the experiments presented in this section, two types of faults are used: permanent and transient faults.

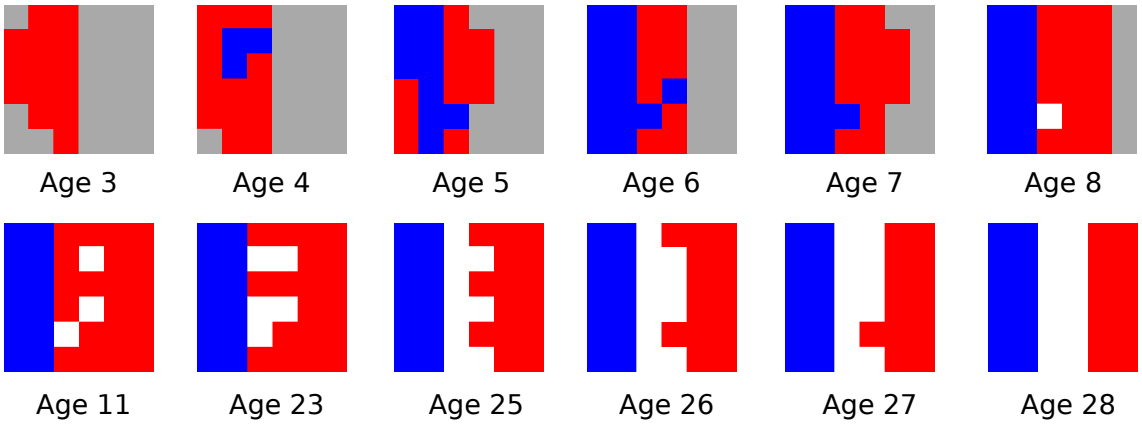


Figure 6.11: *Development of French flag pattern. Once the perfect French flag is achieved the organism stays stable at that pattern for 465 developmental steps.*

6.4.1 Permanent Faults

One way of introducing permanent faults is to simulate a complete failure of a cell. Hence, when a permanent fault is introduced to an organism, one or more of the cells are killed, and the cell is unable to carry out any of the developmental functions (such as growth and signalling).

Figures 6.13 and 6.14, illustrate the change in the course of development when cells fail in the developmental organisms displayed in figures 6.11 and 6.12. In these figures the images where a fault is first introduced are marked with "**F**", and when these organisms reach stable states after the introduction of faults the respective images are marked with "**S**".

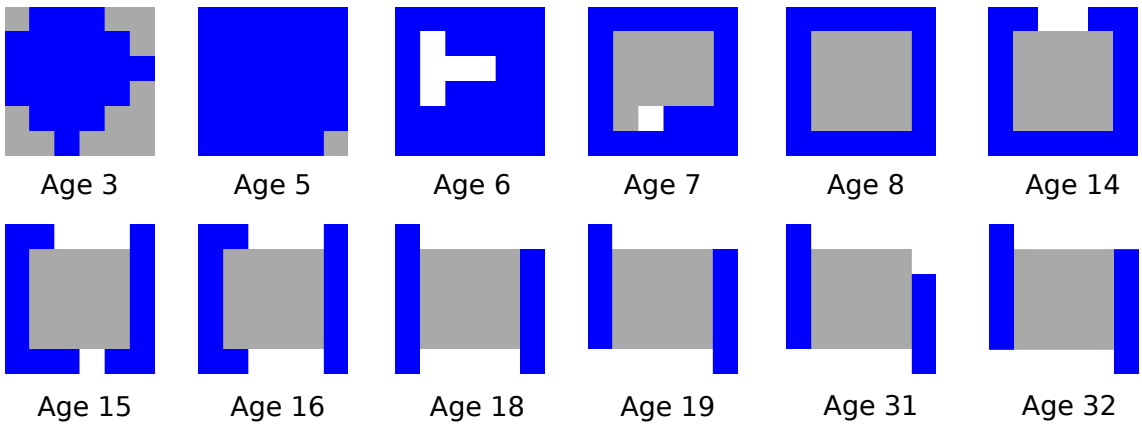


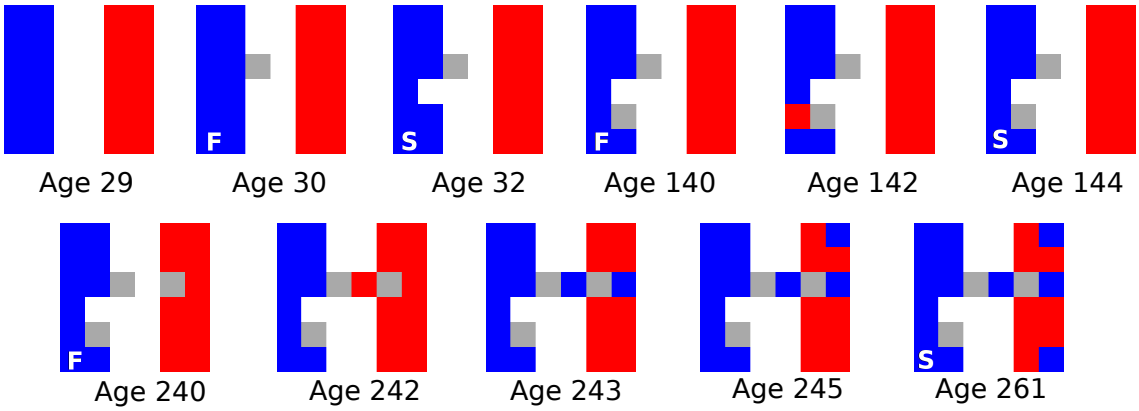
Figure 6.12: *Development of asymmetric borders pattern. The organism stays stable for 15 developmental steps at the target pattern, and every 15 developmental steps it oscillates once between the last two images displayed in this figure.*

In Figure 6.13(a), an already matured organism that formed a French flag incurs complete cell failures at different stages. The effect of the cell failures to the overall organisation of the organism seems rather small; after two cell deaths the organism manages to keep the French flag pattern with only one perturbed pixel (except the dead cells–grey), and after three cell deaths the number of perturbed pixels (again other than the dead cells) goes up to four. The effect of a cell failure in a premature organism, shown in Figure 6.13(b), seems to be larger; three perturbed pixels after one cell failure. However, the overall shape of the French flag pattern is still achieved by the matured organism. The fact that a small number of cell failures do not have a large effect on the overall organisational abilities of the organism suggests that the organism may be highly benefiting from the use of diffusion in the formation of the French flag pattern. Even though certain key cells are killed in a neighbourhood, the state of the majority of the neighbourhood cells are protected. This would not be the case if the organism was purely reliant on direct contact signalling, since the death of a cell means a large change in the information received by its neighbouring cells (a loss of $\frac{1}{4}$ of information).

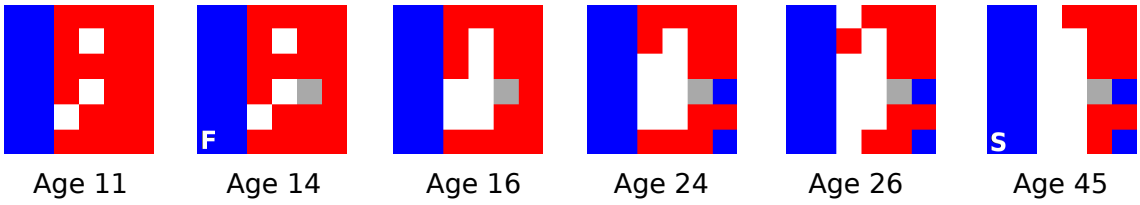
Similar tests are carried out for the asymmetric borders pattern, and the results are illustrated in Figure 6.13. Interestingly, failure of a cell both before and after the organism matured had large effects on the formation of the target pattern. When a cell failure occurred, the pattern formed by the organism diverged from the target pattern. This suggests that the organism evolved for the development of an asymmetric border pattern relied heavily on direct contact signalling; it can also be seen from figures 6.14(a) and 6.14(b) that when a cell is killed all four of its neighbours always change their cell states.

Another method of introducing permanent faults is to knock out genes. The genome of a developmental system is the code required to build and maintain the target organism. Losing part or all of the genetic information would be a permanent fault, and it is expected that such a fault would mean the complete failure of the system. However, it was demonstrated by earlier work that developmental systems can sometimes be unaffected by or provide a "graceful degradation" when genetic perturbations occur rather than a complete failure [Bentley, 2005; Reil, 1999].

Figures 6.15 and 6.16 show some example patterns obtained when some of the genes are replaced with zeroes before the organism is developed. Results obtained in these figures confirm the observations made by [Bentley, 2005] and [Reil, 1999]. Figure 6.15(a)

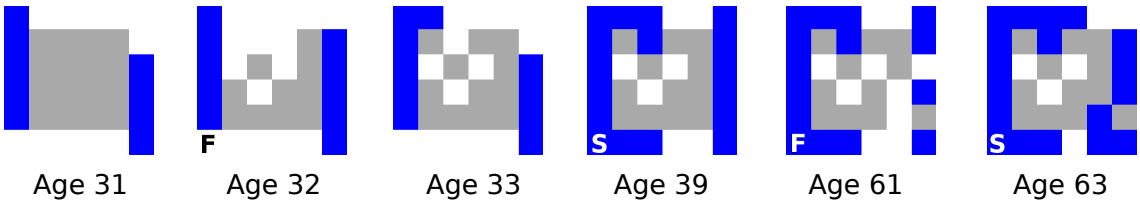


(a) Cell failures after the organism has fully developed.

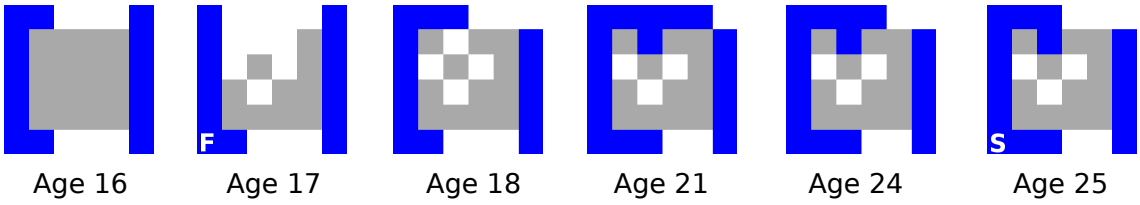


(b) Cell failure at the early developmental stages of the organism.

Figure 6.13: *The changes that occur in the French flag pattern formed by the developmental organism after permanent faults occur. The snapshots where a new fault occur are marked with "F", and snapshots that display the pattern where the developmental organism stabilizes to are marked with "S".*

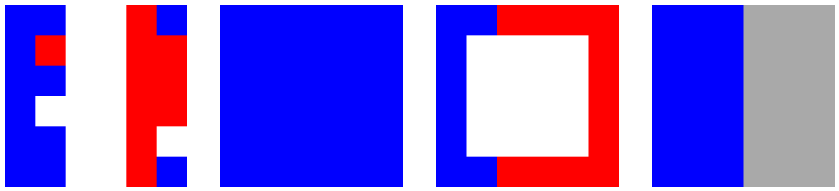


(a) Cell failures after the organism has fully developed.



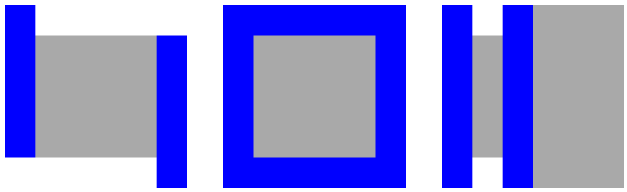
(b) Cell failure at the early developmental stages of the organism.

Figure 6.14: *The changes that occur in the asymmetric borders pattern formed by the developmental organism after permanent faults occur. The snapshots where a new fault occur are marked with "F", and snapshots that display the pattern where the developmental organism stabilises to are marked with "S".*



(a) Gene 16. (b) Gene 6. (c) Gene 0. (d) Gene 26.

Figure 6.15: *Patterns achieved with knocked out genes.*



(a) Gene 1. (b) Gene 30. (c) Gene 29.

Figure 6.16: *Patterns achieved with knocked out genes for the asymmetric borders pattern organism.*

shows a very small amount of degradation in the quality of pattern obtained (with 5 pixel mismatches) when Gene 16 is knocked out from the organism that was evolved to form a French flag pattern. The pattern shown in Figure 6.16(a) matches the target pattern perfectly even though Gene 1 is knocked out from the organism that was evolved to form an asymmetric borders pattern. These results show that degeneracy is inherently present in development and it can provide fault tolerance even in artificial systems. Interestingly, some of the resulting patterns shown in Figures 6.15 and 6.16 show that some genes may have drastic effects on the formation of a target pattern; with roles such as growth (figures 6.15(d) and 6.16(c)), specialisation (figures 6.15(b) and 6.16(b)), or specific structuring (Figure 6.15(c)).

6.4.2 Transient Faults

Transient faults are simulated as temporary loss in information in the neighbourhood of a cell(s). It has been demonstrated that a developmental system can recover from transient changes in the phenotype, despite not being evolved to do so [Federici, 2004; Liu et al., 2005; Miller, 2004; Reil, 1999; Roggen, 2005].

In the experiments presented here, the transient faults are simulated by changing the chemical information present in the system. When a cell is selected to have a transient

fault, the cell itself and its Moore neighbourhood[2] are cleared of all the chemicals that they possess. This way, there are no permanent faults present in the system, but the information gathered over the developmental history of the selected cells are lost.

Figures 6.17 and 6.18 display the changes in the developing pattern when transient faults are introduced to the organisms before and after the organisms have matured. Unlike the permanent faults caused by the killed cells, the organism that forms an asymmetric borders pattern exhibit much better recovery properties from transient faults than the organism that forms a French flag pattern when developed without any faults. In fact, the organism that is evolved to develop an asymmetric borders pattern demonstrate perfect recovery from three different instances of transient faults. When all 36 of the cells in the organism that is evolved to achieve an asymmetric borders pattern are subjected to transient fault all at the same time, the organism still manages to retrieve the target pattern. Whereas the organism evolved to achieve a French flag pattern can not fully recover from even a single transient fault (in case of a single transient fault 2 cells obtain incorrect states). The organism that is evolved to develop a French flag pattern suffers more from transient faults than permanent cell deaths, this may again be due to the heavy reliance of this particular organism on the diffusion mechanism for the organisation of its cells. One can argue that if the organisation of cells is mainly achieved via the diffusion

[2] A Moore neighbourhood of a cell covers the eight cells that surround the central cell.

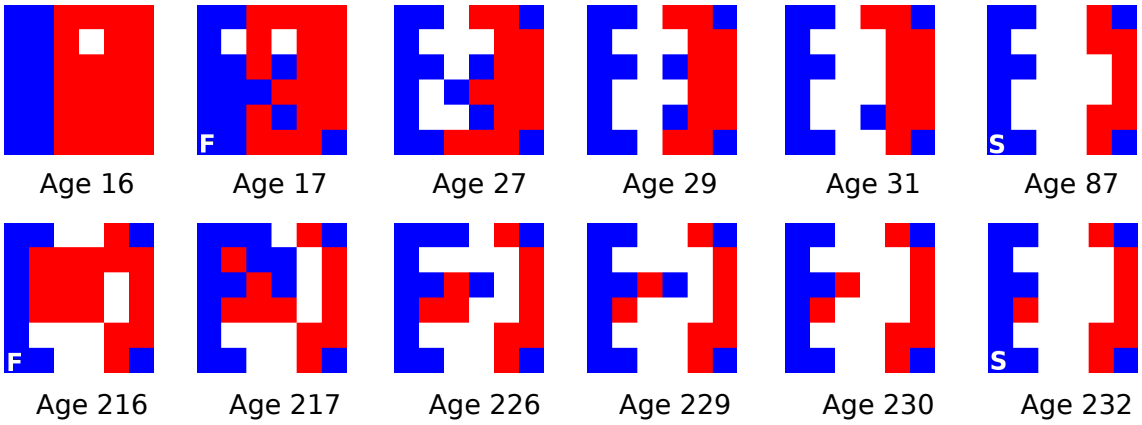


Figure 6.17: *The changes that occur in the French flag pattern formed by the developmental organism after permanent faults occur. The snapshots where a new fault occur are marked with "F", and snapshots that display the pattern where the developmental organism stabilises to are marked with "S". Two sets of transient faults introduced in the same cells of the organism: first one being in the early stages of the development of the organism, and the second fault is after the organism stabilises at a pattern after the first fault.*

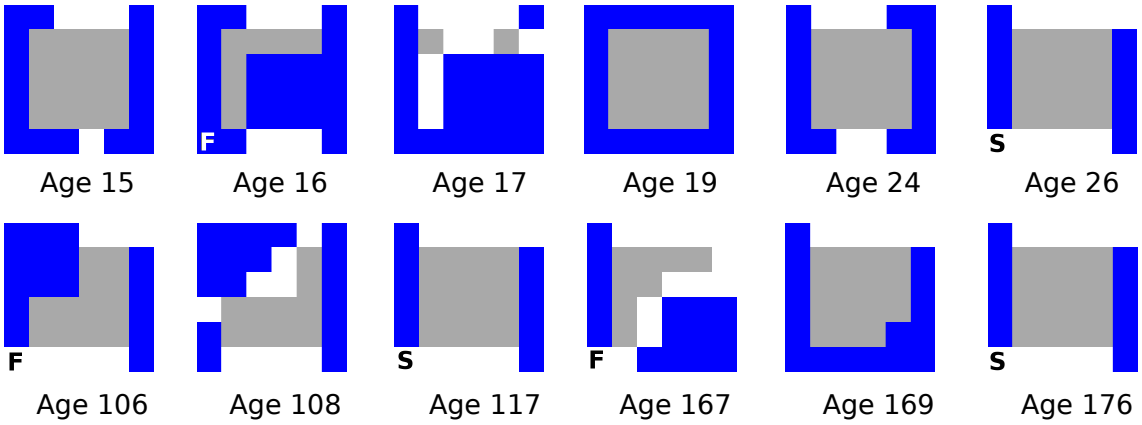


Figure 6.18: *The changes that occur in the asymmetric borders pattern formed by the developmental organism after transient faults occur. The snapshots where a new fault occur are marked with "F", and snapshots that display the pattern where the developmental organism stabilizes to are marked with "S". Three sets of transient faults are introduced in different parts of the organism: first one being in the early stages of the development of the organism, and the other two after the organism stabilizes at a pattern after the previous fault.*

mechanism, then when the chemical information obtained is wiped out in a group of cells the retrieval of the previous states of the cells via diffusion is not possible due to the large change in the chemical gradient. Whereas, if the organisation of cells is mainly achieved via the cell contact signalling, then retrieving the previous states of cells in case of the loss of the chemical information in these cells is possible as long as these cells are surrounded via fully functional cells.

More work in the effects of these signalling mechanisms on the fault tolerant properties of developmental systems is required to support or negate these arguments. Nonetheless, it is interesting to see that one organism that is better at coping with permanent cell failures is not as good as coping with transient faults as another organism. In either of these cases, both patterns demonstrate inherent fault tolerance and recovery, which is an encouraging result on the success of the developmental system presented in this thesis. It is a positive indication that robustness is an inherent property of developmental systems.

6.5 Summary

An initial set of evolutionary experiments are undertaken with the ADS described in Chapter 5. The experiments involved the evolution of the ADS in single cell and multicellular environments.

The single cell experiments demonstrated that the GRN model used is capable of creating internal dynamics in a cell to synchronise itself with the environment. The single cell experiments also showed that the GRN could reach oscillating states purely by it internal dynamics.

The multicellular experiments involved organisation and differentiation of cells to form patterns. These experiments demonstrated that the developmental system is capable of achieving multicellular differentiation and organisation. The states achievable were shown to be both stable patterns and oscillating patterns.

All experiments reveal that the ADS has some biases, and when certain constraints that do not favour these biases are applied during evolution, the process of evolution becomes more time consuming than it should be. It was also shown in the pattern experiments that it is possible to improve the evolutionary performance via various improvements to the ADS and the fitness function. These improvements prompt the question of whether there is more to learn about the ADS in order to improve the ADS and evolutionary fitness function further in order to achieve a more evolvable system.

One of the observations made from both the single cell experiments and the pattern experiments is that the system is more evolvable when the evolutionary constraints are relaxed. For the single cell experiments, achieving the 'ON-OFF' behaviour required $3\times$ more genes in order to keep the 'ON' state always at 0 and OFF state always at '250', instead of a solution with a smaller gap between these two states ('ON' - 150, 'OFF' - 70). In the pattern experiments, it was shown that evolving a pattern by defining the relative locations of colours in the fitness function rather than exact locations improved the evolutionary performance greatly. The improvement in the evolvability of the ADS in these cases is not surprising, since these cases enlarge the number of acceptable solutions within a given search space. This may also suggest that evolution of ADSs is more suitable for problems that does not require precision but complex behaviour. Such a conclusion is not supportive for the use of the evolution of ADSs to build digital circuits, but this will be investigated by simple benchmark problems in Chapter 7.

Finally, two of the selected organisms (the most stable solutions from the evolution of asymmetric borders and French flag patterns) from the multicellular experiments were tested for their fault tolerance properties. It was observed that the two organisms handled

different faults better than the other, but both were shown to exhibit inherent fault tolerant properties. The organisms were shown to be capable of reorganising to the evolved pattern in the event of transient faults that cause loss of developmental data; it was suggested that direct contact signalling may play an important role in this behaviour. The organisms were also shown to be able to keep the evolved pattern in the event of a cell failure; it was suggested that chemical diffusion may be the key to this behaviour. It was also shown that a developmental organism may provide a more graceful degradation in the event of loss of genetic information or even be immune to some loss of genetic information.

The brief demonstrations of organisms'–which were evolved without fault tolerance as an objective function–ability to recover from various faults highlight the large scope of fault tolerant properties that ADSs can bring to evolutionary computation. The developmental model described in this thesis is certainly one of these ADSs.

An interesting observation that was made in all the experiments presented in this chapter was that even though a variable mutation rate was used, the mutation rate was seldom varied by evolution, and the lowest mutation rate was used most of the time for each evolutionary generation. Jin and Trommler note that evolvability increases during evolution more when the mutation step-size is smaller[Jin and Trommler, 2010].

Publications III

The work presented in the next chapter (Chapter 7) has been presented in these published papers:

Kuyucu, T.; Trefzer, M.; Miller, J. F. & Tyrrell, A. M. On The Properties of Artificial Development and Its Use in Evolvable Hardware. *IEEE Symposium Series on Computational Intelligence - IEEE SSCI 2009*, 2009, 108-115

Kuyucu, T.; Trefzer, M.; Miller, J. F. & Tyrrell, A. M. A Scalable Solution to N-bit Parity via Articial Development. *5th International Conference on Ph.D. Research in Microelectronics & Electronics – PRIME 2009*, 2009

Chapter 7

Developing Digital Circuits

It was demonstrated in Chapter 6 that the Gene Regulatory Network (GRN) modelled in the ADS is capable of producing responsive, stable and dynamic states using chemicals for gene interaction within a cell. When the GRN was used as part of a larger network of cells, it was shown that the ADS could be evolved to form various organizational patterns. It was shown that the ADS's evolvability could drastically change with the fitness and physical constraints. The evolution of the ADS to achieve a target state was shown to be highly dependent on successful fitness description and correct developmental parameters. In this chapter the ADS is evolved for the design of digital circuits. A minimally constrained approach will be taken in the hope of discovering the full potential of using ADS for the evolution of circuits, as well as finding out about the weaknesses of evolution of a GRN based ADS in circuit design.

7.1 Mapping the Developmental Organisms to Circuits

The multicellular organism developed in the experiments use a similar structural representation as described in Section 6.1. In the case of circuits, an n-bit (n being a predefined value; a multiple of 64) size binary string that represents the cell function is divided into multiple 16-bit chunks (each 16-bit representing an integer). Four 16-bit integers are then used to create a Cartesian Genetic Programming (CGP) node. A CGP node has 3

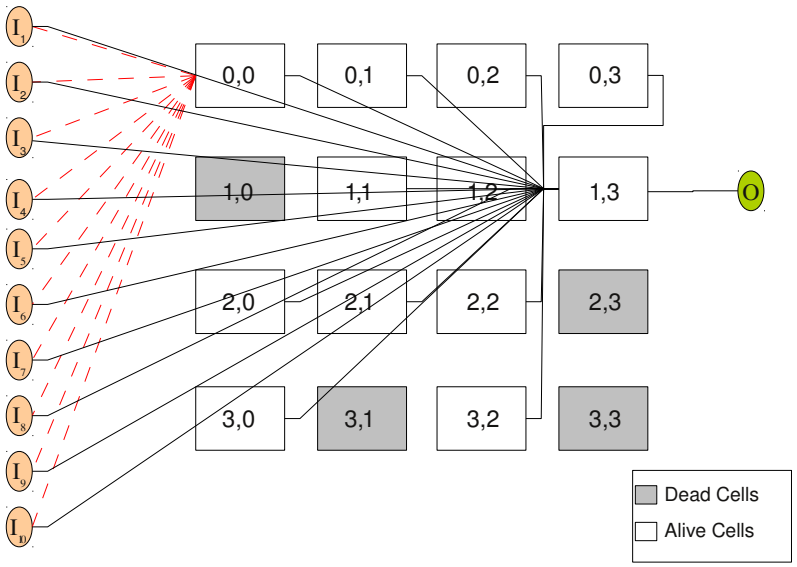


Figure 7.1: *An example* 4×4 *organism with* 10 *inputs is shown. The available structural connections for cells 0,0 and 1,3 are drawn to illustrate how the cells can connect to form a CGP circuit. The cells in the organism are organised into rows and columns. Each available cell is given a coordinate value in the organism as* X,Y*;* X *being the row the cell is at, whereas* Y *represents the column of the cell.*

inputs (from the external inputs or previously defined nodes) and a function (a digital multiplexer in this case). Each cell can have one or more CGP nodes as its cell function. A feed-forward CGP representation is used for building combinational circuits, thus each node within a cell can connect to the other nodes within the cell as well as the outputs of other cells. The maximum organism size is predefined, and the cells are ordered in columns to form an organism of size $n \times n$. Each cell has 3 external connections, and they are allowed to connect to the circuit inputs and cells from previous columns as well as cells from the same column but previous rows, as shown in Figure 7.1. For the first time, the design of a digital circuit via a GRN based ADS includes the connectivity of the circuit components. If the ADS can handle designing the routing as well as the logic configuration, it should remove the need to provide a pre-wired circuit topology, and allow evolution and development to work more freely in the design of digital circuits. However, developing the routing may make the developmental task more complex and remove the scaling benefits that a developmental system may bring about.

