EVOLVING GENE EXPRESSION TO RECONFIGURE ANALOGUE DEVICES

Kester Dean Clegg

Thesis submitted in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy.

THE UNIVERSITY of York

Department of Computer Science

May, 2008

Abstract

Repeated, morphological functionality, from limbs to leaves, is widespread in nature. Pattern formation in early embryo development has shed light on how and why the same genes are expressed in different locations or at different times. Practitioners working in evolutionary computation have long regarded nature's reuse of modular functionality with admiration. But repeating nature's trick has proven difficult. To date, no one has managed to evolve the design for a car, a house or a plane. Or indeed anything where the number of interdependent parts exposed to random mutation is large. It seems that while we can use evolutionary algorithms for search-based optimisation with great success, we cannot use them to tackle large, complex designs where functional reuse is essential.

This thesis argues that the modular functionality provided by gene reuse could play an important part in evolutionary computation being able to scale, and that by expressing subsets of genes in specific contexts, successive stages of phenotype configuration can be controlled by evolutionary search. We present a conceptual model of context-specific gene expression and show how a genome representation can hold many genes, only a few of which need be expressed in a solution. As genes are expressed in different contexts, their functional role in a solution changes. By allowing gene expression to discover phenotype solutions, evolutionary search can guide itself across multiple search domains.

The work here describes the design and implementation of a prototype system to demonstrates the above features and evolve genomes that are able to use gene expression to find and deploy solutions, permitting mechanisms of dynamic control to be discovered by evolutionary computation.

Acknowledgements

I gratefully acknowledge financial support from the EPSRC and Microsoft Research (UK). Additional equipment and software costs were met by pump-prime funding from the University of York for the York Centre for Complex Systems Analysis. Without these two sources of funding, this work would not have been possible.

A PhD is a long, and at times lonely, task. While every PhD student gives thanks for the many hours of supervision they receive, I feel exceptionally fortunate to have had Susan Stepney's inexhaustible energy and good humour driving me on. I cannot think of anyone else with whom I could have done a PhD, and maintained such a harmonious and productive relationship. It was a thoroughly enjoyable experience, and long may the cooperation continue!

Along the way, many people contributed ideas and suggestions. Particular thanks must go to Julian Miller, with whom I often discussed core elements of my work and whose belief in an academic way of life is one I admire. Others who helped with the disentangling of ideas include Tim Clarke, Dave Lovell (of Anadigm), James Walker, Simon Harding, Dave Chesmore, Cristina Costa Santini and Simon Poulding. My thanks to Tim Crosby for sanity checking some chapters, to John Clarke and George Tsoulas for their interest in my work and to Fiona Polack for her close reading of the text.

Further afield, I would like to express my thanks to Pauline Haddow for organising the Workshop on Development near Trondheim in 2008, which helped me write part of the conclusion to this work. I would also like to thank Michael Hübner for inviting me to Karlsruhe in 2008 to disseminate the results of our experiments, and Una-May O'Reilly of MIT for a helpful discussion on the capabilities of the Anadigm FPAA at GECCO 2007.

Away from the whiteboards, I must say an enormous, heartfelt thank you to my friends and family, whose support has been crucial to Cécile and myself in what have been some difficult years. Many of you helped more than you may realise, and more than we were able to let you know at the time.

The final word must of course, go to Cécile, whose unstinting support as my number one fan makes me thank my lucky stars each night that our paths crossed all those years ago.

Author's Declaration

This thesis is the work of Kester Dean Clegg and was carried out at the University of York. Work appearing here has appeared in print (in collaboration with other authors) as follows:

- Clegg, K., Stepney, S., Clarke, T.: Using feedback to regulate gene expression in a developmental control architecture. In Lipson, H., ed.: GECCO, ACM (2007), pg. 966–973, 19.
- Clegg, K., Stepney, S., Clarke, T.: Evolutionary search applied to reconfigurable analogue control. In: Field-Programmable Logic and Applications: FPL07, Amsterdam, IEEE Press (2007)
- Clegg, K., Stepney, S.: Analogue circuit control through gene expression. In press, accepted for EvoHOT 2008 (part of EuroGP 2008).

Chapter 2 was issued in its entirety as Department of Computer Science Technical Report YCS-2008-425, *An Introduction to Evolution for Computer Scientists* (Jan, 2008).[1]

The conceptual architecture described in Chapters 3 and 4 was first presented at the 4th European Conference on Intelligent Systems and Technologies (ECIT, Romania, 2006) and the results in Chapter 6 were first presented at a seminar for the Department of Electrical Engineering and Information Technology, University of Karlsruhe, Germany (Feb, 2008).

[1] http://www.cs.york.ac.uk/ftpdir/reports/YCS-2008-425.pdf.

Contents

Chapter 1

Outline of work presented

This thesis proposes that evidence from biology relating to the mechanisms of gene expression during early development could be useful to those working in evolutionary computation. It demonstrates how a device configured by evolutionary algorithms can be appropriately reconfigured as the environment changes. A mechanism is shown that links information about the search domain to the gene expression controlling reconfiguration. Evolution is thus not only able to discover a series of suitable solutions on the device, but also determine which solution should be deployed in which environment.

1.1 Research hypothesis and interpretation

The research hypothesis is explained in detail in Chapter 3. In this thesis, the process of gene expression during biological development is abstracted and used to guide evolutionary search. In our model, genes relate to configurable elements in an analogue circuit, while the configuration of a circuit represents a stage in phenotype configuration. In this context, we make the following hypothesis:

Hypothesis — *By expressing subsets of genes in specific contexts, successive stages of phenotype configuration can be determined by evolutionary search.*

Thus evolutionary search finds not only one solution, but several, and can control the movement between consecutive phenotype stages. Gene regulatory processes in biology are immensely complex. Any abstract representation of them inevitably conceals details. Only those parts pertinent to our needs have been included in our model.

Our approach has been motivated by trying to help in two areas of weakness in evolutionary computation, scalability and the reuse of modular func-

tionality. Genes are reused during the development of organisms, giving rise to a range of repeated, morphological features such leaves, limbs and hair. Gene reuse is one key to why biological systems scale so well. The developmental processes that lead to cell differentiation and pattern formation in embryos gives an indication of the importance that special proteins, known as transcription factors, play in determining which genes are expressed in which cells. We abstract some key features of biological gene expression to see how they might enable the reuse of genes in evolutionary computation. We examine a single "cell", with different gene expressions over time, responding to different environmental conditions.

Our application area is the adaptation of a Field Programmable Analogue Array (FPAA) to processing an analogue signal. As that signal changes, we allow reconfiguration to occur so that the signal can be processed differently. The mechanism for controlling reconfiguration comes from an evolved genome capable of expressing a subset of genes by examining the output of the FPAA. This "feedback loop" governs how our evolutionary search uses gene expression to explore a dynamic search domain.

1.2 Overview of Thesis Structure

This chapter has given the abstract of the thesis. The remaining chapters are summarised below.

Chapter Two is an introduction to evolution and evolutionary computation. Taking its lead from contemporary thinking in biology, it advocates that we have more to learn from the role that developmental processes play in natural evolutionary search than has previously been admitted by practitioners in evolutionary computation. A brief summary of evolutionary biology is given, from Darwin to Dawkins and beyond, alongside theoretical insights from authors such as Kauffman and Solé. The chapter then sketches the development of evolutionary computation as a field of computer science. It gives some background to how the "central dogma of biology" gained a canonical form as an algorithm and why decomposing tasks for evolutionary computation remains a difficult problem. It ends by indicating how gene reuse and context-dependent gene expression may provide one key to how evolution creates scalable solutions of such complexity.

Chapter Three re-examines the conclusions to Chapter 2 in the light of our conceptual architecture and uses them to formulate and interpret the research hypothesis.

Chapter Four moves from the conceptual architecture to a system architecture composed of hardware and software elements. Justifications for the choice of platform are given, and a clearer breakdown of the proposed processes to allow gene expression using information from the search domain.

Chapter Five covers the implementation of the system architecture. It looks at how our genome representation is able to contain several solutions, including a degree of redundancy that means some forms of the phenotype will not be expressed. The capabilities and difficulties of our hardware platform are discussed and the mechanism for binding an analogue signal to a digital genome is explained.

Chapter Six gives the preliminary results and attempts to deal with the problems of evolution and gene expression on noisy hardware. The main experiment is described. Examples of the solutions discovered and how they were deployed are discussed.

Chapter Seven concludes the work with a summary of what has been presented, why it is novel, a review of the key design decisions during the project and, with the benefit of hindsight, where the work could lead us.

Appendices A-C have additional material germane to the thesis.

Chapter 2

An Introduction to Evolution

This chapter gives some biological background to evolutionary computation. It traces the history of evolutionary biology, from Darwin to Dawkins and beyond, looking both at the gene reductionism of the 1970s–80s and more theoretical work modelling the complexities of evolutionary systems. The section concludes with a look at recent discoveries in developmental evolutionary biology, arguing that the field of evolutionary computation could benefit from a richer representation of the developmental process between genotype and phenotype. A brief summary of the development of evolutionary computation as a search technique is given, showing the canonical form of the evolutionary algorithm and noting troublesome aspects of the field, such problem representation / decomposition and the use of the technique in areas outside multi-objective optimisation. The chapter concludes with a discussion urging for a broader approach to tackling the problems of evolving complex, scalable solutions. The case is put that one way to do this would be to look at gene reuse, as employed in natural biological systems during development, and to see if an abstraction of the mechanisms used would help evolutionary search scale to larger, more complex problems.

2.1 Background

Evolutionary computation uses the process of natural selection as a search algorithm. Like evolution, the algorithm works over successive generations, gradually moving the search closer to the objectives until no more improvement in the population is possible, or the objectives are satisfied.

Evolutionary computation has a long academic and industrial record (§ 2.8.1). Within that time it has branched into variants (§2.8.1), developed a canonical form (§2.8.2) and been deployed in a wide range of industrial ap-

plications (§2.9.1). During the mid 1990s, evolutionary computation began to draw media attention with claims that human-competitive patents were being discovered through the use of evolutionary search techniques (Fonlupt, 2005; Koza et al., 2003). The techniques found industrial application wherever a design required taking a set of competing objectives into account. Examples from this period include Honda's "evolved" gas turbine fan blades (Jin, 2005), while NASA carried out experiments to evolve antennae and other aerospace hardware (Miller, 2000). More recently, claims in popular science journals, such as Peter Bentley's article in New Scientist, have fuelled expectations that evolutionary computation is poised to take over human design and that creativity can now be automated (Bentley, 2004a). The reality is more mundane. The achievements of evolutionary computation remain relatively modest. No one has evolved the design for a car, a house, or anything where the number of parts exposed to random mutation significantly adds to the complexity of the objective.

The problem is one of scale. Evolutionary computation has successfully been used as a search-based optimisation technique. Such optimisation generally involves a handful of factors in the fitness assessment. But as the number of elements sought by evolutionary search goes up, the time required to find a solution increases dramatically (§2.8.1). Where elements can affect one another, the search space size increase is exponential.[1] To get round this, the process is generally "bootstrapped" to the point where evolutionary algorithms optimise just a few variables on an existing solution, poor though that initial solution might be. But bootstrapping incurs a penalty — it constricts the search start point and therefore its trajectory. While we can trace the search trajectory of an evolved artifact, it is harder to estimate the place we should have started from to get a better result, particularly when the nature of the search space is unknown. Tackling these issues requires being well-versed in the art of problem representation and decomposition, an area that continues to cause difficulties (§2.9.2).

Claims that evolutionary computation is inspired by biological evolution must be tempered with the understanding that the representation and process bear only a token correspondence to those in nature. Although evolutionary computation employs a genetic encoding from which a population is generated and upon which selection is made, the step between the genetic encoding and selection of the phenotype is very small. In contrast, natural evolution has brought about a complex developmental process that plays a vital role in the exploration of the functional search space, and by extension, on phenotype

[1] Natural evolution deals with just such multi-dimensional search spaces, but of mind-boggling proportions, see §2.6.3.

selection. It is worth noting the historical legacy at work here, both from the study of evolution by biologists and the "borrowing" done by practitioners of evolutionary computation. The theory of evolution and natural selection, as developed by Charles Darwin (§2.3), was taken up and revised after the 1960s by those who believed that genes alone were the driving forces behind evolutionary change (see §2.3.2). This was the point at which evolutionary computation first settled on an established representation of genetic encoding and mutation (§2.8.1). But the genetic "reductionism" of the late 1970s was seen by some in biology as omitting the bigger picture (§2.3.3). They argued that evolution did not have a free hand to exploit genetic mutations. Instead, developmental processes constrain the degree and type of changes permitted in order to maintain the viability of the organism (discussed in §2.3.3 and §2.5.1). These restrictions have important implications on how natural evolution handles the combinatorial explosion of scale that has so far defeated proponents of evolutionary computation. However, despite the emerging evidence from biology on gene reuse and expression, and criticisms of the field's own shortcomings over the last decade (Banzhaf et al., 2006), evolutionary computation has failed to move away from evolutionary models founded in the 1960s and has largely ignored what phenotype development could bring to the process of evolutionary discovery.

During the 1990s, theoretical biologists began to investigate complexity and evolution by building models of how evolutionary paths were traversed in competitive ecologies (§2.6). The models indicated that rather than striving for continuous perfection, organisms are involved in an evolutionary arms race against other organisms. Those who stand still while the world evolves around them fail in the race for survival. Likewise those who stray too high up an evolutionary peak of specialist adaptation find their evolutionary paths are cut off when they need to adapt to new circumstances (§2.6.3). These theoretical models suggest that natural evolutionary systems are poised close to the edge of chaos, as small changes in one part frequently have knock on effects throughout the system (§2.6.3; page 46). This picture of evolution is closer to one where the system is held in balance and species expand into available niches as they appear, rather than one geared to isolated optimisations over an open landscape. Such theoretical evidence may give clues to how evolution works within highly connected systems and perhaps help evolutionary computation cope with the complexity generated when evolution can act on all parts of a system.

Although it is difficult to do justice to the breadth of developmental and evolutionary biology, the following sections attempt to show where the arguments in favour of a developmental approach to evolutionary computation

Figure 2.1: Insect camouflage: one mimics a shiny seed pod found on the forest floor (left), another mimics bird excrement while it feeds.
Images copyright of BBC Worldwide Ltd. (Attenborough, 1984)

come from, and in doing so try to give a flavour of the inspiration that natural evolution provides.

2.2 The Inspiration of Nature

Nature furnishes us with examples of organisms that have adapted to a bewildering variety of environments. Forms of life extend almost as far below the earth as they do above it. From bacteria thriving in complete darkness in hostile, boiling sulphur springs miles beneath the surface of the Pacific Ocean, to insects blown aloft in the oxygen-deprived, freezing temperatures of our upper atmosphere, there are few places that life has not managed to adapt to and survive. This general purpose, problem-solving ingenuity can be found almost anywhere life exists, but where there are abundant sources of food and water, such as equatorial rain forests, there is more opportunity for specialist adaptation. Camouflage is a vivid manifestation of adaptation to a particular habitat, and insects in particular draw inspiration for camouflage from almost anything in their environment. Examples can be found of beetles on the forest floor that have evolved to mimic fallen seeds pods (Fig. 2.1), while others mimic twigs or diseased leaves (Fig. 2.2). Caterpillars may mimic other, more poisonous caterpillars, or even bird excrement (Fig. 2.1). Birds also make use of their plumage and stance to render themselves invisible (Fig. 2.3).

The variety of evolved forms is one factor that allows life to solve the problem of existence within such a wide range of habitat (see Fig. 2.4). But variety alone would be insufficient to allow phyla to survive indefinitely. If environments change, whether by movement of the organism or from external factors,

Figure 2.2: Stick insects come in huge variety of shapes, but each subspecies is closely adapted to a particular species of flora.
Images copyright of BBC Worldwide Ltd. (Attenborough, 1984)

organisms must change with them. If a species fails to change, it may quickly die out. No organism knows how it will need to adapt for the future and only reproduction permits a species to survive.

In Europe at least until the mid-nineteenth century, what had enabled a species to arise in the first place had traditionally been ascribed to the creative power of God. There was no recognition that things ever changed from the point of creation onward. But a famous debate by the British Association for the Advancement of Science in Oxford in 1860 (Howard, 2001), gave birth to a new vision of how life created and continued to create new forms. Theologians lost the right to impose a single understanding of how life had been created, and although that debate is still engaged in some quarters, the explanation of how life has evolved has been almost universally attributed to the theory of natural selection developed by Charles Darwin.

Figure 2.3: The feathers of Brazilian potoo closely resemble both the colour and textures of bark and lichen, allowing it to mimic a tree stump when sitting motionless on its nest.
Images copyright of BBC Worldwide Ltd. (Attenborough, 1984)

2.3 Darwin's Theory of Natural Selection

Jonathon Howard has commented in his book on Darwin that anyone attempting a biography of the man is "faced with an embarrassment of riches" (Howard, 2001). But while a very full account of Darwin's life would be possible given the notes and records that have come down to us, the following section attempts to summarise in brief terms who Darwin was, with respect to his time and place, and what enabled him to formulate the most famous theory in biology, in his book "On The Origin of Species" (Darwin, 1859).

Grandson of the doctor and "speculative evolutionist" (Howard, 2001) Erasmus Darwin, Charles Darwin was born into a wealthy family, allowing him the opportunity to study what he wished. After giving up a brief career as a medical student in Edinburgh, Darwin took the retrospectively ironic step of moving to Cambridge to become an Anglican priest. But his interest in science developed and led him at the age of 22, thanks to his tutor's recommendation, to be accepted as the naturalist on board a five-year scientific voyage by the *HMS Beagle*. Darwin claimed that the "voyage of the Beagle has been by far the most important event in my life and has determined my whole career" (Darwin and Henry, 1974) and there is no doubt that the places that the Beagle visited, in particular some isolated islands, were to play a large part in shaping Darwin's thoughts about evolution.

Most biographers assert the principal influence on Darwin while aboard the Beagle was Charles Lyell's *Principles of Geology*. Prior to the theory of natural selection, the earliest battles fought by scientists against the scriptural dogma of the Church were fought by geologists:

Figure 2.4: Two specialist adaptations: the lichen beetle (left) exudes a glue so that it can stick pieces of lichen to itself, while a relative of the cockroach, *grylloblattodea* (right), scavenges high up above the snowline on the Himalayan mountains. It is so closely adapted to temperatures below freezing that the heat of your hand would kill it in a few minutes.
Images copyright of BBC Worldwide Ltd. (1984)

> "It was inevitable that a geological science which looked at the surface of the earth as a mobile and changing structure and part of a mobile and changing cosmos should eventually come into direct conflict with theological limitations on the development of science. Historical geology, with its emphasis on slow and continued processes, introduced a new and almost limitless timescale for the past evolution of the earth which recognised none of the miraculous and instantaneous events of the Mosaic creation story." (Howard, 2001)

Alongside evolutionary geology were the natural sciences that represented the prevailing scientific beliefs of the time, some of which were intermingled with religious sentiments regarding man's "place" within nature, and the widespread belief that there was a well-defined hierarchy of species or types, and that such types persisted unchanged through time. For example, as Howard points out, the concept of permanence among species type was closely correlated to the story of Genesis, where God created all living things on a single day and their types continued unchanged to the present day via reproduction.