In a mature organism, not every cell needs to be alive, and even though the dead cells occupy space in the organism, a live cell's circuitry can not create connections to a dead

Table 7.1: *The multiplexers used for the experiments presented. A, B and C are the inputs to the multiplexer.*

Multiplexer Type	The Circuit
MUX1	$(A \,\&\, \overline{C}) \| (B \,\&\, C)$
MUX2	$(A \,\&\, \overline{C}) \| (\overline{B} \,\&\, C)$
MUX3	$(\overline{A} \,\&\, \overline{C}) \| (B \,\&\, C)$
MUX4	$(\overline{A} \,\&\, \overline{C}) \| (\overline{B} \,\&\, C)$

cell. Initially every organism starts with a single live cell, in the middle of the reserved organism area, and can grow to cover as much of the area as required. The output of the circuit built can be taken from any of the cells, but for most of the experiments this will be fixed to the "last" live cell in the organism. In the example shown in Figure 7.1, this is cell *1,3*. The components used for building the structural part of the cells are 4 different types of MUXes, shown in Table 7.1. Mapping the cell function to CGP nodes is shown in Figure 7.2. A single CGP node is represented by a 64-bit binary string, and each CGP node is formed of three input values and a component type value. For more information on CGP mapping and the processing of a CGP network see [Miller and Thomson, 2000].

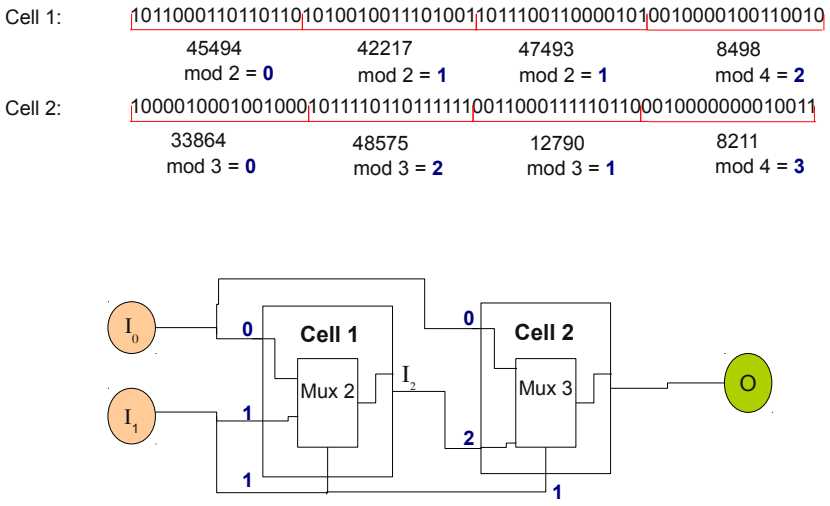


Figure 7.2: *An example decoding process is shown for 2 cells, each with a single CGP node (i.e. 64-bit long cell function).*

7.2 Circuits Developed

In these experiments, the ADS is evolved to develop several different circuits. These circuits are classic examples used in Evolvable HardWare (EHW), namely; parity and multiplier circuits.

The number of chemicals in an organism allowed in these experiments are composed of four proteins; three of them being "structuring" proteins, while the fourth is a "plasmodesma" protein, as detailed in Section 5.2.2. The organism is allowed to age for 10 developmental steps and then evaluated once at age 10. Hierarchical Bit-string Sampling (HBS) fitness function, described in Section 3.5.1, is used to check if the developed circuit functions correctly. Therefore, the developmental system goes off-line once it reaches maturity in these experiments. The Evolutionary Algorithm (EA) used is as described in Section 6.1. The maximum number of evolutionary generations is 2 million.

7.2.1 Development of Even n-bit Parity Circuits

Parity problem is easy to implement and popular amongst EHW researchers. It has been used previously to test/demonstrate the scalability of a proposed system [Gordon, 2005; Koza, 1994]. Thus, it is regarded as a suitable initial test problem for the ADS. The ADS is evolved that solve 5, 7, 9, 10, 11, 12 - bit even parity circuits.

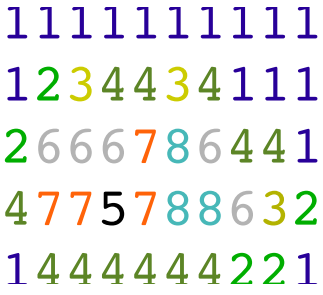


Figure 7.3: *The cells alive at the end of the fully developed organism of a 3-bit even parity circuit are categorized via enumeration from 1-8. The type (category) of each cell is determined by the genes active in the cell.*

In the even n-bit parity experiments, the organism is limited to 100 cells (10×10 cells) with each cell having 3 inputs, 1 output and a 2 component structure, and the number of genes per genome is defined as 50, and each gene is represented by a 64 bit number. Thus, the evolved genome is 3200 bits long ($64[\text{bitspergene}] * 50[\text{genes}]$).

The circuits evolved display a large amount of reuse in genes; Figure 7.4 shows a snapshot of the gene activity in two cells with different structures. A large number of the active genes in the two cells overlap, and the differentiation of the cells emerge from the activation of few different genes. In the cells shown in Figure 7.4, more than half of the genes are inactive in a single cell, which was the case for all other cells in the same organism. For the same organism approximately $\frac{1}{3}$ of the total number of available genes were never used. Even though there is a good deal of gene reuse, when considered in detail, the solution for the 3-bit parity appears to be formed by a highly differentiated organization of cells. Figure 7.3 shows the organisation of differentiated cells in the organism that solves a 3-bit parity circuit using MUXes. Although there is a large amount of reuse for cell type 1, the number of cell types is surprisingly large (8 different types) for a symmetric circuit such as 3-bit parity.

Table 7.2: *Results of the even parity experiments. Each experiment is carried out 30 times; the number of successfully evolved circuits, and total number of generations per average successful run are listed.*

Circuit Evolved	Evolutionary Mechanism	Results	
		Solution Found	Average Generations
5-bit Parity	Direct Evo.	30	1060
5-bit Parity	Development	30	2072
7-bit Parity	Direct Evo.	29	4430
7-bit Parity	Development	30	7216
9-bit Parity	Direct Evo.	30	12744
9-bit Parity	Development	30	22024
10-bit Parity	Direct Evo.	30	79160
10-bit Parity	Development	29	88222
11-bit Parity	Direct Evo.	30	150456
11-bit Parity	Development	30	161858
12-bit Parity	Direct Evo.	30	395582
12-bit Parity	Development	30	288481

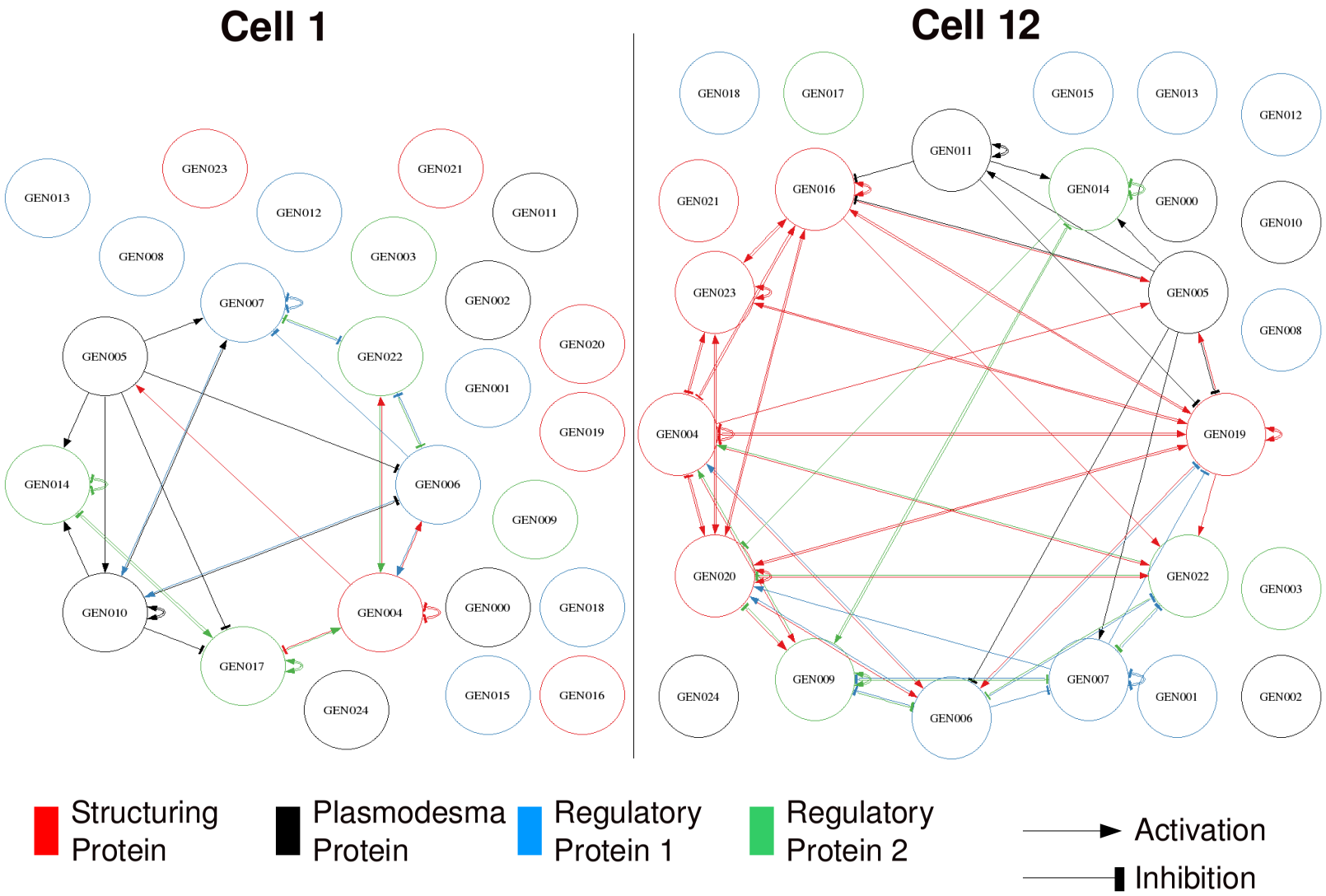


Figure 7.4: *The GRN interaction graph of two fully developed cells in a 3-bit parity circuit is illustrated as examples. Each circle refers to a gene in the DNA; the genes that are used (active) in a cell are the ones that are connected to other active genes via activation/inhibition arrows. An arrow head pointing to a gene means that the gene can be activated by the activation of the gene the arrow head is pointing from, and a flat head pointing to a gene means that the gene can be inhibited by the activity of the gene the flat head is pointing from. Each gene is coloured by the type of protein they produce, i.e if a gene produces a structuring protein it's coloured red, if it produces a plasmodesma protein it's coloured black and vice versa.*

On almost all of the runs presented in Table 7.2 development has a perfect success rate in building parity circuits. It is a good achievement for development that it is able to directly produce functional circuit structures in a reasonably small amount of time. When compared with direct evolution, the average number of evolutionary generations for development to reach a successful run is most of the time higher than direct evolution. However, the gap between the number of evolutionary generations required for each method decreases as the size of the parity circuit is increased, and for the largest parity evolved (12-bit) the evolution of the developmental system requires less generations than the direct evolution of the 12-bit parity circuit. The trend seen in Table 7.2 suggests that using development for larger size parity circuits would be more efficient. The common way of developing circuits in literature has been to use pattern mapping – that removes the need to deal with routing while developing the circuit – when a circuit was evolved using a developmental system. The developmental system can then develop a pattern rather than a complete description of a circuit [Gordon, 2005; Liu, 2007; Tufte, 2008a].

The figures presented in Table 7.2 show that the ADS's performance (in terms of number of average evolutionary generations required to find the target circuit) decreases as the circuit size gets bigger. This is an undesirable trend as it suggests that development is not exploiting the scalable nature of the parity problem. This is also shown in the earlier example of the 3-bit parity circuit. In Figure 7.3, it's shown that there are 8 different types of cells in order to implement 3-bit parity circuit, and the pattern shows no particular order or symmetry to it. There are a number of factors that could be the reason for this, but it's hard to point where to start. The immediate and most obvious constants that may have adversely affected the ADS performance in developing parity circuits are; the organism size, number of components used in a single cell, and the organism maturing age. Larger circuits may require a larger number of cells available in order to implement a scalable solution. Using two MUXes per cell may create a larger challenge for development to come up with a general solution to parity. Whereas it is possible to create an XOR gate using a single MUX, which is the basic building block for a parity circuit. Using ten developmental steps to mature the developmental organism may be too low for the larger parity circuits to emerge.

Table 7.3: *Results of 9-bit parity experiments with different developmental steps. 30 evolutionary runs were done for each experiment listed.*

Circuit Evolved	Cell Age	Solutions Found	Average Generations	Total Developmental Computations
9-bit Parity	7	30	29321	1,436,729
9-bit Parity	10	30	22024	1,541,680
9-bit Parity	15	30	17038	1,788,990
9-bit Parity	25	30	17153	3,001,775
9-bit Parity	35	30	14361	3,518,445
9-bit Parity	42	30	13837	4,068,078
9-bit Parity	50	30	13601	4,760,350
9-bit Parity	75	30	10780	5,659,500
9-bit Parity	upto 30	30	37321	$\sim 3,918,705$

A simple set of experiments are undertaken to investigate the effects of using larger maturing ages for the developmental organism in building 9-bit parity circuits. Table 7.3 gives the results to these tests.

Looking at Table 7.3, it is interesting to see that the number of evolutionary generations required for the developmental system in developing a fully functional circuit decreases as the number of developmental steps (cell age) required before the organism is considered mature is increased. This indicates that the developmental system might become more evolvable if it is developed for longer. However, every time the maturing age is increased, the total number of developmental computations increase. Developmental computation is the total number of steps an organism is developed before the evolutionary run ends. This is the total number of evolutionary generations multiplied by the number of developmental steps the organism is run every generation multiplied by the population size (7 in this case). Using a non-deterministic maturing, as suggested in Section 6.3, with a maximum developmental age of up to 30 did not improve the evolvability of the ADS either.

It should be noted that development is able to determine the number of cells it needs to use for the problem evolved rather than using every single available cell location, thus the maximum number of cells available for the organism should not determine the size of the final organism.

Table 7.4: *Results of the development of larger parity experiments (10–12 bit) with bigger organism sizes. 30 evolutionary runs were done for each experiment listed.*

Circuit Evolved	Organism Size	Sol Found	Avg Gens	Std Dev
10-bit Parity	10x10	29	88222	112452
10-bit Parity	11x11	30	93594	168272
11-bit Parity	10x10	30	161858	166388
11-bit Parity	12x12	30	110454	123963
12-bit Parity	10x10	30	288481	378401
12-bit Parity	13x13	28	257757	349526
12-bit Parity	15x15	27	242716	176216

The developmental system was provided with a fixed maximum organism size regardless of the size of the tackled problem in order to determine whether it would be capable of determining the amount of resources it needs to use. In the experiments carried out, development is able to determine the number of cells it needs to solve the problem, rather than using every single available cell location. Thus, the maximum number of cells provided is not always required for the final organism, and the developmental system is capable of controlling the growth of the organism (even without the use of programmed cell death). For example, for all the 12-bit parity circuits all the cells were alive for the fully developed organism, whereas for the 5-bit parity circuit most of the time only 30-50 cells (out of 100) would be alive for the fully developed organism. This example is illustrated in Figure 7.5, where the number of cells alive/dead of actual organisms for fully developed 5-bit and 12-bit parity circuits are shown.

To investigate the effects of using a bigger organism on the performance of the developmental system in finding valid circuits, the experiments for developing 10, 11, and 12 bit parity problems are carried out with more resources. Increasing the organism size did lower the number of generations in most cases as it can be seen from Table 7.4. In one case (namely the 12 bit parity), the success rate slightly dropped with the increase of organism size. However, the figures shown in Table 7.4 are not significant enough to suggest any change in the evolvability of the ADS with change in organism size.

Figure 7.5: *The final cellular states of the two fully developed organisms are shown: for 5-bit parity only 30 of the cells are alive, whereas for 12-bit parity all 100 of the cells are alive.*

Table 7.5: *Results of the 2-bit multiplier experiments are given. 30 evolutionary runs were done for the experiment listed.*

Circuit Evolved	Sol Found	Avg Gens	Std Dev
2-bit Multiplier - Development	20	821668	482611

7.2.2 Development of a 2-bit Multiplier

As a second demonstrator application, a 2-bit multiplier is evolved to see if the proposed developmental system is able to tackle problems that have multiple outputs, and see how it manages solving problems that are not as symmetric and repetitive as even parity. For the multiplier experiments the organism is limited to 100 cells with each cell having single component structure (using the MUXes in Table 7.1). The number of genes per genome is defined as 100. The number of proteins used are four; three of them being structuring, while the fourth is a plasmodesma protein. The organism is allowed to age for 15 developmental steps and then evaluated. The maximum number of generations is 2 million and the EA parameters are as described in Section 6.1.

The developmental system is successfully evolved to develop a 2-bit multiplier 20 out of 30 times, which shows that the ADS is capable of developing multiple output circuits. The number of successful runs in developing multiplier circuits is lower than the number of successful runs in parity circuit experiments, which is not surprising since the 2-bit multiplier does not have as symmetric and repetitive structure.

From the experiments presented here it is observed that development is capable of developing digital circuits. However, the amount of scalability that is expected from an ideal developmental system is not demonstrated. Due to the unordered structure of a 2-bit multiplier, it is expected that evolving a 2-bit multiplier is not suited for a developmental system, and being able to do so presents a positive result to show that the developmental system is capable of growing highly differentiated cells.

It was stated at the start of this section that HBS fitness function is used for the evolution of developmental organisms that grow to be digital circuits. It was demonstrated in Chapter 3 that HBS is effective in taking care of transient effects as well as defining a positive bias for the parity circuit. Although HBS is a valuable fitness function for the evolution of circuits in hardware, it lacks useful information and provides unnecessary information for the evolution of scalable developmental circuits in simulation. This is because the ability of HBS to detect intermittent circuits due to transient effects in the evolved circuits is not useful for extrinsic evolution, where transient effects do not exist. On the other hand, HBS fails to define a scalable description of a parity circuit. By using HBS or bitwise fitness functions to evaluate the circuit developed, the fitness function is not asking evolution to search for scalable solutions, and without any guidance the evolutionary search is easily trapped in a local optima.

It is also possible to guide evolution in achieving scalable solutions via the evolutionary environment, i.e. by constraints. An example of this is provided by Gordon's example of n-bit adder problem; he provides the correct routing for a scalable adder circuit and evolves a developmental organism that provides the contents of the look up tables that make up the circuit. Without the fixed routing that favours an n-bit adder circuit, evolving a scalable adder organism would not be as feasible. Therefore, allowing the developmental organisms to manage the connectivity of their phenotype was possibly another reason for not achieving scalable solutions to the parity circuits.

7.2.3 Developing a Parity Solving Organism

In order to develop a scalable parity circuit, the fitness function for evaluating the developmental organism is changed. An organism is initially evaluated to behave as a 2-bit parity, i.e. an XOR gate. The evaluation is done when the organism reaches develop-

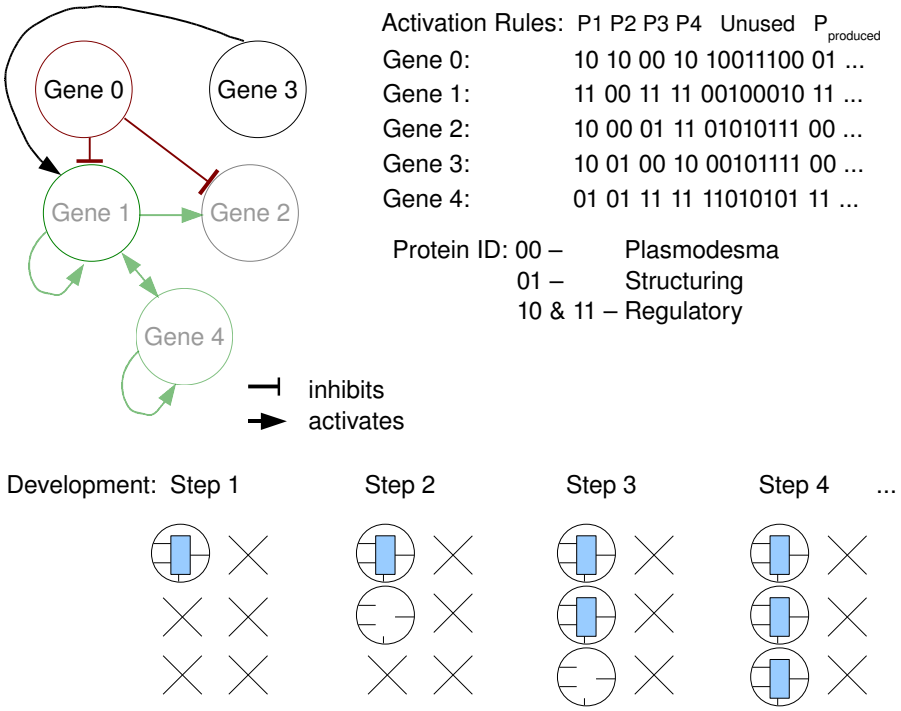


Figure 7.6: *An organism that was evolved to act as a growing parity circuit is shown, with the list of its evolved genes and the GRN graph for the alive cells.*

mental step 3, and the maximum size of the organism is limited to 2×2. Once the 2-bit parity is achieved, the organism is then developed further with larger organism size and expected to provide a larger parity solving circuit each developmental step.

To achieve this behaviour, a few other changes were made to the developmental model. The number of external inputs were allowed to be variable rather than a constant, and as well as allowing the cells to pick their inputs, they were also allowed to specify a more generic connectivity such as "next set of unconnected inputs". This last change provided a convenient way for the developmental model to manage the connections of a growing circuit.

The number of genes were reduced to five, and four proteins were used again, this time only one as the structuring protein, one plasmodesma protein, and two regulatory proteins. An organism is successfully evolved that is a growing n-bit parity solver. As shown in Figure 7.6, the GRN of the evolved organism keeps replicating an XOR gate for each row of cells.

The growth of the parity solving organism in more detail is shown in Figure 7.6. The example shown demonstrates the proposed developmental system's ability to achieve modularity and scalability. At developmental step 1 only one cell is alive in the organism. Looking at the GRN rules it is seen that Gene 0 and 3 do not need any promoting proteins in order to activate and the inhibiting protein (*P3* for both genes in this case) initially does not exist (initially no proteins exist in the system). Only Gene 0 and 3 would be active at the first stages of the GRN as the other genes require the presence of promoting proteins in order to be active. Gene 3 produces plasmodesma protein that triggers a growth process to the south of the parent cell (encoded in the postconditional bit-string, which is not shown in the figure). On developmental step 2 the second cell grows and shares its proteins with the parent cell, creating an identical structure and connecting its inputs to the next available input and the previous cell (i.e. the "next set of unconnected inputs"). By step 3 the organism has two identical cells, each implementing an XOR with a MUX. The developed organism in this example demonstrates a modular and redundant behaviour by replicating the same physical structure in both of the cells, which are desirable features for a scalable system. More importantly, when the organism was allowed to develop further (beyond step 3), the daughter cell from step 2 triggers a growth process into its southern neighbour in step 3, which would then become a chain of growth events, resulting in a growing parity solving organism. Hence achieving *n*-bit parity circuit in *n* developmental steps.

By defining the fitness function carefully and altering the genotype–phenotype mapping to allow scalable circuit design, a scalable n-bit parity is shown to be possible to evolve using a small number of genes.

7.3 Summary

It was demonstrated in this chapter that the ADS is capable of designing digital circuits. Both the logic and routing configurations were presented to the ADS as part of the circuit design problem, and the ADS was able to provide valid solutions.

The results demonstrated a large amount of cell differentiation as well as design redundancy. In fact, the amount of cell differentiation observed was surprisingly high; for a

3-bit parity circuit there were 8 different types of cells used. The fact that the developmental system required so many specialized cells to produce a simple repetitive circuit like 3-bit parity suggests that the evolutionary conditions were preventing a simpler scalable solution. Two of the major reasons for this were shown to be the unsuitable fitness function and environmental constraints. Allowing the ADS to design the circuit connectivity caused the circuit problem to be organisationally a tough problem. By adjusting the fitness function and providing better phenotype mapping, it was shown that the evolution of a scalable n-bit parity becomes a simple task for the ADS. One other reason that may have reduced the evolvability of the developmental system in the earlier parity and multiplier examples can be the large number of genes used that created a search space of 3200 bits. It was mentioned in Chapter 2 that the search space size of an evolutionary algorithm may be limited to 1000 bits [Sekanina, 2006]. However, when the number of genes were reduced in these experiments, there was no visible increase in the evolutionary performance.

It was shown that a suitable fitness function is important for the evolution of scalable systems. It was argued in Section 7.2.2 that, HBS as a fitness function for the evolution of a developmental system lacked some necessary information. If the aim is to achieve scalable systems, a favourable fitness function for the evolution of a developmental system, whatever the task may be, should be encouraging scalable behaviour. One way of achieving this is to divide the evaluation of the organism phenotype into multiple stages [Federici, 2004].

Although the evolution of an ADS building more complex digital circuits could be seen as the next natural step, the observations made thus far from chapters 6 and 7 suggest that the information known on the evolution of ADS for the design of scalable computational systems and circuits is insufficient. The increasing computational effort for evolving ADSs that build larger sizes of parity circuits via the use of multiplexers was not expected, and there remains a largely unknown number of parameters and mechanisms that can be altered to influence the behaviour of development. The influence on evolvability of a large list of developmental mechanisms and parameters are currently unknown; such as diffusion, contact signalling, protein production rate, chemical consumption rate, chemical thresholds for protein to gene binding, the method of mapping the developmental organism to the desired phenotype, and the list goes on. All these

mechanisms and parameters can have a substantial effect on the evolvability of the developmental system, however, not much investigative work exist in the literature on the effects of these mechanisms. Therefore, it seems more natural to investigate further the effects of some of the mechanisms and parameters in order to obtain a better understanding of multicellular development. With a better understanding of ADS, the ADS and its working environment could be tuned better for real applications. Chapter 8 undertakes a detailed investigation of the importance of many developmental mechanisms and parameters on the behaviour of the developmental system. The work presented should lead to an improved understanding of artificial multicellular development, which will assist in its utilization in the application of evolutionary computation. The experiments undertaken in Chapter 8 may also provide a better understanding of mechanisms of biological development.

Chapter 8

Developmental Mechanisms and Parameters

Gene Regulatory Network (GRN)s are complex dynamic systems, and there is little understanding about how they work and what their best domain of application in computing would be. There is existing work that undertakes detailed experiments in order to demonstrate and investigate the evolution and behaviour of artificial GRNs; such as Banzhaf's work on investigating the dynamics of artificial GRNs [Banzhaf, 2003], and Knabe's work on investigating the evolvability of artificial GRNs [Knabe et al., 2008, 2006].