But our ability to classify distinct types or species had already been questioned. Linnaeus had begun his *System Naturae* in 1735 with the conviction that a distinct categorisation was possible for all species, but ended his life doubting it was achievable (Howard, 2001). This wasn't the only messy problem that failed to fit with the vision of a divinely created natural system. Others more firmly attached to "natural theology", such as the Anglican cleric Malthus, sought in 1809 to justify why an ideal creation countenanced a system that al-

lowed a species population to exhaust its resources and suffer. Malthus argued that populations always increase geometrically where there is no competition for resources. What made Malthus different from previous justifications for nature's "unreasonableness" was that he declared that competition between species was a "law of nature" and that man was no different from any other creature with respect to competition. This was at least a shift from most classifications that painted man at the head of creation.

One can draw from early Victorian thinking to depict the background from which the theory of natural selection sprung, but whether such opinion would have had much sway on a passenger of the *Beagle* when it was several thousand miles from home is doubtful. What we do know is that Darwin took with him Lyell's work, and Lyell's second volume of the *Principles of Geology* reached Darwin midway through his voyage. In that volume, Lyell dealt with biological evolution and in particular, responded to a theory of evolution put forward by Jean Baptiste de Larmark in his book *Philosophie Zoologique* (1809). Lamark not only emphasised Linnaeus's doubts about the difficulties of classification, he noted the specialist adaptations of which nature was capable and presented a theory to explain such adaptations. Lamark claimed, without evidence, that the evolution of animals was propelled by their recognition of "new needs", which in turn provoked behavioural change to satisfy those needs, and this in turn caused structural change to make the behavioural change more efficient (Howard, 2001). This neat circularity was complete when the structural changes made in the creature's lifetime were inherited by its offspring. But while Lyell came up with his own ideas about biological evolution (later rejected by Darwin), his importance to Darwin was that he effectively dismissed Lamark's ideas as speculative, i.e. that the *mechanism* of evolution had not been proven, even by argument (Howard, 2001). It was Lyell's insistence that the mechanism required a scientific explanation that led Darwin to concentrate on this first, rather than why variations occurred between species, or the more established problem of species classification.

2.3.1 The three generalisations

The mechanism involving the flawed replication of DNA code through which inheritable variation operates was not found in Darwin's lifetime. However, the attributes manifested by the process were deduced by Darwin through observation and argument. Darwin drew up three independent generalisations which allowed him to argue for the theory of natural selection:

Variation: no two individuals are identical within a population.

Hereditary characteristics: the variation expressed as individual characteristics is inheritable from parents to offspring.

Multiplication and competition: Malthus's observation — about population increase where no competition for resources exists — means that as resources are always finite, competition must act as a brake on population growth.

If these processes were acting, then hereditary variations within a population that allowed an organism to survive would stand a greater chance of being passed on to their offspring. This was the method that explained how characteristic adaptations to the environment evolved within species and why species diverged.

However, geographical divergence between species was not entirely explained by natural selection. A new species isolated on an island or separated by a mountain range from a similar species presented an easy case, but Darwin found it harder to provide convincing arguments for why speciation still occurred where there was no geographical barrier. This area would continue to cause difficulty for Darwin in his later years, even to the extent of damaging his reputation when he published an account of it (his hypothesis of *panegenesis*) which veered dangerously close to Lamark in explaining the process of heredity. Richard Dawkins points out that this may have had its roots in 19th century views that heredity was a blending process. He comments:

> ... if heredity is of this blending type, it is almost impossible for Darwinian natural selection to work because the available variation is halved in every generation. Darwin knew this, and it worried him enough to drive him in the direction of Lamarckism. (Dawkins, 1998)

Despite the difficulties Darwin faced trying to explain heredity and variation, his theory of natural selection went on to become the dominant explanation of evolution which the discovery of DNA, almost a century later, would do little to change. Indeed, some would say the discovery has strengthened the theory of natural selection, even though in the words of Richard Dawkins, biologists now had to turn "from the fact of evolution to the less secure theory of its mechanism" (Dawkins, 1998).

2.3.2 Neo-Darwinism

> "The definition that I want comes from G.C. Williams. A gene is defined as any portion of chromosomal material that potentially

> last [sic] for enough generations to serve as a unit of natural selection." (Dawkins, 1989)

The solution to Darwin's problem over the blending nature of hereditary was published, unknown to him, by the German Gregor Mendel in 1865. Mendel's theory was that heredity was particulate, rather than blending in nature, so that parents pass on to their offspring discrete hereditary particles. The mechanism underlying this was not provided until the twentieth century, with the discovery that how particular genes are inherited from a parent — and genes are particulate in nature, either they are inherited or they are not. There is no half-way, partial inheritance. It is claimed (by neo-Darwinists) that this makes all the difference to the mathematical plausibility of the theory of natural selection. Dawkins states that Hardy and Weinberg were the first to realise:

> "there is no inherent tendency for genes to disappear from the gene pool. If they do disappear, it will be because of bad luck, or because of natural selection — because something about those genes influences the probability that individuals possessing them will survive and reproduce. The modern version of Darwinism, often called Neodarwinism, is based upon this insight." (Dawkins, 1998)

Dawkins was one of those who championed neo-Darwinism, and he summarised the modern genetic theory of natural selection as follows.

The genes of interbreeding animals constitute a gene pool. The genes compete, but in practice spend their time either sitting in individual bodies which they helped to build, or travelling from body to body via sperm or egg in the process of sexual reproduction. Sexual reproduction shuffles the genes, and it is in this sense that the long-term habitat of a gene is the gene pool. Any given gene originates in the gene pool as a result of a mutation, a random error in the gene-copying process. Once a mutation is formed, it can spread by means of sexual mixing provided that its carrier is able to sexually reproduce. Good carriers will contribute more to the gene pool than reproductively less successful ones. Any given gene in a gene pool is said to have a frequency, as it is likely to exist in the form of several copies, all descended from the original mutant. Some genes such as the albino gene are rare in the gene pool, others are common. At a genetic level, evolution may be defined as the process by which gene frequencies change in gene pools (adapted from (Dawkins, 1998)).

For Dawkins, although natural selection accounts for the "perfection of adaptation" in nature, it is of primary importance only because of its consequences for the survival of genes in the gene pool. If a gene is successful in

creating a good body that reproduces successfully, then it ensures its own survival. Dawkins (1989) developed this idea in great detail in his first book, *The Selfish Gene*. In it he suggested that in the world of the selfish gene ("what is a single selfish gene trying to do? It is trying to get more numerous in the gene pool") there is no individual altruism in the carriers of genes other than kin selection, that is to say, "a gene might be able to assist replicas of itself that are sitting in other bodies". The logical extension to this, Dawkin argues, is that almost all behaviour is genetically predetermined to aid the survival of the gene and nothing else. This he argues, explains all manner of bizarre behaviours:

> Mantises are large carnivorous insects...If the female gets the chance she will eat [the male]...It might seem most sensible for her to wait until copulation is over before she starts to eat him. But the loss of the head does not seem to throw the rest of the male's body off its sexual stride. Indeed, since the insect head is the seat of some inhibitory nerve centers, it is possible that the female improves the male's sexual performance by eating his head. (Dawkins, 1989)

Dawkin's one caveat to the selfish gene is that humans may be the one organism capable of resisting genetically determined behaviour through cultural and moral values.

Dawkin's ideas helped swing popular opinion towards a belief that genes explain "life", and persuaded many that the notion of competitive genes fitted perfectly into the theory of natural selection. But such genetic reductionism began to look less convincing as more became known about the developmental process of molecular biology and the constraints of the environment. It became apparent that genes do not have a free hand when it comes to altering features that could affect the viability of the organism. Conflicting genetic evidence also started to muddy the waters concerning the separate evolution of similar organs across species, such as the eye (see §2.3.2). But even outside these discoveries, there were there those who remained sceptical about the presence of minute variations in the gene pool explaining the eventual appearance of new species.[2] Brian Goodwin, one of those who sought to bring the generic complexity across nature (including developmental biology) back to the forefront of the evolutionary debate, commented a few years after *The Selfish Gene* was published that neo-Darwinism failed to explain "large scale aspects of evolution, including the origin of species":

> "New types of organisms simply appear upon the evolutionary scene, persist for various periods of time, and then become ex-

[2]When I refer to such scepticism, I am referring to those in the scientific community who wish to be convinced of better scientific arguments, rather than those who counter such arguments with religious bias.

> tinct. So Darwin's assumption that the tree of life [i.e. speciation] is a consequence of the gradual accumulation of small hereditary differences appears to be without significant support. Some other process is responsible for the emergent properties of life, those distinctive features that separate one group of organisms from another..." [3] (Goodwin, 1994)

Goodwin questioned whether the mechanisms of evolution could be reduced to the action of genes alone. Genetic mutation painted a convincing picture for small scale changes — the "fine tuning" of varieties — but not differences of *type*: fish from amphibian, worms from insects, or horsetails from grasses. Darwin's original difficulty to explain speciation began to rear its head again, despite the best efforts of neo-Darwinists to put the argument to bed. This is a broad topic and one we can summarise only a part of. The next section tries to indicate the nature of current debates over speciation and functional features, and how these debates now include evidence from diverse areas, such as developmental biology and gene regulation, and the development of similar functional features across the species divide.

2.3.3 The role of complexity in evolution

Fully developed forms of animals and plants, with their millions of eukaryotic cells acting in concert to provide much larger scale functionality, can seem a long way from the genetic code that translates a string of amino-acids to some proteins. The gap between the two is the part investigated by developmental biology.

Goodwin, in his book *How the Leopard Changed its Spots*, argued that genes are not the whole picture, and that complex organs have developed the way they are due to a robust, morphological process of development that was both helped and constrained by its environment (Goodwin, 1994). Goodwin's choice of the eye as one of his principal examples was deliberate. It was the organ that Darwin had gone to enormous effort to convince people that such "organs of extreme perfection and complication" could have developed by the infinitely gradual process of inheritable variation and natural selection, even though in his own words, it "seems I freely confess, absurd in the highest possible de-

[3] Goodwin and others, such as Gould (1989), may not have had access to Darwin's notebooks, particularly some of his pencil sketches, which give a less rigid idea of his tree of descent than is often portrayed. Darwin was perfectly aware that his tree of descent could not simply be an ever-growing tree of diversification, but instead was somewhat fragmented, more "like a piece of coral, with some parts dead and missing at the root, some parts alive and growing at the tips." (Howard, 2001)

gree".[4] It was his defence of the evolution of the eye that provided one of the more famous quotes in *The Origins of Species*:

> "If it could be demonstrated that any complex organ existed, which could not possibly have been formed by numerous, successive, slight modifications, my theory would absolutely break down. But I can find out no such case." (Darwin, 1859)

Darwin marshaled all manner of examples to suggest that primitive eyes were present in many organisms in different stages of evolutionary development, that nerve endings in many creatures were sensitive to light and that the "unerring skill of natural selection" would forever refine the simplest form of eye towards greater perfection.

By re-examining the evolution of the eye, Goodwin created a long-running debate between molecular, developmental and evolutionary biologists. Dawkins wrote a direct response to Goodwin from the neo-Darwinist perspective in Chapter 5 of his book *Climbing Mount Improbable* (Dawkins, 1996). In it he gives a picture by Mike Land that tries to represent the evolutionary landscape of all known eyes (Fig. 2.5). Dawkins argued that climbing an evolutionary peak of optical sophistication was easy enough for a species, but no species could jump from one mountain top to another. Others have pointed to the fact that the simple compound image-forming eye appears to have been invented at least three times during the course of evolution (D. Fogel 2000), and therefore types of eye couldn't be explained by a diagram such as Fig 2.5.

But it seems now as if neither Dawkins (for neo-Darwinists and gene reductionism) nor Goodwin (influence of developmental processes and morphological constraints) can claim an emphatic victory over the root causes of differences in species type and that the truth lies somewhere between the two. Russell Fernald, writing in (2001), sums up how the arguments swing this way, then that, almost as each new research paper is published:

> Have the structural similarities among eyes resulted from evolutionary convergence due to similar selective pressures (analogous) or from descent from a common ancestor (homologous)? This distinction is particularly hard to draw when comparing eyes because the physical laws governing light greatly restrict the construction of eyes. Similar eye structures may have arisen in unrelated animals simply because of constraints imposed by light. ... However...opsin has a significant DNA sequence homology across all phyla. Remarkably, recent work by Gehring and Ikeo (Gehring and

[4] To counter the scepticism he met, Darwin felt compelled to extend his arguments for the evolution of the eye between the first (1859) and sixth (1872) edition of the *The Origins of Species*.

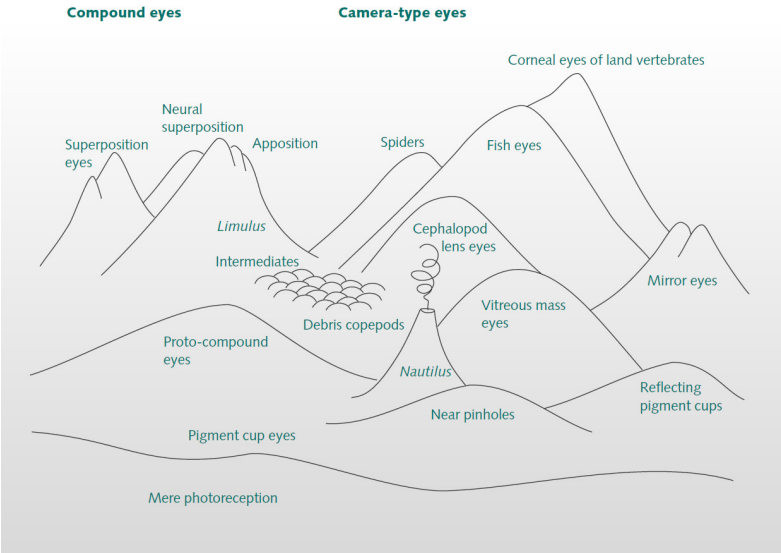


Figure 2.5: Possible landscape of eye evolution created by Mike Land. Height represents optical quality and the ground plane evolutionary distance. Land writes that "Climbing the hills is straightforward but going from one hilltop to another is near impossible" (Quoted by Ferald (2001)).
Image and text taken from Ferald (2001) (originally in Dawkins (1996))

> Ikeo, 1999) has shown that features of ocular development in different phyla can be coordinated by a homologous "master" gene, Pax-6. That a single gene could trigger construction of an animal's eye in diverse species led to their proposal that eyes are monophyletic, i.e. evolved only once. (Ferald, 2001)

What can be said about the ongoing debate on the evolution of complex organs is that developmental processes and environmental constraints seem to be playing a larger part in the discussion than in the early eighties, when they were largely discounted by those advocating gene reductionism. The following sections examine the physical development of the phenotype and assess why such developmental processes are important in their own right to the creation of complex organisms.

2.4 Aspects of Development

Development controls whether evolutionary designs can be built or not, and helps exploit the complexity of the physical world to allow life to be constructed. In evolutionary terms, genes play a vital role as the instigators of

change, but the viability of the organism during development restricts what mutations will be passed on in the phenotype. This section draws heavily from the introduction to Kumar and Bentley's book *On Growth, Form and Computers* (2003), which in turn takes much of its information from Wolpert's *The Principles of Development* (1998).

Wolpert describes development as "the emergence of organised structures from an initially very simple group of cells" (Wolpert, 1998). The process of moving from simple cells to more complex structures is almost entirely governed by proteins, but in order to understand how proteins influence development, we need to first become familiar with development at the level of the cell. This overview therefore starts with a high level view of cell development, then looks at the internals of cells, covering the synthesis of proteins within cells and cell signalling, before ending with a look at DNA and the role of proteins in gene regulation.

2.4.1 Cells

Cells are complicated. They have their own internal logic, they can act as sensors to respond to external signals and are able to emit signals that govern the behaviour of other cells. Cells and their proteins have resulted in a sophisticated control mechanism that not only dictates how development proceeds, but which also controls the running of bio-machinery after development.

Cells have two forms: *prokaryotic* (bacteria, including the important, large group of cyanobacteria, sometimes referred to as "blue green algae") and *eukaryotic* (everything else). The latter encases its DNA within a membrane, giving advantages in terms of control, defence and the ability to process information. Many regard the eukaryotic cell as the foremost achievement of evolution. Wolpert goes as far to claim that:

> "Once you had the eukaryotic cell, from the point of view of evolution and development, it was downhill all the way: very, very easy... Among the basic components required for development, I can think of virtually nothing that eukaryotic cells did not have which is required for the developmental process." (Wolpert, 2003)

Certainly when we look at the extraordinary information processing capabilities of the cell one can only marvel that the system scales so well.

Cell signalling

The cell container is a membrane (Fig. 2.6), a subtle discriminator over what passes through it — some proteins may enter, others can leave, the membrane

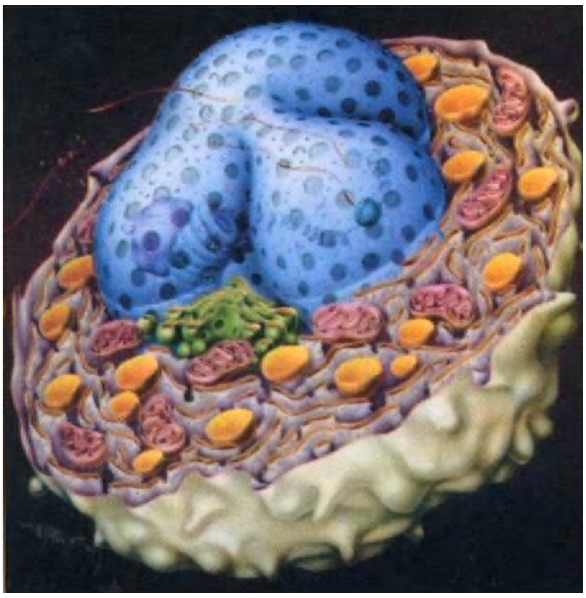


Figure 2.6: "Eukaryotic cells have an extensive array of membrane-bound compartments and organelles with up to 4 levels of nesting. The nucleus is a double membrane. The external membrane is less than 10% of the total."
Image and text taken from Cardelli (2005)

determines which. This *selective permeability* gives cells a filtering mechanism that can listen to broadcast messages as though tuned into a single hormonal frequency. Cells act as marvellously sensitive sensors and emitters of signals. They can operate in a wide range of media, from insect pheromone signalling in an air stream over miles of open space, to hormone signals carried in the blood stream in animals, from cell membrane surface proteins signalling to their immediate neighbours during development, to intra-cellular signalling for the presence of invading pathogens, cells can do it all. The mechanisms they use are equally complex and diverse (Hancock, 2003). Without cell signalling, it is hard to envisage how multicellular organisms, that are awash with information, could evolve, develop or exist at all.

The physical sequence of a cell signal generally follows the pattern of a molecule being released by one cell to be detected by the receptors on another cell, but there are variations on this, such as the detection of membrane proteins on one cell by receptors on another or the transfer of small molecules through "gap junctions".

Cell signalling plays an important part in development, particularly mechanisms such as juxtacrine signalling (surface to surface) in early development when the cells may be closely packed together (Wearing et al., 2000). But more

generally, without cell signalling an organism could not even begin to differentiate positional information to set up the axes of the body plan (see Speman's 'organiser experiment' [5] on amphibian embryos in Wolpert (1998)). Without cell signalling, only asymmetric distribution of transcription factors during cleavage could cause cells to become different from one another (see §2.5.2, also Kumar and Bentley, 2003). Signalling provides an important, if still poorly understood part of the context for early cell differentiation, laying the pathways for later developmental processes that require context-specific gene expression (§2.5.2–2.5.4).