Although multicellular Artificial Developmental System (ADS)s have been demonstrated to provide scalability, fault tolerance, and adaptivity in Evolutionary Computation (EC), they are also poorly understood. Hence, tuning the parameters and mechanisms of a GRN based ADS is a real challenge. Implementations of artificial development in EC has been proposed since early 90's; e.g. [Dellaert and Beer, 1994; Fleischer and Barr, 1993]. But most of the developmental models designed still rely on educated guesses, and various assumptions on the suitability of the biological developmental processes for EC. The need to investigate the behaviour and effective ways of implementing artificial development is already acknowledged by various researchers [Devert et al., 2007; Haddow and Hoye, 2009, 2007; Stanley and Miikkulainen, 2003; Steiner et al., 2008]. The physics of the digital medium where the artificial development is modelled is different from the biological counterpart. A biological cell has many advantages and disadvantages when

compared with a digital cell, e.g. the ability to copy the genotype perfectly from one cell to another is a given in digital systems whereas small errors are inevitable in biological organisms. On the other hand, the cell growth in biology is naturally allowed by the physics of the organism whereas this has to be engineered in digital systems. Hence when implemented artificially, the usefulness of a mechanism can not always be directly correlated to biology.

It was demonstrated in chapters 6 and 7 that the GRN based ADS can be evolved successfully to achieve stable pattern organisation and scalable circuit design. However, many parameters and mechanisms of ADS and the GRN are not well understood, and it is unknown whether the results achieved in chapters 6 and 7 reflect the best results that can be achieved by the GRN based ADS used in this thesis.

It was shown in Section 6.3 that it is possible to improve the evolvability of the ADS by using non-deterministic maturing during the evolution of patterns. This hints that the ADS has the potential to be further improved. Undertaking experiments to understand several of the mechanisms and parameters of an ADS will enable the improvement of the ADS, and provide an idea on how much more the ADS can be improved by further investigations.

In the experiments presented so far, the tuning of the developmental system and the inclusion/design of multicellular developmental mechanisms (e.g. cell signalling) was mostly done by making educated guesses about the behaviour and effects of these mechanisms and parameters. Although there are various implementations and uses of developmental systems in the literature, there is little exhaustive work on the investigation of the effects of developmental parameters and mechanisms on artificial multicellular development. Without thorough investigation the tuning of a complex dynamical system like an ADS is not feasible, and before using an ADS to solve complex problems, it is important to understand the effects of mechanisms and parameters that might have a drastic impact on an ADS's performance.

8.1 Experiments on Mechanisms and Parameters

In this chapter we investigate the importance of various mechanisms and parameters on the ability of the ADS to successfully achieve the intended behaviour. These mechanisms and parameters are:

- *Diffusion*
- *Direct contact signalling*
- *Protein production and chemical consumption rates*
- *Chemical thresholds for gene binding*
- *Various chemical/protein types*
- *Various ways of constructing the phenotype of a multicellular developmental system*
- *Preconditional and postconditional decision methods*
- *Artificial simulation of food reliance*

The problem of explaining how genetic mechanisms can produce the many differentiated patterns of cells found in multicellular biological organisms remains a fundamental research challenge in molecular and developmental biology [Wolpert et al., 2002]. Inspired by this a series of benchmarks based on pattern formation are devised. Although developing spatial patterns does not appear directly useful in engineering applications, it proves to be very useful for examining and comparing micro-model developmental approaches and ascertaining the importance of mechanisms and parameters. Three types of patterns are used, which were also examined in [Flann et al., 2005], namely:

- *Mosaic patterns*
- *Border patterns*
- *Patch patterns*

Three main pattern types were selected in order to test the ADS on building ordered structures of different types. ADSs have been well established in successfully matching ordered patterns, and ordered patterns are a common part of biological development as well. Hence, using ordered patterns of different regularities is a convenient and intuitive way of providing initial benchmarking experiments to ADS.

These three types of patterns demonstrate different ordering characteristics and complexities in patterns for the experiments testing the characteristics of the ADS. A mosaic pattern is a periodic pattern that repeats a smaller motif over the whole pattern area. A border pattern is a group of cells (pixels) isolated by a thin layer of different type of cells. Finally, a patch pattern is the division of different groups of cells into an aperiodic partitioning of groups of cells of the same type. These three types of patterns are also common in natural development, and they can be seen in various parts of different developmental organisms [Flann et al., 2005]. Out of these three categories six patterns are chosen as test patterns for all the experimental scenarios. These patterns are illustrated in further detail in Figure 8.1. Each pixel of a pattern is represented by a cell, which specifies the colour of the pixel as the cell type. For all the patterns except the 8 patches pattern in Figure 8.1, the maximum number of colour types is set to 4 (i.e. colours range from $0 \rightarrow 3$). For 8 patches pattern the maximum number of colour types is increased to 8.

The experiments described in this chapter are composed of 50 evolutionary runs for each case, each evolutionary run being limited to 1 million evolutionary generations. The evolutionary algorithm is the one described in Section 6.1. Non-deterministic maturing (first explained in Section 6.3.1) is used for all the runs, and the maximum maturing age is limited to 30. The value 30 is chosen to allow the developmental organism plenty of time to develop a mature organism. This value was chosen as a result of general knowledge obtained from previous experiments, and it does not rely on any exhaustive data about its effect on the evolvability of the developmental system. Each run that achieves the perfect representation of the target pattern is evolved further (until the generation limit is reached) until a stable solution is found. To test the stability of a run, the organism is developed for ten extra developmental steps (e.g. $19 \rightarrow 28$) and for each step matched to the target pattern. If an organism can form a fixed pattern for *ten* consecutive developmental steps, it is marked as a stable solution. Of course in reality not all the solutions marked as stable are infact stable. Testing a candidate solution for 100%

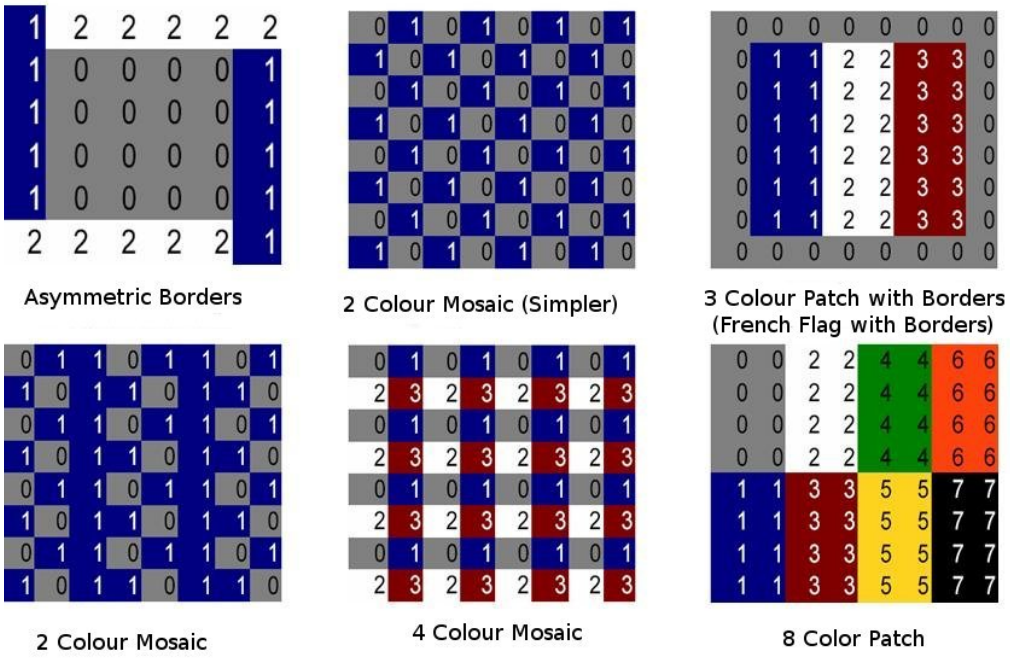


Figure 8.1: *The patterns used for investigating the influence on evolvability of developmental mechanisms and parameters. A border pattern with asymmetric borders of size 6x6 is the first pattern. The next three patterns (size 8x8) are of mosaic nature with different complexities. The simple 2 colour (simplest of the three) and 4 colour mosaic patterns have a 2x2 size for the periodic pattern, whereas the 2 colour mosaic pattern (most complex of the three) has a 4x2 size for its periodic pattern. The next two patterns (size 8x8) can be classified as patch patterns, one being the popular French flag pattern and the other one a patch pattern of 8 colours. A border around the French flag pattern is added making it a hybrid – bordered patch pattern.*

stability is too time consuming to be incorporated into the evolutionary phase. The test used here filters most of the transient solutions, and encourages (but not force) evolution to find stable solutions.

8.2 Direct Contact Signalling

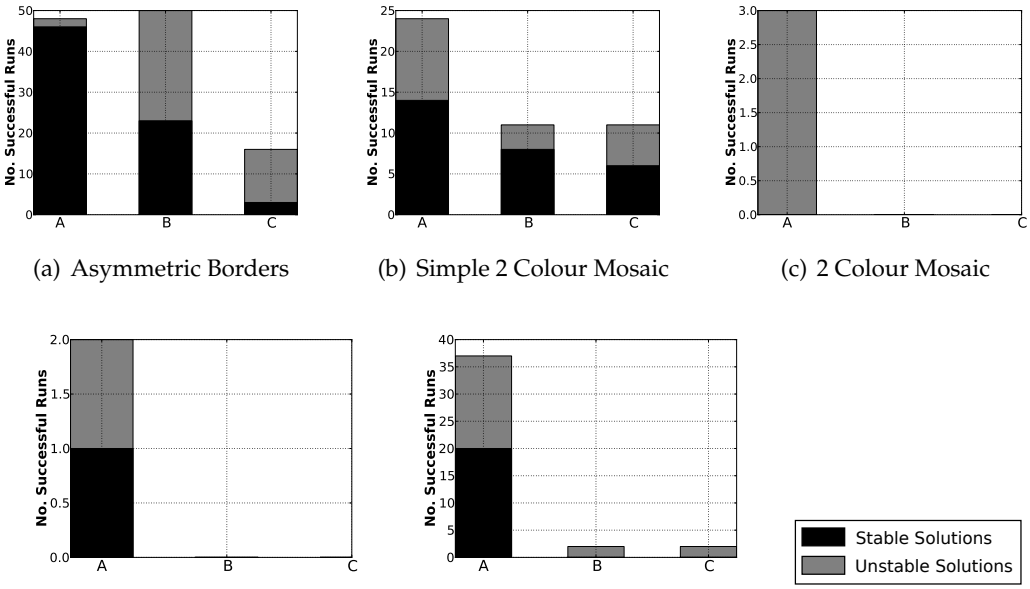
Contact signalling is the most popular cell signalling model in ADSs, see Table 5.1. It is simple to implement, and biological evidence strongly suggests that direct contact signalling is an essential part of development [Fagotto and Gumbiner, 1996]. Experiments in this section will investigate the usefulness of direct contact signalling. Three versions of the developmental model with different contact signalling mechanisms are investigated in the experiments. In the ADS used here, contact signalling is established

by the mutual transcription of plasmodesma proteins (described in Section 5.2.3), which form connections between two neighbouring cells. These connections are formed as plasmodesmata, a tunnel, between the two cells.

- **Simple tunnels**: The initially designed version of plasmodesmata between two cells involved a "simple tunnel"; no filtering of chemicals was done, and all the chemicals available in both cells were entirely shared amongst the two cells.
- **Controlled tunnels**: A more complex version of the original design. When a plasmodesmata is formed, the chemicals are not freely shared between the two cells. A chemical may pass through the tunnel only as a result of further gene regulation. A chemical is transferred through an existing tunnel when a plasmodesma protein coded with the concentration and identity of the chemical to transfer through the tunnel is expressed by a gene. This approach of contact cell signalling is more complex than the original design, but it provides individual cells the chance to protect their specialisation in case of an emerging plasmodesmata with their neighbour. In the original design, the cells that are connected together were forced to share all their chemicals creating two identical cells at the cost of loss of the identity of both cells, causing loss of information. Simple tunnels may have had adverse effects on the ADS performance by making cell specialisation a tougher task.
- **No contact signalling**: In this case direct contact signalling is disabled, i.e. no plasmodesmata (tunnels), leaving diffusion as the only cell signalling process within the organism.

All three contact signalling mechanisms described above are accompanied with a second signalling mechanism, which is the diffusion cell signalling mechnisms described in Section 5.2.

The success rates of the experiments are presented in Figure 8.2. All of the experiments suggest that a contact signalling mechanism is essential for effective multicellular development. The developmental system had a poor performance for all the patterns when it lacked contact signalling (labelled as *No Tunnelling*); for some patterns the success rate was 0/50. This strongly suggests that a simple diffusion mechanism on its own is unable to organise a multicellular system effectively.



(a) Asymmetric Borders (b) Simple 2 Colour Mosaic (c) 2 Colour Mosaic

(d) 4 Colour Mosaic (e) French Flag

Figure 8.2: *Bar charts displaying the number of successful runs out of fifty runs for five of the six experimental patterns tried by developmental models using different contact signalling mechanisms. For the 8 patches pattern none of the models returned a successful run. Grey Bars show the total number of successful runs and black bars show the successful runs that are also stable. The aliases used in the x-axis are explained in Table 8.1.*

Table 8.1: *The direct contact signalling cases used in the experiments.*

Reference in Plots	Mechanism
A	Controlled Tunnels
B	Simple Tunnels
C	No Tunneling (i.e. only diffusion)

The newly introduced direct contact signalling model (marked as *controlledTunnels*) improved the success rate as well as the number of stable solutions in all patterns (except 8 patches pattern). In most cases the *simple tunnels* model is unable to provide an effective signalling mechanism. This suggests that even though the *controlled tunnels* model is a more complex signalling model than the *simple tunnels*, the developmental system greatly benefits from the larger control over the shared chemicals between individual cells.

Since none of the versions of the ADS were able to find a perfect match for the 8 patches pattern, a box and whisker plot (referred to as box plot) for the fitness achieved over the 50 runs for each experiment is displayed in Figure 8.3. Box and whisker plots are explained with an example in Appendix C.

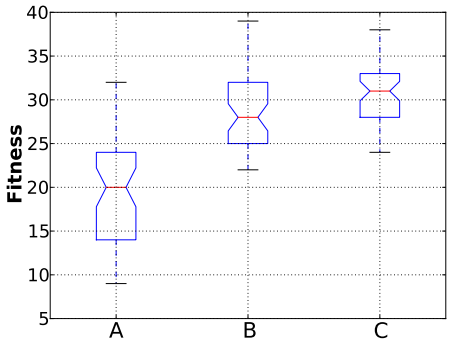


Figure 8.3: *A fitness box and whisker plot of the different contact signalling models for 8 patches pattern. The box plot shows the best (minimum) fitness achieved as well as the fitness distribution via quartiles.*

The best fitness is achieved by the *controlled tunnels* contact signalling mechanism was 9; meaning there were 9 mismatched pixels from the target image. The *controlled tunnels* also achieves the best fitness distribution among the three signalling models, *simple tunnels* having the second best distribution. The box plots indicate that *controlled tunnels* mechanism provides a significantly different data sample from the other two mechanisms, which has the lowest median value at 20. The *simple tunnels* and *controlled tunnels* were analysed for *statistical and scientific significance* with rank-sum (p-value) and Vargha-Delaney A statistic tests as well [Vargha and Delaney, 2000].

White and Poulding provide a consice definition for scientific and statistical significance [White and Poulding, 2009], where they define these two terms to be:

1. **Statistical Significance:** "A statistical analysis of the difference in observed algorithm performance must provide evidence that the observed difference is unlikely to be a chance result."

2. **Scientific Significance:** "The difference in observed algorithm performance–the effect size–must be sufficiently large in comparison to the stochastic noise (arising from the choice of random seeds) to be scientifically important."

A 5% significance level is used for all the rank-sum tests presented in this section, hence a p-value of $< 5\%$ indicates a significant difference in the results of the two ADS versions being compared. To analyse scientific significance, the Vargha-Delaney A statistic is calculated, which is a measure of effect size compared to stochastic noise [Vargha and

Delaney, 2000]. This statistic is independent of the sample size and has a range of 0-1: a value of 0.5 indicates identical performance between the two samples. Values smaller or larger than 0.5 suggests increasingly large effect sizes; a value larger than 0.5 indicating a better performance for the first of the two samples. An A statistic value greater than 0.64, or less than 0.36, indicates a medium or large effect size [Vargha and Delaney, 2000]. Any comparison demonstrating medium or large effect sizes is considered to be scientifically significant.

Vargha-Delaney A statistic and rank-sum tests are non-parametric statistical tests, which do not include specific assumptions about the distribution of the measured data sets, unlike parametric tests such as a t-test. When using parametric tests, small deviations from the required assumptions can invalidate the results [Leech and Onwuegbuzie, 2002]. Therefore it is essential to use non-parametric statistical tests in the experiments presented here, since the data from evolutionary experiments generally feature skewed distributions.

The comparison of *simple tunnels* with *controlled tunnels* yield a p-value of $< 10^{-6}$ and an A statistic of 0.104, which suggests a statistically and scientifically significant difference between the performance provided by the two methods, *controlled tunnels* being significantly better.

Observing the data provided by figures 8.2 and 8.3, we can conclude that uncontrolled diffusion is not a sufficient signalling mechanism on its own, and contact signalling provides a complementatry signalling method for multicellular organisation. However a careful design of contact signalling is required in order to fully benefit from its use in multicellular organisation. Since the *controlled tunnels* version of the contact cell signalling mechanism used in the experiments here proved to be the most effective, the rest of the experiments presented in this section use an ADS with *controlled tunnels* contact signalling mechanism.

8.3 Diffusion

Table 5.1 lists various developmental models. More than half of these models implement some form of diffusion mechanism as a distant cell signalling mechanism. The only

exception of a distant cell signalling mechanism that does not involve diffusion is the routed signalling implemented by Bentley and Kumar [Bentley and Kumar, 1999]. In this mechanism, two cells form direct connections with each other regardless of the distance between them. Not limiting the direct connections to the nearest neighbours, a symmetry breaking behaviour via a distant signalling mechanism is achieved. Such a connection scheme may be too complex for ADSs with a large number of cells. Since diffusion is a more popular and biologically inspired form of short-medium distance signalling, a simple diffusion mechanism is also modelled in the developmental system used here (the same mechanism used in [Miller, 2003]). The mechanism involves constant diffusion of chemicals from each cell to their four cardinal neighbours.

The diffusion mechanisms implemented in the developmental systems displayed in Table 5.1 vary in their implementation, possibly providing different effects on the multicellular developmental system in each case. Zhan et al. [Zhan et al., 2008] extends the model used by Miller [Miller, 2003] by the addition of a cell membrane and a chemical pathway. The cell membrane prevents the diffusion of chemicals below a certain threshold level, and the pathway provides a conversion table to convert the type of the chemicals to be diffused out to another chemical type. The cell membrane in [Zhan et al., 2008] could be a good improvement since it provides a more controlled diffusion mechanism, and as discussed earlier; unrestricted diffusion in biological embryos is undesirable [Fagotto and Gumbiner, 1996], which may also apply to ADSs. Steiner et al. [Steiner et al., 2007] use a diffusion layer for the diffusing chemicals, the chemicals are diffused into the layer constantly but it is accompanied by cell adhesion and sorting which indirectly controls the direction of diffusion by moving cells within the organism.

Whether a more complex and bio-inspired diffusion mechanism is required for a more evolvable developmental system or a simple diffusion mechanism is sufficient is not clear without empirical data. Biological analogy could be used to suggest that uncontrolled diffusion is undesirable, but biological data can not always be relied on for models designed for EC. On the other hand most of the applications could be tackled by cell contact signalling [Flann et al., 2005], hence a diffusion mechanism may increase the ADS complexity and might not be beneficial at all.

Six different diffusion models are investigated in this section for their effects on the developmental system's ability to achieve the patterns shown in Figure 8.1.

- **Unrestricted constant diffusion**: The diffusion mechanism that was included in the initial design. The diffusion process is carried out for all chemicals available in every cell each developmental step; half of all the available chemicals diffuse out of a cell equally to the four nearest neighbours, i.e. each neighbour obtains $\frac{1}{8}$ of the cell's chemicals each developmental step.

- **Generic diffusion proteins**: In this case the diffusion process only takes place when a diffusion protein is produced by a gene. When the dedicated diffusion protein is produced by a gene, the diffusion protein uses the information provided by the postconditional part of the gene to determine the amount of chemical concentration (same for each chemical type) to diffuse out. Therefore the diffusion of chemicals is controlled by gene regulation within a cell, but same amount of all the chemicals diffuse out to neighbouring cells every developmental step.

- **Chemical Specific diffusion proteins** (Diffusion Protein (DP)): In this case when the dedicated diffusion protein is produced by a gene, the diffusion protein uses the further information provided by the postconditional part of the gene to determine **how much of which** chemical to diffuse out. Therefore the diffusion of each chemical is controlled individually by the gene regulation within a cell, so that not all the chemicals might diffuse out to the neighbouring cells every developmental step.

- **Diffusion layer**: An extra layer for the diffusing chemicals is included to simulate a more realistic diffusion of chemicals throughout the organism. Hence, a diffusing chemical does not travel directly between two neighbouring cells. Also in this case, a chemical only diffuses from a direction of high concentration to low concentration and not vice versa. The amount of chemical that diffuses is described by Equation 8.1: the flow of chemical a from position p to position q is calculated as their difference in their concentrations Y. DC is the diffusion constant.

- **Diffusers**: These are chemical sources placed around the organism diffusing out chemicals at a constant rate. These diffusers are used as well as an unrestricted constant diffusion of chemicals from the cells themselves. The idea of diffusers was used in Roggen's morphogenesis model [Roggen, 2005].

- **No diffusion**: Diffusion mechanism is completely removed from the system, only leaving cell to cell contact signalling.

$$\Delta Y_{pat} = \begin{cases} (Y_{pat} - Y_{qat})/DC & Y_{pat} > Y_{qat} \\ 0 & Y_{pat} \leq Y_{qat} \end{cases}. \tag{8.1}$$

Figure 8.4 contains bar charts displaying the number of runs the developmental system was successfully evolved to match the patterns shown in Figure 8.1 for different diffusion mechanisms. Unlike the contact signalling experiments there is no single diffusion mechanism that outperforms the other mechanisms in achieving all the experimental patterns. However just like contact signalling, diffusion had a big impact on the ADS performance. Most of the problems suffered from uncontrolled presence (except the bordered French flag pattern) or total absence (except asymmetric borders pattern) of diffusion.

Unrestricted constant diffusion was only favoured by the popular French flag pattern, and had negative effects on the formation of all other patterns. Although lack of diffusion reduced the performance of the developmental system in general, its effects were not as big as having constant diffusion. Once more, introducing a more controlled method of communication seemed to improve the ADS performance in general. Figure 8.4 shows that 8 patches and 4 colour mosaic patterns benefited from the presence of an extra diffusion layer in the system whereas 2 colour mosaic, simple 2 colour mosaic, and 8 patches patterns benefited from the use of diffusion proteins. Use of diffusion proteins that control the diffusion of single chemicals rather than the diffusion of all chemicals at once slightly improved the performance in 4 colour mosaic, 8 patches, and asymmetric borders patterns due to the higher precision control in the ADS; whereas the performance deteriorated in other cases due to the increase in the required number of genes in the control of chemical diffusion. The use of diffusers did not have any notable benefits, in fact in most of the cases the performance of the ADS deteriorated when diffusers were included in the system.

For some of the experiments with a diffusion layer, the diffusion constant was evolved instead of being pre-set. For most of the experiments this provided an improvement in evolutionary performance, although not a significant improvement.