Finally, binding a protein to a cell's receptor can trigger internal cell reactions that relay information back to the genome via *signal transduction* pathways. Such pathways can be viewed as signal cascades, sometimes involving many events that can be used by the cell as an amplification mechanism. Pathways are complex, as during the "cascade" there are opportunities for increased interaction or influence from other pathways, leading to a complex interplay of genes, proteins, even conflicting signals.

Cell division

Cells multiply by duplicating their contents and splitting in two. The cycle of cell division contains several phases: *interphase*, where DNA replication and the production of proteins occurs, *mitosis* or nuclear division, and *cytokinesis*, which concerns the division of a cell's cytoplasm after nuclear division (Kumar and Bentley, 2003).

Cell division is either symmetric or asymmetric. Symmetric division occurs when the plane of cleavage divides the cell into equal sizes with equal proportions of cytoplasmic proteins. Asymmetric division results in unequal sizes of daughter and parent cells containing different cytoplasmic factors. The different levels of cytoplasmic factors plays a large part in local regulation of gene expression in the embyro (§2.5.2).

2.4.2 Proteins

The broad behaviour range of eukaryotic cells is due to their interaction and production of proteins. With each cell containing several thousand proteins, the scope of potential functions a cell can achieve is huge. Proteins form not just the structural components of cells and tissues, but are involved in both signalling and the general "house keeping" of the cell, such as transporting

[5] Speman's early experiments involved lassoing a fertilised newt's egg with a baby's hair to force cell divisions to be on one side only. This led to the discovery that "grey crescent cells" initiate gastrulation of other cells.

or storing oxygen or haemoglobin molecules. Proteins are also involved in defensive mechanisms such as the production of antibodies.

In terms of their chemical structure, all proteins are polymers comprised from twenty different amino acids. These amino acids are joined by peptide bonds giving long *polypeptide* chains, hundreds or thousands of amino acids in length. The chains "fold", taking on distinctive three dimensional shapes which are critical to their function. Conceptually, proteins might be described as rather like a scrunched up ball of string, with the string itself being composed of up to 20 differently coloured segments, which can be repeated, and whose sequence order is specified by the order of nucleotides in the gene (see next section). The *spatial proximity* of the segments in their scrunched up ball largely determines the functionality of the protein. However understanding proteins in the real world is less easy than this simple, visual conceptualisation. 3D modelling software can help indicate the physical complexities of proteins by producing "ribbon" drawings (see Fig. 2.7), but understanding the modelled structures remains difficult, particularly when trying to determine a functional role from protein folding.

2.4.3 Genes code for proteins

The genome specifies when and where proteins are synthesised, and there are those who argue that genes have no function other than to specify proteins (Wolpert, 2003). As hinted at in the introductory section on proteins (§2.4.2), complicated networks of interactions involving proteins and genes are built up within cells. Proteins can promote or inhibit other proteins, and the absence or presence of certain proteins can affect the expression of a gene, which would in turn affect the production of another protein. These forms of "cascading" control sequences in protein production are termed gene regulatory networks and can be extremely complex. A minor industry has built up trying to infer such networks by analysing the massive amounts of data produced during the study of gene expression. It is important to understand that it is via gene regulatory networks and protein signalling pathways that physical feedback loops are possible during the development of the organism. Such feedback and control during development gives the organism a measure of flexibility in response to its environmental conditions.

Deoxyribonucleic Acid (DNA) and the translation process

In what would appear to be a case of massive redundancy, every cell in every living organism contains the unique instruction set for that organism's construction. That might seem a lot of information for each cell to contain, but

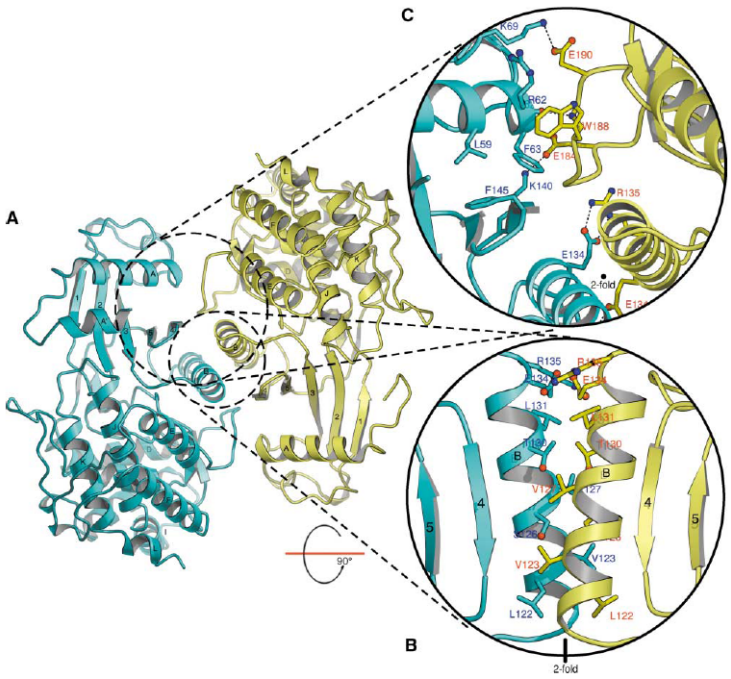


Figure 2.7: The structure of the Choline Kinase CKA-2 Dimer. (A) Shows a ribbon drawing of the CKA-2 dimer looking straight down the two-fold axis of symmetry. (B) and (C) show enlarged views of the residues involved in interactions between the two symmetry-related helices. The view in (B) is orthogonal to those in (A) and (C).

Image and text reproduced from Peisach et al. (2003)

the cell as it divides and multiplies will make use of just a tiny part of it. The cell must construct itself from proteins, and the rules for synthesising those proteins are contained in the DNA sequence that specifies the proteins for that cell.

The discovery of the structure of DNA by Crick and Watson in 1953 helped understand how genetic information copied itself. Rather like the polypeptide chains of amino acids that make up proteins, DNA contains two polynucleotide chains of nucleic acids, linked to form the two strands of a double helix. The nucleic acids are simpler than their amino acid counterparts in proteins and contain just four bases (usually represented by their first letters), two purines: adenine (A) and guanine (G), and two pyrimidines: thymine (T) and cytosine (C) (see Fig. 2.8). Along the two polynucleotide chains of the helix, purines and pyrimidines face inward and pair up in what are termed the complementary base pairings: G-C and A-T. [6] One consequence of complementary base pairings is that a single strand of DNA or RNA can act as a template for

[6] In RNA, thymine is replaced with uracil (U), giving a base pairing of A-U.

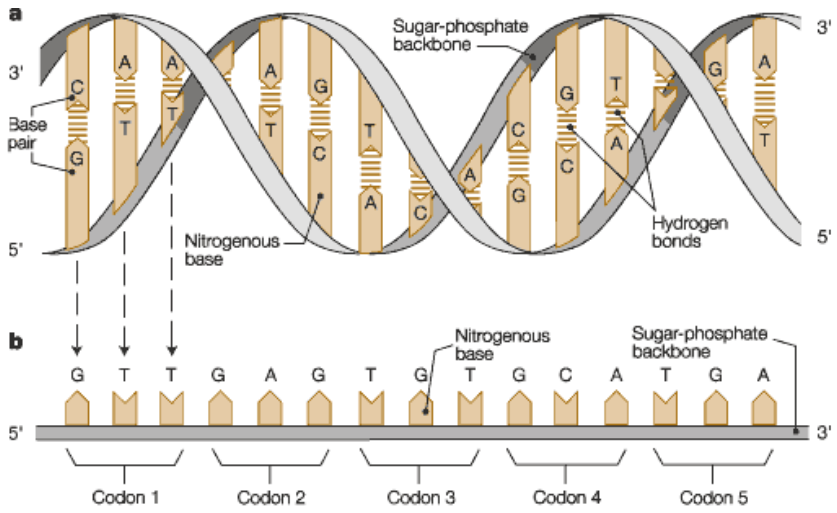


Figure 2.8: The double helix structure of DNA.
Image reproduced from Alberts et al. (2002)

the synthesis of its complementary strand, allowing nucleic acids the capability of directing their own replication (Kumar and Bentley, 2003).

In a section of the DNA, the order of the nucleic acid sequences governs the creation of amino acids that make up the protein. However, DNA is passive. It does not directly control the protein synthesis generated in the cell cytoplasm. That process is controlled by synthesised RNA, known as messenger RNA (mRNA), itself synthesised from the DNA template. In order to construct a protein, the nucleotide sequence in the mRNA is read three bases at a time, in what are termed nucleic triplets, or *codons*, with each codon corresponding to a single amino acid (Kumar and Bentley, 2003). This allows some redundancy, with some of the amino acids being encoded by more than one codon.

The translation of the codon into an amino acid is carried out by transfer RNA (tRNA) molecules, with at least one tRNA molecule being specific for an amino acid and a particular codon (although some amino acids may require the services of two or three different tRNAs). The amino acid is attached to the tRNA by an enzyme (aminoacyt-tRNA synthetase) which is again specific to that amino acid and its corresponding tRNA molecule. In a process similar to DNA replication, each kind of tRNA has a sequence of 3 unpaired nucleotides known as the *anticodon*, which bind into complementary base pairs in exactly the same way as the double helix strands in DNA, except this pairing is to the codon in the mRNA molecule.[7]

[7] Although I can't include it here, I can recommend John Kyrk's online animation of how the translation process constructs proteins: `http://www.johnkyrk.com/DNAtranslat.swf`.

Having covered the basic building blocks of protein transcription, we can now look at what restrictions development places on evolution, how developmental processes differentiate cells and how evolution is able to reuse functional genes in new locations.

2.5 Evolutionary Developmental Biology

Although the supporting evidence changes, evolutionary developmental biology is still tussling with the same question that Darwin hesitatingly put to his readers: how did organs "of extreme perfection and complication" such as the eye evolve?

> "One of the most important concepts in evolutionary developmental biology is that any developmental model for a structure must be able to account for the development of earlier forms in the ancestors." Carroll et al. (2001).

Over a hundred years after Darwin first posed the question, and even with our increased knowledge about the developmental process, being able to account for earlier ancestral forms presents a fascinating challenge and as we shall see in later sections, its answers may help computer scientists as much as biologists.

Evolution and development are closely interwoven. In §2.3.3 it was argued that the importance of developmental processes has begun to be given greater credence in evolution. Wolpert is not being provocative when he states that "DNA is rather boring and passive" (Wolpert, 2003), he is promoting the extent to which proteins are responsible for interpreting the complex information contained within DNA. While it is true that all changes of form and function are down to changes in DNA, as DNA dictates which proteins are made, change to DNA material does not imply that there is a "one-way flow of information, from DNA to proteins" (Banzhaf et al., 2006) that forms the basis of functional exploration by evolutionary search. Developmental processes also play an important role in saying which genes will be expressed in which places. Organisms evolve *into* environments, and this would be impossible unless the flow of information was two-way. As the environment into which an organism develops isn't a given constant, the developmental process needs constant feedback and gene regulatory networks cannot obtain information about their environment except via protein interaction. The presence of certain proteins (transcription factors) inhibits or promotes gene expression, so that genes encode not only for the proteins that build the biological infrastructure, but also for the proteins that control their own self-expression.

The ability of genes to encode for proteins that allow them to self-construct their own rules of operation is one of the more remarkable features of development. But there are restrictions on how those rules evolve to explore functional space, and the evidence for that comes from the field of evolutionary developmental biology.

2.5.1 The development of complexity

For evolutionary developmental biologists,[8] every structure has two histories that relate to how it developed: *ontogeny* (its complete development to maturity) and *phylogeny* (its evolutionary history). Wolpert states that, "ontogeny does not recapitulate phylogeny", as many embryos pass through common phases that their ancestors passed through (Wolpert, 2003, pg. 47).[9] For example, all vertebrates pass through a similar phylotypic stage involving the development of the nervous system (neurolation) and the formation of somites (body segments). This suggests that a distant ancestor of all vertebrates passed through this stage, and despite the stages before and after the phylotypic stage diverging in many species, neurolation has persisted to become a feature of all vertebrate development.

Sharing an embryonic stage provides evidence of common ancestors. An alternative source of evidence is to trace the alteration of structures present in ancestral forms in early embryonic stages. An example is the evolution of the branchial arches and clefts that are present in all vertebrate embryos, including humans. During evolution the branchial arches have produced both gills in primitive jawless fishes, and in a later modification, given rise to jaws. But Wolpert makes an important point:

> "These are not the relics of the gill arches and the gill slits of an adult fish-like ancestor, but of structures that would have been present *in the embryo* of the fish-like ancestor." Wolpert (2003) (my emphasis)

This has profound implications. It suggests that while all changes to form result from changes in the DNA, the changes are limited to *where* they can occur. Early embryonic stages appear to be robustly protected against change, perhaps because change here would be dangerous to the organism, but also because change appears to be easier once critical stages of development have passed.

[8] This section draws heavily from a single source (Wolpert, 2003).

[9] This is a common rebuttal of Ernst Haeckel's theory of recapitulation put forward in 1866, which claims that embryos pass through all their evolutionary stages. A full discussion of the debate (with references) can be found on Wikipedia, http://en.wikipedia.org/wiki/Ontogeny_and_phylogeny.

Striking evidence of the restriction imposed on gene mutations that control early development is the almost universal conservation of a group of genes called *Hox* gene clusters. Hox gene clusters control a wide range of developmental processes, such as limb bud or body plan development, and range across species from fruit flies to elephants. An important function of some Hox genes is to specify positional identity in the embroyo. These positional values are interpreted differently in different embryos, so that cells develop into, for example, segments and appendages (Wolpert, 2003). This means that the same genes expressed in a different location, time or context, may give rise to a different morphological form (Carroll et al., 2001). This aspect of reuse is examined in more detail in the following section, as it demonstrates how nature has invented a few, very useful genes, and re-used them widely across species. If we want evolutionary computation to mimic this trick, we need to discover what allows context-specific expression of a gene, and allow that context to in-part define the gene's functional role.

But the universal presence of Hox gene complexes should not be used as blanket evidence that such genes cannot mutate. Quite the reverse is true, but the manner they have mutated gives an insight into a key evolutionary mechanism. Gene duplication can occur in a variety of ways during DNA replication and provides the embryo with an additional copy of the gene. The beauty of this is that:

> "... this copy can diverge in its nucleotide sequence and acquire a new function and regulatory region, so changing its pattern of expression and downstream targets without depriving the organism of the function of the original gene." (Wolpert, 2003)

Haemoglobins (an oxygen carrier in red blood cells) in humans are an example of the evolution of new proteins and patterns of gene expression that have occurred by gene duplication. The duplication that gave rise to Hox genes means that they can be compared across a variety of species, allowing one to reconstruct how they are likely to have evolved from "a simple set of six genes in a common ancestor of all species" (Wolpert, 2003). Thus Hox genes are evidence on the one hand of conservation in development due to their widespread deployment, and on the other hand they indicate how successful genes that provide functional features can be both kept *and* changed at the same time. Both these aspects are pertinent to evolutionary computation, where there is a need to protect a good solution as part of solving a larger problem.

If gene mutation is limited to where it can occur, it suggests that functional complexity deriving from development may be hierarchical in nature, with change easiest at the "leaves" of the tree. The supporting evidence from evolu-

tionary biology is two-fold. Firstly, entirely new structures are rare, evolution tends to "tinker with existing structures":

> "New anatomical features usually arise from modification of an existing structure. A nice example is provided by the evolution of the mammalian middle ear. This is made up of three bones that transmit sound from the eardrum (the tympanic membrane) to the inner ear. In the reptilian ancestors of mammals, the joint between the skull and the lower jaw was between the quadrate bone of the skull and the articular bone of the lower jaw, which were also involved in transmitting sound. During mammalian evolution, the lower jaw became just one bone, the dentary, with the articular no longer attached to the lower jaw. By changes in their development, the articular and the quadrate bones in mammals were modified into two bones, the malleus and incus, whose function was now to transmit sound from the tympanic membrane to the inner ear." (Wolpert, 2003).

Although mammals and reptiles appear to have evolved separate mechanisms for hearing, the mechanism actually stems from a common structure. Secondly, comparisons of embryos suggests that those characteristics that are shared by a group of animals appear earliest in their evolution:

> "In the vertebrates, a good example of such a general characteristic would be the notochord (a skeletal rod of tissue enclosed by a firm sheath), which is common to all vertebrates and is also found in other chordate embryos. Paired appendages, such as limbs, which develop later, are special characters that are not found in other chordates and which differ in form among different vertebrates." (Wolpert, 2003)

The evolutionary record therefore provides us with some interesting evidence: shared characteristics occur earlier in evolution, and entirely new structures are, if not infeasible, at least rare. While the fossil record has long suggested these observations, fossil evidence is often patchy and its discovery down to chance. It is the more recent genetic evidence from embryology and developmental biology that has thrown light on the restrictions that evolution operates under. Although mutations may occur at random, developmental processes place restrictions on changes to the DNA, as any change must leave the organism viable. The scope of potential change is therefore narrowed to produce the "tinkering" effect described by Wolpert.

Evidence provided by Sean Carroll and his colleagues suggests that most changes are in the *cis*-regulatory region of the genes, rather than in the nature

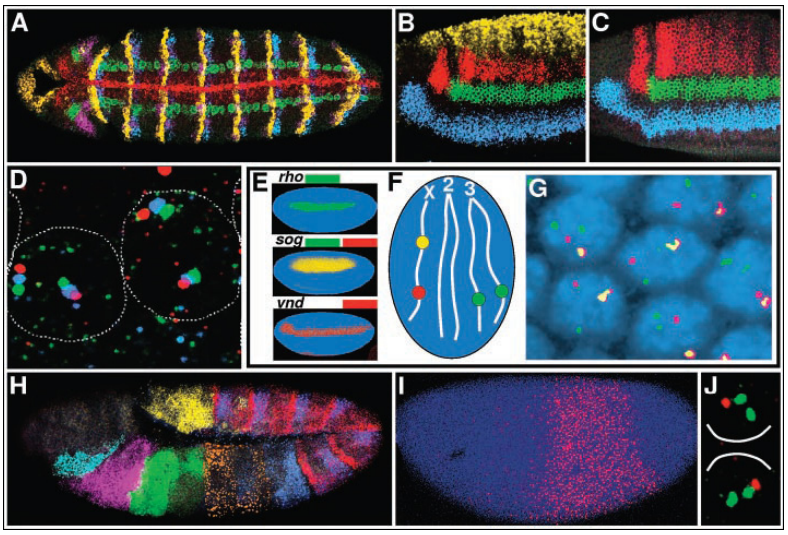


Figure 2.9: Drosophila embryos showing pattern and stripe formation.
Image reproduced from Kosman et al. (2004)

of the protein for which the genes encode (Carroll et al., 2001). The reason for this is that the *cis*-regulatory region is where protein factors — the transcription factors — bind and determine whether or not the gene will be transcribed. The subtle variations in morphological form, or "tinkering", is a by-product of this process, and its role in evolutionary development explained in the following section.

2.5.2 Evolutionary development

Work on developmental processes has uncovered much of what drives morphological variation in organisms. The literature is full of examples of drosophila embryos showing early body plan layout or stripe formation. These pictures are created by attaching fluorescent proteins to certain transcription factors (see Fig. 2.9) which enables biologists to see the distribution of such proteins across the embryo.