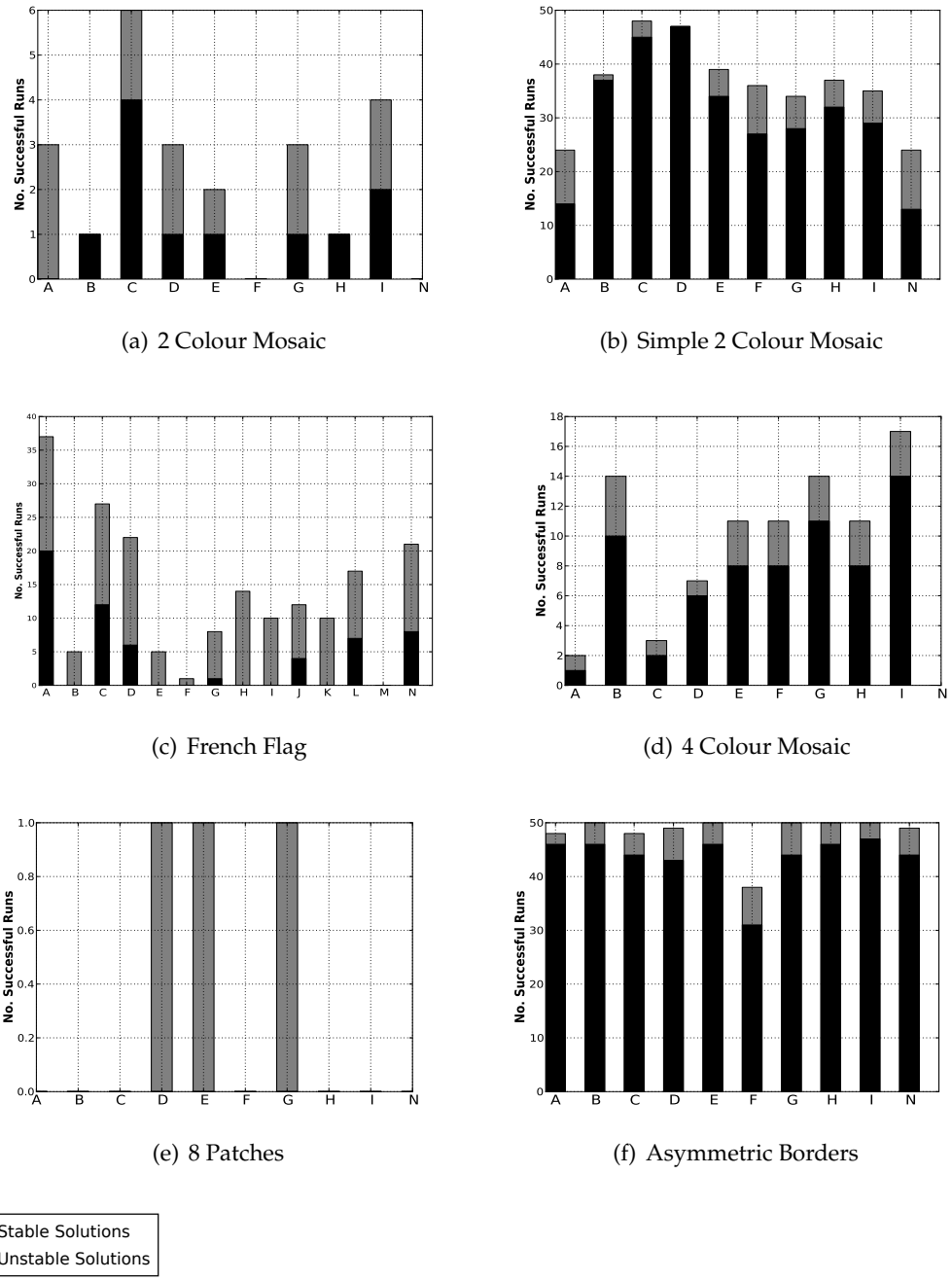


Figure 8.4: *Bar charts displaying the number of successful runs out of fifty runs for different diffusion mechanisms in achieving all six patterns. The aliases used in the x-axis are explained in Table 8.2.*

Deciding whether the use of a diffusion layer, diffusion protein, or both for controlling the diffusion process to achieve the overall best diffusion mechanism for the ADS is not very clear from the bar charts shown in Figure 8.4. To provide more information on the performance, Figure 8.5 provides box plots on the fitness distribution on all the runs of each experiment. Since almost all the experiments on asymmetric borders pattern

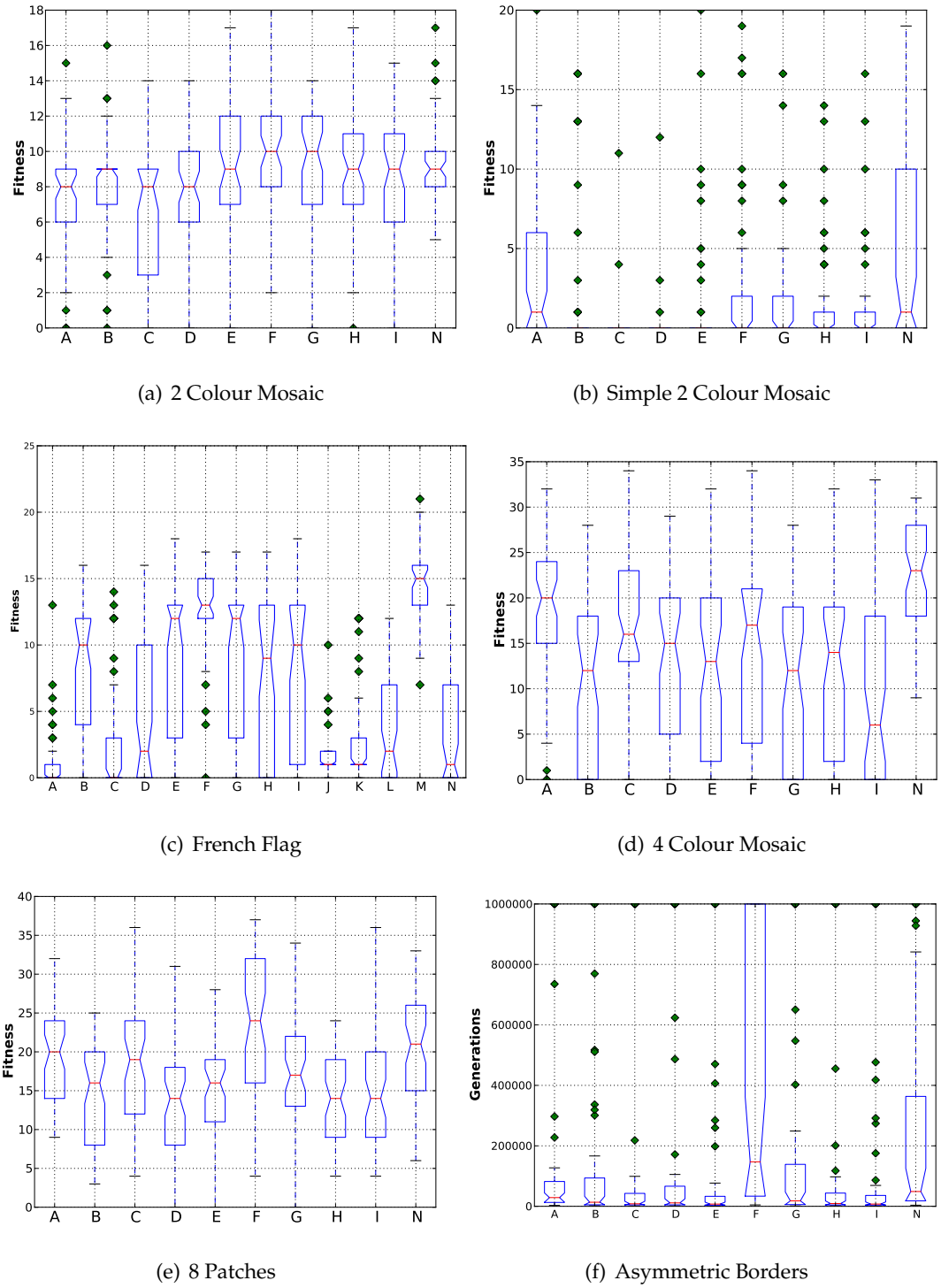
Table 8.2: *The diffusion mechanisms used in the experiments.*

Reference in Plots	Mechanism
A	Constant diffusion
B	No diffusion
C	Generic diffusion proteins
D	Chemical specific diffusion proteins (DP)
E	Diffusion layer with evolved Diffusion Constant (DC)
F	Diffusion layer with fixed DC
G	Diffusion layer that uses DP
H	Diffusion layer with evolved DC that uses DP
I	Diffusion layer with evolved DC that uses generic DP
J	Diffusers in all cells - 127
K	Diffusers in all cells with no don't care genes - 127
L	Single diffuser - 127
M	Single diffuser with no don't care genes - 127
N	Single diffuser - 255

had high success rates (i.e. fitness 0), the box plots on the distribution of the number of generations to achieve stable solutions is shown instead of the fitness distribution.

Looking closely at the results of all the options for diffusion mechanisms in Figure 8.5, using a diffusion protein as the diffusion control mechanism seems to give the best results. But in every pattern except the French flag, the confidence interval of the best diffusion protein runs overlap with the confidence interval of the best diffusion layer runs. Hence it is worth taking a closer look at the best cases in a side by side significance comparison for a better conclusion. Table 8.3 compares chemical specific diffusion protein with six different versions of the diffusion mechanisms using Vargha Delaney A statistic and Mann-Whitney-Wilcoxon p-value. As described earlier a p-value of $< 5\%$ is considered to represent a statistically significant difference between the two data sets being compared, as well as an A value above 0.64 or below 0.36 representing a scientifically significant difference.

Addition of a diffusion layer or a diffusion protein in the ADS improves the overall ADS performance compared to constant or no diffusion. However, the use of diffusion protein on its own seems to be sufficient, and the addition of a diffusion layer –a more complex mechanism– does not seem to make a significant difference.



(a) 2 Colour Mosaic

(b) Simple 2 Colour Mosaic

(c) French Flag

(d) 4 Colour Mosaic

(e) 8 Patches

(f) Asymmetric Borders

Figure 8.5: *Box plots of fitness distribution in fifty runs for the different diffusion mechanisms in achieving all the patterns except the asymmetric borders pattern; box plots of the number of evolutionary generations in finding stable solutions are shown for the asymmetric borders pattern. The aliases used in the x-axis are explained in Table 8.2.*

Hence the optimal diffusion mechanism is a diffusion protein in which the GRN controls the chemical diffusion. Chemical specific diffusion protein introduces extra complexity

Table 8.3: *Statistical comparison of diffusion mechanism using chemical specific diffusion protein (labelled as "diffusion protein") with six other diffusion mechanisms. Scientifically better cases for the chemical specific diffusion protein are marked with tick marks, and worse cases are marked with crosses; explained further below the table. For the statistical significance the actual p-values are shown, and the statistically significant difference in the effects of the two mechanisms are recorded in bold white text. For all examples except asymmetric borders, the final fitness distribution for each experiment is used. For the asymmetric borders, number of evolutionary generations required to achieve stable solutions is used to calculate the A statistic and the p-value.*

Diffusion Protein vs	Statistic	2 Colour Mosaic	4 Colour Mosaic	Simple 2 Colour Mosaic	8 Patches	Bordered French Flag	Asym. Borders
Generic Diffusion Protein	A Statistic	XX	✓	=	✓✓✓	X	X
	p-value	**0.024**	0.062	0.062	**0.001**	**0.048**	0.277
Diffusion Layer-EvoDC	A Statistic	✓✓	X	✓	✓	✓✓✓	X
	p-value	**0.037**	0.326	**0.035**	0.105	**5.1×10^{-5}**	0.101
Diffusion Layer-DP-EvoDC	A Statistic	✓	X	✓✓	✓	✓✓✓	X
	p-value	0.243	0.343	**0.014**	0.264	**0.006**	0.239
Constant Diffusion	A Statistic	X	✓✓✓	✓✓✓	✓✓✓	XXX	✓✓
	p-value	0.271	**0.001**	**3.7×10^{-6}**	**6.3×10^{-5}**	**0.0003**	**0.015**
No Diffusion	A Statistic	✓	X	✓✓	✓	✓✓✓	✓
	p-value	0.488	0.100	**0.0212**	0.174	**0.0003**	0.483
Diffusion Layer-Generic DP-EvoDC	A Statistic	✓	XX	✓✓	✓	✓✓✓	X
	p-value	0.191	**0.015**	**0.006**	0.229	**0.001**	0.177

A Statistic	Legend	Meaning
0.5	=	Same Performance
0.4 - 0.5	X	Slightly Worse
0.36 - 0.5	XX	Worse
<0.36	XXX	Significantly Worse

A Statistic	Legend	Meaning
0.5-0.6	✓	Slightly Better
0.6-0.64	✓✓	Better
>0.64	✓✓✓	Significantly Better

to the system, reducing the ADS performance when compared to the generic diffusion protein in some of the experimental patterns, but in the experiments provided in this section the effects of this complexity is mostly negligible. When a larger number of chemicals are used, these effects may become more significant. It is noteworthy that in the most complex of all the patterns tried in this section, the 8 patch pattern, the use of chemical specific diffusion protein made a significant improvement in the fitness distribution. The performance difference between the two different implementations of the diffusion protein are almost marginal, but there was one case (the 8 patch pattern) where the chemical specific diffusion protein provided better results with a scientific significance. The performance differences between the two implementations of the diffusion protein in all other experimental problems were below the A statistic significance threshold.

8.4 Mapping The Phenotype

The phenotypic function of the cell (referred to as the cell phenotype from now on), which is part of the target function for the ADS (e.g. a pixel value in the case of patterns) can be built in a number of ways. The common method of obtaining the cell phenotype is to use a mechanism within the developmental processes of the cell to provide a structural output as an emergent result of the developmental processes. The most popular way of building the cell phenotype in systems that use GRNs is to use specialised proteins that change the cell phenotype when they are being expressed during the developmental process. Other developmental systems that do not use GRNs tend to use the evolved developmental cell program both for regulatory purposes and building the cell phenotype. Therefore the cell phenotype is determined by one or more of the outputs of the cell program. The behaviour of the cell phenotype in these cases is directly affected by the developmental processes. There are few other methods of building the cell phenotype, and most of these methods are an emergent result of the developmental progress of the organism. This is the case for all except three of the developmental models listed in Table 5.1; Gordon [Gordon, 2005], Kitano [Kitano, 1995] and Zhan et al. [Zhan et al., 2008] do not use integrated mechanisms within development that build the cell phenotype as an emergent result of the respective developmental system. Instead they use the chemical concentrations at the end of a developmental step or phase to map the cell phenotype (in the application domain).

In this section some experiments were carried out to investigate whether there is an advantage of using an extra mechanism within the GRN based developmental system to build the cell phenotype, or whether it is as good to make use of the chemical concentrations within each cell at the end of a developmental step to build a cell phenotype for every cell. Using the chemical concentrations at the end of a developmental step to create a cell phenotype is not biologically plausible, however it simplifies the cell structuring process. Hence once more this imposes a question on the evolvability of bio-inspired versus simplistic mechanisms.

The experiments in this section only use the French flag, simple 2 colour mosaic, 2 colour mosaic and the 8 patches patterns as test patterns. There are three versions of cell structuring methods tested.

Table 8.4: *Statistical comparison of developmental system using the concentration of a single protein for cell structuring to the developmental system using structuring protein.*

Statistical Comparison	French Flag	Simple 2 Colour M.	2 Colour M.	8 Patches
Vargha Delaney A Statistic	0.160	0.012	0.035	0.527
p-value	$< 10^{-6}$	$< 10^{-6}$	$< 10^{-6}$	0.387

- **The structuring protein**: This uses a dedicated protein to change the cell phenotype when it is produced during the gene processing. This is part of the initial design of the developmental system described in Section 5.2.2.

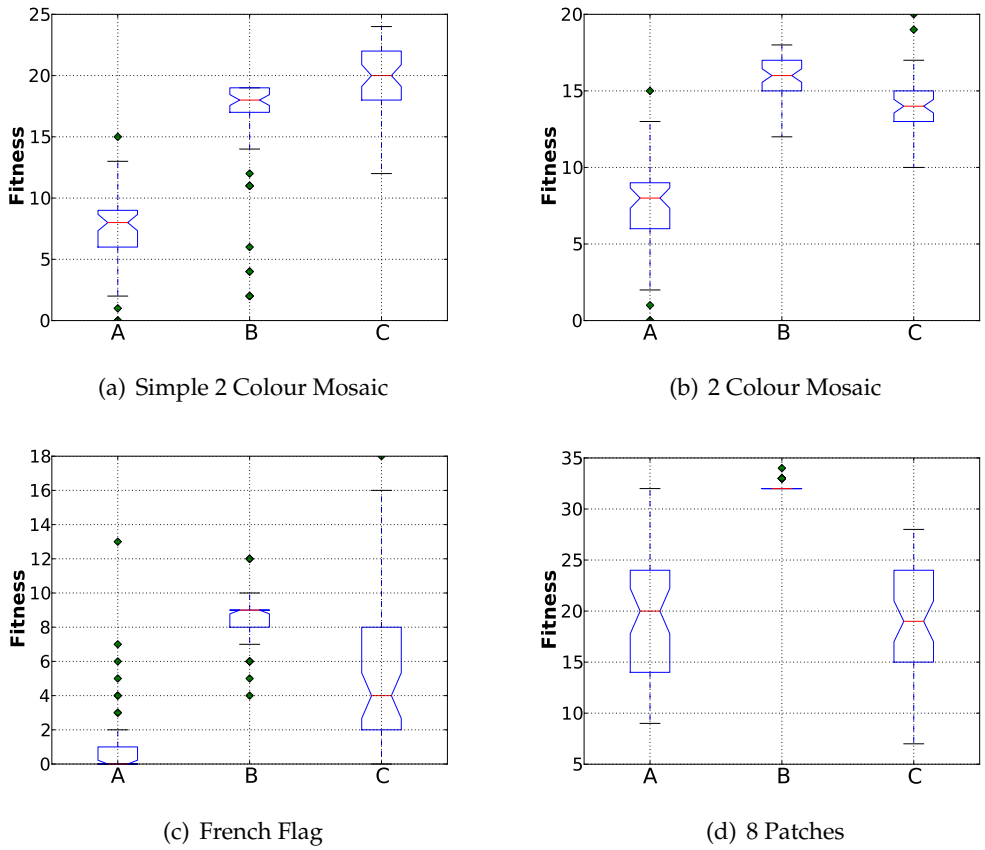
- **Concentration of a dedicated protein**: In this case, the structuring protein has no action when it is produced during the gene processing stages. At the end of a developmental step, the concentration of one of the proteins is used as an output. In the examples presented here the concentration value is divided by the *maximum* chemical concentration level, multiplied by the number of possible cell phenotypes and rounded down to an integer value in order to obtain the cell phenotype.

- **Concentration of all chemicals**: In this case all the chemicals in the system are used for determining the cell phenotype. The chemical with highest concentration is used to set the cell phenotype, and each chemical corresponds to a specific cell phenotype.

For all the experiments the ADS used contact signalling with "controlled tunnels" as discussed in Section 8.2, and a constant diffusion mechanism as described in sections 5.1.2 and 8.3.

In all test cases the runs where multiple and single protein concentrations were used to define cell phenotypes showed poor success rates. Therefore, rather than using success rate bar charts, the final best fitness values are used to create box plots (Figure 8.6). In almost every case using a structuring protein to build the cell phenotype resulted in better performance. Further statistical analysis of structuring protein mechanism versus the other two methods of mapping the cell phenotype are shown in Tables 8.4 and 8.6.

In all cases except one using the structuring protein proved to be significantly better. The exceptional case is the 8 Patches problem where the use of single protein concentration gives slightly better results. However, in this case the use of single protein concentration



(a) Simple 2 Colour Mosaic

(b) 2 Colour Mosaic

(c) French Flag

(d) 8 Patches

Figure 8.6: *Box and whisker plots on the fitness distribution of the different methods of constructing cell phenotype. The aliases used in the x-axis are explained in Table 8.5.*

does not provide a scientifically significant improvement (A statistic of 0.527 against structuring protein), and the distribution difference from the case with structuring protein is not statistically significant (p-value of 0.387).

Using an emergent mechanism employing structuring proteins to build the cell phenotype proved to be more successful than using the final protein concentrations at the end of a developmental phase. From the observations made, using protein concentrations to build the cell phenotype made the developmental system's ability to create special-

Table 8.5: *The labels for each structuring mechanism used in the experiments.*

Reference in Plots	Mechanism
A	Structuring proteins
B	Concentration of all proteins
C	Concentration of a dedicated protein

Table 8.6: *Statistical test results for the developmental system that uses the concentration of multiple proteins to map the cell phenotype compared with the developmental system with structuring protein.*

Statistical Comparison	French Flag	Simple 2 Colour M.	2 Colour M.	8 Patches
Vargha Delaney A Statistic	0.036	0.121	0.018	0.018
p-value	$< 10^{-6}$	$< 10^{-6}$	$< 10^{-6}$	$< 10^{-6}$

ized cells harder. It was observed that the cell specializations arise from the different developmental path taken by each cell rather than their final developmental state. For example every cell in an ADS might always end up having similar concentration levels of chemicals at the end of each developmental step, but the changes to the chemical concentration levels during the processing of the GRN would be different for each cell. Thus using information relating to changes in the chemical levels during the processing of the GRN rather than the chemical levels at the end of a developmental step provides a more precise way of differentiating a cell.

8.5 Parameters for Transcription Factors

In a GRN system the interactions between chemicals and genes are the direct determinants of the computational pathway the GRN system undergoes, and the final states it stabilises to. As seen earlier (in Figure 4.2), the presence of chemicals determine whether a gene will be activated, and the activation of a gene increases the amount of a protein (which is a chemical) available in the system, which also regulates the activation of genes. The amount of chemical needed to activate a gene and the amount of protein produced after the activation of a gene affects the number of genes that will be active at one time as well as the number of genes that will never be active within the GRN. The amount of chemical needed to activate a gene is referred to as "Gene Activation Threshold"; the minimum concentration of a chemical required to bind a gene. The amount of protein produced after the activation of a gene is referred to as "Protein Production Rate". When a chemical binds to a gene, a certain amount of that chemical is "used up" by the gene, this is referred to as "Chemical Consumption Rate". These three values directly or indirectly control the gene activation that can be evolved during the evolution of the developmental system or they can be set to a constant value before the evolution phase.

Although each target organism might have a different optimal set of values for the gene activation threshold, protein production and chemical consumption rates, evolving these values might not be beneficial as it increases the dimensionality of the genotype.

In order to find out the most evolvable approach to determining the developmental parameters of gene activation thresholds, protein production and consumption rates for a developmental system, a series of experiments involving all six patterns shown in Figure 8.1 were performed.

8.5.1 Protein Production and Chemical Consumption Rates

Three different experiments involving setting the protein production and chemical consumption rates are investigated.

- **Preset values**: A protein production rate that is the same for all genes in the GRN, and a chemical consumption rate that is also the same for all genes and chemicals are initialized at the start of an experiment. Various combinations of different values for both are used for all the experiments.
- **Evolved values**: A protein production rate that is the same for all genes in the GRN is evolved, as well as a chemical consumption rate that is also the same for all genes and chemicals. Therefore only two extra values need to be evolved for each developmental system.
- **Evolved values per gene**: A protein production rate and a chemical consumption rate are evolved for each gene. This provides a finer tuning of the protein production and chemical consumption rates but increases the evolutionary search space further by the increasing number of genes.

The experiments provided in this section use "controlled tunnels" as the contact signalling mechanism, and a constant diffusion mechanism. Table 8.7 provides the details for the labels used in the plots for the protein production and chemical consumption rates experiments. These rates are detailed in the table as; "Production:X" for protein production rate of X, "Consumption: Y" for chemical consumption rate of Y, and "Plasmodesma prod: Z" for the production rate of plasmodesma protein: for experiments that

used a different production rate for plasmosdema protein (the contact cell signalling and growth protein). One of the experimental cases in the plots presented in this section is detailed as "Original setting"; this represents a protein production rate of $\frac{255}{96}$ for the plasmodesma protein and $\frac{255}{24}$ for the rest of the proteins, and a chemical consumption rate of $\frac{255}{48}$ for all chemicals (255 is the maximum chemical concentration). The "Original setting" is used for the experiments in the other sections of this chapter. The "Original setting" was obtained during the initial design as a result of manual tuning of the system.

Figure 8.7 shows the results of the experiments. Various combinations of protein production rate at the activation of a gene and chemical consumption rate at the binding of a chemical to a gene are tested. The aim is to see whether there exist a single best combination for these developmental parameters or whether they need to be tuned for each application. If the latter, then evolving the protein production and chemical consumption rates will create a more adaptive system by allowing evolution to tune the ADS. If all the applications can be easily developed with a single set of these parameters then using a fixed set of optimised values would be more beneficial as it would reduce the load on the evolutionary algorithm.

The protein production and chemical consumption rates are evolved using two different approaches. The first approach evolves a protein production and a chemical consumption rate for the whole organism. So, every cell and every gene use the same pair of values for protein production and chemical consumption rates. The second approach evolves the protein production and chemical consumption rates for each gene. Hence, the genome length is extended in the second approach to accommodate these rates for each gene. Evolving these rates per gene will provide evolution the ability to tune these parameters at a finer level, but if the effect of these parameters is not large then the evolutionary performance will deteriorate due to the increase in evolutionary search space.

The results in Figure 8.7 show that different patterns have different optimal protein production and chemical consumption rate values. A compromise setting between the patch and border patterns would be a protein production rate at $\frac{255}{48}$ and chemical consumption rate at $\frac{255}{96}$, which provides not the best but close to best performance for all three patterns. But for the mosaic patterns, this combination of protein production and chemical consumption rates is far from ideal.

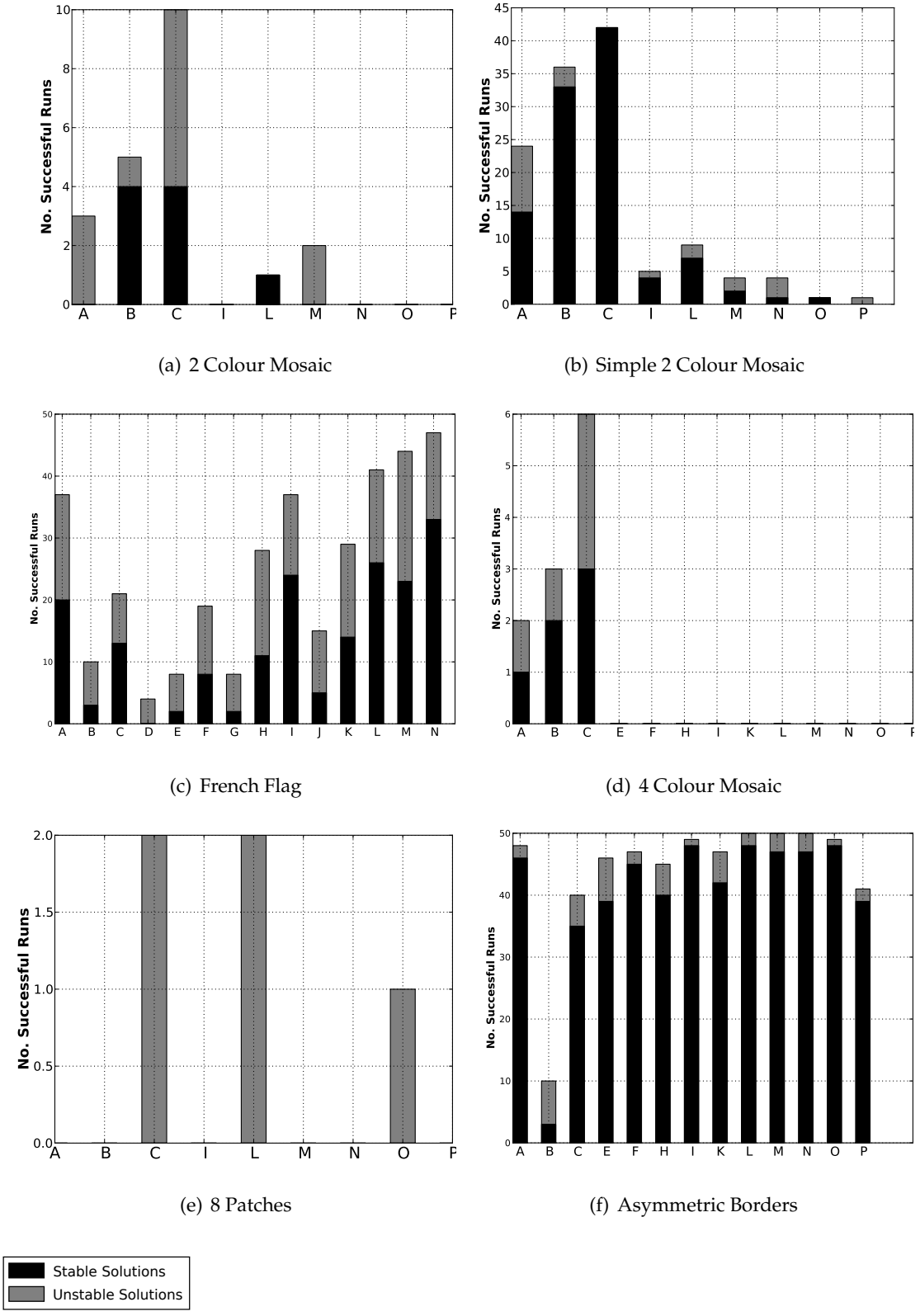


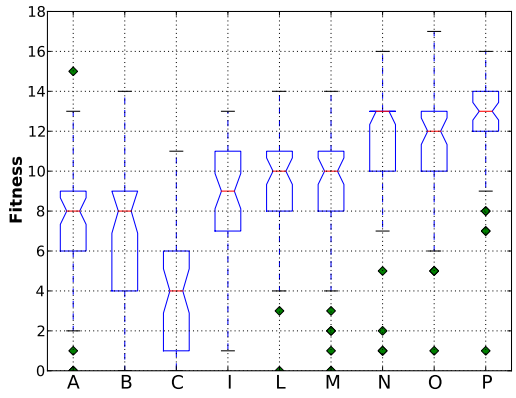
Figure 8.7: *Success rates of experiments with different protein production and chemical consumption rates are displayed. The aliases used in the x-axis are explained in Table 8.7.*

Table 8.7: *The labels for each protein production and chemical consumption values used in the experiments.*

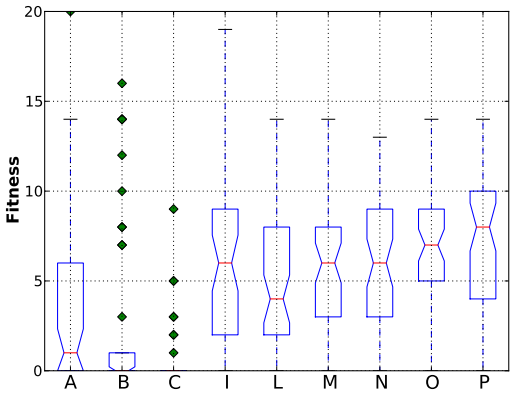
Reference	Mechanism
A	Original setting - Prod:$\frac{255}{96}$,Cons: $\frac{255}{48}$, Plas prod: $\frac{255}{24}$
B	A single evolved value
C	A value evolved per gene
D	Production: $\frac{255}{12}$, Consumption: $\frac{255}{6}$
E	Production: $\frac{255}{24}$, Consumption: $\frac{255}{12}$
F	Production: $\frac{255}{48}$, Consumption: $\frac{255}{24}$
G	Production: $\frac{255}{12}$, Consumption: $\frac{255}{12}$
H	Production: $\frac{255}{24}$, Consumption: $\frac{255}{24}$
I	Production: $\frac{255}{48}$, Consumption: $\frac{255}{48}$
J	Production: $\frac{255}{12}$, Consumption: $\frac{255}{24}$
K	Production: $\frac{255}{24}$, Consumption: $\frac{255}{48}$
L	Production: $\frac{255}{48}$, Consumption: $\frac{255}{96}$
M	Production: $\frac{255}{48}$, Consumption: $\frac{255}{96}$, Plasmodesma prod: $\frac{255}{192}$
N	Production: $\frac{255}{64}$, Consumption: $\frac{255}{128}$
O	Production: $\frac{255}{64}$, Consumption: $\frac{255}{96}$
P	Production: $\frac{255}{64}$, Consumption: $\frac{255}{128}$, Plasmodesma prod: $\frac{255}{255}$

Evolving these parameters is the best compromise amongst all six patterns, and evolving a set of these rates per gene rather than a single set for the whole organism seems to make a significant difference. For a better understanding of the performance difference obtained via the evolution of these rates, fitness box plots of the results are provided in Figure 8.8, with the exception of asymmetric borders pattern. For the asymmetric borders pattern the number evolutionary generations to achieve stable solutions is used rather than final fitness values.