The individual contexts provided by the varying distribution of transcription factors allows the expression of repetitive morphological structures, such as backbone vertebrae or body segments. In the following sections we concentrate on how regulation works at the level of a single cell nucleus, rather than tackle the complexity caused by cascading networks of control, which is beyond a short introduction such as this.

As mentioned briefly in §2.5.1, the *cis*-regulatory regions or "switches" employed by gene regulatory networks (GRNs) determine the contexts in which

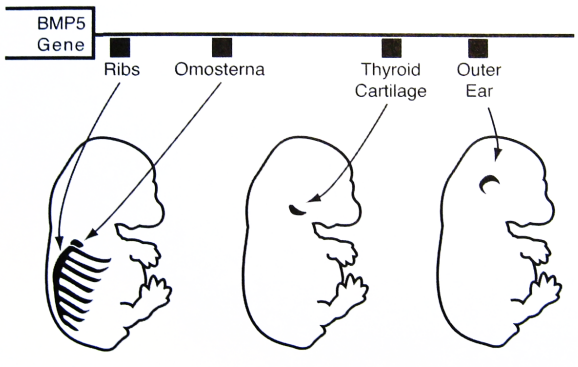


Figure 2.10: Different switches cause the gene for bone material protein 5 (BMP5) to be expressed in different locations in a mouse embryo.
Image taken (with permission) from Carroll (2006)

a particular gene is expressed or inhibited from transcribing proteins in the nucleus of a cell. These "switches" allow a gene to be re-used in a variety of contexts, which means that the protein a gene encodes will have the opportunity of interacting with different sets of proteins according to the cell's location in the embryo. For example, the different bones in our body are not created by different genes encoding separate proteins for particular bones, but by the same gene being used in different contexts to create the bone material protein for a rib, a sinus, an outer ear and so on (see Fig. 2.10).

Gene switches work by certain proteins being able to *bind* to small sections of DNA material upstream of where the gene is located. These transcription factors or "binding proteins" act on DNA to inhibit or promote gene expression. Whether a transcription factor is present or not in a particular cell type is determined by that cell's location in the embryo. For example, in Fig. 2.11, a promoter for a gene is distributed in vertical stripes that extend to the horizontal axis of an embryo. However, the presence of inhibiting transcription factors for the same gene in the lower third and back half of the embryo results in a net expression of the gene as a series of dots along the horizontal axis. Pattern formation is the basis of all gene reuse. The following section looks at some aspects of gene reuse that are of particular interest to evolutionary computation: context-specific functionality, and the mechanism that some extent determines the degree of gene reuse in an organism.

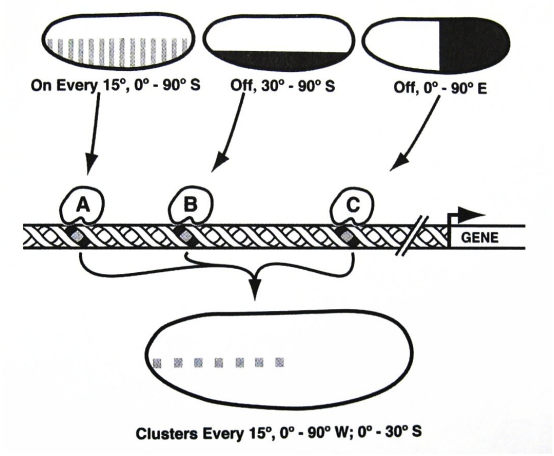


Figure 2.11: Gene switches acting on a drosophila embryo results in the gene being expressed as a series of dots extending halfway along the horizontal axis. *Image taken (with permission) from* Carroll (2006)

2.5.3 Gene expression and reuse

Research suggests that hardly any morphological features are created *de novo* from new genes (Carroll, 2006; Wolpert et al., 2002). Instead different morphological features between species are the result of the same genes (usually one of the four Hox clusters) being employed in different contexts. The Distal-less gene, essential for the formation of appendages, such as limbs or wings, is one example. Fig. 2.12 shows how in butterflies, this gene has evolved an additional "switch". The switch provides a new context for the Distal-less gene to be expressed — in this particular case, that location is on the wing. In the new context, rather than forming a limb bud, the gene results in an entirely different morphological feature: a spot of colour (Carroll, 2006).

The gene switch mechanism allows reusable, configurable instances of a gene to be expressed in the different contexts of embryo development. Repeated use of a gene in different contexts is called *modularity* by biologists, and gives rise to repetitive morphological structures such as vertebrate backbones, thorax segmentation, rib cages, leaf and wing venation, limbs, etc.. Such structures are common in nature, but it has taken researchers a long time to understand the link between switches and gene reuse. An important part of this interaction is how the switches work to allow the binding process some degree of flexibility.

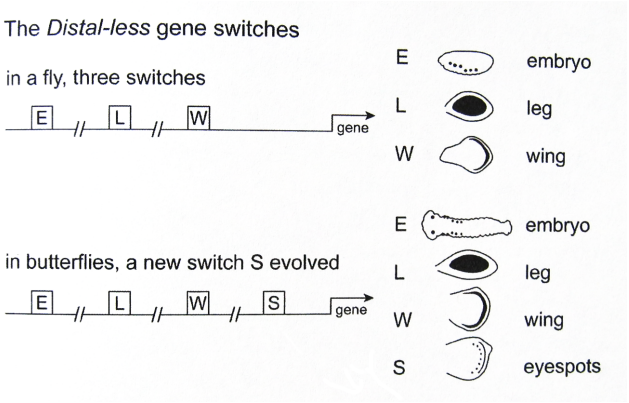


Figure 2.12: Switches in the Distal-less gene control expression in the embryo, larval legs and wing in both flies and butterflies, but butterflies evolved an additional switch providing a new context for Distal-less (giving wing eyespots — a different morphological feature).

Image and text taken (with permission) from Carroll (2006)

2.5.4 Binding signatures

Transcription factors attach to stretches of DNA by recognising signature sequences of base pairs. For example, a single switch for a gene may consist of several hundred base pairs (bp), lying perhaps several thousand bp upstream of the gene. Within the gene switch, there are usually 6–20 signature sequences (each ∼6–9 bp in length) that affect the expression of the gene concerned (a gene contains ∼1000 or so bp, and a chromosome contains thousands of genes, so millions of bp). Even a short signature length has a huge number of possible combinations (Carroll, 2006).

Signature sequences sometimes require exact matches for every position, sometimes they contain wildcards. Wildcard positions can be filled by all four nucleic acids (Cytosine, Thymine, Adenosine, and Guanine) but are more often limited to pairs of alternatives (e.g T or A, C or G, etc.). For example, Tinman, a gene related to heart development in most species, is highly specific. However, Pax-6 (the gene supposedly controlling the development of sight across species) and the gene Dorsal use wildcards in their binding signatures, represented by **K** (G or T), **Y** (C or T), **M** (C or A), **W**(...), etc. (example below is from Carroll (2006)):

```
Tinman           TCAAGTG
Pax-6(eyeless)   KKYMCGCWTSATKMNY
Dorsal           GGGWWWWCCM
```

Thus Pax-6 has a signature with only 6 specific sites out of 16 possible bp com-

binations, indicating that it could bind at a variety of locations. This is borne out by experimental evidence that shows eyes can be "grown" in other contexts — such as on wings or legs — by altering the transcription factors present at those locations (Carroll, 2006).

Binding signatures and proteins permit the genome to maintain a *set* of solutions from which it selects how to explore its functional domain. The action of "binding" is one of feedback: the information relayed by the presence of binding proteins in the cell nucleus feeds back to the DNA and determines which genes will be expressed in that context. But a by-product of exploring the functional search space in this way is that developmental processes have a fundamental impact on which genes are conserved. Their continued presence means that they are not only more likely to be reused by evolution during later developmental processes, but they also stand a greater chance of being available to evolutionary action such as gene duplication. Genes expressed in a particular location have the opportunity to construct morphological functions that will differ if the same gene is expressed elsewhere at another point in development. To summarise our look at evolutionary developmental biology, we can say that although all change is driven by changes to DNA, it is the developmental processes of construction that largely determine whether those changes can have an effect.

2.6 Models of Evolution and Complexity

So far we have taken examples from nature, looked at how theories of evolution have developed, and tried to investigate how the evidence provided by our greater understanding of genetics has influenced those theories. We have also looked at the process of construction, including the basic building blocks of life and combined them with a brief excursion into evolutionary developmental biology to see if they could shed some light on the evolution of complex features. However, it should be noted that not all the progress in this area has been done by practitioners gathering samples and amassing data from real life examples. Some important concepts have been developed by theoretical biologists, particularly those such as Kauffman (1995) and Solé and Goodwin (2000) working on abstract models of evolution and complexity.

One consequence of the restrictions that developmental processes impose on evolution's design is a "smoothing out" effect. Large jumps, as we saw in the theoretical landscape of all possible eyes, are not possible; small, incremental changes of direction are.

2.6.1 Fitness landscapes

The analogy of the landscape that represents peaks and troughs of evolutionary success within a population was first devised by the American geneticist Sewall Wright, who coined the phrase "adaptive landscape" to describe the shifting balance of population genetics (Wright, 1932). The adaptive landscape and its associated heuristics is an influential model in evolutionary biology, but its use is not universally accepted. It was strongly criticised by Wright's own biographer, Provine, who declared the heuristic was mathematically uninterpretable (as there appears to be no way of generating the continuous "surface" of the landscape) (Provine, 1986). But it has also been defended by Ruse (1996) and most recently by Skipper (2002) who, in a short survey on the influence of Wright's work, claims that the adaptive landscape diagram remains of use in the study of dynamic behaviours. Despite apparent weaknesses with the model, Wright's adaptive landscape has gone on to be extensively developed by Kauffman, Levin, Johnsen and others, in the investigation of what they term "adaptive walks" by organisms (Kauffman and Levin, 1987). They term their models "fitness landscapes".

The model is a simple one. An individual within a species is represented as a string of genes that defines its genotype. The string itself has a real number associated with it. This number defines the *fitness* of the string in terms of the phenotype it produces. The assessment of fitness as a single or two dimensional trait is one aspect of the model that has been criticised (see following paragraph). The distribution of fitness values over the space of all genotypes gives the *fitness landscape*, and all members of the population map onto that landscape according to their fitness value. If an individual has a high fitness value, it falls somewhere near a peak on the landscape; if it has a low fitness value, it is in a trough. A schematic fitness adaptive landscape is shown in Fig. 2.13, although most landscapes are considerably more rugged (and higher dimensional, see §2.6.3) than this, something that has important implications. Initially (at least conceptually), the population of phenotypes falls over the landscape with a random distribution, according to the fitness values given to them by their genes. However, after undergoing selection and mutation (more details of this are given in §2.8.1), individuals start to gain higher fitness values and start to "walk" towards the nearest peak. For the purpose of the model, the process of adaptation or improving a phenotype's fitness values is equivalent to walking up a peak.[10]

A crucial part of the model is the definition of the fitness landscape, as it

[10]One could select for negative values and this might be more apt, particularly for aspects of development as one could model "basins of attraction" (see D. Fogel (2000) and also §2.6.4).

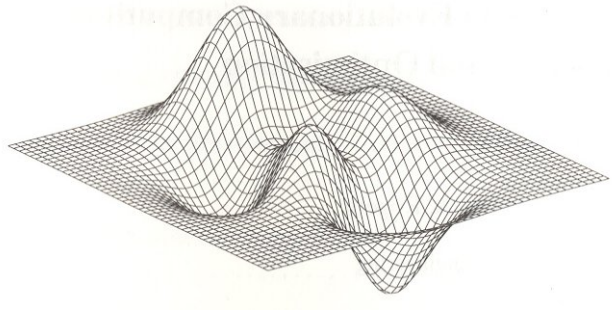


Figure 2.13: Schematic "adaptive" or "fitness" landscape.

is this which measures how the fitness of the population moves towards some optimal configuration. Although the model is intuitive to visualise for two fitness traits that can be isolated and easily quantified, the situation becomes more difficult as we try to represent complex, interrelated factors. But let us first examine the case of single fitness traits. In an imaginary world, I might have a creature that is predated on by an animal that can run fast. The environment therefore selects individuals who can run fast as these individuals have more chances of escape, by reproducing those individuals who are located higher up on the fitness peak that represents the ability to run fast and discarding others. Over time, the individuals who survive are all fast runners. At least that is the theory of the model. In reality, there is no evidence that evolution works on single traits. It may select for longer legs, faster muscles, and so on, but it could equally evolve the strategy of growing sharp horns, becoming poisonous or evolving a thick skin. Even in the former case, there might be many different ways of achieving the same fitness value as it is measured solely in terms of running speed. For the purpose of modelling selection based on a single trait the model is adequate, and some authors even go so far to claim that the model reflects reality:

> "Selection acts on collections of interactive phenotypic traits, not on singular traits in isolation. The appropriateness of an organism's holistic functional behaviour in light of the physics of its environment is the sole quality that is optimised through selection". (D. Fogel 2000)

D. Fogel thus suggests that providing one selects (or assesses) on the basis of an aggregated, holistic functional behaviour, the complex genetic relationships that cause the behaviour can safely be ignored. This perspective is one that is interested in the perceived "optimisation" of that behavioural trait.

The problem for fitness-landscape models is that we are forced to assess

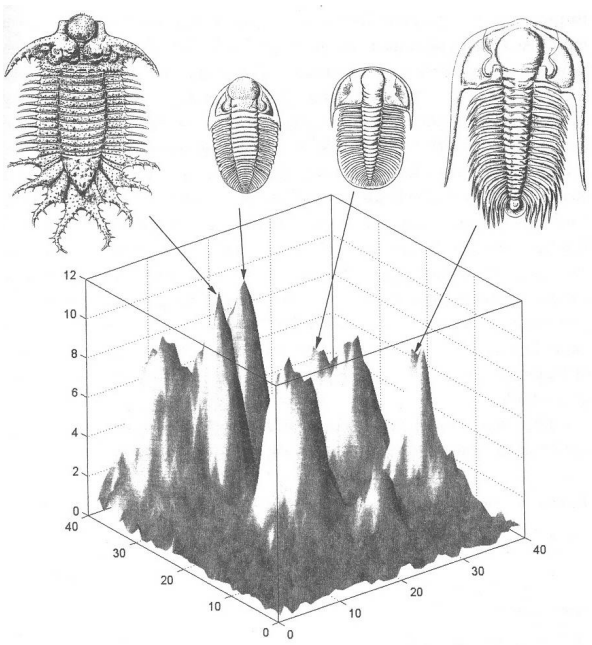


Figure 2.14: Imaginary rugged fitness landscape, showing optimal shape configurations of fossil tribolites.
Image reproduced from (Solé and Goodwin, 2000)

all fitness in terms of such amalgamated functional features, which in the real world are the complex expressions of genes and proteins. But as this would lead to a very difficult to visualise model, involving thousands of dimensions, fitness landscapes are instead generally plotted against just two dimensions representing two traits in the phenotype. Often, there is a sleight of hand at work here, in that we need to forget the multi-dimensional nature of the genotype space. Stuart Kauffman gives a flavour of this easy-to-imagine model:

> "Consider a set of all possible frogs, each with a different genotype. Locate each frog in a high-dimensional "genotype space", each next to all genotypes that differ from it by a single mutation. Imagine that you can measure the fitness of each frog. Graph the fitness of each frog as a height above that position in genotype space. The resulting heights form a fitness landscape over the genotype space, much as the Alps form a mountainous landscape over part of Europe". (Kauffman, 2000)

The appealing nature of such illustrations is shown in Fig. 2.14 from Solé and Goodwin (2000), where a fictitious landscape has been plotted based on fossil tribolites and fitness assessed in terms of shape. In any one set of conditions,

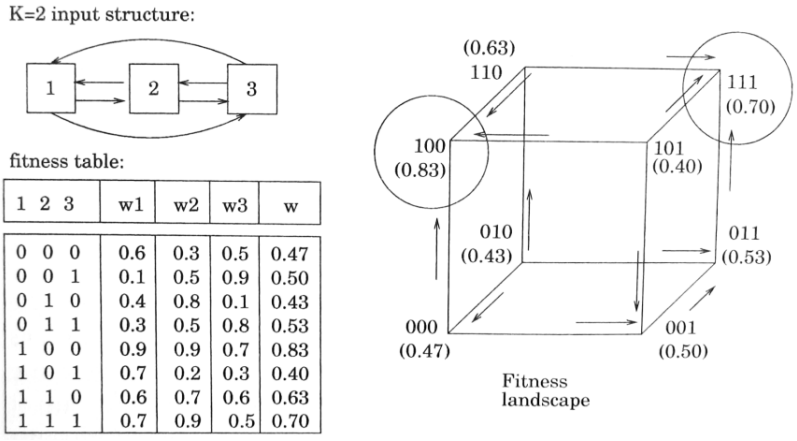
there are some optimal configurations for the organism, represented by the peaks in the fitness landscape. However, the extent to which this has any real meaning in nature is doubtful. Even Wright (1932) realised that such a representation hugely over-simplified the case:

> "... *accurately representing the population genetics of the evolutionary process requires thousands of dimensions. This is because the field of possible gene combinations in the field of gene frequencies of a population is vast.* ... Wright used the two dimensional graphical depiction of an adaptive landscape ... as a way of intuitively conveying what can only be realistically represented in thousands of dimensions. The surface of the landscape is typically understood as representing the joint gene frequencies of all genes in a population graded for adaptive value." (Wright's words italicised by me) (Skipper, 2002)

But Provine (1986) argues that Wright's original illustrations have no gradation along the axis or even any indication of what the units are (as they represent "genotype interpretation") and neither are there points along them to indicate where a gene combination is placed. Provine therefore claimed that there is no way of generating the continuous surface of an adaptive landscape. A second, more serious criticism by Gavrilets (1997), is that many gene combinations are incompatible, the number of which rises with the number of genes under consideration. Therefore the idea of representing gene combinations by a smooth continuous surface is itself specious; reality more closely resembles a landscape pock-marked with variously sized holes, where the holes indicate unachievable gene combinations. Others, notably Stadler (2002), have extended fitness landscapes with views similar to Gavrilets based on the impossibility of certain genotypes realising phenotypes due to developmental processes (in essence the same restrictions placed on evolution that were noted in §2.5.1). As a result of these issues, most biologists have abandoned Wright's original genotype interpretation in favour of one that assumes a population-based interpretation: "joint frequencies of all genes in a population graded for adaptive value" (Skipper, 2002). But even if one takes this model as realistic, the problems are not over for fitness landscapes.

2.6.2 The NK model

The difficulties with the genotype interpretation led to the development in the late 1980s of a fitness landscape that tried to tackle the interdependency between genes, known as *epistasis*. In Kauffman and Levin's well-known NK model (Kauffman and Levin, 1987), N represents the number of genes (and



1 2 3	w1	w2	w3	w
0 0 0	0.6	0.3	0.5	0.47
0 0 1	0.1	0.5	0.9	0.50
0 1 0	0.4	0.8	0.1	0.43
0 1 1	0.3	0.5	0.8	0.53
1 0 0	0.9	0.9	0.7	0.83
1 0 1	0.7	0.2	0.3	0.40
1 1 0	0.6	0.7	0.6	0.63
1 1 1	0.7	0.9	0.5	0.70

Figure 2.15: Building up fitness landscape. Each gene receives inputs from two other genes ($K = 2$) that affect the fitness contribution of the gene. Each gene in each of the $23 = 8$ possible genomes is randomly assigned a fitness contribution between 0 and 1. The fitness value of each genome is then computed as the mean value of the fitness contributions of the three genes. A fitness landscape is constructed as a Boolean hypercube. Circled vertices on the cube represent local optima, arrows represent "uphill" directions *(text from Kauffman (2000)).*
Image reproduced from Solé and Goodwin (2000)

therefore the dimension), while K indicates how many other genes influence any given gene, i.e. the K other genes are epistatic inputs to the fitness of the considered gene. If $K = 0$, so that no gene influences any other gene, then it results in a fitness landscape of only one peak with smooth sides (known as the Fujiyama landscape). But as K increases, the number of peaks on the landscape increases and the mean fitness of the nearest peak decreases toward that of an entirely random genotype (Skipper, 2002). Typically, interconnection results in a rugged fitness landscape. Genes (or traits) are represented as binary alleles, so that they are either expressed (1), or not (0). For computer scientists, the model starts to sound familiar:

> "the 2 to the N combinations of alleles of the N genes are therefore located on the vertices of the N-dimensional Boolean hypercube. The fitness of each type of organism, or vertex, is written on that vertex and can be thought of as a height. Hence the NK model creates a fitness landscape over the N-dimensional Boolean hypercube" (Kauffman, 2000).

The NK model and its statistical properties (i.e. the effects of changing values of N and K) have been widely explored (Solé and Goodwin, 2000).

As before, a species evolves by "adaptive walks". Essentially this means that we can choose a given trait, mutate the bit and then examine the fitness

table. If the average fitness of the resulting configuration is higher, an adaptive walk has occurred and the species moves in the landscape (it starts to climb). As already mentioned, when $K = 0$ the system is disconnected and there is a single global optimum. But when K is the theoretical maximum, $N - 1$, the system is entirely interconnected and every gene influences every other gene. Kauffman explored the generic properties of interconnected landscapes by assigning random fitness values across each of the allele states affecting a given gene (i.e. alleles of other genes whose expression affects the gene you are looking at). The fitness value of a specific allele at each of the N genes is then the average of the fitness contributions of the other N genes, yielding a random fitness landscape over the N dimensional hypercube (see Fig. 2.15). These random, highly interconnected landscapes yield interesting properties:

> "A main feature of random landscapes is that there are nearly exponentially many local peaks, indeed the number of local peaks is 2 to the $N/(N + 1)$. For $N = 1000$, there are 10^{297} local peaks on the landscape. Finding the global peak by hill climbing is improbable, and the system becomes trapped on a local peak. Other features include the lengths of walks via fitter neighbours to nearby peaks, which scales as the logarithm of N, and the way directions uphill dwindle on walks uphill. At each step uphill, the fraction of directions uphill is cut in half, yielding exponential slowing in the rate of finding fitter variants." (Kauffman, 2000)

These features of highly interconnected, rugged landscapes are crucial to understanding the nature of genotype search space. Because the landscape is big and interconnected, finding a global optimum becomes not just improbable, but of dubious value even as a strategy. The odds are stacked against the organism. Furthermore, the fact that the rate of improving fitness slows exponentially with each uphill step, and that the system gets trapped on local optima, correlates to the earlier suggestion by Wolpert that evolution merely "tinkers" with existing structures. Big jumps are not possible. The reason you are forced to tinker with the edges, making only incremental movements in any direction, is because each step uphill seriously restricts the other directions you can move in. If an organism wants to stay flexible in a dynamic environment, it can't afford to get trapped on a local peak of specialist perfection having abandoned its options for adaptive movement. Unfortunately organisms have no way of knowing whether their adaptive movements may strand them or keep them in the race of poor, but flexible competitors.

2.6.3 Coevolution

Natural systems are much more complicated than single genotype populations. In a Malthusian world of limited resources, everything is fighting for survival. For cheetahs to stay alive and reproduce, the species must keep up with the gazelles who are constantly evolving to outrun the fastest cheetah. Organisms not only affect each other's environment, they compete in an evolutionary race against other species. Losing means extinction. The idea of organisms competing merely to "stay in the race" was first put forward by Van Valen (1973) and is known as the *Red Queen hypothesis*.[11]

Solé and Goodwin (2000) explain the hypothesis thus: based on the fossil record, Van Valen observed that a species may become extinct at any time, regardless of how long it had previously existed. But if evolution is a process of constant improvement, why are modern species as equally likely to disappear as their ancestors were? Van Valen's hypothesis suggests that if continual improvement were the case, we would expect to see a decreasing probability of extinction the longer a species had existed. Instead, the fossil record shows the probability remains constant. That constant probability can only mean that continuous improvement is not possible for any species.[12] Van Valen claims that this means species are compelled instead to continuously adapt to each other's changes. Rather than continuous improvement, we have continuous re-adjustment. And despite natural selection doing its best to improve your chances of genetic survival, you might find you can no longer hill-climb as well as you could because someone else is affecting your ability to do that. Thus you can drop out of the race at any moment, and according to Van Valen, that would probably be the moment you failed to adapt to someone else's advantageous change. So we have yet another route to extinction. But this time, rather than getting trapped on a local peak due to your own adaptive movements, you get trapped because *the landscape moves faster than you do*.

Modelling adaptive landscapes takes on a whole new level of complexity when competing species are able to affect the fitness criteria of the genotype search space. Kauffman uses a simple, fictitious model of a frog and fly in evolutionary competition (see Fig. 2.16)

> "Each of the N genes in the frog receives inputs from K genes in the frog and C genes in the fly, and vice-versa. Thus, the sticky tongue of the frog affects the fitness of the fly via the presence or

[11] The name of the hypothesis comes from the Red Queen in Lewis Carroll's *Alice Through the Looking Glass*, in which she explains to Alice "Here, you see, it takes all the running *you* can do, to keep in the same place" Solé and Goodwin (2000).

[12] As pointed out in the previous section, the evidence suggests that continuous improvement is actually dangerous to the continued existence of a species.

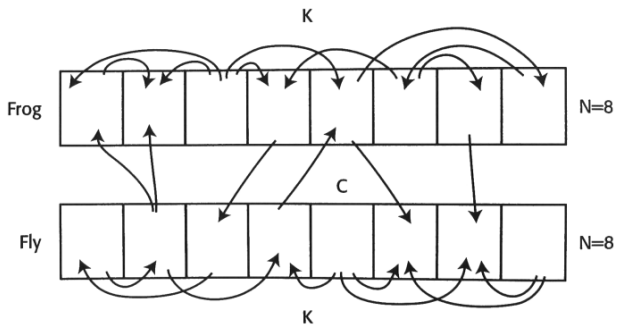


Figure 2.16: Interconnected genes affecting two coevolving species where K represents epistatic influence, C represents the degree of epistatic coupling between species.

Image reproduced from Kauffman (2000)

> absence in the fly of slippery feet, sticky stuff dissolver, or a strong sense of smell for sticky frog tongues ... Now when the frog population moves by mutation and selection uphill on the frog landscape, those moves distort the fly's landscape and vice-versa. Coevolution is a game of coupled, deforming landscapes." Kauffman (2000)

The NKC model developed by Kauffman and Johnsen (1991) introduces the new parameter C to represent the coupling between species. Kauffman and Johnsen claim that these models "generally behave in two regimes: an ordered regime and a chaotic regime, separated by phase transition" (Kauffman, 2000). Solé and Goodwin describe these regimes as a) low-K, ordered or frozen, where species settle on local optima, and b) high-K, chaotic or Red Queen, where the ecosystem is in constant flux. They also comment that the system appears finely, if not critically balanced:

> "At the boundary between these regimes, species in a finite system reach local peaks, but any small perturbation generates a coevolutionary avalanche of changes through the system. The distribution of these avalanches follows a power law, as expected for a critical state." (Solé and Goodwin, 2000)

Such changes are usually interpreted as extinction events. Kauffman and Johnsen (1991) mapped these avalanches to extinction events in the fossil record and although initially unsuccessful in finding a correlation, once the model was adapted to allow connections between species themselves to co-evolve, a correlation was found (Kauffman, 1995). Kauffman concluded that as avalanches of extinction events can propagate across species, it appears that species survive

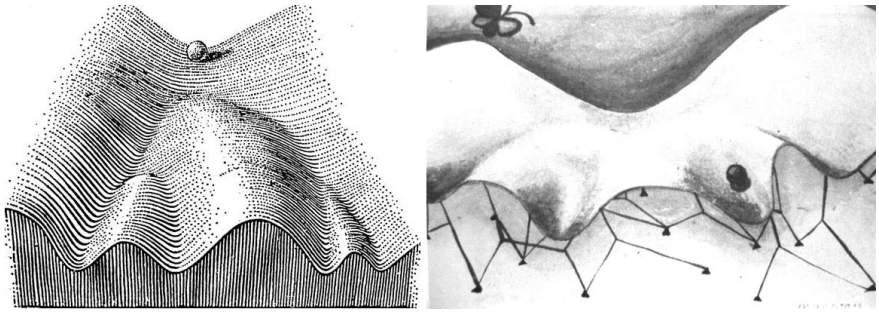


Figure 2.17: Waddington's epigenetic landscapes: his original drawing on the left showing developmental pathways (Waddington, 1957) and a later alternative showing epigenetic influences on the same landscape. Genes pull guy ropes attached to the landscape, deforming it and fixing the path of the ball, but a slight alteration in the genotype will not significantly change the final state, due to the stability (*homeostasis*) of development.
Right-hand image reproduced from Saunders (1993)

by niching on local optima and thus protecting themselves against too much "evolutionary competition" from other species. This niching, in a highly coupled, adaptive landscape is akin to each species "tuning" the ruggedness of its landscape (i.e. managing its interconnectedness) so that it retains both a degree of independence from the adaptive movements of other species and the ability to make its own adaptive moves. Both Solé and Kauffman further claim that by tuning their own landscapes, species poise the entire system as close to the critical boundary as possible. That boundary line is *the edge of chaos* between the two system states, low-K and high-K, as described above. The knack of maintaining the system near that edge of chaos is termed "self-organised criticality".[13] This is an interesting property of interconnected evolutionary systems, and perhaps one that those in evolutionary computation should bear in mind as they attempt to scale their models.

2.6.4 Deforming landscapes of development

The NK and NKC models of adaptive landscapes are not the only theoretical models that use the metaphor of deforming landscapes. Waddington developed a model in the 1950s of an epigenetic landscape that uses a slightly different metaphor to explain stability during development. In Waddington's model, rather than adaptive walks over a fitness landscape, the image is one of a ball rolling down the hills and valleys of a landscape of potential developmental paths, as shown in Fig. 2.17. The hills and valleys are created by the competing influence of genes (the genes were later shown as pulling on

[13] Per Bak (1996) was largely responsible for developing ideas around self-organised criticality.

guy ropes that are attached both to each other *and* to the surface of the landscape with different degrees of force). Waddington's model does not suffer the problems of some gene combinations being impossible, as we are looking at the developmental process. Instead, unrealisable developmental paths (due to the viability of the organism) are represented by peaks in the landscape that deflect the path the ball can take. The smooth continuous surface is the result of gene expression during development, itself an interconnected system of guy ropes where many genes can affect the influence of a specific gene's tension on the landscape. Waddington was fascinated by the ability of the system to return to a stable state even after being perturbed by environmental or genetic effects, a property he termed *homeostasis*. He also used the word *canalization* to describe the property that development can typically proceed to one or more of a restricted number of alternative end states, rather than to a broad spectrum (Saunders, 1993). Waddington's point was that a system, especially a dynamic, non-linear system such as an organism, is unlikely to have stability in the traditional sense of a single point equilibrium. Waddington sought to emphasise that dynamic, nonlinear systems had a richer notion of stability, one closer to a *path* or *trajectory*, which could be returned to if travel along it was deflected at some point.

The importance of stability is crucial in determining the viability of the organism and although this acts as a restriction on evolutionary change, it also brings benefits. For example, it is through developmental stability that great genetic variation can be supported in a population of nearly identical phenotypes (Saunders, 1993). The model also has explanatory value when we want to view how large evolutionary changes might affect the developmental process. From Waddington's model, we can see that, even if the influence (i.e. the tension on the guy rope) of a gene is increased dramatically, change is mitigated by the opposing tensions from other genes' guy ropes that are attached to it. Thus for one change to have a large impact, it would have to affect many other genes with a similar degree of force, and make them somehow conspire to work together to allow a large alteration to affect the shape of the landscape. Large changes to the outcome of developmental processes are thus both difficult and unlikely by random mutation.

2.7 Concluding Remarks on Biological Evolution

The invention of nature appears almost limitless. Nevertheless, evolutionary adaptation is tightly constrained by natural selection and the viability of the organism throughout both its evolutionary and developmental history. The in-

terplay of so many aspects in evolution makes the process difficult to model. From complex developmental processes that show little evidence of tampering with the early embryonic stages, to theoretical models showing the interplay of epigenetic forces through to evidence from the avalanches of extinction events in the fossil records, no single model can capture it all. The best we can hope for, it seems, is to take as much evidence as we can from natural and theoretical biology when hoping to understand evolution in its broadest sense, and investigate how the mechanics of the process tune and govern themselves.

If there were a single criticism of the field of evolutionary computing, it would be that too little evidence from biology has been used during adoption of the evolutionary paradigm. While no one would wish to try to replicate the intricacy of biology, evolutionary computation has historically taken a very narrow interpretation of evolution, one that is predominantly based on models of fitness landscapes. These models are highly abstract and perhaps rather sterile as a consequence. There is little opportunity for the rich, complex interactions we see in real biological processes to take place. For example, very little work has demonstrated the role developmental processes play in evolutionary search, despite evidence from biology suggesting its fundamental importance.

In the following sections we briefly examine the history of evolutionary computation, taking in the major trends in the field and examining where the current research effort is focused. We see how some of the early successes of evolutionary computation were in part responsible for the direction of later work. We look at interesting results on the evolution of logic circuits in hardware, which suggest that physical complexity or "richness" is a quality we could exploit using evolutionary algorithms, and end by looking at some models that have shifted their emphasis from evolutionary search towards development.

2.8 Evolutionary Computation

Towards the end of the twentieth century, biotechnology increasingly made headline news. There was a growing awareness, even hysteria, about the extent to which genes determined many aspects of our lives. Genes were discovered, it was disturbingly claimed, for homosexuality, schizophrenia, even criminality (Hutcheon, 1996). Other spinoffs from biotechnology, such DNA "fingerprinting" have become commonplace and large scale. Publicly funded research such as the Human Genome Project[14] kept biotechnology in the public eye. The famous double helix even features on a British sterling two pound

[14] http://www.genome.gov/ and http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml

coin minted specially for the fiftieth anniversary of the discovery of DNA.

Biotechnology's rise to fame and its increasing influence on people's lives is perhaps matched only by that of computer science over the same period. As biologists have become increasingly reliant on computers, it was inevitable that the latest findings and theories in biology would feed back into computer science research. Just a decade after Crick and Watson's discovery of the structure of DNA, computer scientists were already investigating what ideas could be taken from biology to use in their own field (L. Fogel, 1963). But although the borrowing has been somewhat piecemeal to date, the trend to adopt ideas from biology shows signs of becoming even stronger in the coming years. For some, such as Peter Bentley writing the New Scientist in 2004, the future of computer science will be inextricably linked to paradigms of biological processes:

> "You could say we are going back to nature. I am convinced that in the future, software will evolve and grow instead of being designed and built. In place of programmers there will be digital horticulturalists who plant, prune and grow software from seeds that they have cultured. Not a single line of code will ever be typed into a computer again." (Bentley, 2004a)

Whatever the truth to such grandiose claims, links between computer science and biology are gaining strength. One could point to areas such as neural networks and immune systems as examples of this, but perhaps the greatest recipient of bio-inspired ideas is the area covered by evolutionary computation.

2.8.1 A brief history of evolutionary computation

The earliest attempts at simulating evolution were linked to machine learning. Turing (1950) suggested how an evolutionary or genetic search might be used in general machine learning, while Friedman (1956) speculated on the use of feedback, selection and mutation to design "thinking machines" (an idea which has raised its head again recently, with the work of Bongard and Lipson (2004)). There were others too, perhaps less closely tied to academia, such Friedburg (1958) (who suggested "a population-based hill climbing search") and Box (1957). Box's work is interesting in that it dates the involvement of industrial systems control engineering to the earliest days of evolutionary computation. While never purely an engineering discipline, the practical side of evolutionary computation has remained influential and been partly responsible for the direction and narrowness of later research (see §2.9.1). However, despite these early pioneers, evolutionary computation as a field didn't really develop until the early 1960s, when several branches appeared independently.

Evolutionary programming

Evolutionary programming was part of the attempt to create artificial intelligence. L. Fogel (1962) used finite state machines and simulated evolution on a population of contending algorithms to demonstrate intelligent behaviour. The machine in question had to predict an input symbol and its prediction was an output based on previous experienced input symbols. L. Fogel exposed a population of machines to the learning environment of input symbols and selected machines on fitness criteria. The selected machines were then randomly mutated and the process repeated with their offspring.

It is worth noting that L. Fogel presented his work initially in an industrial research journal and it would be fair to say received mixed reviews. While some were positive, others such as Solomonoff (1966) were critical of the inefficiency of random, hill-climbing searches. But the criticisms stemmed from comparisons with other artificial intelligence research, rather than an objective look at the potential of the method for its own sake. More recent commentary on the early work in automated programming by Lenat (1983) is equally critical, but perhaps unfairly so given the considerable benefit of hindsight:

> "... early (1958–1970) researchers in automatic programming were confident that they could succeed by having programs randomly mutate into desired new ones. This hypothesis was simple, elegant, aesthetic and incorrect. The amount of time necessary to synthesise or modify a program was seen to increase exponentially with its length. Switching to a higher level language ... merely chipped away somewhat at the exponent, without muffling the combinatorial nature of the process. All the attempts to get programs to 'evolve' failed miserably, casualties of the combinatorial explosion." Lenat (1983)

D. Fogel (2000) argues that claims about the amount of time to evolve a solution increasing exponentially with its length are unsubstantiated in the literature. However, the "combinatorial explosion" of the total search space is a well-documented issue with other branches of evolutionary computation, particularly genetic programming, but also more generally when the representation length is increased or given greater complexity. Practitioners still advocate throwing more computing power at this problem, much as they did in the 1960s (discussed in §2.9).

Evolutionary strategies

Evolutionary strategies developed as general function optimisation algorithms to solve difficult real-valued parameter optimisation problems. The work was started in the mid-1960s at the Technical University of Berlin by Rechenburg (1963; 1964; 1973) and Schwefel (1975; 1977; 1981). Evolutionary strategies had some noteworthy features. For example, "the components of a trial solution are viewed as behavioural traits of an individual, not as genes along a chromosome" (D. Fogel, 2000). Although a genetic source for phenotypic traits is assumed, the nature of that linkage is not made explicit. The genetic transformations result in behaviour changes that follow a Gaussian distribution, allowing many phenotypic characteristics to change following a genetic alteration.

Another interesting feature was the self-adapting strategy parameters, enabling the degree of mutation of a parent to change dynamically and for the parameter to be mutated and undergo evolution itself. This work bears comparison with the more recent dynamic parameter encoding in genetic algorithms (Schraudolph and Belew, 1992). D. Fogel (2000) claims that "strong similarities exist between evolution strategies and evolutionary programming ... In many cases, the procedures are virtually equivalent even though they developed independently". More recent work on evolutionary strategies can be found in Voigt *et al* (1996).