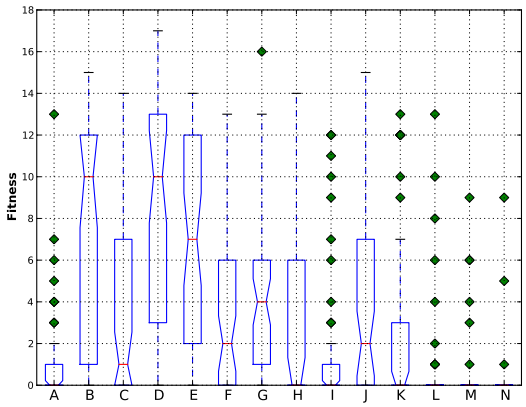
Although evolving the protein production and chemical consumption parameters for the ADS during the evolution phase made a large performance improvement for the mosaic patterns, this is not true for the patch and border patterns, see Figure 8.8. However, even though the patch and border patterns suffer from the use of evolved protein production and chemical consumption parameters, overall, it is worth evolving these parameters for each gene. By evolving the protein production and chemical consumption parameters for each gene:



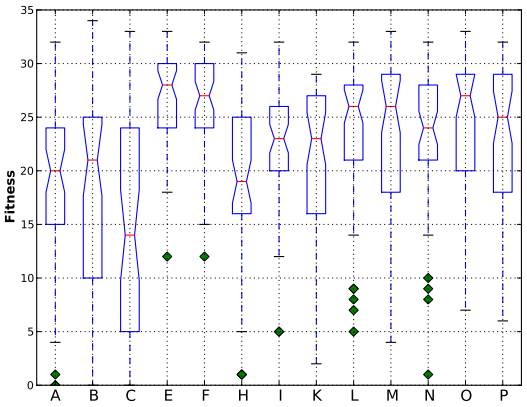
(a) 2 Colour Mosaic



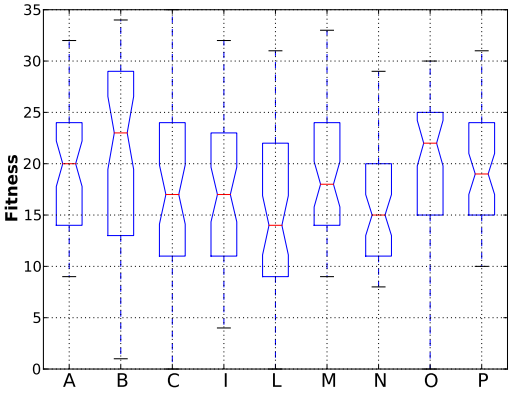
(b) Simple 2 Colour Mosaic



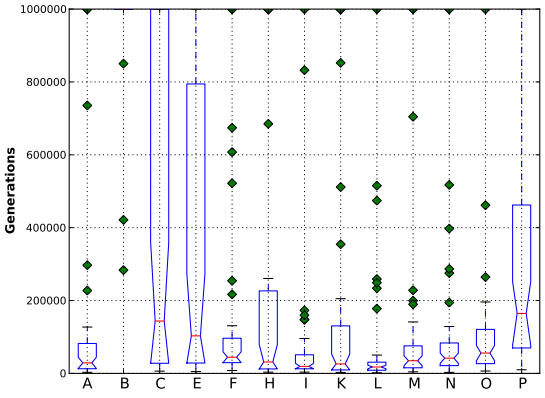
(c) French Flag



(d) 4 Colour Mosaic



(e) 8 Patches



(f) Asymmetric Borders

Figure 8.8: *Box plot of fitness values (the number of evolutionary generations required to achieve stable solutions for the asymmetric borders pattern) reached at the end of each run out of fifty runs of each experiment with different protein production and chemical consumption rates are displayed. The aliases used in the x-axis are explained in Table 8.7.*

1. The performance loss in the worst case problems (e.g. evolving asymmetric borders pattern) is balanced by the time it would take to determine the optimal combination of these parameters before running experiments for each problem.

2. There is more control for evolution to fine tune the ADS.

3. Due the the drastic effects these rates might have on the evolvability of the ADS, evolving these rates per gene resulted in better results when compared with the evolution of these rates as global values for all genes.

8.5.2 Gene Binding Threshold

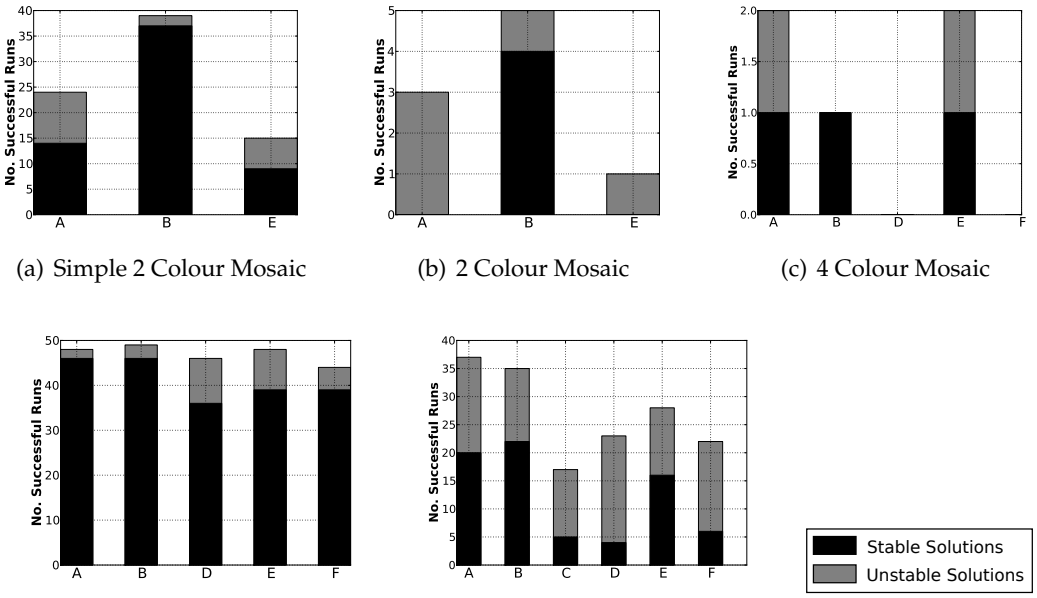
Three different approaches of setting the chemical concentration threshold for gene binding were investigated.

- **Pre-set thresholds**: The minimum concentration required for a chemical to bind a gene is pre-set to a constant value before evolving the developmental system. All the genes and chemicals use the same threshold value.

- **Evolved thresholds per gene**: The minimum concentration required for a chemical to bind a gene is evolved for each gene during the evolution phase.

- **Soft Thresholds**: The idea of relaxed threshold limits are implemented in a simplistic way. The "Original setting" is used for this case but when a protein concentration drops or increases to the chemical threshold level, the switching from present to absent or absent to present is not done immediately but a further X amount of chemical concentration increase (if switching to present state) or decrease (if switching to absent state) is allowed. Concentration X is determined via a random number generator that returns a value between 0 and 15.

The experiments provided in this section use "controlled tunnels" as the contact signalling mechanism, and a constant diffusion mechanism. Table 8.8 provides the details for the labels used in the plots for the chemical to gene binding thresholds. These rates are detailed in the table as; "Gene inhibition thresh: X" X being the minimum concentration required of a chemical to bind an inhibitory site of a gene, and "Activation thresh: Y"



(a) Simple 2 Colour Mosaic (b) 2 Colour Mosaic (c) 4 Colour Mosaic

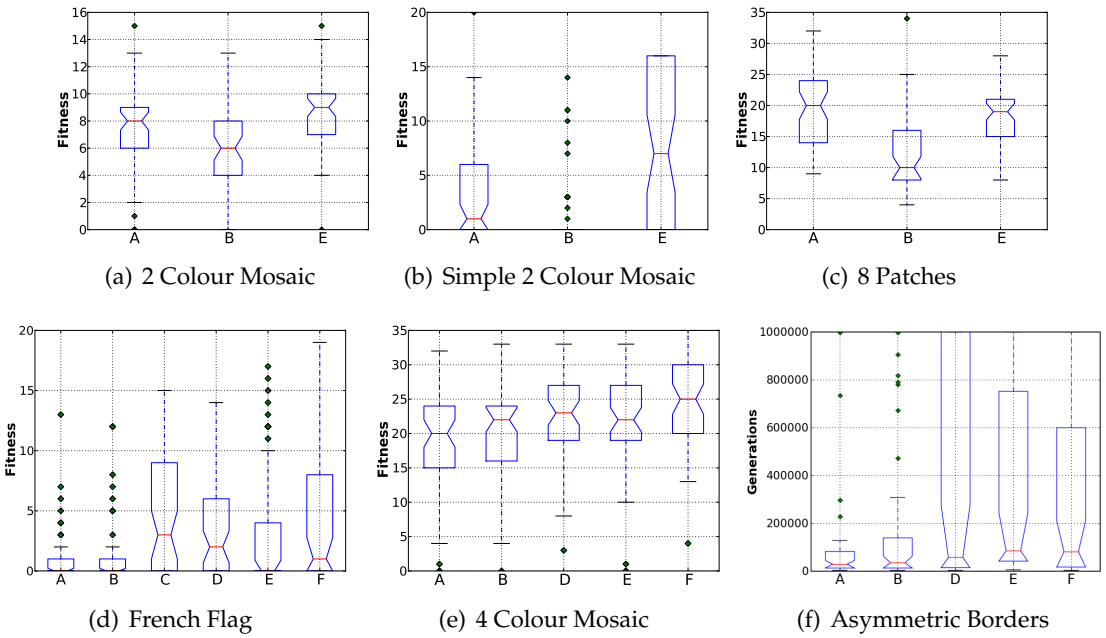
(d) Asymmetric Borders (e) French Flag

Figure 8.9: *Success rates of experiments with different chemical to gene binding thresholds. Only the most successful settings are shown for some of the test patterns e.g. 2 colour mosaic. 8 patches pattern is omitted because none of the runs in this section returned a success. The aliases used in the x-axis are explained in Table 8.8.*

Y being the minimum concentration required of a chemical to bind an excitatory site of a gene. One of the experimental cases in the plots presented in this section is labelled as "Original setting"; this represents a chemical to gene binding threshold level of 127 ($\frac{max}{2}$) for both inhibiting and enhancing chemicals. The "Original setting" is used for the experiments in the other sections of this chapter.

Table 8.8: *The labels for protein concentration threshold values used in the experiments for activating or inhibiting a gene.*

Reference in Plots	Mechanism
A	Original setting
B	Evolved thresholds
C	Gene inhibition thresh: $\frac{255}{4}$, Activation thresh: $\frac{255}{4}$
D	Gene inhibition thresh: $\frac{255}{2}$, Activation thresh: $\frac{255}{4}$
E	Gene inhibition thresh: $\frac{255}{4}$, Activation thresh: $\frac{255}{2}$
F	Soft thresholds at $\frac{255}{2}$



(a) 2 Colour Mosaic (b) Simple 2 Colour Mosaic (c) 8 Patches

(d) French Flag (e) 4 Colour Mosaic (f) Asymmetric Borders

Figure 8.10: *Fitness box plots of every pattern except the asymmetric borders pattern are displayed in this figure for different chemical to gene binding threshold experiments. Due to the high success rate of most experiments in achieving the asymmetric borders pattern, the number of evolutionary generations to achieve stable results is used for its box plots. The aliases used in the x-axis are explained in Table 8.8.*

Once more evolving a developmental parameter improved the overall evolvability of the ADS. Evolving the chemical to gene binding thresholds improved the performance in most patterns when compared to experiments with pre-set values, and in two cases (french flag and 4 colour mosaic) the results were comparable to the best case ("Original setting"), see figures 8.9 and 8.10. Although none of the experiments returned a perfect matching result for the 8 patches pattern, evolving the chemical to gene binding thresholds improved the fitness distribution (with the best fitness being 4 mismatched pixels) when compared to pre-set cases, see Figure 8.10. The "soft thresholds" implementation have only complicated the chemical to gene binding processes and had no benefits in any of the tested cases.

8.6 Miscellaneous Developmental Mechanisms

The use of extra mechanisms in an ADS may enrich the model by providing access to areas of design space that were not accessible before. Some mechanisms may allow the developmental system to reach certain points in the design space more easily by creating a less rugged search space. On the other hand extra mechanisms will increase complexity of the ADS potentially slowing it down. Mechanisms that only duplicate the already existing abilities of a system may create larger biases towards a specific area of a design space forming a more rugged search space. Whether the inclusion of a mechanism in an ADS is beneficial is often not clear. A biologically inspired developmental system is highly dynamic, and it is poorly understood. Predicting the changes in the dynamics of a dynamical system is a challenging task. Extensive empirical data on the behaviour of the ADS with and without the mechanism in question can help with the decision on whether the mechanism is worth keeping as part of the ADS.

Six developmental mechanisms that were not used in the previous experiments are investigated in this section.

- **Local proteins**: The effect of proteins that are only involved in the regulation of the GRN within the source cell is investigated. This is similar to the role of the chemicals in Haddow and Hoye's model [Haddow and Hoye, 2007]. The local proteins are not used for cell communication, hence they are not diffused or allowed to pass onto the neighbouring cells via plasmodesmata (tunnels). In [Haddow and Hoye, 2007] it was demonstrated that chemicals which are only used for local cell regulation increase the complexity of the search space without any visible advantages.
- **Messenger Molecules**: In Section 5.2.2 the role of messenger molecules was explained, but for the experiments presented thus far they were not used. The idea of messenger molecules is to provide a more adaptive system in changing environments, which does not really correspond to stable patterns. Nonetheless the effect of messenger molecules on the evolvability of the ADS for evolving stable patterns is investigated. In the pattern forming experiments presented here, the sensor proteins are used to monitor the phenotype of the neighbouring cells and

produce messenger molecules accordingly. The use of phenotypic information from the neighbouring cells to regulate the developmental system has already been used in many developmental systems before. In fact, almost half of the ADSs listed in Table 5.1 include a form of this mechanism in their systems [Bentley and Kumar, 1999; Dellaert and Beer, 1994; Federici, 2004; Haddow and Hoye, 2007; Miller, 2003; Tufte and Haddow, 2003].

- **Voter decision mechanism**: The decision mechanism on the activation of a gene in the initial design is simply a conjunctive expression. As an alternative mechanism a voter is used for deciding the activation of a gene. The voter mechanism counts the number of bound enhancers and inhibitors, and activates the gene when there are higher number of enhancing bindings or zero inhibitors. The aim of the voter decision mechanism is to provide a smoother transition between active and inactive genes rather than a sharp transition provided by a conjunctive expression.

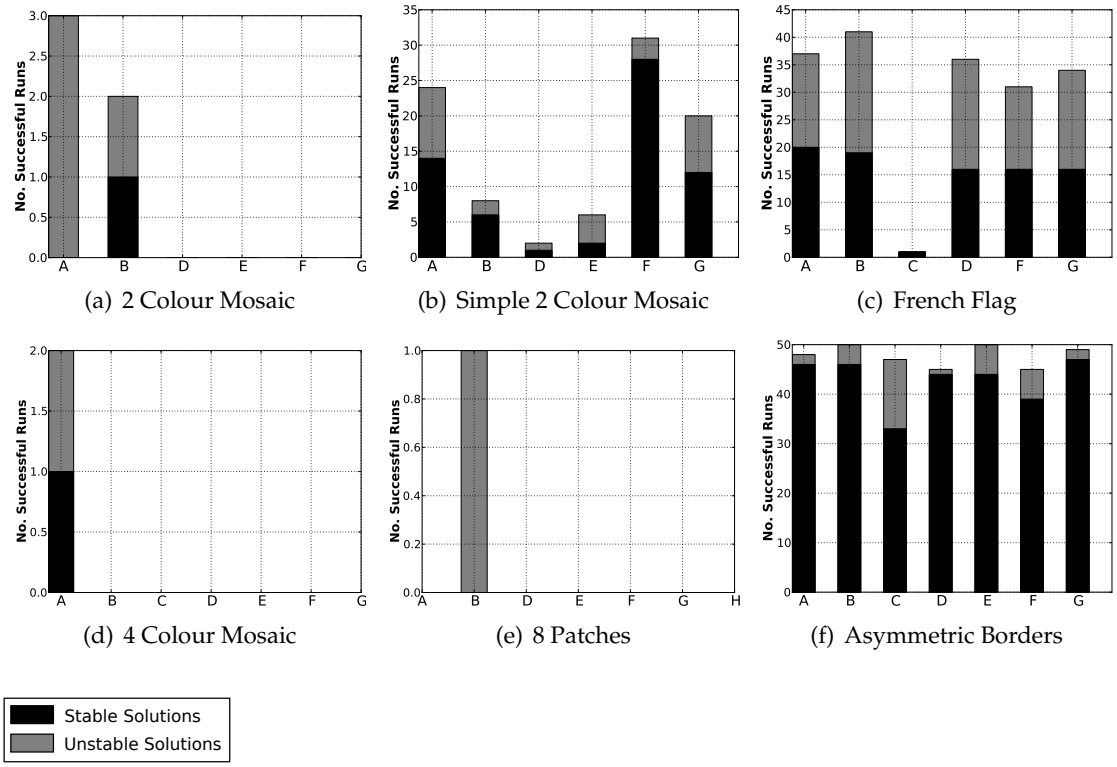
- **Protein Consuming Genes**: Although the genes consume proteins when proteins bind to a gene, protein consuming genes use up proteins as the result of their postcondition as well. The ability to further consume a protein may allow the GRN to regulate the protein chemical levels more precisely, and create redundant behaviour for further gene degeneracy in the system.

- **Unproductive Genes**: Genes that do nothing when activated are named unproductive genes. Similar to protein consuming genes these genes may be useful for protein regulation.

- **Food reliance**: A new type of chemical that is modelled to act as artificial food for the cells in the ADS is introduced. The food chemical is required by the cells to process genes, and every time a gene is activated the food chemical is used up. The absence of food chemical prevents the activation of genes. This chemical is diffused within the organism but it is not shared between neighbouring cells via contact cell signalling. The food chemical is supplied via an outside source which is located around the organism, and the source supplies a constant amount of food chemical every developmental step to the cells on the outside borders of the organism. The food chemical can not bind the genes and it can not be produced by the active genes. In Table 8.9, the food reliance is given a "supply level", which is the amount of food chemical being supplied by the outside source every developmental step.

Table 8.9: *The labels for the charts and plots of experiments with various developmental mechanisms.*

Reference in Plots	Mechanism
A	Original setting
B	2 local proteins
C	Voter decision mechanism
D	4 messenger molecules
E	Food reliance - supply level 255
F	Protein consuming genes
G	Unproductive Genes
H	8 messenger molecules
I	Food reliance - supply level 32
J	Food reliance - supply level 64
K	Food reliance - supply level 128

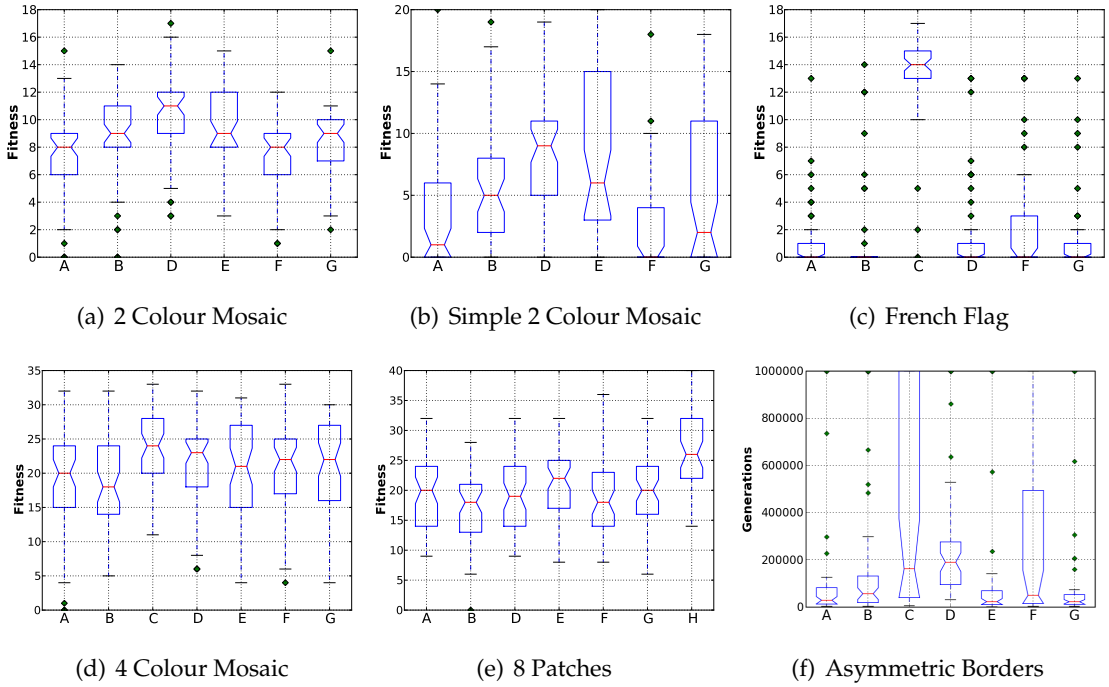


(a) 2 Colour Mosaic (b) Simple 2 Colour Mosaic (c) French Flag

(d) 4 Colour Mosaic (e) 8 Patches (f) Asymmetric Borders

Figure 8.11: *The success rates achieved by the different developmental mechanisms. The aliases used in the x-axis are explained in Table 8.9.*

Unlike most of the previous experiments the inclusion of different mechanisms presented in this subsection did not have a large effect in the performance of the ADS. The patch



(a) 2 Colour Mosaic (b) Simple 2 Colour Mosaic (c) French Flag

(d) 4 Colour Mosaic (e) 8 Patches (f) Asymmetric Borders

Figure 8.12: *The box plots of the results from the experiments with the different developmental mechanisms. The aliases used in the x-axis are explained in Table 8.9. The fitness values achieved at the end of each run is used for all the experiments except the asymmetric borders pattern experiments. Due to the high success rate of most experiments in achieving the asymmetric borders pattern, the number of evolutionary generations to achieve stable results is used for its box plots.*

patterns slightly benefited from the use of "local proteins", whereas the mosaic patterns suffered from their use, see Figure 8.11. The same is true for "protein consuming genes", which gave the "simple 2 colour mosaic" pattern a boost in performance, and deteriorated the performance of the other patterns. In conclusion, none of the mechanisms described in this section made any real improvement on the evolvability of the ADS; some of them simply increased the biases in achieving certain patterns. The best overall performance was achieved without the use of any of these mechanisms. It is not surprising that the messenger molecules does not improve the evolvability of ADS in pattern experiments, since these chemicals are designed to adapt the ADS in a changing environment. The use of local proteins have already been shown to degrade the performance of a multicellular developmental system, due to its limited use and the extra complexity involved with its simulation [Haddow and Hoye, 2007]: the results presented in this section confirms this. The use of "food reliance" created a similar situation as the local proteins and increased the complexity of the system. In fact in this

case evolution had less control over the "food chemical" causing the resulting system to be less evolvable. The unproductive genes and protein consuming genes created redundancy in the control of chemicals available in the system. The evolvability of the system did not show a significant change with these two new functionalities, which suggest that the developmental system already had a sufficient "tools" in controlling the chemical levels in the organism.

Figure 8.12 illustrates the changes in the evolvability of the ADS with the new mechanisms well; most of the time the use of new mechanisms did not affect the fitness distribution (number of generations for the asymmetric borders pattern). Only changing the decision mechanism to a voter (not shown in all the plots) had really deteriorated the ADS performance.

8.7 Improving the ADS

The previous subsections have investigated various mechanisms and developmental parameters that affect the evolvability of a bio-inspired GRN based multicellular ADS. It was shown that several of these mechanisms (e.g. cell signalling, adaptive parameters, etc.) had large effects on the evolvability of the multicellular ADS. Table 8.10 summarises the results of these experiments. The mechanisms discussed and demonstrated to improve the evolvability of the ADS to achieve stable patterns are simple to implement and during the simulations had negligible effects in the computational complexity of the system. These mechanisms (B–E in Table 8.11) should also be simple to implement in an embedded system (on a microprocessor or Field Programmable Gate Array (FPGA)).

In this section the best of the mechanisms that have significant effects on the evolvability of the ADS are shown side by side along with the original design of the ADS, and with a combination of all these "best" mechanisms. The aim is to obtain an idea about the amount of effect one mechanism may have on the evolvability of the ADS in comparison to the others, as well as demonstrating the amount of improvement that can be achieved via the use of correct combination of developmental mechanisms in an ADS.