Genetic programming

Genetic programming was extensively developed by Koza as a means to automate programming, but some of its greatest successes have been in the field of machine generated analogue circuit designs (Fonlupt, 2005; Koza, 1992, 1994; Koza et al., 1999, 2003). An individual in genetic programming is a computer program rather than a chromosome. Each program is evaluated by being run and a fitness is then assigned to it (although this may be over multiple runs with different inputs).

Programs themselves are usually represented as parse tree structures, with subtree nodes acting as the points on which mutation or recombination occurs. For example, in recombination, two different subtrees in the same node position might be swapped between parent trees, or if using a mutation operator, a node might be selected and the subtree replaced by a randomly generated subtree. Other variants of mutation exist.

Like other branches of evolutionary computation, genetic programming has been successful in industrial applications (Koza et al., 2004), particularly with respect to human-competitive solutions in analogue electrical circuit design, some of which have been patented (Streeter et al., 2003). Unfortunately

the physical construction of analogue circuits is slow, and has meant that the evolutionary design process and fitness evaluations in Koza's work have run as software simulations. The simulations use a specially modified version of the freely available SPICE program. Even with this version of the program (which is not freely available), Koza requires large populations running over many generations, and despite his success in discovering patentable designs, the simulations need immense computing power.[15]

However, a big problem with genetic programming has historically been the issue of "bloat". Operators can grow large structures that have no effect or that are wasteful. This feature of genetic programming has been likened to "junk DNA", in that while it doesn't alter the semantics of the program it represents junk code that is either unused or wasteful. There are many papers suggesting ways to tackle bloat (Brameier and Banzhaf, 2003; Fernández et al., 2004; Langdon, 2000; Langdon and Banzhaf, 2000; Langdon and Poli, 1997; Tomassini et al., 2004; Vanneschi, 2004). Some good papers theorising about the shapes of parse trees and the causes of bloat have been recently been published by Diada *et al* (2004; 2003; 2005), in particular an analysis of the visual form of evolved tree structures which has lead to the hypothesis that evolved trees are inherently "deep and narrow rather than wide" due to the numbers of nodes on deep subtrees leading to them being more likely to be selected. Variants of GP have appeared over the years, including Linear GP (Banzhaf, 1998; Brameier, 2003), Grammatical Evolution (O'Neill and Ryan, 2003) and others, with some designed explicitly to avoid the tree structure of traditional GP. Cartesian Genetic Programming (Miller, 2001) is one such example and as this form of genetic programming has particular relevance to this thesis, we now give a detailed summary of the development and variants of this technique.

Cartesian Genetic Programming (CGP)

CGP appeared in an early form as an evolutionary algorithm, in a paper discussing the evolution of arithmetic circuits (Miller et al., 1997). It used a "linear chromosome of cell functionalities and connectivities based on a rectangular array of logic cells". From a later perspective, particularly given Miller's developmental work on cells (Miller, 2004), it is interesting to note that Miller contemplated the relationship between cell connectivity and evolvablilty, and wondered whether "a concept of cell-neighbourhood" would help the evolutionary process, something he would later go on to investigate.

[15]Stanford offered Koza a thousand parallel node Beowulf cluster with 1/2 teraflops capacity *(interview with EvoNet, 14 Aug, 1998)*.

The name CGP first appeared in 1999, when Miller compared the performance of his "linear integer chromosomes in the form of connections and functionalities" (Miller, 1999) with GP and Evolutionary Programming (EP) (Koza, 1994). Miller's results demonstrated that CGP was more efficient at learning Boolean functions than either GP or EP.[16] The Cartesian part of the name came about because the method considered a grid of nodes addressed in a Cartesian co-ordinate system. CGP was finally described explicitly in Miller and Thomson (2000). Since then several variants have appeared, including an embedded form that allows the acquisition of modular functionality (Walker and Miller, 2007a,b) and a self-modifying version (Harding et al., 2007). Claims of improved performance against CGP were made by Lones (2003), but this work has not been replicated and Lones's results were refuted (along with others that had used results from CGP using restricted genome lengths) in a paper comparing the performance of CGP against a range of other methods (Walker and Miller, 2008).

Genetic algorithms

Genetic algorithms (GA) were largely developed by John Holland and his students from the 1960s onwards (1962; 1992),[17] with theoretical work being added by Goldberg (1989; 2002) and Vose (1999a; 1999b). Holland's original motivation was to "understand the principles of adaptive systems" (Dimutrescu et al., 2000) and in common with other branches of evolutionary computation, the early papers presented the process of evolution in a highly abstract form, so that key elements of the simplified process could be identified and understood in terms of what made the process effective as a search algorithm.

GAs are generally comprised of a population of candidate solutions encoded as chromosomes in a binary string representation.[18] The process is simple: take the best members from the candidate population of solutions and use those to form your next generation of solutions by combining them with randomly chosen individuals or each other. Assess your new population and repeat. Natural selection ensures successive generations move the population towards your fitness objectives. The canonical form of GAs is given in the following section.

Exactly how each successive generation should be formed soon became a major topic of debate, with many forms of crossover between individuals be-

[16] This was not difficult in the case of GP, as the method was known to suffer "bloat" and required large populations over many generations. Miller compares GP bloat and CGP in Miller (2001).

[17] D. Fogel (2000) also cites Bremermann (1962; 1966) and Fraser (1957; 1968) as among the early developers.

[18] Binary representation has declined in recent years, but the canonical form is generally given as a fixed length binary string representation. See §2.8.2.

ing tried. A large part of the debate focused on improving the performance of GAs. Running a simulation over many generations containing large populations was computationally expensive and early workers in the field frequently struggled with limited computing power (D. Fogel, 1998). This, combined with the pressure to achieve practical results on engineering problems, meant it became the dominant area of research.

The GA is considered to be the main paradigm of evolutionary computation (Dimutrescu et al., 2000) and as so much work has been done on aspects of the model, we will look in greater detail at its main components and theories relating to their influence in §2.8.2.

Evolutionary computation

The previous four sections have outlined the historical branches of evolutionary computation. Although each of the variants presents slightly different models of evolution, the representation of individuals and the mechanism of exploring the population search space, the differences are not sufficient to consider any of the variants unique. D. Fogel (2000) notes that since 1993 and the formation of the journal *Evolutionary Computation*, "evolutionary computation" (or "evolutionary computing") has become an accepted umbrella term for all the variants. D. Fogel even doubts whether any value can continue to be gained by using specialist terminology from one of the branches:

> "It is no longer possible to identify a particular effort in evolutionary computation as a genetic algorithm, an evolution strategy, or an evolutionary program, simply by examining the representation chosen, the selection method, the use of self-adaptation, recombination or any other factor. In fact, the practical utility of each of these terms has evolved to be essentially useless: Little or no information is conveyed by identifying a particular effort as a genetic algorithm, evolution strategy or evolutionary program." (D. Fogel 2000)

In agreement with this sentiment and notwithstanding the historical importance of the variant branches, the term *evolutionary computation* will be used for the remainder of this thesis.

2.8.2 The canonical genetic algorithm

The basic framework of evolutionary computation is one based on population convergence over optimal peaks in a fitness landscape of the genotype's population. In the breeding of successive generations, selection occurs according

to individuals being assessed against some fitness criteria, thus some of their "good" genes are carried over into the next generation. The canonical process uses fixed-length binary strings to represent chromosomes. In terms of actual algorithms, the genetic operators are procedures that modify the individuals represented as chromosomes by mutation (or inversion) or by combining them (crossover). As individuals usually map to a represented solution, it is common in evolutionary computation to refer to the population as containing *candidate solutions*.

The canonical or simple GA is as follows (from Dimutrescu et al. (2000)), where t means time step:

1. Set $t = 0$
2. Initialise chromosome population $P(t)$.
3. Evaluate $P(t)$ using fitness criteria.
4. **while** termination condition not satisfied **do**
   begin
   (a) Select best individuals from $P(t)$. Let $P_1(t)$ be the set of selected chromosomes. Choose individuals from $P_1(t)$ to enter mating pool (MP).
   (b) Recombine chromosomes in MP forming populations P_2. Mutate chromosomes in P_2 forming P_3.
   (c) Select replacements from P_3 and $P(t)$ forming $P(t+1)$.
   (d) Set $t = t + 1$
   end

This simple process provides certain key elements to the model that have been investigated in great detail. For example, there are many forms of crossover. Holland created a straightforward crossover between two parent chromosomes to get two offspring by "selecting a random position along the coding and splicing the section that appears before the selected position in the first string with the section that appears after the selected position in the second string, and vice versa" (see D. Fogel (2000) and Fig. 2.18). Other types exist that use multiple crossover points.

Generally each chromosome is assigned a probability of reproduction so that its chances of being selected are proportional to its fitness. One method of doing this is the *roulette wheel*, which divides up the population such that all

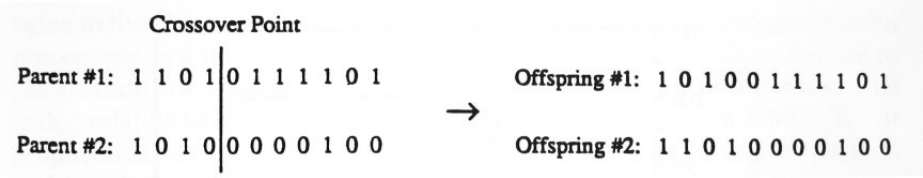
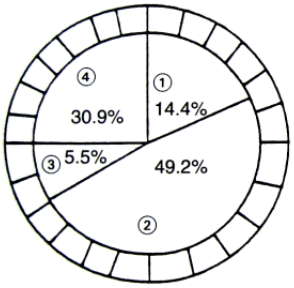


Figure 2.18: The one-point crossover operator applied to two parents.
Image and text reproduced from D. Fogel (2000)



No.	String	Fitness	% of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

Figure 2.19: Roulette wheel selection.
Image and text reproduced from D. Fogel (2000)

chromosomes receive a probability in relation to their fitness (Fig. 2.19). Another popular method is tournament selection, and there are several others.

The use of binary encoding for fixed-length chromosomes has been criticised since it was first proposed as a universal encoding by Holland (1975). Binary encoding has its historical roots in the introduction of the schema theorem and building blocks, also by Holland. D. Fogel (2000) states that "schemas allow a way of determining the usefulness of finding out fitness values for strings that match your schema, as a partial match should also mean a partial fitness." His example explains how using a wild card [*] in a schema where the evaluation of the string [0000] has some fitness, the schema would suggest that partial information is also received about the worth of sampling the variations in [****], [0***], [*0**], [0*0*], [*00*] and so on (D. Fogel 2000). This characteristic is called *implicit parallelism* and indicates that a single sample can provide information with respect to many schemas. It is claimed that certain representations and problem spaces are more amenable to implicit parallelism in schema design, particularly those where individual genes are not epistatic (MacKay, 2003).

But although Holland claims to have proved maximum implicit parallelism (i.e. the effectiveness of using schemas) occurs when the encoding is binary (1975), others have found no practical advantage. Michalewicz (1992) finds

that real-valued numerical optimisation problems are best encoded in floating-point representations (faster, more precise), and others had similar experiences after practical experimentation (Koza, 1989; Syswerda, 1991; Wright, 1991). Nowadays, binary representations are rarely used except when the representation can be easily mapped to a series of Boolean decisions or a bit mask. Whatever representation is chosen for the chromosome encoding, it would be well to remember that Fogel and Ghozeil (1997) "proved that there are essential equivalencies between any bijective representations, regardless of cardinality. . . Thus, no intrinsic advantage accrues to any particular representation" (D. Fogel, 2000).

According to the *building block hypothesis* (Goldberg, 1989; Holland, 1975), genetic algorithms work by locating and maintaining "good" building blocks. Building blocks are defined as "low order, low defining-length schemata with above average fitness".[19] Good building blocks are joined to other building blocks to create sequences that are associated with above average fitness. The hypothesis rests on the assumption that combinations of good schemata are likely to result in higher fitness more quickly than could be achieved if every possible combination of bits in a string were tried. Goldberg states "instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings" (Goldberg, 1989). The building block hypothesis has been criticised as having no theoretical basis (Wright et al., 2003) and experimental evidence has shown that single point crossover does not result in identifiably better solutions (Syswerda, 1989). Despite the uncertainty around the building block hypothesis, it is notable that there is no other well developed philosophy about *how* genetic algorithms work and such a theory (or practical understanding) is needed if evolutionary computation is ever to scale to tackling large, complex problems (see §2.9.2).

GAs have been applied to a wide variety of real-world tasks. As that experience has been gained, practitioners discovered there were issues that reduce the technique's attractiveness as a search-based optimisation algorithm. Premature convergence is a common problem that occurs when the population of chromosomes reaches a generation where crossover no longer provides offspring that are capable of out-performing their parents. Although one might suspect this is the natural fate of any hill-climbing search, premature convergence is peculiar in that the means to avoid it often seem landscape dependent. An example was the attempt to introduce *dynamic parameter encoding* (DPE) by Schraudolph and Belew (1992). The technique seems to offer promise

[19] Definition taken from Wikipedia: http://en.wikipedia.org/wiki/Genetic_algorithm.

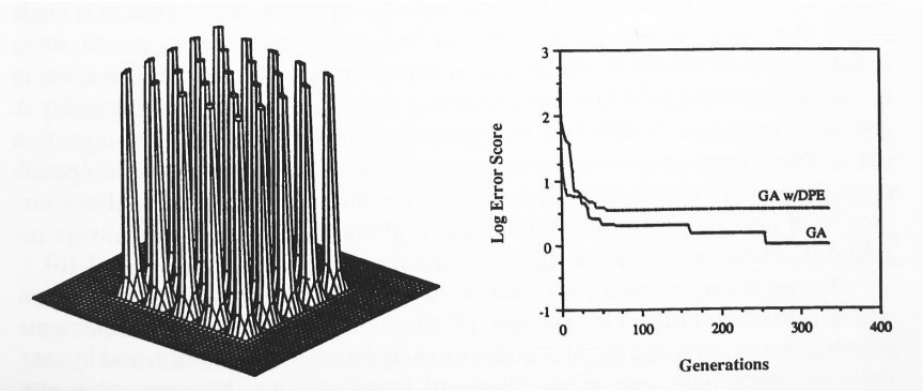


Figure 2.20: DPE performance on an extreme landscape (inverted Shekel's foxholes).

Image reproduced from (D. Fogel, 2000)

on quadratic bowl shaped landscapes; however, it actually fares worse than a simple GA on multimodal type landscapes (such as Shekel's foxholes, see Fig. 2.20). Further doubt was cast on the wisdom of pursuing solutions to premature convergence by coming up with landscape-specific algorithms. The "No free lunch theorem" by D. Wolpert and Macready (1995; 1997) states that averaged over all landscapes, no search algorithm performs better than any other. The same may be true for landscape-specific solutions to premature convergence. In addition to which, landscape-specific solutions require prior knowledge of the search space — something which it may be impossible to ascertain.

Despite these shortcomings, there is no doubting the popularity of GAs, either in academia or industry. They are now part of the standard toolbox of search algorithms where the search space is large and unpredictable, and have become the default method for tackling traditional, NP-hard problems, such as the travelling salesman. Unsurprisingly, their ability to tackle multiple objectives and to find optimal (or "good enough") solutions grouping those objectives finds many applications in control systems and other industrial applications.

However, the range of problems that evolutionary computation has tackled outside those relating to optimisation is not as wide as one might imagine. Perhaps a victim of its own (industry-based?) success, evolutionary computation and in particular, GAs, have themselves evolved little beyond their basic operational framework that was first described in the 1960s.

2.9 Weaknesses in EC Models

No one has yet evolved a design for a car, a house, or anything that has a high number of parts, each of which can be exposed to evolutionary change. In the current model, as complexity grows, the length of the chromosome string representation grows, and large numbers of generations start to be required to reach good solutions. This increasingly hampers the effectiveness of an evolutionary search and during the 1980–1990s researchers attempted to address the issue. Initially, in an echo of the 1960s, researchers tackled the problem by simply throwing more computing power at it (Koza et al., 2003). Parallel GAs running on parallel machines were also tried (Cantu-Paz, 1998; Cantu-Paz and Goldberg, 1997). But the promised breakthrough hasn't happened. To date, no one has cracked the problem of scale when it comes to complexity, and evolutionary computation remains tied to addressing the same problems of multi-objective optimisation that it first started investigating over twenty five years ago.

2.9.1 Obsessed by optimisation

By tracing the historical successes in evolutionary computation we can understand better the influence of those successes on the direction of subsequent research. Industrial success is often a good thing, but there is no doubt that evolutionary computation as a field has been heavily influenced by the need to fulfill its practical promise. Success in industrial applications, such as circuit design (Koza et al., 2004) and control systems (Robinson and McIlroy, 1995; Sharman et al., 1995), has meant that evolutionary computation was always being pushed towards making the evolutionary process more efficient, more practical. Such implementation concerns are not usually the domain of academia, but Chris Stephens being interviewed in 2003 for the EvoNet website, admits that research in the field is driven by those who want to use evolutionary computation for practical design problems:

> "Evolutionary computation, at least in terms of the fraction of papers dedicated to it, is mainly driven by the practitioners. ... there is a big gap between the mathematical perspective and the engineering perspective." (Stephens, 2003)

While research programs can be forgiven for focusing on ways to improve the performance of genetic algorithms, that same focus has produced a rather blinkered view of evolutionary computation, one that sees nothing more in evolution than a set of optimising search algorithms. For example, David Fogel, in his introduction to *Evolutionary Computation: the fossil record* firmly states

Multi-objective GA	Year
Schaffer's Vector Evaluated GA	1985
Syswerda & Palmucci GA with weighted sums	1991
Fonseca and Fleming propose MOGAs	1992
Wilson & MacLeod goal based GA	1993
Goldberg's Fast Messy GA	1993
Srinivas & Deb Nondominated sorting GA	1993
Horn, Nafliotis and Goldberg's Niched Pareto GA — co-operative sharing	1993
Coello Min-Max Optimisation — ideal non-Pareto feasibility vetting	1996
Priaux et al. GA-based approach with game theory	1997
Gary Lamont & David Van Veldhuizen's survey of MOGAs	2002
Tan, Khor, Lee &. Yang Tabu-based exploratory GA	2003

Table 2.1: List of multi-objective GAs (MOGAs), adapted and much reduced from Coello (2000).

that "natural evolution is a population-based optimization process" (Fogel, 1998). Martin Keane, in a similar introductory chapter, describes evolutionary computation as "design search and optimization" (Keane, 2000). Both of these views stem from a practitioner's perspective, a perspective which has built up authority after the success GAs had in particular with multi-objective optimisation problems. So the last two decades have seen a continuous stream of papers published on the performance of GAs and optimisation, perhaps to the detriment of work that could have explored other features of the evolutionary process, such as greater exploration, better bootstrapping to deal with complexity, alternative mechanisms for encoding or problem representation and so on.[20] For example, a survey by Coello (2000) for the IEEE on multiple objective GAs (MOGAs) managed to list almost fifty separate applications and variants of MOGAs, and one wonders whether even industrial applicants would wish to wade through them all to find one appropriate to their needs (see Table. 2.1).