Figure 8.13 displays the number of successful runs (as well as the stable ones) out of fifty runs for each version of ADS in solving the six different patterns shown in Figure 8.1.

Table 8.10: *Summary of the investigations done on the evolvability of a multicellular ADS in forming various patterns.*

Mechanism	**Result**
Contact Signalling	It was shown that contact signalling is an essential part of an ADS, and its absence made pattern formation almost impossible for some examples. It is also important to design a contact signalling mechanism carefully, in order to allow the ADS to be able to control the intensity of signalling.
Diffusion	It was shown that both the lack of and constant presence of diffusion is undesirable. The best overall performance was obtained via a simple method that gave ADS control for the diffusion process without the need for more complicated mechanisms.
Gene-Chemical Interactions	The optimal values for ADS parameters were shown to be problem dependent. Most of the time a non optimal combination of these parameters was shown to be undesirable. Evolving these parameters during the evolution of the genotype of the ADS provided the best overall performance.
Non-Standard Chemicals	The use of messenger molecules were introduced to monitor the phenotypes of the neighbouring cells. Local proteins were introduced to regulate the dynamics within a cell without the direct interaction of other cells, and "food reliance" was introduced to create extra level of control on the number of active genes. All these chemicals provide alternative ways of guiding the ADS by gene regulation, but none showed any real improvement in the overall performance of the ADS.
Mapping Development	It was observed that a cell has more information in the oscillations of its chemicals' concentrations that arise during gene interactions rather than the concentration of its chemicals at the end of a developmental step. Thus it was concluded that using the final concentrations of chemicals to map the cell phenotype is inefficient.
Regulatory Logic	Only two different decision mechanisms for activating a gene were compared: a conjunctive expression and a voter. There were large negative effects of using a voter decision mechanism when compared to a conjunctive expression. It would be worth investigating the effects of a larger number of mechanisms.
Chemical Control	The results showed that uncontrolled supply of chemicals via diffusers or constant diffusion is not desired, and in most cases the performance of the ADS deteriorates under such conditions. Protein consuming genes and unproductive genes were introduced as redundant controls of chemical supplies. But the results showed that the ADS did not need need these mechanisms.

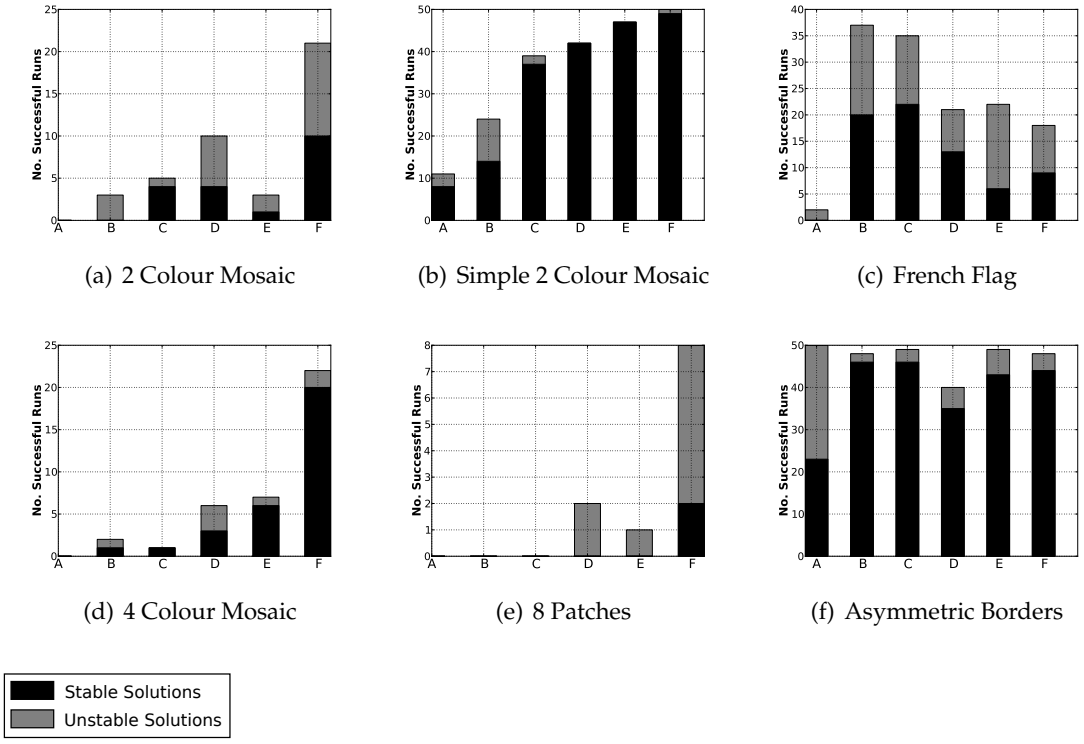
"Original setting" in the charts refer to the initial design of the developmental system; "simple tunnels", constant diffusion, pre-set chemical vs gene interaction constants. All the other experiments use "controlled tunnels", and have only one other mechanism implemented differently from the "controlled tunnels" experiments. In a final case all the improved mechanisms are used as part of the ADS.

Table 8.11: *The labels for the charts displaying the results for experiments with the best mechanisms.*

Reference	Mechanism
A	Original setting
B	Controlled tunnels
C	Evolved binding thresholds
D	Evolved chemical production and consumption rates per gene
E	Diffusion protein
F	Combination of B,C,D,E

The mosaic and the 8 patches are the patterns that generally benefit from the combination of all the improved mechanisms. In most cases the initial design of the ADS was unable to find any perfectly matching patterns in fifty runs, and for the patterns it did, the success rate was low (except asymmetric borders pattern). French flag pattern benefited mainly from a single different implementation of a mechanism –the "controlled tunnels" implementation of contact signalling–, and all the other "improved" mechanisms lowered the performance of the ADS in achieving French flag patterns. Asymmetric borders pattern was generally not affected by the changes made to the ADS, however the ADS achieved more stable asymmetric borders patterns with the use of "controlled tunnels" contact signalling. The most complex patterns – 8 patches, 4 colour mosaic and 2 colour mosaic– benefited from the combination of all the "improved" mechanisms, obtaining an improvement of 100-400% from the best case achieved among the other experimental runs. Using the combination of all the improved mechanisms, stable solutions were found for the 8 patches pattern for the first time. A small number of experiments looking at the stability and fault tolerance properties of some of the evolved organisms from this section are discussed in Appendix D.

It seems that the contact signalling is the most important multicellular developmental mechanism investigated here, and a careful design of contact signalling is important in obtaining an evolvable ADS. However contact signalling alone is not sufficient for a flexible ADS in tackling EC problems, and a careful design of other mechanisms is as important. It is essential when solving unknown problems that the ADS is not biased on a specific type of multicellular ordering, and it is equally evolvable for all EC problems.



(a) 2 Colour Mosaic (b) Simple 2 Colour Mosaic (c) French Flag

(d) 4 Colour Mosaic (e) 8 Patches (f) Asymmetric Borders

Figure 8.13: *Combination of all the best mechanisms compared to the "original setting" and each best mechanism. The aliases used in the x-axis are explained in Table 8.11.*

The experiments show that a version of an ADS may perform well on certain problems and equally poorly on others. This strongly suggests that using a single class of experimental problems for the empirical validation of an evolutionary system is not sufficient and often misleading.

8.8 Summary

In chapters 4 and 5, a novel GRN based multicellular ADS was introduced in detail with a description of the design process, using the literature in biology and EC as a guide. Several mechanisms and design constraints were embodied as part of the ADS during the design process. But the logic behind these decisions was merely an interpretation of the biological development and artificial development in EC. Although the literature on multicellular ADS is rich with various models and impressive demonstrations on the capabilities of ADSs, the understanding of important mechanisms and parameters has been poor.

Consequently, a detailed investigation into some of the poorly understood mechanisms and design constraints were undertaken in this chapter. These investigations were done with the hope of optimising the presented ADS as well as providing a better understanding of some of the developmental mechanisms. 2D patterns were used as the experimental problems. Such 2D patterns are easy to understand, implement, and make a distinction between different organisational ordering. Using problems of different orders proved to be important throughout the experiments, as patterns with different types of orders usually ended up providing different results. This was an important lesson in choosing the set of problems carefully for empirical investigations on the properties of a system. An example of this was demonstrated by the experiments on the effects of constant diffusion; one of the patterns strongly favoured constant diffusion, three of the patterns produced acceptable results with it, and two of the patterns produced poor results. These results as well as the ones observed from the "chemical-gene interaction parameters", highlighted that most of the time the optimal parameters, and well performing mechanisms are highly problem dependent. The best mechanism/parameter value was found to be the most flexible one that provided the best overall performance, but not necessarily the best performance for each problem. Table 8.10 provides a summary all the results of the experiments undertaken.

In almost all the experiments it was observed that an ADS is more evolvable when evolution has more control over it. The ADS was the most evolvable:

- When its parameters were included in the genotype being evolved.
- With a diffusion cell signalling mechanism, where the diffusion rate could be adjusted by the ADS; indirectly giving control to evolution.
- With a contact cell signalling mechanism, where the ADS controlled the flow of chemicals; again indirectly giving control to evolution.

A large number of experiments (of over 40 sets of 50 runs) were done with six different patterns. Each run took approximately an hour of computational time on the single core of a 2.83GHz Intel Q9550 CPU based Unix system. This totalled up to approximately 12000 hours of CPU time for all the investigations presented in this chapter. The experi-

ments presented in this chapter were run on a Unix based cluster PC that was built of 10 Intel Q9550 CPUs and 80GB of memory.

A few of the important mechanisms and parameters in multicellular development were explored by the investigations provided, but many mechanisms were left out and even more mechanisms and parameters that need detailed investigations were found. Growth and cell division, cell movement and adhesion, importance of the order of cell processing in a developmental step, the use of signalling pathways, long distance cell signalling (i.e. auxins and endocrine) are some of the important and only partially explored mechanisms that can have a significant effect on the performance of an ADS, and these mechanisms would be worth investigating in future. It was also found in the experiments provided in this chapter that a detailed investigation of the decision mechanisms used in the rule based GRN models and the techniques of mapping a developmental system to a phenotype may provide important advancements in the use of ADS in EC.

Understanding the capabilities of the mechanisms that accompany ADS is an important step towards harnessing the full potential of ADS for designing and optimising engineering problems. With the advancing technology and a better understanding of what is needed from EC, ADS has the potential to be used in the evolutionary design of real life systems.

Chapter 9

Conclusions

Biological processes such as evolution and development are intricate systems. These partially understood processes provide engineers and computer scientists with new inspirations for new design techniques and computing paradigms. However, using models of these processes to achieve reliable design methods is arduous. In order to discover the full potential of these approaches, one needs to sample a large number of cases exploiting the properties of these models.

Using evolution, a stochastic meta-heuristic, to design circuits on a hardware platform is a non-trivial task and there lie many traps created by the complex dynamics of evolution and the hardware environment, which can divert progress. It has already been shown in the literature that evolution on a hardware substrate can result in unconventional designs (see Chapter 2). These designs are in a design space unreachable by conventional engineering techniques. Although these examples in the literature reflect the good aspects of evolution on hardware, it was shown in Chapter 3 that when uncontrolled, evolution of circuits in hardware can be infeasible. The input pattern ordering during the evaluation phase of evolution was shown to be essential for the evolution of valid digital circuits, along with effective definition of fitness functions. It was discussed and demonstrated that the fitness functions define the fitness landscape, and in a hardware substrate the amount of information that can be detected by the fitness function contributed greatly to the successful evolution of valid circuits. Along with the evolution of valid circuits, multiple techniques were also developed for quick and effective evolution of circuits in hardware. These methods involved effective use of input pattern ordering, fitness

function, partitioning of the circuit outputs and applying inputs of the evolved system at multiple locations on the chip. Using these methods, it was shown that the evolution process was considerably faster when evolving circuits in a hardware platform.

After a review of the Evolvable HardWare (EHW) field in Chapter 2, and the development of various mechanisms for the efficient evolution of valid circuits in hardware in Chapter 3, it was discussed that the complexity achievable when evolving circuits (intrinsically or extrinsically) is limited when direct genotype–phenotype mapping is used, and in fact the experiments presented in Chapter 3 represent circuits with complexities close to this limit. Therefore, artificial multicellular development was introduced in Chapter 4 as an effective genotype–phenotype mapping technique that has the potential to promote the evolution of scalable circuits. It was further discussed that development is more than just a genotype–phenotype mapping technique in biology and it has more to offer for Evolutionary Computation (EC). Fault tolerance and adaptivity were discussed as two of the most important benefits of evolving developmental systems. A new method of classification was introduced to provide a clearer distinction in the existing artificial implementations of development. This new method divides Artificial Developmental Systems (ADS) into two categories depending on their sources of inspiration: macro-modelling was used to refer to Artificial Developmental System (ADS)s that model the overall behaviour of biological development (taking a ”high-level” view), and micro-modelling was used to refer to ADSs that model biological development at a small scale (taking a ”low-level” view), i.e. modelling cells and their interactions that lead to multicellular development. Chapter 5 provided the description of a new bio-inspired Gene Regulatory Network (GRN) based developmental system. A table of relevant developmental models was created and further discussed as the details of the new developmental system were explained. The new developmental system was designed to be biologically plausible, but the representation and processing of the developmental system were kept simple in order to allow its implementation in embedded systems with limited memory space and processing power.

A series of experiments investigating the evolvability of the GRN and the multicellular developmental system were reported in Chapter 6. These experiments demonstrated that the presented GRN is capable of creating stable dynamics using chemicals for interactions, inputs and outputs. The initial experiments with GRNs also showed that

they can be evolved to be responsive to environmental changes reflected as changes in chemical concentrations, and create clocks via the changes in chemical concentrations and internal dynamics. The experiments with the multicellular developmental system in Chapter 6 demonstrated that the developmental system is able to achieve multicellular organisation and differentiation. It was shown that using non-deterministic maturing and fitness functions that define the order of the pattern rather than the exact pattern create a more evolvable developmental model. It was also shown that the developmental model presented in this thesis is capable of fully recovering from transient faults, and maintaining target cell organisation (either perfectly or with small imperfections) in the presence of cell death or gene knock-outs.

The experiments with the ADS in Chapter 6 were followed by initial experiments developing digital circuits in Chapter 7. Circuits were developed using the ADS, where the ADS built both the routing and logic side of the circuits. Although successful, it was observed that developing the circuit connectivity limited the scalability properties of the ADS. By providing a simpler way of creating circuit connectivity and a more effective fitness function, it was shown that the evolution of an ADS that can develop a generic even n-bit parity circuity. Once more, the importance of the fitness function and careful design of the evolution environment was demonstrated to be of utmost importance for evolving designs.

Chapter 7 concluded that there is a need to further investigate the mechanisms and parameters that directly affect the evolvability of the developmental system. Therefore, an in-depth investigation on the effects of some of the important developmental mechanisms and parameters was undertaken. The results and discussions on this investigation were presented in Chapter 8. A large number of experiments were undertaken using six different patterns representing different test cases for multicellular organisation. The experiments were aimed at testing the evolvability of the ADS in achieving **stable** multicellular organisation. It was shown that several of the mechanisms tested (such as cell signalling and developmental parameters) had drastic effects on the evolvability of the multicellular ADS. The results were gathered in a table (Table 8.10), which summarised the experiments. It was later shown that using the information gathered, an optimized version of the ADS could be obtained. Although this optimized version of the ADS did not provide the best possible evolvability for every experimental pattern, it was

demonstrated to provide the best evolvability overall. Some of the contradictory results from different patterns showed that using a single pattern or a group of patterns with the same organisational properties can lead to misguided conclusions.

It was interesting to see that unrestricted diffusion was unfavourable, a conclusion that confirms the observations from embryo development [Fagotto and Gumbiner, 1996]. Similarly, the investigations in contact signalling–which was modelled after the plasmodesmata in plant cells–revealed that creating direct links between two cells which allows the uncontrolled movement of chemicals from one cell to another is not favoured. In plant cells, when a plasmodesmata is created, the size of the "tunnel" is small and only allows molecules of small sizes to travel through directly (passive movement). Larger sized proteins can only go through the plasmodesmata via "movement proteins" that can carry the larger proteins through the plasmotesmata channels (active transport) [Leyser and Day, 2003]. From the investigations in Chapter 8, it was found that using "controlled tunnels" that made the use of transport proteins had greatly enhanced the evolability of the ADS.

The chapters in this thesis reviewed EHW and the use of artificial development in EC. Evolution of circuits on hardware was suggested as an effective way of evolving valid circuits with unusual properties that extrinsic evolution or traditional circuit design techniques could not achieve. However, it was shown that evolving circuits on real hardware can be problematic. Thus a set of techniques were developed to ensure the successful evolution of valid circuits on hardware. For the evolutionary experiments on hardware, a novel hardware evolution platform (Reconfigurable Integrated System Array (RISA)) was used and consequently, the properties of this platform were investigated. Once a set of techniques were developed for evolving circuits on hardware, it was suggested that a more scalable way of mapping the evolved genotype to the target phenotype (circuits) is needed. A GRN based ADS was designed in the following chapters to satisfy this need. The designed developmental system was demonstrated to be evolvable at a single and multicellular level. It was also demonstrated to show inherent fault tolerant properties both at genotypic and phenotypic levels. The developmental system was evolved to build various circuits, including a scaling parity circuit. Finally, a detailed investigations into some of the developmental mechanisms and parameters were done in order to optimize the ADS designed and provide a better understanding on the use

of GRN based ADSs in EC. The investigations undertaken provided some crucial results on understanding the effects of various developmental mechanisms on the evolvability of a GRN based ADS for achieving multicellular organisation. By the end of this thesis, a list of crucial mechanisms for the evolution of circuits on hardware as well as a better understanding of the capabilities of evolution and artificial development in the design of computational systems are obtained. The thesis' novel contributions provide creating a better understanding of:

- Evolving digital circuits on real hardware by addressing reliability issues that emerge in intrinsic hardware evolution.
- The desired characteristics in a hardware evolution platform.
- Evolvability on the artificial models of GRN.
- Evolvability on the artificial models of multicellular development.
- Accurate evaluation of the properties of an evolutionary and developmental model.

The work presented proves the hypothesis stated at the start of this thesis. Evolution of digital circuits in hardware can provide interesting and novel designs, but not complex and human competitive results (Chapter 3). Multicellular development in biology can be simulated to create a scalable system for the evolutionary design of electronic systems (Chapter 7). By understanding evolution of circuits on real hardware and the behaviour of multicellular development in a computational environment, the key factors that determine the evolvability of an evolutionary developmental system can be determined (chapters 3, 6, 7, and 8 and Appendix D).

Although a large number of experiments and research has been undertaken, there is still a large scope for future investigative work.

9.1 Future Work

A large number of time-consuming experiments were reported in Chapter 8 of this thesis, and only a few of the important mechanisms and parameters in multicellular development could be explored. There is still a large number of developmental mechanisms that

are poorly understood for their effects on the evolvability of a developmental system. Growth and cell division, cell movement and adhesion, importance of the order of cell processing in a multicellular organism, the use of signalling pathways, long distance cell signalling (i.e. auxins and endocrine) are some of the important and only partially explored mechanisms that can have a significant effect on the performance of an ADS, and these mechanisms would be worth investigating in future. It was also found in the experiments provided in Chapter 8 that a detailed investigation into the decision mechanisms used in the rule based GRN models and the techniques of mapping a developmental system to a phenotype may provide important advancements in the use of ADS for EC. As shown in Chapter 8 using multiple types of patterns provided more reliable information on how evolvable the developmental system is with the use of a particular developmental mechanism. However, the decision of choosing the most evolvable developmental mechanism became tough in some cases due to the contradicting results from different patterns. It would be worth investigating and including fitness function independent evolvability measures into future investigations as well as fitness function dependent measures with experimental applications other than patterns (such as Artificial Neutral Networks (ANN), control systems, and image compression) for even more reliable results. Jin and Trommler developed a fitness function independent evolvability measure for GRN based developmental systems, where the GRN is modelled using differential equations [Jin and Trommler, 2010]. It should be possible to extend the method they present for developmental systems using rule-based GRN models.

In Chapter 6, it was shown that some of the developmental organisms could achieve full recovery from transient faults, be unaffected by the loss of genetic information, and maintain the target pattern after permanent cell failure within the organism. In Appendix D, some of these properties along with the ability to maintain a stable pattern are further discussed. It would be interesting and perhaps highly rewarding to investigate and understand in detail the key mechanisms that enhance the fault tolerance properties of developmental systems. By understanding the causes of different fault tolerant and stable behaviours, ADSs can be used for fault tolerance as well as for fault detection in computational applications. Chapter 4 had highlighted adaptivity as one of the benefits of artificial development, but investigating the adaptive properties of multicellular artificial development was not part of the work provided in this thesis. Developmental organisms have been shown in another work to dynamically respond

and adapt to environmental information [Tufte, 2008b]. It was also shown in the first section of Chapter 6 of this thesis that the presented GRN model is capable of providing a responsive network and adapting its environment. Further experiments investigating the adaptive properties of the presented multicellular ADS would be a valuable extension to this thesis.

Understanding the capabilities of the mechanisms that accompany ADSs is an important step towards harnessing the full potential of ADSs in the design of computational systems. With the advancing technology and a better understanding of what is needed from EC, ADSs have the potential to provide the right approach to "evolving" problems into solutions. Applying evolution and development to the right computational problem is one of the foremost important steps that needs to be taken, before impressive designs can emerge with the use of evolution and development. During the writing of this thesis, the developmental model described in Chapter 5 has been used with many of the improvements obtained from Chapter 8 as an image compression tool that produced better results than JPEG image compression and provided robust results where a single genome can encode multiple images [Trefzer et al., 2010]. With the right choice of applications, the achievable designs by the evolution of a multicellular developmental system seems promising. However, finding the right computational application to tackle remains to be one of the grand challenges of EC using multicellular ADSs. Including the Chapter 7 of this thesis several other publications have demonstrated the use of development to design scalable digital electronic circuits; such as [Bidlo and Škarvada, 2008], and [Harding et al., 2009], who evolve developmental models that build generic multiplier and parity circuits respectively. Although good demonstrators of the power of developmental systems, these applications are not practical as they present already existing solutions.

As a better understanding of GRNs and ADSs is obtained, the use of ADSs will be key to the emergence of complex and intelligent artificial systems that can achieve comparable behaviours to biological systems. The use of developmental approaches to the growth of neural networks has been suggested and shown to be successful already [Khan et al., 2008]. A more bio-inspired and better tuned approach to growing neural networks has the potential of creating brain-like intelligent networks. With the successful application of ADSs, adaptive and intelligent systems can be created to be used in situations that

require the intuition of humans. Security checking, handling of dangerous materials, and human interactions are tasks that require intelligent, intuitive and adaptive systems. Although achieving such systems via engineered designs is not possible, bio-inspired approaches provide a gateway to these systems. With further research, implementation of real applications via the evolution of ADSs will be possible in the near future.

Acronyms

ADS Artificial Developmental System

ADF Automatically Defined Function

AI Artificial Intelligence

ANN Artificial Neutral Networks

ASIC Application Specific Integrated Circuit

BIST Built-In Self Testing

BMH Bitwise Fitness Modified for Hardware

CA Cellular Automata

CAB Configurable Analogue Block

CE Cellular Encoding

CGP Cartesian Genetic Programming

CLB Configurable Logic Block

COP Constrained Optimisation Problems

CPLD Complex Programmable Logic Device

DC Diffusion Constant

DP Diffusion Protein

DNA DeoxyriboNucleic Acid

EA Evolutionary Algorithm

EC Evolutionary Computation

ECGP Embedded Cartesian Genetic Programming

EHW Evolvable HardWare

EP Evolutionary Programming

ES Evolution Strategy

FPAA Field Programmable Analogue Array

FPGA Field Programmable Gate Array

FPTA Field Programmable Transistor Array

GA Genetic Algorithm

GP Genetic Programming

GRN Gene Regulatory Network

HBS Hierarchical Bit-string Sampling

HIFF Hierarchical IF-and-only-iF

IO Input/Output

L-Systems Lindenmayer Systems

LUT Look-Up Table

mRNA messenger RNA

NMR N-Modular Redundancy

RNA RiboNucleic Acid

RISC Reduced Instruction Set Computer

RISA Reconfigurable Integrated System Array

SNAP Simple Networked Application Processor

TMR Triple Modular Redundancy

tRNA transfer RNA

Appendix A

Resource Consumption on RISA

In Chapter 3, evolution of circuits on the RISA hardware platform was described, and various sequential and combinational circuits were evolved. Some of the circuits were

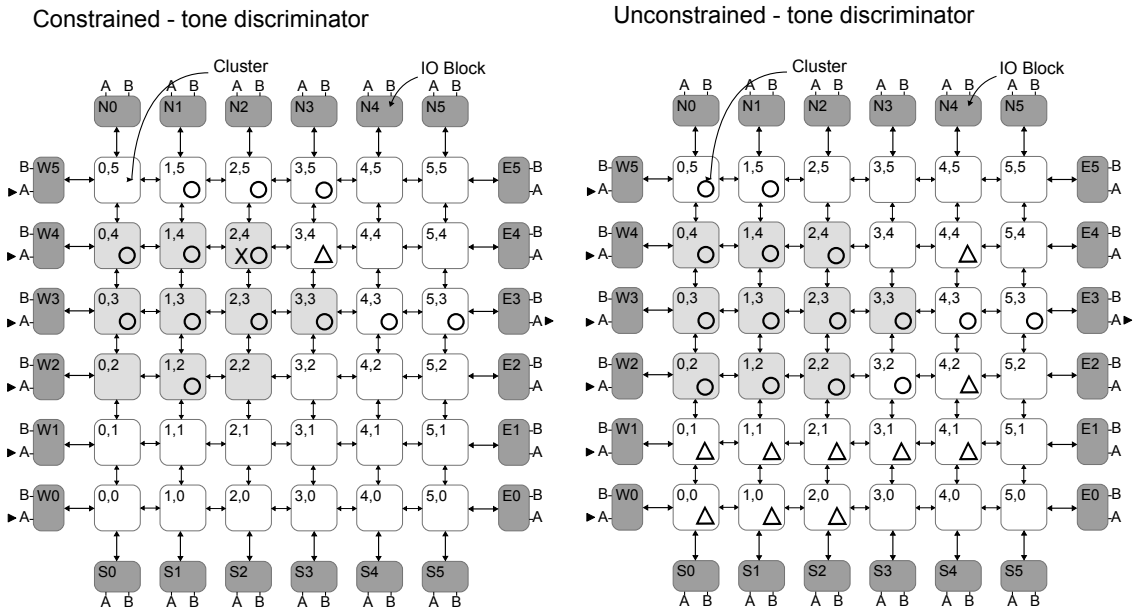


Figure A.1: *A map of the RISA clusters that are relevant for the operation of typical solutions found with constrained and unconstrained evolution respectively. Clusters that were relevant for all tested random input patterns are marked with a circle. Those, which became relevant in some cases are marked with a triangle and those which are no longer necessary when the ones with the triangles become relevant are marked with a cross.*

evolved on both a constrained and unconstrained versions of the RISA's reconfigurable architecture. Two of the evolved tone discriminator circuits, one in unconstrained and the other in constrained RISA platform, are analysed for the amount and style of cluster usage. In the constrained RISA platform the size of configurable bits per cluster were reduced from 472 to 58 bits.

The experiments were done only involving ten clusters out of 36 available on the RISA chip for both constrained and unconstrained cases (marked as grey blocks in Figure A.1). All the other clusters were configured to pass the incoming signals from west to east, and their configurations were not altered during evolution. The information displayed in Figure A.1, shows the clusters that were used in the final design of fully functional tone discriminators.

The clusters, which are actually relevant for the operation of a solution found, are identified by successively clamping configuration bits to zero until the fitness gets worse. It is interesting to see that evolution exploits clusters that were not explicitly subject to evolution and configured with the default, unchanged pass through configuration. This effect is present in both constrained and unconstrained evolution cases, however, it is stronger in the unconstrained case. Another surprising observation is related to testing with random input patterns: in order to provide the correct output for certain input patterns, additional clusters became relevant, as can be seen from Figure A.1.

Appendix B

Cluster and IO Routing in RISA

The configuration options and the connectivity of the IO pads going in and out of RISA FPGA is shown in Figure B.1. Each IO pad can be connected to only one line going in or out of the RISA FPGA, and can only be used as an input or output port.

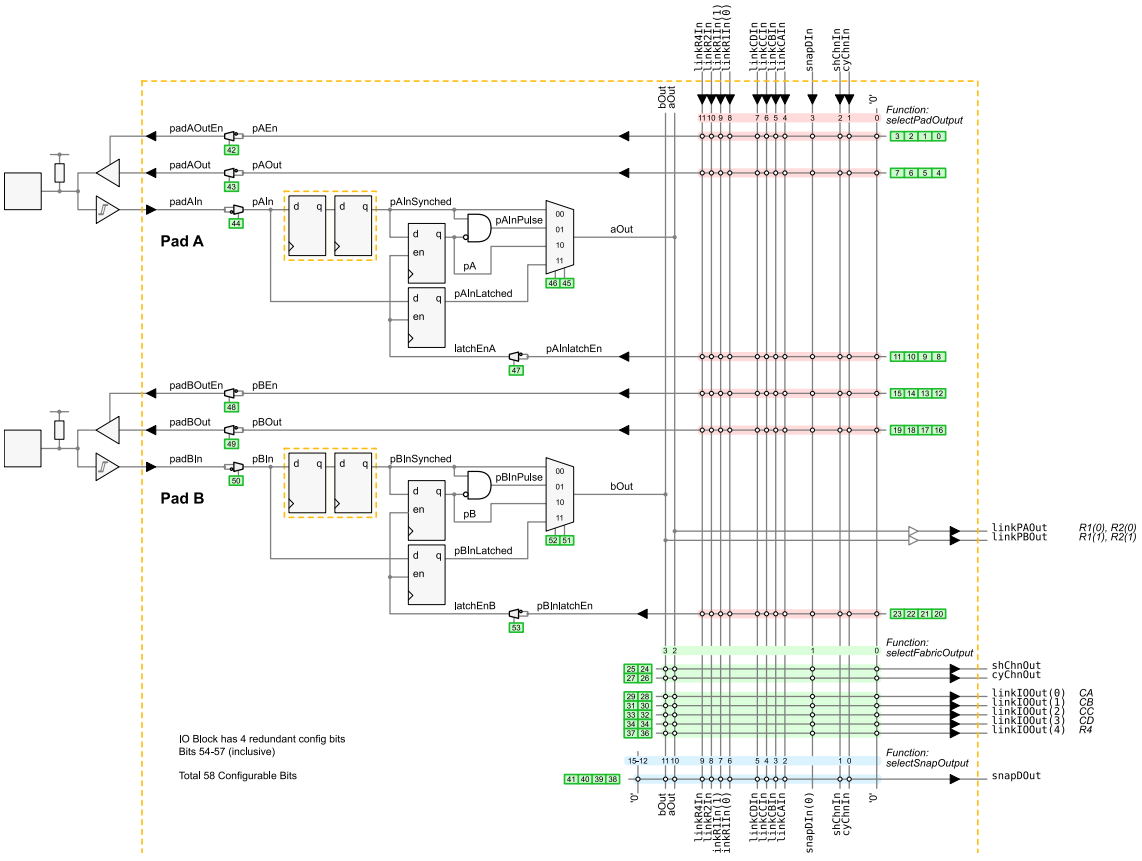


Figure B.1: *The complete schematic for one of the IO blocks surrounding the RISA FPGA. Each IO block has two IO pads, which can be configured as input or output to the FPGA substrate of RISA. The configurable bits are displayed in boxes.*

Figure B.2 shows the connectivity of the IO blocks to the RISA FPGA, and the connectivity among the RISA clusters. The maximum number of IO ports that can be used in each IO block is two since there are only 2 IO pads. Each cluster has four function units that are dedicated to inputs and outputs from and to a single direction (east, west, south, north). For example, accessing a function unit dedicated for "north" via inputs coming from "east" is not possible via combinational pathways. The only possible way to use these inputs is to route them to the function unit dedicated for inputs coming from "east", and a single output from the latter function unit can be registered to be used in the next clock cycle for the function unit dedicated for "north". Each function unit has a 16-bit look-up table, which require 4 inputs in order to be fully used. The limited number of inputs (maximum of 12 in each direction) limits the usability of the substrate. For example, a

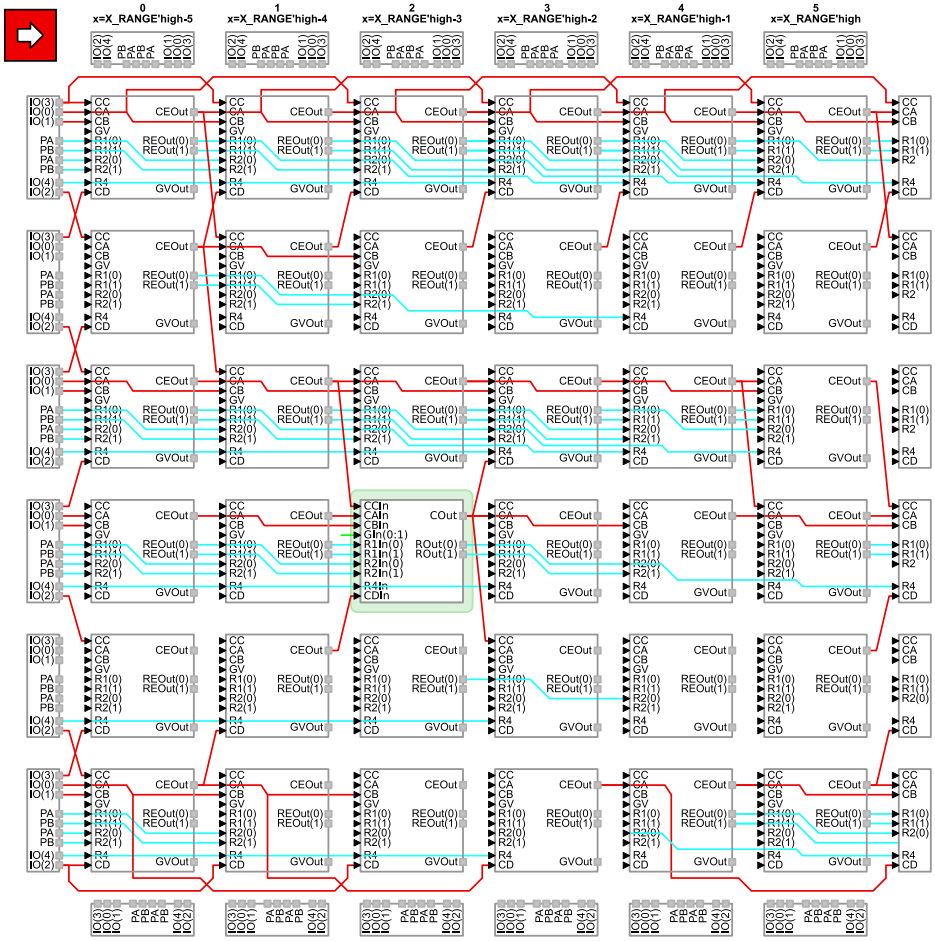


Figure B.2: *The schematic for the routing between the clusters and IO blocks of RISA FPGA. The routing at this level is fixed and not configurable. In the schematic only the eastward direction is shown, the same routing scheme is present in all four directions.*

maximum of only 2 function units out of the 24 (6 clusters $\times$ 4 function units) nearest function units to the IO blocks can be fully utilized.

Even though the RISA FPGA substrate provides 144 16-bit look-up tables for building combinational and sequantial digital circuits, the total number of usable look-up tables in one direction is less then 10. This is a main contributing factor for obtaining large performance increase in evolution of multiple output circuits when the outputs of the evolved circuits were partitioned, see Section 3.5.3. Therefore, the small number of available IOs and routing for the comparably large amount of logic provided in RISA makes this platform hard to evolve circuits on. This is because, the number of IOs provided in RISA only allow a very small portion of the FPGA substrate to be utilized, hence a large part of the logic circuitry create unusable design space. Therefore when unconstrained, it is possible for evolution to spend a long time searching for an optimal design by changing the configuration of the vast amount of unusable logic provided in RISA. Although it has been shown that in evolution of circuits, "neutral search" can improve the evolutionary performance [Harvey and Thompson, 1997; Vassilev and Miller, 2000a]. The architectures where these "neutrality" experiments with redundant hardware were done, provided the possibility of connecting the redundant hardware to the actual design. As it was termed by Harvey and Thompson, neutrality of the right kind with "potentially useful junk" helps eliminate local optima [Harvey and Thompson, 1997]. Unfortunately, in RISA there is a large amount of redundancy that is unusable by evolution, thus instead of being eliminated, the local optima are enhanced.

Appendix C

Explaining Box and Whisker Plots

A box plot displays a variety of information about the sample data being plotted; an example plot illustrated in Figure C.1. The median of a sample data is shown as a line in the middle of a box. The 25^{th} and 75^{th} percentiles of a sample are drawn as the lower and upper lines of the box. The distance between the lower and upper lines of the box is referred to as the inter-quartile range. The rest of a sample is shown by the whiskers of the plot, which covers up to 1.5 times the inter-quartile range of the sample. If there are any data outside the range covered by the whiskers, these are then marked by diamond symbols above and below the whiskers and they are referred to as outliers. The notches

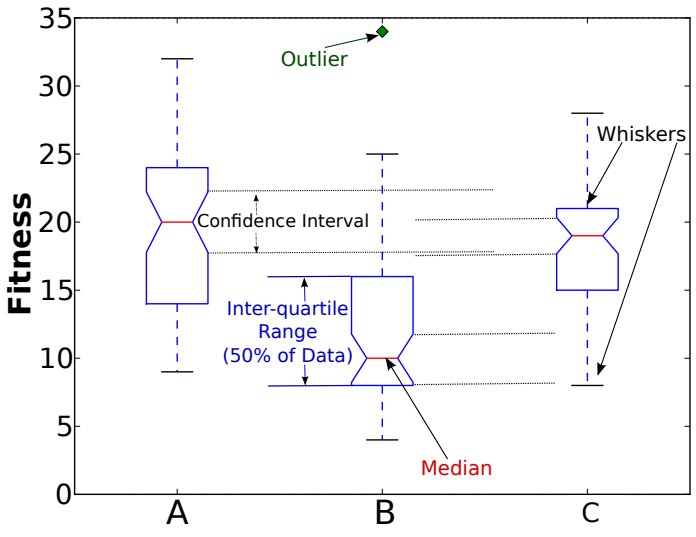


Figure C.1: *Example box and whisker plots of three different data sets.*

in the box plot, first introduced by [McGill et al., 1978], are a graphic confidence interval about the median of a sample. A small confidence interval demonstrates a good sample of data. A side-by-side comparison of two notched box plots can provide information on the statistical difference between the two data sets. If the notches of the two data sets do not overlap the medians are significantly different at about a 95% confidence. In the example plot shown in Figure C.1, the confidence intervals of data set A and C overlap, suggesting that these two data sets are not significantly different, whereas the confidence interval of data set B does not overlap with either of the other two data sets, suggesting that data set B is significantly different than the other two data sets.

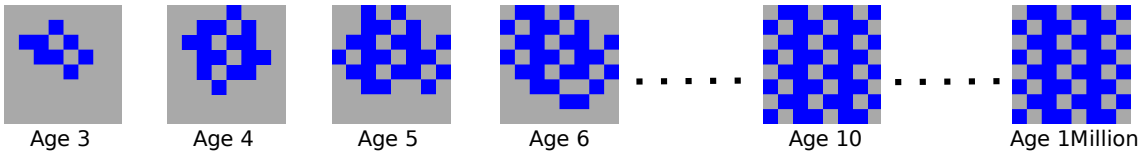
Appendix D

Stability and Fault Tolerance

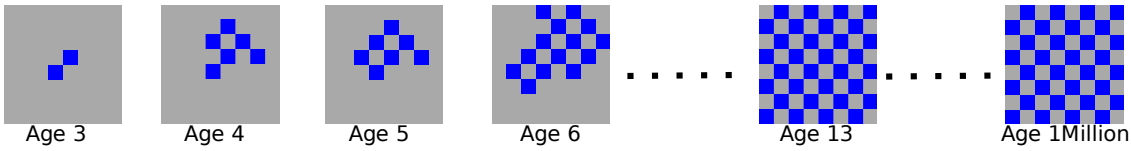
It was shown in Section 6.4 of Chapter 6 that the developmental system presented in this thesis can achieve emergent fault tolerance and recovery in the presence of various faults such as; gene knock-outs, cell death and transient faults. The most stable two organisms from a total of 80 different runs (40 runs for each pattern) that achieve two different patterns were used for these fault tolerance and recovery demonstrations. However, neither of these organisms were able to maintain the target pattern for a very long time; the asymmetric pattern would go in to an oscillatory state with one of the cells changing state every 15 developmental steps, and the French flag would stay stable only for the first 465 developmental steps.

One organism from each of the pattern experiments in Section 8.7 of case 'F' (the combination of "best" mechanisms) is chosen at random from the solutions marked as "stable" (i.e. successfully tested during evolution to keep the target pattern for 10 developmental steps). These are the organisms evolved with the final version of the developmental system after the developmental mechanisms and parameters are tuned by the investigations in Chapter 8. All six of the chosen organisms are evaluated for 1 million developmental steps for their ability to remain stable at the pattern they were evolved to form. Each of these (semi)randomly chosen organisms remain stable for 1 million developmental steps without diverting from their respective target patterns. Figure D.1 illustrates the developmental path of each organism used in this section.

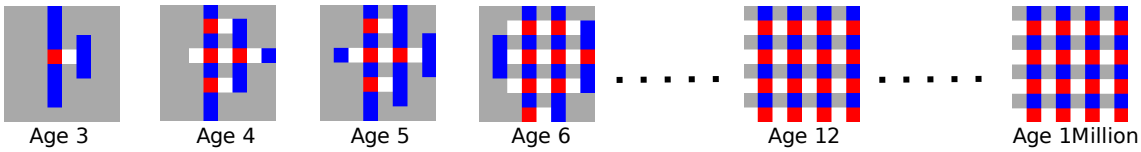
Furthermore, two of the organisms are tested for their fault tolerant capabilities by repeating tests used in Section 6.4. The organisms that form the same patterns used in



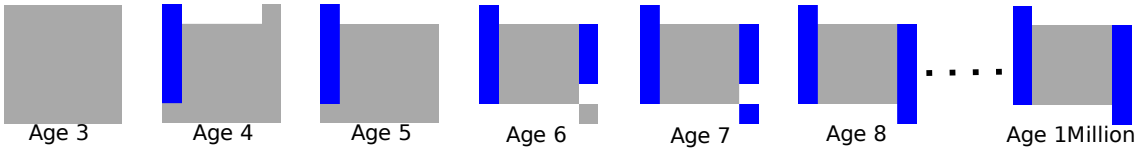
(a) 2 colour mosaic pattern



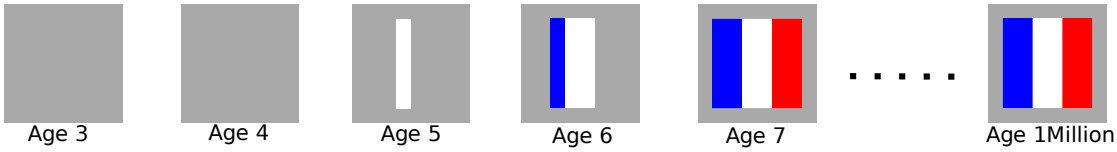
(b) Simple 2 colour mosaic pattern



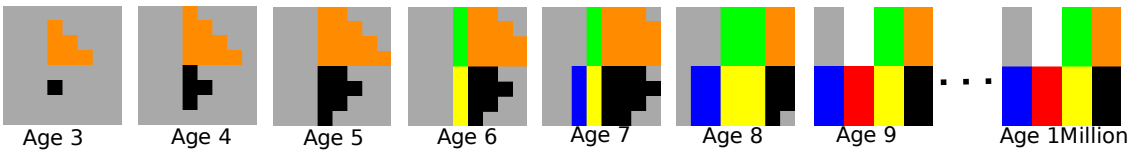
(c) 4 colour mosaic pattern



(d) Asymmetric borders pattern



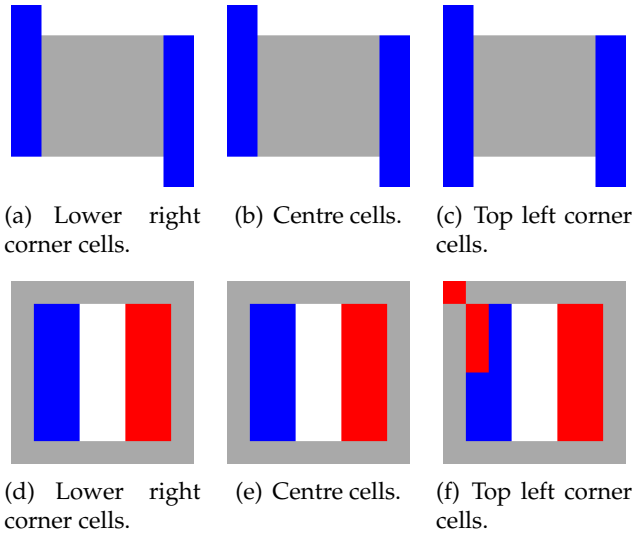
(e) French flag pattern



(f) 8 patches pattern

Figure D.1: *The development of organisms for 1 million developmental steps for each pattern used in Chapter 8.*

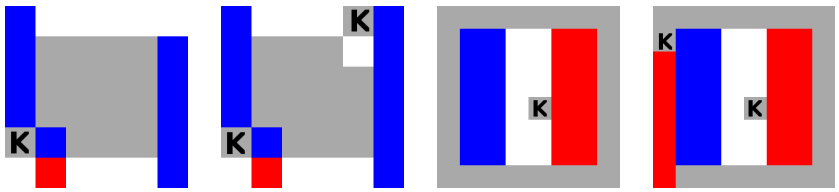
Section 6.4 –French flag and asymmetric borders patterns– are used for the fault tolerance tests. Similar or slightly better results when compared to Section 6.4 are obtained in the case of cell death or transient faults (see figures D.2 and D.3). However, both of the organisms display tolerance to a higher number of gene knock-outs. Both organisms can tolerate the loss of 7 genes. The organism forming French flag still forms the same pattern even if all 7 of these genes are knocked out at the same time. The organism forming asymmetric borders pattern can achieve the same pattern up to a knock-out of 6



(a) Lower right corner cells. (b) Centre cells. (c) Top left corner cells.

(d) Lower right corner cells. (e) Centre cells. (f) Top left corner cells.

Figure D.2: *Transient faults on organisms forming the French flag and asymmetric borders patterns. Only the final stable patterns achieved after the faults are shown in these figures. The cells where the transient faults were introduced are noted below each figure.*



(a) (b) (c) (d)

Figure D.3: *Cells are killed in the organisms forming the French flag and asymmetric borders patterns. Only the final stable patterns achieved after the faults are shown in these figures. The location of the cells killed are marked with letter 'K' on each figure.*

genes at a time (even though 7 different genes can be knocked out individually without disturbing the pattern). This is a large increase in the number of redundant genes from the examples shown in Section 6.4, where the French flag pattern had no redundant genes and the asymmetric borders pattern only had one redundant gene.

Although there is not much improvement to the fault tolerant properties of the evolved organisms with improved combination of mechanisms in the face of transient faults and cell death, these organisms have a considerably larger number of redundant genes than the ones in Chapter 6. Investigating the fault tolerance and long term stability of the organisms evolved in the experiments presented in Chapter 8 in detail may provide further insight into the effects of the mechanisms and parameters discussed as well as a better understanding of how different types of fault tolerant properties emerge in a developmental organism.

Bibliography

(1997). *Xilinx Xc6200 FPGA-Based Reconfigurable Co-Processor Data Sheet*. Xilinx Inc.

Alpert, D. and Avnon, D. (1993). Architecture of the pentium microprocessor. *IEEE Micro*, 13(3):11–21.

Arad, B. and El-Amawy, A. (1994). Robust fault tolerant training of feedforward neural networks. In *Circuits and Systems, 1994., Proceedings of the 37th Midwest Symposium on*, pages 539–544, Lafayette, LA, USA.

Araujo, S. G., Mesquita, A. C., and Pedroza, A. (2003). Using genetic programming and high level synthesis to design optimized datapath. In *ICES'03: Proceedings of the 5th international conference on Evolvable systems: from Biology to Hardware*, pages 434–445.

Back, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17.

Banzhaf, W. (1993). Genetic programming for pedestrians. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, page 628, University of Illinois at Urbana-Champaign. Morgan Kaufmann.

Banzhaf, W. (2003). On the dynamics of an artificial regulatory network. In *Advances in Artificial Life, Proceedings of the 7th European Conference (ECAL-2003),*, pages 217–227. Springer Berlin / Heidelberg.

Banzhaf, W., Beslon, G., Christensen, S., Foster, J. A., Kepes, F., Lefort, V., Miller, J. F., Radman, M., and Ramsden, J. J. (2006). Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7:729–735.

Bentley, P. (2005). Investigations into graceful degradation of evolutionary developmental software. *Natural Computing: an international journal*, 4(4):417–437.

Bentley, P. and Kumar, S. (1999). Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proc. of the Genetic and Evolutionary Computation Conf.*, pages 35–43, Orlando, Florida, USA. Morgan Kaufmann.

Berleth, T. and Sachs, T. (2001). Plant morphogenesis: long-distance coordination and local patterning. *Current Opinion in Plant Biology*, 4:57–62.

Bidlo, M. and Škarvada, J. (2008). Instruction-based development: From evolution to generic structures of digital circuits. *Int. J. Know.-Based Intell. Eng. Syst.*, 12(3):221–236.

Bland, A. S., Kendall, R. A., Kothe, D. B., Rogers, J. H., and Shipman, G. M. (2009). Jaguar: The Worlds Most Powerful Computer. In *CUG 2009 Proceedings*.

Bode, H. R. (2003). Head regeneration in hydra. *Developmental dynamics*, 226:225–236.

Boers, E. J. and Kuiper, H. (1992). Biological metaphors and the design of modular artificial neural networks. Master's thesis, Leiden University.

Bonner, J. T. (1998). The origins of multicellularity. *Integrative Biology: Issues, News, and Reviews*, 1(1):27–36.

Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101.

Bradley, D., Ortega-Sanchez, C., and Tyrrell, A. (2000). Embryonics + immunotronics: A bio-inspired approach to fault tolerance. In *The Second NASA/DoD Workshop on Evolvable Hardware*, page 215.

Bradley, D. and Tyrrell, A. (2002). Immunotronics: Novel finite-state-machine architectures with built-in self-test using self-nonself differentiation. In *IEEE Trans. Evolutionary Computation*, pages 227–238.

Braitenberg, V. (2001). Brain size and number of neurons: An exercise in synthetic neuroanatomy. *Journal of Computational Neuroscience*, 10(1):71–77.

Bramlette, M. F. and Bouchard, E. E. (1991). Genetic algorithms in parametric design of aircraft. In Davis, L. D., editor, *Handbook of Genetic Algorithms*, chapter 10, pages 109–123. Van Nostrand Reinhold.

Bremermann, H. J. (1962). Optimization through evolution and recombination. *Self-Organizing Systems*, pages 93–106.

Brown, S. and Rose, J. (1996). Fpga and cpld architectures: A tutorial. *IEEE Des. Test*, 13(2):42–57.

Canham, R. and Tyrrell, A. (2002). Evolved fault tolerance in evolvable hardware. In *Congress on Evolutionary Computation*, pages 1267–1272, Hawaii.

Canham, R. and Tyrrell, A. (2003). A hardware artificial immune system and embryonic array for fault tolerant systems. *Genetic Programming and Evolvable Machines*, 4(4):359–382.

Chavoya, A. (2009). *Foundations of Computational, Intelligence Volume 1*, volume 201/2009 of *Studies in Computational Intelligence*, chapter Artificial Development, pages 185–215. Springer Berlin / Heidelberg.

Claverie, J.-M. (2001). What if there are only 30,000 human genes? *Science*, 291(5507):1255–1257.

Corno, F., Prinetto, P., and Reorda, M. (1996). A genetic algorithm for automatic generation of test logic for digital circuits. In *Proceedings of the IEEE International Conference On Tools with Artificial Intelligence*, Toulouse, France.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In Grefenstette, J. J., editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1st edition.

Davidson, E. (2006). *The Regulatory Genome: Gene Regulatory Networks In Development And Evolution*. ACADEMIC PRESS, 1 edition.

Davidson, S. (2005). Bist the hard way. *IEEE Design and Test of Computers*, 22:386–387.

Dawkins, R. (2003). *On Growth, Form and Computers*, chapter The evolution of Evolvability, pages 239–255. Elsevier Academic Press.

de Jong, H. (2002). Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9(1):67–103.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

Dellaert, F. and Beer, R. (1994). *Toward an Evolvable Model of Development for Autonomous Agent Synthesis*. MIT Press Cambridge.

Devert, A., Bredeche, N., and Schoenauer, M. (2007). Robust multi-cellular developmental design. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 982–989, New York, NY, USA. ACM.

Edelman, G. M. and Gally, J. A. (2001). Degeneracy and complexity in biological systems. *Proceedings of the National Academy of Sciences of the United States of America*, 98(24):13763–13768.

Eggenberger, P. (1997). Evolving morphologies of simulated 3d organisms based on differential gene expression. In *Proceedings of 4th European Conference on Artificial Life*, pages 205–213.

Fagotto, F. and Gumbiner, B. M. (1996). Cell contact-dependent signalling. *Developmental Biology*, 180:445–454.