2.9.2 The black art of decomposition

Despite the attention to optimisation issues, to say that nothing had been reported about other interesting aspects of the evolutionary process would be wrong. Indeed the ability of evolution to "invent" things was widely publicised in popular journals like *Scientific American* (Koza et al., 2003). As noted, human-competitive, even patented designs have been produced by evolution-

[20]This remains the case. A count of papers submitted to EvoWorkshop 2003 and 2004 shows the majority (over 60%) in areas related to optimisation.

ary computation, and practitioners such as David Goldberg claim that what was going on in these processes was more than mere optimisation:

> "... the design of effective GAs [is] ultimately helping us create first-order *computational models of innovation*." Goldberg (2002) (my italics)

However, a genuine computational model of innovation is something we are far from having. Innovation is hard to quantify or model in any sense, and working out *how* people (or GAs) invent things has proved equally difficult (although Thompson (2002) has some interesting comments on how evolution does this).

Goldberg, following on from Holland, believes that the knack of getting your problem effectively solved by GAs lies in the correct representation of the problem, and that representation itself relies on the problem being broken down in the correct "chunks", so that the right building blocks can be chosen. This may seem something of a black art to the uninitiated, an impression unlikely to be diminished by Goldberg's interesting, if unconvincing, description of the invention of human flight by the Wright Brothers in 1903. According to Goldberg, the Wright brother's success was down to how they decomposed the problem. While the evidence for this is sketchy and based on Goldberg's analysis of events, Goldberg nevertheless makes some interesting observations about the nature of invention and human design.

Goldberg demonstrates that human design isn't always a rational process of problem decomposition. When the problem domain is poorly understood, people will apparently try anything, no matter how deeply it may run against the grain of common sense. Fig. 2.21 gives an idea of just *how wild* human invention can be when unconstrained by design principles.[21] But it equally demonstrates just how dangerous a little knowledge can be, as an incomplete understanding of aerofoils led to people misapplying what little knowledge they had and trying designs that were doomed to fail. Goldberg described these early pioneers of aviation as appearing to "flail about in design space, hoping for good luck" (Goldberg, 2002).

But even if Goldberg were correct, and the reason for the Wright brothers' success was the correct decomposition of their problem, there is still no convincing rationale behind *why* they broke it down the way the did — was it fluke, intuition, or did they use a set of rules that could be applied to unknown problem domains everywhere with the same degree of success? Goldberg claims the latter, but offers only a vague method, while the record of those

[21] I can find no authoritative source for the photographs in Fig. 2.21, which appear on many websites. A good source for explanations about the machines can be found at http://www.ctie.monash.edu.au.

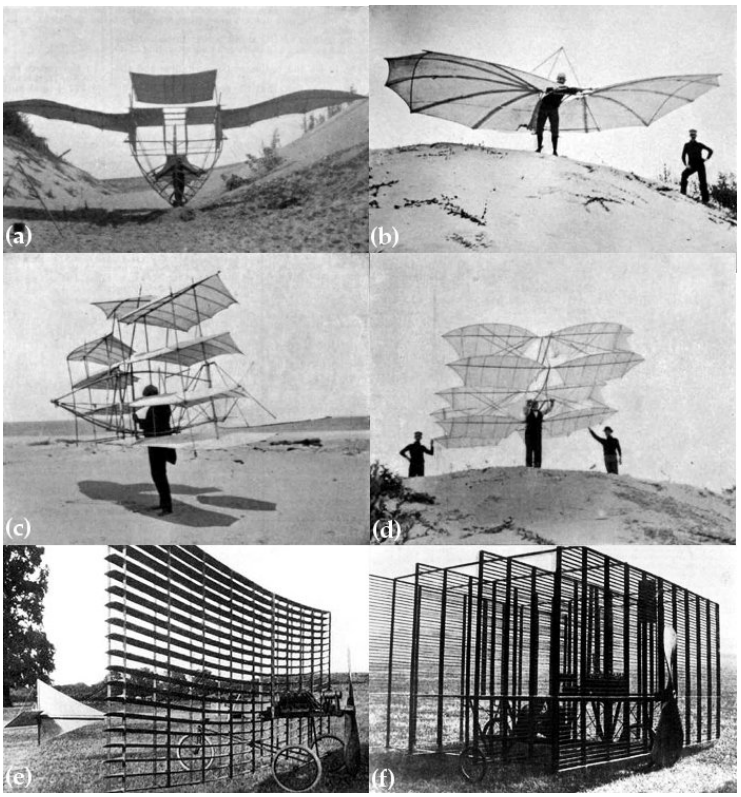


Figure 2.21: Bats and bikes as flying machines from early aviation pioneers: (a) Le Bris, The Albatross, 1868; (b)–(d) Adapted Lilienthal c.1890; (e) Phillips Multiplane of 1904; (f) Phillips Multiplane of 1907.

aviation pioneers shows that while many tried, hardly any had success. One argument that supports finding the "lucky" decomposition of the problem is that *over a population*, the success rate of so many failures is consistent with an evolutionary process exploring a large search space. However, it is one thing for human engineers to have discovered "good building blocks" that could then be combined to solve a bigger problem, it quite another to suggest that evolutionary computation can do the same. The problem again comes back to scale.

Practitioners, such as Goldberg, advocate "careful" decomposition for complex problems: one should decompose the problem into small chunks, then run an evolutionary computation process over them. Leaving aside whether your problem decomposition is correct, for larger, more complex problems, decomposed solutions must "bolt back together" so that the whole thing works as a single solution. But although problem decomposition is a typically human approach to finding a solution, there is no evidence that natural evolution tackles large scale problems this way. In fact the evidence is to the contrary. We have

seen that epistatic fitness landscapes in co-evolution appear to be self-tuned close to the critical point of interconnectedness (see page 46), meaning that although species appear to be evolving in isolation, they are in fact responding to a wider ecosystem that cannot easily be broken into parts. Despite ideas about species niching on local optima, there is no evidence that natural evolution allows a chunk of the system to evolve in isolation, with the aim of making it fit into the wider whole at a later point. Such a proposal seems counter-intuitive, but this is what has been proposed as a way to scale evolutionary computation (Goldberg, 2002). However, work by Torrensen (2000) again reinforces the fact that evolutionary algorithms exploit the particular characteristics of their search space to find a solution, and this results in highly localised solutions for local problems.

Torrensen (2000) investigated issues of scalability and complexity while trying to evolve a signal filter over a range of inputs for a road image recognition exercise. Torrenson first broke the problem into a series of subproblems (each evolving a logic circuit on part of an FPGA (Field Programmable Gate Array)). For each subproblem, he then subdivided his inputs. Although the circuits evolved individually to high fitness, Torrensen found that when they were reassembled, the circuits failed to work due to noise from other inputs. This is a characteristic of evolved solutions in that evolution is typically "lazy"; it does the minimum possible to achieve a satisfactory result. It is also highly environment sensitive, solutions are not generally portable (see also Thompson (1997), discussed in more detail in §2.9.3). Torrensen's solution was to evolve his decomposed filters by exposing them to the full range of inputs. This worked when the decomposed elements were reassembled, but meant that the evolutionary process was now much slower for each subunit than before, resulting in a less than ideal solution to the problem of complexity and scale (see also investigations by Vassilev et al. (2000).

2.9.3 Towards richer invention

Humans "flailing about in the design space" of early aviation is an example of highly unconstrained invention, but it is not a realistic example of evolutionary invention. We know from evidence in evolutionary developmental biology, that evolution tends to tinker with successful structures rather than create entirely new structures out of the blue. A better example of evolutionary design by humans is the evolution of golf balls in the latter part of the twentieth century (Thompson, 2002). In contrast to early aviation, where a little knowledge led to many misguided designs, the evolution of golf balls was carried out in ignorance of why the changes led to improvements.

The earliest balls were called featheries and were made of hide case densely packed with feathers. Around 1850, a new type of ball appeared, a guttie, of solid *gutta percha* (a sort of rubber). They were cheap and smooth, but didn't fly as far as the older featheries. Gradually it was noticed that used featheries travelled further than brand new ones, so people experimented adding nicks and cuts to make their balls fly further. Over time, the manufacturers started to produce balls that had similar textures on them. Modern balls prefer a variety of dimples. However, it is only recently that the aerodynamics causing a rough ball to travel further have been understood. Two things stand out in this example of design evolution; i) changes were made at random and in ignorance of why they were good changes to make, ii) the changes were incremental.

Thompson wanted to see if a blind, incremental evolutionary search could still be encouraged to generate truly innovative designs. In 1995 he set up a ground-breaking experiment designed to promote design innovation through the relaxation of constraints (Thompson, 1996). His experiment evolved a circuit to distinguish between two frequencies on an FPGA, at the lowest level of abstraction possible — that of the physical behaviour of the platform. A 10x10 area of Xilinx 6126 bitstream was evolved (i.e. all the bits in this area were evolved directly as chromosome bits in a GA). The evolved circuit had to discriminate between 1kHz and 10kHz bursts of signals, no other input was given. Even the clock on the chip was turned off (Gordon and Bentley, 2002).

The experiment was a success. However, when Thompson tried to copy the evolved circuit onto another FPGA chip, he found that the circuit wasn't portable. He then tried to move the circuit onto another part of the original chip used in the experiment. Again the circuit failed to work. Thompson discovered that the circuit made use the *physical properties* of the silicon on the FPGA chip. It was extremely sensitive to any alteration in its environment — temperature, electricity supply, even the silicon of the chip — a change in any of them could stop the circuit working. The result again demonstrates the "laziness" of evolution.

Thompson's circuits lacked both robustness and portability.[22] But there was another feature of the evolved circuit that caused puzzlement. A lengthy analysis of the circuit concluded that its functionality was "bizarre" and some parts of it are still not understood (Thompson, 1997; Thompson et al., 1999). By utilising physical characteristics of the platform, the algorithm made some very unusual and complex circuitry. Commentators on Thompson's work have suggested that the reason for the innovative nature of his evolved circuit was not because evolution had searched a bigger design space than human designers, but that evolution had navigated through that search space differently (Gor-

[22]Thompson later evolved more robust circuits by varying the environmental conditions.

don and Bentley, 2002). It seems inescapable that the reason for the search trajectory is due in part to evolution making use of the physical properties of its environment.

Miller and Downing (2002) have investigated what it would mean for complexity and innovation if evolved solutions were given a free reign to make use of the physics embedded in a rich medium. Rather than being surprised at evolution exploiting the complex physical properties of silicon, Miller states we should be impressed that it was able to do anything at all, given that silicon as a material was chosen expressly for its stability in electronics. He suggests the time is ripe to abandon media traditionally chosen for its physical stability and even the conventional components of electronic circuit boards: "... artificial intrinsic evolution may be best attempted in physical substrates that are rich and complex, rather than conventional transistor based technology" (Miller and Downing, 2002). To this end, Miller and Harding investigated evolving robot controllers in media such as liquid crystal (Harding and Miller, 2003).

2.9.4 The gap between genotype and phenotype

Environments select for fitness based on the capabilities of the phenotype. In the natural world, the viability of the phenotype not only acts as a brake on the random mutation of genetic material, it also acts to link evolutionary search to the process of construction. In most models of evolutionary computation, the development of the phenotype prior to selection is conspicuously absent. But as greater importance began to be attached to developmental processes in biology, so researchers in evolutionary computation began to question why it was missing from their models (Shipman et al., 2000). The standard representation for genetic algorithms, for example, is that shown in Fig. 2.22 (Lewontin, 1974). While such diagrams show the mappings between genotype and phenotype populations, and even some cursory epistasis, they stem from the period of neo-Darwinism that saw genes as the source of all phenotypic features and behaviours, and the role of development was credited with less influence than it has today. But these diagrams still constitute the majority of models employed in evolutionary computation. Genotype to phenotype mapping remains insignificant. The developmental process of the organism interacting with its environment and the restrictions imposed on evolution by development — such as the viability of the organism and the dependence of complex features on phylogenic predecessors — is ignored. But if these things help nature handle the combinatorial explosion of complexity when all parts of the system are potentially exposed to mutation and selection, then why would they not also be of advantage to evolutionary computation?

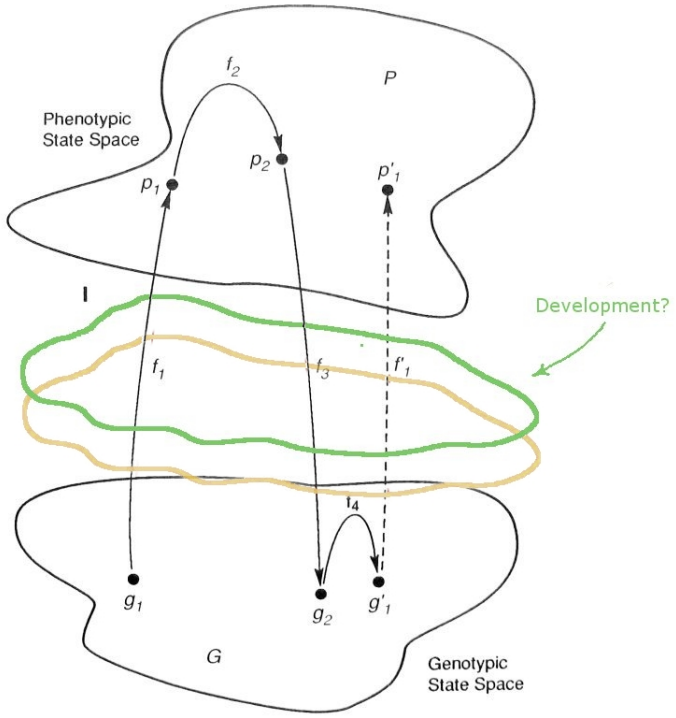


Figure 2.22: Lewontin's (1974) distinction between the two state spaces of genotype population space (informational/encoding) and phenotype population space (behavioural/performance). The middle spaces have been added to indicate the missing stages of development in the model.
Image adapted from (D. Fogel 2000)

Genetic evidence from evolutionary developmental biology provides us with evidence of why evolution has to tinker with existing structures. Embryology has given us clues about the hierarchical nature of developmental structures and how evolution is constrained to act at the later stages of development. Evolutionary computation has yet to take account of such evidence. Instead, hypotheses such as building blocks and schema theorems that have no basis in biology have been allowed to dominate the research agenda, while failing to tackle either complexity or scalability. Thompson's work opened researchers' eyes to a whole new world of search potential. To explore it we need to allow our search algorithms to exploit physical resources. If we want to evolve solutions beyond human design space, we must move evolutionary computation out of sterile software abstractions into a much richer environment. However, providing access to richer resources does not guarantee they will be used. We know that biological development feeds back information about the search environment to the genome. The genome in its turn, dictates

how to respond to that environment. To attempt a similar exploratory system in evolutionary computation, we need a mechanism capable of dynamic gene expression that can control the developmental process.

2.9.5 Models of development

Development in evolutionary computation is a still nascent subject area. Other than an interesting collection of essays edited by Kumar and Bentley (2003) and some isolated submissions in the field of evolvable hardware (Gordon and Bentley, 2002; Lones, 2003; Tufte and Haddow, 2003), there has been little work done on modelling development within the evolutionary computation community.[23] Some early papers that are often cited, such as Fleischer and Barr (1994); Hogeweg (2000a,b) have presented models that, while impressive, have not been further developed by other authors. However, the field is growing and we highlight below some of the better known frameworks.

One relatively successful framework with a considerable body of research behind it is Lindenmayer, or L-system, grammars (Lindenmayer and Prusinkiewicz, 1989). The approach is capable of modelling the growth of plants and simple cell development. The use of generative grammars such as L-systems provides a means of modelling structure, and in particular, the growth of that structure. Structural elements represented in L-systems may not have or need a close mapping to the microscopic units that comprise real biological structures — in fact, successful models have been built using macro-level abstractions of plant parts, such as petals and leaves.

A powerful feature of L-systems is their brevity of expression. A relatively small rule set can generate surprisingly complex structures. Another is that generative grammars lend themselves to repetitive modular structures, so that structural elements such as branches or hair can be elegantly represented. One side effect of the "abstraction" of macro-level units is that some irregularity has to be introduced into the models so that forms acquire "roughness" (see Fig. 2.23). This can be done using context-sensitive or stochastic means. Some success has also been achieved using fractals for this purpose (Ferraro et al., 2005) and when rendered with turtle graphics, such models can be startlingly realistic in both behaviour and appearance (see Jacob (1999); Prusinkiewicz (2000) and Fig. 2.24).

For those looking at particular influences on development, feedback points can be introduced into L-system structures in conjunction with turtle graphics to produce models that respond to changes in their environment. Work by

[23] Until recently there were few places to get such work published. For example GECCO, the main conference for evolutionary computation, featured its first track in development in 2007.

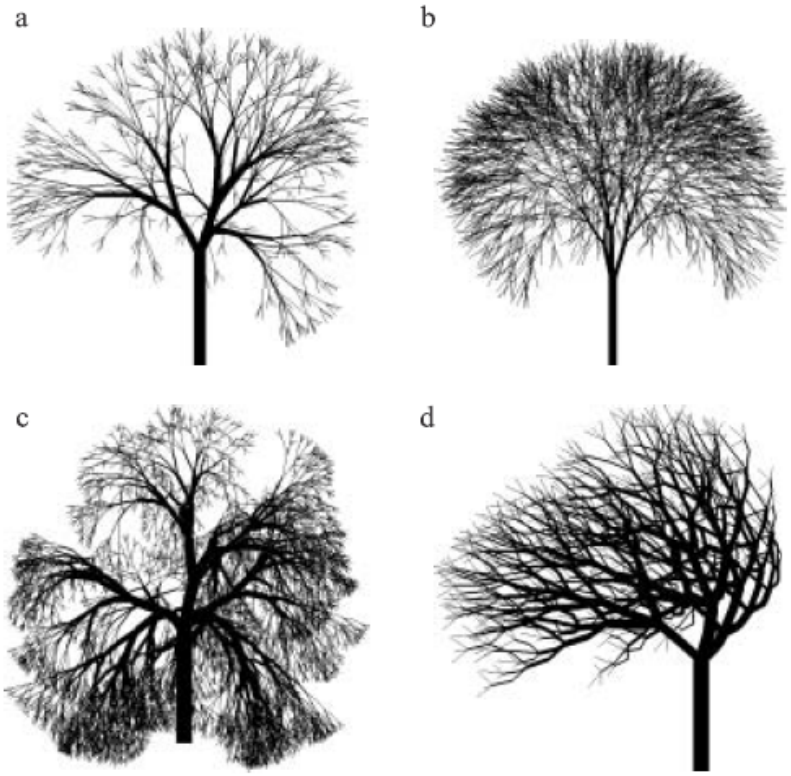


Figure 2.23: Some trees with different branching structures produced by L-systems.
Image reproduced from Prusinkiewicz and Lindenmayer (1990, pg. 60, Fig. 2.8)

Měch and Prusninkiewicz (1996) has shown how feedback can be incorporated into open L-systems to show variance in the developmental outcome according to environmental conditions (see Fig. 2.25). Thus the potential to have open systems that develop in a natural way by interaction becomes a possibility.