Fahrmair, M., Sitou, W., and Spanfelner, B. (2006). Unwanted behavior and its impact on adaptive systems in ubiquitous computing. In *ABIS 2006 : 14th Workshop on Adaptivity and User Modeling in Interactive Systems*.

Federici, D. (2004). Using embryonic stages to increase the evolvability of development. In *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*. Springer Verlag.

Flann, N., Jing, H., Bansal, M., Patel, V., and Podgorski, G. (2005). Biological development of cell patterns : Characterizing the space of cell chemistry genetic regulatory networks. In *8th European conference, ECAL*. Springer Berlin / Heidelberg.

Fleischer, K. and Barr, A. H. (1993). A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In *Third Artificial Life Workshop*, pages 389–416, Santa Fe, New Mexico, USA.

Floreano, D. and Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *From Animals to Animats 3: Proceedings of Third Conference on Simulation of Adaptive Behavior*, Cambridge, MA. MIT Press/ Bradford Books.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, USA.

Friedberg, R. M. (1958). A learning machine: I. *IBM Journal of Research and Development*, 2(1):2–13.

Gao, P., McConaghy, T., and Gielen, G. (2008). Importance sampled circuit learning ensembles for robust analog ic design. In *Computer-Aided Design, International Conference on*, pages 396–399, Los Alamitos, CA, USA. IEEE Computer Society.

Garvie, M. and Thompson, A. (2004). Scrubbing away transients and jiggling around the permanent: Long survival of fpga systems through evolutionary self-repair. In *Proc. 10th IEEE Intl. On-Line Testing Symposium*, volume 2606 of *LNCS*, pages 155–160. IEEE Computer Society.

Giustina, A. and Manelli, F. (2001). *Growth Hormone and The Heart*. Springer.

Goldbeter, A. (1996). *Biochemical Oscillations and Cellular Rhythms: The Molecular Bases of Periodic and Chaotic Behaviour*. Cambridge University Press.

Gordon, T. G. W. (2005). *Exploiting Development to Enhance the Scalability of Hardware Evolution*. PhD thesis, University College London.

Greensted, A. and Tyrrell, A. (2003). Fault tolerance via endocrinologic based communication. In *LNCS Evolvable Systems: From Biology to Hardware*, 2606, pages 24–34, Trondheim. ICES.

Greensted, A. and Tyrrell, A. (2004). An endocrinologic-inspired hardware implementation of a multicellular system. In *2004 NASA/DoD Conference on Evolvable Hardware proceedings*, Seaprele.

Greensted, A. and Tyrrell, A. (2007a). Extrinsic evolvable hardware on the risa architecture. In *International Conference on Evolvable Systems proceedings*, Wuhan, China.

Greensted, A. and Tyrrell, A. (2007b). Risa: A hardware platform for evolutionary design. In *IEEE Workshop on Evolvable and Adaptive Hardware, 2007*, pages 1–7, Honolulu, HI. IEEE.

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

Gwaltney, D. A. and Ferguson, M. I. (2003). Intrinsic hardware evolution for the design and reconfiguration of analog speed controllers for a dc motor. In *5th NASA / DoD Workshop on Evolvable Hardware (EH 2003)*, pages 81–90, Chicago, IL, USA.

Haddow, P. and Hoye, J. (2009). Investigating the effect of regulatory decisions in a development model. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 293–300.

Haddow, P. and Tufte, G. (1999). Evolving a robot controller in hardware. In *In Proc. of the Norwegian Computer Science Conference (NIK-99)*, pages 141–150.

Haddow, P. and Tufte, G. (2000). An evolvable hardware fpga for adaptive hardware. In *In Congress on Evolutionary Computation (CEC00)*, pages 553–560.

Haddow, P. C. and Hoye, J. (2007). Achieving a simple development model for 3d shapes: Are chemicals necessary? In *Proc. of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1013–1020, New York, NY, USA. ACM.

Haddow, P. C. and Tufte, G. (2001). Bridging the genotype-phenotype mapping for digital fpgas. In *EH '01: Proceedings of the The 3rd NASA/DoD Workshop on Evolvable Hardware*, page 109, Washington, DC, USA. IEEE Computer Society.

Haddow, P. C., Tufte, G., and van Remortel, P. (2001). Shrinking the genotype: L-systems for EHW? In *ICES*, pages 128–139.

Harding, S. (2006). *Evolution In Materio.* PhD thesis, University of York.

Harding, S., Miller, J. F., and Banzhaf, W. (2009). Self modifying cartesian genetic programming: Parity. In *2009 IEEE Congress on Evolutionary Computation*, pages 285–292.

Harding, S. L., Miller, J. F., and Banzhaf, W. (2007). Self-modifying cartesian genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1021–1028, New York, NY, USA. ACM.

Harvey, I. and Thompson, A. (1997). Through the labyrinth evolution finds a way: A silicon ridge. In *Proc. 1st Int.Conf.on Evolvable Systems (ICES'96)*, volume 1259 of *LNCS*, pages 406–422. Springer-Verlag.

Hereford, J. and Kuyucu, T. (2006). Neural network with distributed nodes provides fault tolerance. *SPIE.org*.

Hereford, J. and Pruitt, C. (2004). Robust sensor systems using evolvable hardware. In *6th NASA / DoD Workshop on Evolvable Hardware (EH 2004)*, pages 161–168.

Higuchi, T. (1994). Evolvable hardware with genetic learning.

Hilder, J. A., Walker, J. A., and Tyrrell, A. M. (2009). Optimising variability tolerant standard cell libraries. In *CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, pages 2273–2280, Piscataway, NJ, USA. IEEE Press.

Holland, J. H. (1973). Genetic algorithms and the optimal allocation of trials. *SIAM J. of Computing*, 2:88–105.

Hollingworth, G., Smith, S., and Tyrrell, A. (2000). The intrinsic evolution of virtex devices through internet reconfigurable logic. In *3rd International Conference on Evolvable Systems: from Biology to Hardware*, pages 72–79, Edinburgh,. Springer-Verlag.

Hong, J.-H. and Cho, S.-B. (2003). Meh: modular evolvable hardware for designing complex circuits. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2003, 8 - 12 December 2003, Canberra, Australia*, pages 92–99. IEEE.

Hornby, G., Kraus, W. F., and Lohn, J. D. (2008). Evolving MEMS resonator designs for fabrication. In *Proceedings of the 8th International Conference Evolvable Systems: From Biology to Hardware, ICES 2008*, volume 5216 of *Lecture Notes in Computer Science*, pages 213–224, Prague, Czech Republic. Springer.

Hornby, G., Lohn, J., and Linden, D. (2007). Computer-automated evolution of an x-band antenna for nasas space technology 5 mission. *IEEE Transactions on Evolutionary Computation*.

Hornby, G. S. and Pollack, J. B. (2001). The advantages of generative grammatical encodings for physical design. In *In Congress on Evolutionary Computation*, pages 600–607. IEEE Press.

Hounsell, B. I. and Arslan, T. (2001). Evolutionary design and adaptation of digital filters within an embedded fault tolerant hardware platform. In *3rd NASA / DoD Workshop on Evolvable Hardware (EH 2001)*, pages 127–135.

Huelsbergen, L., Rietman, E. A., and Slous, R. (1999). Evolving oscillators in silico. In *IEEE Transactions on Evolutionary Computation*, volume 3, pages 197–204.

Imamura, K., Foster, J. A., and Krings, A. W. (2000). The test vector problem and limitations to evolving digital circuits. In *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*, page 75, Washington, DC, USA. IEEE Computer Society.

Jakobi, N. (1995). Harnessing morphogenesis. In *International Conference on Information Processing in Cells and Tissues*, pages 29–41.

Jeng, S.-L., Lu, J.-C., and Wang, K. (2007). A review of reliability research on nanotechnology. *Reliability, IEEE Transactions on*, 56(3):401–410.

Jin, Y. and Trommler, J. (2010). A fitness-independent evolvability measure for evolutionary developmental systems. In *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 69–76.

Kajitani, I., Hoshino, T., Kajihara, N., Iwata, M., and Higuchi, T. (1999). An evolvable hardware chip and its application as a multi-function prosthetic hand controller. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference*, pages 182–187, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Kalganova, T. (2000a). Bidirectional incremental evolution in extrinsic evolvable hardware. In Lohn, J., Stoica, A., and Keymeulen, D., editors, *The Second NASA/DoD workshop on Evolvable Hardware*, pages 65–74, Palo Alto, California. IEEE Computer Society.

Kalganova, T. (2000b). An extrinsic function-level evolvable hardware approach. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J. F., Nordin, P., and Fogarty, T. C.,

editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802, pages 60–75, Edinburgh. Springer-Verlag.

Kauffman, S. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467.

Kauffman, S. (1996). *At Home in the Universe: The Search for the Laws of Self-Organization and Complexity*. Oxford University Press.

Khan, G. M., Miller, J. F., and Halliday, D. M. (2008). Breaking the synaptic dogma: Evolving a neuro-inspired developmental network. In *SEAL '08: Proceedings of the 7th International Conference on Simulated Evolution and Learning*, pages 11–20, Berlin, Heidelberg. Springer-Verlag.

Kirschner, M. and Gerhart, J. (1998). Evolvability. volume 95, pages 8420–8427, Department of Cell Biology, Harvard Medical School, Boston, MA 02115, USA. marc@hms.harvard.edu.

Kitano, H. (1995). A simple model of neurogenesis and cell differentiation based on evolutionary large-scale chaos. *Artif. Life*, 2(1):79–99.

Kitano, H. (1998). Building complex systems using developmental process: An engineering approach. In *ICES '98: Proceedings of the Second International Conference on Evolvable Systems*, pages 218–229, London, UK. Springer-Verlag.

Knabe, J. F., Nehaniv, C. L., and Schilstra, M. J. (2008). Regulation of gene regulation - smooth binding with dynamic affinity affects evolvability. In *IEEE Congress on Evolutionary Computation (CEC 2008). Proc WCCI 2008*, pages 890–896. IEEE Press.

Knabe, J. F., Nehaniv, C. L., Schilstra, M. J., and Quick, T. (2006). Evolving biological clocks using genetic regulatory networks. In *Proceedings of the Artificial Life X Conference, Alife 10*, pages 15–21. MIT Press.

Kovel, J. (2002). *The Enemy of Nature: The End of Capitalism or the End of the World?* Zed Books.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, J. R. (1994). *Genetic programming II: automatic discovery of reusable programs*. MIT Press, Cambridge, MA, USA.

Koza, J. R., Bennett, III, F. H., Andre, D., and Keane, M. A. (1996). Automated wywiwyg design of both the topology and component values of electrical circuits using genetic programming. In *GECCO '96: Proceedings of the First Annual Conference on Genetic Programming*, pages 123–131, Cambridge, MA, USA. MIT Press.

Kramer, O. (2008). *Self-Adaptive Heuristics for Evolutionary Computation*, volume 147. Springer.

Krohling, R., Zhou, Y., and Tyrrell, A. (2003). Evolving fpga-based robot controllers using an evolutionary algorithm. In *1st international conference on Artificial Immune Systems*, Canterbury.

Kumar, S. and Bentley, P., editors (2003a). *On Growth, Form and Computers*. Elsevier Academic Press.

Kumar, S. and Bentley, P. J. (2003b). Biologically plausible evolutionary development. In *In Proc. of ICES 03, the 5th International Conference on Evolvable Systems: From Biology to Hardware*, pages 57–68.

Lala, P. K. (2001). *Self-checking and fault-tolerant digital design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Lambert, C., Kalganova, T., and Stomeo, E. (2006). Fpga-based systems for evolvable hardware. In *WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY*, volume 12.

Langeheine, J., Becker, J., Fölling, S., Meier, K., and Schemmel, J. (2001). A cmos fpta chip for intrinsic hardware evolution of analog electronic circuits. In *Proc.of the Third NASA/DOD Workshop on Evolvable Hardware*, pages 172–175, Long Beach, CA, USA. IEEE Computer Society Press.

Lawrence, P. A. (1992). *The making of a fly: The genetics of animal design.* London. Blackwell Scientific.

Leech, N. L. and Onwuegbuzie, A. J. (2002). A call for greater use of nonparametric statistics. Technical report, Educational Resources Information Center.

Leyser, O. and Day, S. (2003). *Mechanisms in Plant Development*. Blackwell.

Linden, D. S. (2001). A system for evolving antennas in-situ. In *EH '01: Proceedings of the The 3rd NASA/DoD Workshop on Evolvable Hardware*, page 249, Washington, DC, USA. IEEE Computer Society.

Linden, D. S. and Altshuler, E. E. (1999). Evolving wire antennas using genetic algorithms: A review. In *EH '99: Proceedings of the 1st NASA/DOD workshop on Evolvable Hardware*, page 225, Washington, DC, USA. IEEE Computer Society.

Lindenmayer, A. (1968). Mathematical models for cellular interactions in development. i. filaments with one-sided inputs. *Journal of Theoretical Biology*, pages 280–299.

Liu, H. (2007). *Biological Development model for the Design of Robust Digital System*. PhD thesis, University of York.

Liu, H., Miller, J. F., and Tyrrell, A. M. (2005). Intrinsic evolvable hardware implementation of a robust biological development model for digital systems. In *EH '05: Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*, pages 87–92, Washington, DC, USA. IEEE Computer Society.

Lones, M. and Tyrrell, A. (2001). Enzyme genetic programming. In et al, J. K., editor, *Proc. 2001 Congress on Evolutionary Computation*. IEEE Press.

Mattiussi, C. and Floreano, D. (2006). Analog genetic encoding for the evolution of circuits and networks. In *IEEE Transactions on Evolutionary Computation*.

McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of box plots. *American Statistician*, 32:12–16.

Metta, G., Sandini, G., and Konczak, J. (1999). A developmental approach to visually-guided reaching in artificial systems. *Neural Networks*, 12(10):1413–1427.

Miller, J. F. (2003). Evolving developmental programs for adaptation, morphogenesis, and self-repair. In *7th European Conference on Artificial Life*, pages 256–265. Springer LNAI.

Miller, J. F. (2004). Evolving a self-repairing, self-regulating, french flag organism. In *Genetic and Evolutionary Computation GECCO*, pages 129–139. Springer Berlin / Heidelberg.

Miller, J. F. and Downing, K. (2002). Evolution in materio: Looking beyond the silicon box. In *EH '02: Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02)*, page 167, Washington, DC, USA. IEEE Computer Society.

Miller, J. F., Job, D., and Vassilev, V. (2000). Principles in the evolutionary design of digital circuits - part i. *Genetic Programming and Evolvable Machines*, 1(1):8–35.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP'2000*, pages 121–132. Springer-Verlag.

Miller, J. F. and Thomson, P. (2003). A developmental method for growing graphs and circuits. In *Evolvable Systems: From Biology to Hardware, 5th International Conference*, pages 93–104.

Muller, S. (2002). *Bio-Inspired Optimization Algorithms for Engineering Applications*. Springer Verlag.

Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., and Higuchi, T. (1996). Hardware evolution at function level. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 62–71, London, UK. Springer-Verlag.

N., W. and A., R. (1946). The mathematical formulation of the problem of conduction of impulses in a network of connected excitable elements, specifically in cardiac muscle. *Archivos del Instituto de Cardiologa de Mxico*, 16:205–265.

Neumann, J. V. (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA.

Pearce, A. C., Senis, Y. A., Billadeau, D. D., Turner, M., Watson, S. P., and Vigorito, E. (2004). Vav1 and vav3 have critical but redundant roles in mediating platelet activation by collagen. *Biological Chemistry*, 279:53955–53962.

Periaux, J., Setriroui, M., and Mantel, B. (1995). Robust genetic algorithms for optimization problems in aerodynamic design. *Genetic Algorithms in Engineering and computer Science*, pages 251–270.

Poli, R. (1996). Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, B15 2TT, UK.

Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog.

Reil, T. (1999). Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny. In *ECAL '99: Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 457–466, London, UK. Springer-Verlag.

Roggen, D. (2005). *Multi-Cellular Reconfigurable Circuits: Evolution Morphogenesis and Learning*. PhD thesis, EPFL.

Sakanashi, H., Iwata, M., and Higuchi, T. (2001). A lossless compression method for halftone images using evolvable hardware. In *ICES '01: Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*, pages 314–326, London, UK. Springer-Verlag.

Samie, M., Dragffy, G., Popescu, A., Pipe, T., and Kiely, J. (2009). Prokaryotic bio-inspired system. In *Adaptive Hardware and Systems, NASA/ESA Conference on*, pages 171–178, Los Alamitos, CA, USA. IEEE Computer Society.

Sekanina, L. (2003). Virtual reconfigurable circuits for real-world applications of evolvable hardware. *Lecture Notes in Computer Science*, 2003(2606):186–197.

Sekanina, L. (2006). Evolutionary design of digital circuits: where are current limits? In *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, pages 171–178.

Shanthi, A. P., Muruganandam, P., and Parthasarathi, R. (2004). Enhancing the development based evolution of digital circuits. In *NASA/DoD Conference on Evolvable Hardware*, pages 91–94. NASA/DoD, IEEE.

Shiveley, R. (2006). Dual-core intel itanium 2 processors deliver unbeatable flexibility and performance to the enterprise. *Technology@IntelMagazine*.

Sims, K. (1994). Evolving 3d morphology and behavior by competition. *Artif. Life*, 1(4):353–372.

Skobtsov, Y. A., Ivanov, D. E., Skobtsov, V. Y., and Ubar, R. (2004). Evolutionary approach to the functional test generation for digital circuits. In *In Proc. of 9th Biennial Baltic Electronics Conf., BEC 2004*, pages 229–232, Tallinn Univ. of Techn.

Spector, L. and Stoffel, K. (1996). Automatic generation of adaptive programs. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4*, pages 476–483, Cape Code, USA. MIT Press.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127.

Stanley, K. O. and Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artif. Life*, 9(2):93–130.

Steiner, T., Jin, Y., and Sendhoff, B. (2008). A cellular model for the evolutionary development of lightweight material with an inner structure. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 851–858, New York, NY, USA. ACM.

Steiner, T., Schramm, L., Jin, Y., and Sendhoff, B. (2007). Emergence of feedback in articial gene regulatory networks. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 867–874. IEEE.

Stoffel, K. and Spector, L. (1996). High-performance, parallel, stack-based genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 224–229, Stanford University, CA, USA. MIT Press.

Stoica, A., Zebulum, R. S., Keymeulen, D., Tawel, R., Daud, T., and Thakoor, A. (2001). Reconfigurable vlsi architectures for evolvable hardware: From experimental field programmable transistor arrays to evolution-oriented chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):227–232.

Stomeo, E., Kalganova, T., and Lambert, C. (2006). Generalized disjunction decomposition for the evolution of programmable logic array structures. In *AHS '06: Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems*, pages 179–185, Washington, DC, USA. IEEE Computer Society.

Stomeo, E., Lambert, C., Lipnitsakya, N., and Yatskevich, Y. (2005). On evolution of relatively large combinational logic circuits. In *EH '05: Proceedings of the 2005*

NASA/DoD Conference on Evolvable Hardware, pages 59–66, Washington, DC, USA. IEEE Computer Society.

Strogatz, S. H. (1994). *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry, And Engineering (Studies in nonlinearity)*. Studies in nonlinearity. Perseus Books Group, 1 edition.

Sundaralingam, S. and Sharman, K. (1998). Evolving iir filters in multipath environments. In *Evolutionary Programming VII, 7th International Conference*, pages 397–406, San Diego, CA, USA.

Teerakittikul, P., Tempesti, G., and Tyrrell, A. (2009). The application of evolvable hardware to fault tolerant robot control. In *IEEE Symposium Series on Computational Intelligence*.

Teller, A. (1993). Learning mental models. In *Proceedings of the Fifth Workshop on Neural Networks: An International Conference on Computational Intelligence: Neural Networks, Fuzzy Systems, Evolutionary Programming, and Virtual Reality*.

Tempesti, G., Mange, D., Petraglio, E., Stauffer, A., and Thoma, Y. (2003). Developmental processes in silicon: An engineering perspective. In *EH '03: Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 255–264, Washington, DC, USA. IEEE Computer Society.

Tempesti, G., Mange, D., and Stauffer, A. (1999). The embryonics project: a machine made of artificial cells. *Rivisita di Biologia-Biology Forum*, 92:143–188.

Terry, M. A., Marcus, J., Farrell, M., Aggarwal, V., and O'Reillym, U.-M. (2006). Grace: Generative robust analog circuit design. In *Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2006: (EvoHOT), Lecture Notes in Computer Science*, pages 332–343. Springer Verlag.

Thompson, A. (1995a). Evolving electronic robot controllers that exploit hardware resources. In *Advances in Artificial Life: Proc. 3rd Eur. Conf. on Artificial Life (ECAL95)*, volume 929 of *LNAI*, pages 640–656. Springer-Verlag.

Thompson, A. (1995b). Evolving fault tolerant systems. pages 524–529, Sheffield, UK. IEE/IEEE, IEEE.

Thompson, A. (1996). Silicon evolution. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 444–452, Stanford University, CA, USA. MIT Press.

Thompson, A., Harvey, I., and Husbands, P. (1996). Unconstrained evolution and hard consequences. In Sanchez, E. and Tomassini, M., editors, *Towards Evolvable Hardware: The evolutionary engineering approach*, volume 1062 of *LNCS*, pages 136–165. Springer-Verlag.

Torresen, J. (1998). A divide-and-conquer approach to evolvable hardware. In *ICES '98: Proceedings of the Second International Conference on Evolvable Systems*, pages 57–65, London, UK. Springer-Verlag.

Torresen, J. (2002). A dynamic fitness function applied to improve the generalisation when evolving a signal processing hardware architecture. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, pages 267–279, London, UK. Springer-Verlag.

Torresen, J. (2003). Evolving multiplier circuits by training set and training vector partitioning. In *Proc. ICES'03:From biology to hardware*, volume 2606, pages 228–237. Springer-Verlag.

Trefzer, M. (2006). *Evolution of Transistor Circuits*. PhD thesis, Ruperto-Carola-University of Heidelberg.

Trefzer, M. A., Kuyucu, T., Miller, J. F., and Tyrrel, A. M. (2010). Image compression of natural images using artificial gene regulatory networks. In *GECCO'10*.

Tufte, G. (2008a). Discovery and investigation of inherent scalability in developmental genomes. In *8th International Conference on Evolvable Systems: From Biology to Hardware*, LNCS, pages 189–201. Springer.

Tufte, G. (2008b). Evolution, development and environment toward adaptation through phenotypic plasticity and exploitation of external information. In Seth Bullock (Chair), Jason Noble, R. W. M. B., editor, *Artificial Life XI (ALIFE XI)*. MIT Press.

Tufte, G. (2009). The discrete dynamics of developmental systems. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2209–2216.

Tufte, G. and Haddow, P. (2000). Evolving an adaptive digital filter. In *The Second NASA/DoD workshop on Evolvable Hardware*, pages 143–150, Palo Alto, California. IEEE Computer Society.

Tufte, G. and Haddow, P. (2003). Identification of functionality during development on a virtual sblock fpga. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, pages 731–738 Vol.1.

Turney, P. (1999). Increasing evolvability considered as a large-scale trend in evolution. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program (GECCO-99 Workshop on Evolvability)*, pages 43–46.

Tyrrell, A., Hollingworth, G., and Smith, S. (2001). Evolutionary strategies and intrinsic fault tolerance. In *Proceedings of the 3rd NASA/DOD Workshop on EHW*.

Tyrrell, A., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., Moreno, J., Rosenberg, J., and A.E.P. (2003). Poetic tissue: An integrated architecture for bio-inspired hardware. In *5th International Conference on Evolvable Systems*, pages 129–140, Trondheim.

Tyrrell, A. M. and Sun, H. (2006). A honeycomb development architecture for robust fault-tolerant design. In *AHS '06: Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems*, pages 281–287, Washington, DC, USA. IEEE Computer Society.

Vargha, A. and Delaney, H. D. (2000). A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132.

Vassilev, V. and Miller, J. (2000a). The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware*, pages 252–26. Springer.

Vassilev, V. K. and Miller, J. F. (2000b). Scalability problems of digital circuit evolution: Evolvability and efficient designs. In *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*, pages 55–64, Washington, DC, USA. IEEE Computer Society.

Walker, J. and Miller, J. (2004). Evolution and acquisition of modules in cartesian genetic programming. In *EuroGp*, volume 3003/2004, pages 187–197. Springer Berlin / Heidelberg.

Walker, J. and Miller, J. (2006). Embedded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, pages 911–918, Seattle, Washington. GECCO.

Walker, J. A. and Miller, J. F. (2007). Predicting prime numbers using cartesian genetic programming. In *Proceedings of 10th European Conference on Genetic Programming*, volume 4445/2007, pages 205–216. LNCS.

Walker, J. A., Miller, J. F., and Cavill, R. (2006). A multi-chromosome approach to standard and embedded cartesian genetic programming. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 903–910, New York, NY, USA. ACM.

Watson, R. A., Hornby, G. S., and Pollack, J. B. (1998). Modeling building-block interdependency. *Lecture Notes in Computer Science*, 1498:97–106.

White, D. R. and Poulding, S. (2009). A rigorous evaluation of crossover and mutation in genetic programming. In Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., and Ebner, M., editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 220–231, Tuebingen. Springer.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Champaign, IL, USA.

Wolpert, L., Beddington, R., Jessell, T. M., Lawrence, P., Meyerowitz, E. M., and Smith, J. (2002). *Principles of Development*. Oxford University Press.

Xie, H., Zhang, M., and Andreae, P. (2006). Automatic selection pressure control in genetic programming. In *6th International Conference on Intelligent System Design and Applications*, pages 435–440. IEEE.

Yao, X. and Higuchi, T. (1999). Promises and challenges of evolvable hardware. In *IEEE Transactions on Systems, Man and Cybernetics, Part C*, volume 29, pages 87–97.

Yasunaga, M., Yamaguchi, Y., Nakayama, H., Yoshihara, I., Koizumi, N., and Kim, J. H. (2008). The segmental-transmission-line: Its design and prototype evaluation. In *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg.

Zebulum, R. S., Pacheco, M. A., and Vellasco, M. (1998). Analog circuits evolution in extrinsic and intrinsic modes. In *ICES '98: Proceedings of the Second International Conference on Evolvable Systems*, pages 154–165, London, UK. Springer-Verlag.

Zebulum, R. S., Stoica, A., Keymeulen, D., Ferguson, M. I., Duong, V., Guo, X., and Vorperian, V. (2003). Automatic evolution of signal separators using reconfigurable hardware. In *Evolvable Systems: From Biology to Hardware, 5th International Conference*, pages 286–295, Trondheim, Norway. ICES.

Zhan, S., Miller, J. F., and Tyrrell, A. M. (2008). A developmental gene regulation network for constructing electronic circuits. In *8th International Conference on Evolvable Systems: From Biology to Hardware*, pages 177–188. Springer-Verlag.

Zhan, S., Miller, J. F., and Tyrrell, A. M. (2009). An evolutionary system using development and artificial genetic regulatory networks for electronic circuit design. *BioSystems*, 93(3):176–192.