To date, such systems have been implemented in software-based virtual environments (as opposed to embedded solutions using real sensor data). Jacob was one of the first to use evolutionary computation with L-systems to explore the evolution of plants and branching structures under light deprivation (Jacob, 1999). Following on from this, work by Hornby has shown that evolution is able to quickly make use of the structural forms that L-systems can describe, and that the use of a generative encoding to produce such structures is advantageous in that "good" structures can be built more quickly (Hornby et al., 1999, 2001; Hornby and Pollack, 2001a,b). His work suggests that there may be a link between these sorts of highly compressed, generative descriptions and the complex, cascading control of genetic regulatory expression, and this

Figure 2.24: Photograph of wild crocus (left) and rendered image produced by a "hairy" L-system (right).
Image and text reproduced from Fuhrer et al. (2006)

would be interesting to explore. Hornby and Pollack have also evolved controllers based on L-systems (in conjunction with neural networks) to produce realistic gaits in simulated walking robots (Hornby and Pollack, 2002). The use of L-systems to model developmental processes is attractive to computer scientists as generative grammars are easy to represent and much of the work to render the structures graphically has already been done.

Developmental characteristics, such as canalization, have been viewed as a useful attribute for systems seeking fault recovery or robustness. Such approaches generally take the view of development as a robust construction process, rather than an adaptive control response to exploration. One example is Miller's French flag "multicellular organism" (Miller, 2004), which formed the starting point for several other pieces of work. Miller evolved two solutions demonstrating interesting capacities for self-repair based on cell growth (i.e. cell replication). The cell behaviour was based on chemical input bits and determined whether the cell would live, die or differentiate as it grew in the Moore neighbourhood (the 8 cells surrounding a 2-D cell). The number of chemicals varied but all chemicals followed a diffusion rule as they spread to new neighbourhoods. Miller's choice of a flag as the task map was based on Wolpert's description of positional information in early embryo development (Wolpert, 1998). Miller's experiments demonstrated that pattern recov-

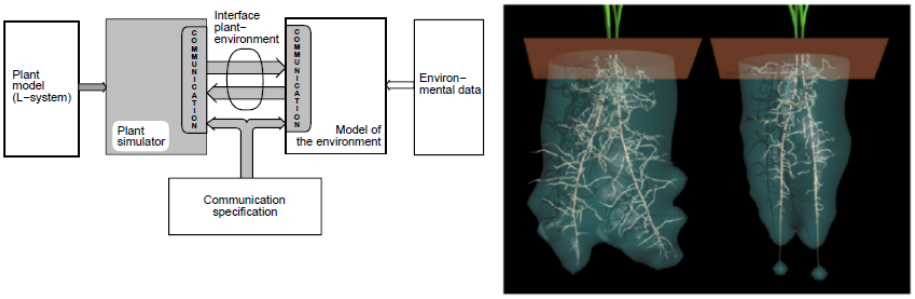


Figure 2.25: Model for an open L-system allowing environmental feedback, in this case modelling root exploration.
Image and text reproduced from Měch and Prusinkiewicz (1996)

ery, even after damage, was possible following a simple developmental model. In his models, the presence of more chemicals had a positive effect on fitness. Miller proposed to take the work forward and implement the system as a control mechanism giving cell growth a function, however to my knowledge this has not yet been done.

Pauline Haddow's group in Norway have also emphasised the importance of a developmental approach, initially using FPGAs (Haddow and Tufte, 2001) and examining the benefits of an extended genotype-phenotype mapping and redundancy / self-repair, and more recently, extending Miller's work and looking at cell development in three dimensions (Haddow and Hoye, 2007). In the latter, they asked whether the presence of chemicals was helpful or a hindrance to the developmental process, but rather than having cell behaviour solely determined by chemicals, proteins requests are used. The chemicals form part of a precondition to protein production, and in contrast to Miller's results it was found that too many chemicals hindered phenotype fitness. Haddow's work is possibly the closest models we have to investigating development in the light of gene expression control. However, even here development is essentially modelled at the level of the single cell (albeit with some parallelism). No gene regulatory networks are modelled, nor are attempts made to see if intermediate fitness can be assigned to developmental stages. Such cascading regulatory control is common in natural systems, but it isn't clear what role evolution plays in all aspects of regulatory control. Like many areas of biological development, this could be a rich vein for evolutionary computation to explore.

As a final note, a taxonomy has been proposed by Stanley and Mikkulainen (2003) for artificial developmental systems, or what the authors call "artificial embryogeny". Under their classification, models of development fall into two camps: L-systems (or similar generative grammars) and cell chem-

istry approaches. The latter is inspired by the early models by Turing (1952), who defined mathematical models of diffusion and reactions in physical substrates. Stanley and Mikkulainen claim that the division between grammatical systems and cell chemistry approaches is the easiest to make in what is admittedly a fledgling field. However, they go on to say that differences between the two are "largely superficial and [do] not reflect how phenotypes can develop" (page 106). Instead, their taxonomy draws up five dimensions of development, which help position an artificial developmental system in terms of what it is trying to achieve:

Cell Fate The eventual role of a cell during development;

Targeting Connections made by cells to target locations;

Heterchrony Timing and ordering of events in the phylogeny of an organism;

Canalization Stable development despite genetic perturbation;

Complexification The addition of new genes.

These dimensions are sliding scales with respect to nature. Stanley and Mikkulainen point out that being closer to natural systems isn't necessarily a measure of whether the system is "better", and that the dimensions instead inform you about the broad characteristics and capabilities of the system. For example, not being faithful to nature could give an artificial development model considerable advantage in terms of its computing speed. However, Stanley and Mikkulainen's dimensions contain some bias towards neural networks in particular and machine intelligence more generally. The dimension of *targeting* seems focused on the quality (or ability) of cells to form extensions such as dendrites and axons used in neural connections and nervous systems. Such qualities relate more to animal cytology than a general measurement of biological development. This criticism aside, their system of classification is at least a useful reminder of some of the qualities that artificial development models should emulate in order to get closer to nature and their report contains a useful summary of work in this area.

2.10 Summary

The previous section has taken an overview of the field of evolutionary computation. Some omissions were necessary in both this overview and that of biological evolution and development. The purpose of covering what has already become well-trodden ground to some in computer science is to try and emphasise the importance of a developmental perspective on evolutionary processes.

The genetic reductionism of the 1970s in biology left a lasting impression on evolutionary computation, and it is one that has become the *de facto* viewpoint.

However, there is evidence that calls for more inclusive models of evolutionary search are gaining acceptance. A research agenda calling for evolutionary computation to abandon its "restricted and dated understanding of natural evolution" has recently appeared (Banzhaf et al., 2006). That article asks the field to challenge its long held assumption that there is a "one-way flow of information, from DNA to proteins" that forms the basis of solution discovery by evolutionary search algorithms. The view prevalent among practitioners of evolutionary computation is that genetic material is essentially symbolic rather than physical. But ignoring the physical aspects of gene translation may have led the field to underestimate the importance of developmental processes on issues like scalability and reuse. Advances in developmental biology have given us fresh insights into how evolution explores a functional domain and the constraints it operates under. Criticisms of this nature have appeared elsewhere (Kumar and Bentley, 2003) but have had little impact on the field, which continues to be dominated by efforts to optimise evolutionary search.

One agenda, to investigate *how* evolutionary algorithms find solutions and *what* they are capable of finding, was set in motion by Adrian Thompson in the mid 1990s (Thompson, 1996). His *in silico* experiments were designed to encourage as much innovation from the evolutionary process as possible. By allowing free access to the physical nature of the search domain, Thompson discovered that evolution was capable of finding solutions in areas that humans would find difficult or impossible to operate (see discussions in Gordon (2001); Harding and Miller (2004); Miller and Downing (2002)). An outstanding task for evolutionary computation — for those who want to pursue Thompson's aims — is to find ways of introducing the equivalent richness of real world physics into virtual environments.

However, the introduction of richer resources does not guarantee their accessibility. In order to access interesting physical properties in evolved solutions, we may need a physical embodiment of the developmental mechanisms employed by nature. This requires a two-way flow of information that allows a phenotype to explore a functional domain in a manner controlled by the genome. A crucial ability of the developmental process is to sense environmental inputs and respond. Things grow in accordance with their surroundings, using a feedback mechanism that tells cells when to start producing certain proteins or inhibit the production of others. Evidence from the study of gene regulatory networks suggests that evolution has exploited developmental mechanisms to allow the reuse of "useful" genes in different contexts (Carroll, 2006; Carroll et al., 2001).

The issue of reuse prompts another criticism of current models of evolutionary computation, namely the "single solution genome". This artifact results from the proximity between genotype and phenotype, the translation process between them being so direct a mapping as to make them often indistinguishable. In nature, a genome controlling the developmental process selects from many potential responses, according to the developmental context. A gene used in one place will have *a different role* somewhere else, roles that are separated by time and space. The repeated morphological features we witness throughout nature are the product of developmental processes. By contrast, in evolutionary computation, selection is carried out on "instant" phenotypic solutions, randomly mutating from one generation to the next, in a process that has no natural mechanism for exploring the functional search space, or conserving and re-using useful genes (see discussion in §2.5.4). Such evolved solutions only work as single, fixed answers to static environments.

Biological evolution has come up with a neat trick: DNA encodes for proteins, and those proteins can govern the production of other proteins. Thus it encodes for the rules that dictate how it explores a particular functional domain. Not only that, but a tiny fraction of what could be expressed is ever realised in a phenotype. A genome contains solutions for countless *sets* of contexts. Change the contexts and the genome still has room for developmental exploration. Without similar mechanisms of interaction and feedback, digital genomes cannot guide themselves across functional search spaces in a way that fully exploits a domain's resources, and this is particularly true where that domain includes the complexity provided by real-world physics.

While the wet manufacture of life is hardly a practical ambition for computer scientists, it contains clues, patterns if you like, of how subtle, scalable structures can be built that allow evolution to explore and interact with the world about it. DNA alone can't do that: it's a passive instruction set. To paraphrase Lewis Wolpert, it's *proteins* that do all the work (Wolpert, 2003).

2.10.1 Key points from Literature Review and Background

The previous section argued for a broader developmental approach to evolutionary computation. There seem good reasons to look at abstracting the processes behind gene reuse in biological systems as one way to address issues such as scalability in evolutionary computation. The following bullet points list the motivation for the approach taken in this thesis, and hopefully can be more easily kept in mind by the reader as they read the next chapter.

- Gene reuse during development suggests that a few, very useful genes can produce a wide range of morphological function.

- The key to reuse is the mechanism that governs which genes are expressed. The mechanism also determines when and where.
- Context is vital to the functional role a gene will play during development. If different contexts were not possible, a gene would always have the same role.
- Applying evolutionary search to areas too difficult for human design is a challenging research task. Often these design spheres are those that have to deal with physical complexities that are difficult to analyse.

Chapter 3

Hypothesis and Experiment Aims

The summary to Chapter 2 gives an overview of the research agenda this thesis sets out to explore. It argues that a developmental perspective could help overcome some of the problems of reuse and scalability in evolutionary computation. It maintains that genome representations should contain, like biological DNA, *sets of solutions*, only some of which are realised during phenotype development. Finally, it suggests the mechanisms that control gene expression during development lie at the heart of what enables phenotype exploration.

For a digital genome to explore dynamic design domains, a system architecture is required that permits gene expression from the genome to govern stages of phenotype exploration. For this we need some way of "binding" responses from the search domain to gene expression, so that the genome can dictate how it explores a particular developmental stage. Our proposal is based on feedback to the genome, in a manner reminiscent of the role of transcription factors discussed in §2.5.4.

3.1 Conceptual Requirements

The conceptual architecture in Fig. 3.1 shows how we link developmental aspects of gene expression in an analogous process for exploration of a physical domain by evolutionary search algorithms. The top half of the diagram shows an abstract view of gene regulation according to the presence of transcription factors in the cell nucleus. These factors are context specific, and change their distribution in cells across the embryo as shown in §2.5.4. The manner in which these proteins control the later production of other proteins can be thought of

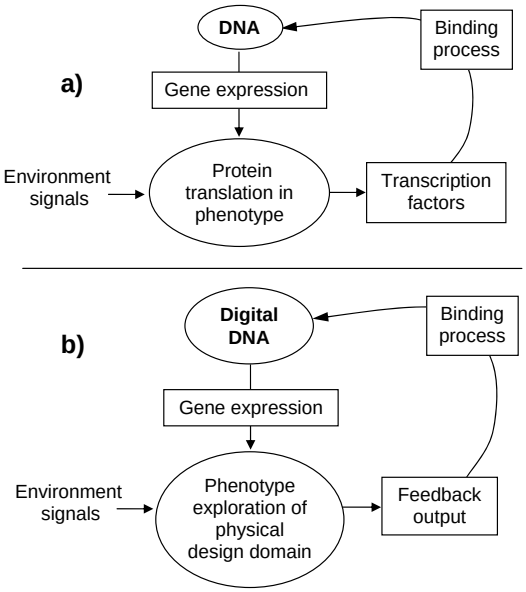


Figure 3.1: Diagram (a) shows an abstract representation of natural gene expression, where expression is affected by the presence of transcription factors in the cell nucleus. (b) A conceptual architecture to allow gene expression to control how a phenotype would explore a complex design domain, for example one affected by real world physics.

as a feedback process from earlier protein translation in other cells. Our model captures this feedback loop conceptually, it is not our intention to claim that this is how all protein translation occurs. We also acknowledge the omission of many other important stages to protein translation. However, for our purposes of abstracting this process for adaptation into a dynamic search algorithm, we include only the minimum possible so that implementation and understanding the algorithm will be made easier. Using a feedback process to control gene expression (and hence phenotype exploration) gives the conceptual architecture a number of benefits:

- many exploratory solutions can be contained in a genome, as a solution is based on a *subset* of genes that have been expressed;
- the changing environmental inputs determines *when* a solution is exploring the search space at which moment;
- by varying the context of expression, different functional behaviours can be obtained from reused genes.

These benefits bring something new to evolutionary computation: they permit evolutionary algorithms not only to search for adaptive designs, but to control

how and when phenotype exploration moves between each adaptive solution. For our purposes, we will define each adaptive solution as a *developmental stage*.

3.2 Experiment Requirements

The conceptual architecture in Fig. 3.1 gives us a framework on which to base our requirements for an experimental system. While the architecture lends itself particularly to a software configurable platform, we know that physical complexity is hard to capture in software environments (§2.10 and §4.1). Physical environments can be difficult to interface with software, as they may be noisy or unpredictable. For ease of implementation, most evolutionary computation experiments are therefore run in simulated environments, or on simulated devices, or both. Software exploration of virtual environments has several weakness (§4.1). But there are software configurable devices that have a design environment containing rich physics which could suit our purpose. The challenge with such hardware is that it is not generally intended to be used in an automated design search that makes use of its *physical* characteristics. Notwithstanding this difficulty, the following points represent the system requirements for our experiments:

- a design domain of exploration affected by real physics (i.e. not simulated);
- a software configurable physical device;
- a means of incorporating physical feedback from the search domain into the reconfiguration process.

Our aim is to build a system that meets satisfies these requirements and the conceptual aims listed §3.1, so that the following hypothesis can be tested.

3.3 Research Hypothesis

As stated in §1.1 and repeated here, in the context of this hypothesis, the process of gene expression during biological development is abstracted and used to guide evolutionary search. The search domain we have chosen is analogue circuit design. The motivation for selecting this domain is fully detailed in the following chapter (§4.1). For the interpretation of this hypothesis, genes relate to configurable elements in an analogue circuit, while the configuration of a circuit represents a stage in phenotype configuration.

Hypothesis —*By expressing subsets of genes in specific contexts, successive stages of phenotype configuration can be determined by evolutionary search.*

This thesis allows us to set ourselves the following objectives with regard to constructing a system to test the hypothesis:

- As subsets of genes are used to define a phenotype solution, the reuse of genes must be demonstrated.
- For the system to have practical use, the different combinations of genes in subsets must demonstrate different functional possibilities.
- To back the claim that evolution was able to configure successive stages of phenotype configuration, the experiments must demonstrate that a previous stage of phenotype configuration in some part determines the ability of the system to move to the next configuration.

If these objectives are demonstrated by the system, then the claim of the hypothesis can be backed up by experimental evidence.

The next chapter connects the conceptual architecture in Fig. 3.1 to a system architecture more grounded in specific hardware and software processes. We give the rationale behind our choice of search domain and hardware platform, and propose a mechanism that will allow the feedback of physical output to control the evolved, developmental exploration of a dynamic search space.

Chapter 4

System Architecture

4.1 Selecting a complex design domain

One of the most challenging areas for evolutionary search is to apply the algorithms to design domains containing real world physics. Searches operating in these domains have the opportunity to access physical properties that would be difficult or even impossible for human engineers to take into consideration. The area has been termed *evolvable hardware*, and has found practical application on deep space missions, where it is difficult to shield Field Programmable Gate Arrays (FPGAs) from potential damage by radiation sources. The Jet Propulsion Lab (JPL) at the California Institute of Technology,[1] has shown how *in-situ evolution* can overcome physical damage in extreme environments by exploiting hardware properties (Stoica et al., 2007), and there is similar work on fault recovery for robots (Berenson et al., 2005). Most work on evolvable hardware uses an external search process that downloads a configuration bitstream onto an FPGA, which then performs a logic-based function. However, despite small analogue FPGAs[2] being available since 1995, and Thompson's original work that turned a digital, clocked FPGA into an unclocked analogue device (Thompson, 1996), the use of software configurable analogue platforms has been limited.

Analogue electronics provides the sort of physical complexity where an automated search for better circuit design could bring significant benefits. For example, analogue component behaviour can be affected by environmental conditions such as temperature range, but also by complex physical effects, such as strong electric or magnetic fields. Such issues complicate the design of

[1] http://www.jpl.nasa.gov/index.cfm

[2] This term was used by Thompson (Thompson et al., 1999), presumably to refer to early hybrid devices.

circuits and components in close proximity. Evolutionary computation has a long track record of addressing the design difficulties in this area, with notable successes by John Koza and his colleagues (Koza et al., 2004). As already discussed in §2.8.1, there are issues relating to the use of program such as SPICE to simulate an analogue circuit. Testing a software simulation of analogue circuits is several factors slower than testing the same circuit in real-time on hardware, but simulations have the advantage of being able to test many circuit configurations across a range of inputs, without having to construct any of them. Another drawback with simulating analogue hardware is the accuracy of simulated behaviour. Software is constrained to work within certain limits (e.g. number of components, temperature, etc.); outside those boundaries the accuracy of predictable behaviour goes down. This hampers one of the most interesting capabilities of *in materio* evolutionary computation[3] — which is to use parts of the design space that are too difficult or complex for human engineers. Software simulation forces experimentation to stay within known bounds, limiting the opportunity for novel exploration.

Selecting analogue electronics for *in materio* evolution provides an opportunity for the search algorithms to operate on a range of application design that uses analogue technology to interface with the real world. For example, electronic sensors feed analogue inputs into circuits, where they usually require other analogue components to treat the signal in some way, before the signal is passed to an analogue-to-digital converter (ADC) for digital processing. Digital processing of analogue signals is always subject to compromise, so it helps to have the signal condition as good as possible from the sensor before it is digitised. The design space for analogue treatment of signals is huge, as can be witnessed by the range of commercial sensors that offer particular operating characteristics so that signal treatment is as easy as possible. Each of these sensors is likely to have a circuit built around it to meet a particular functional specification. Even if we take a trivial circuit specification that contains an operational amplifier (op-amp) and a sensor, such as a photo-diode, it is obvious that there is a wide variety of potential circuits that could process the sensor's output.

To summarise, the principal difficulties of working with traditional analogue circuit design are:

- the design space of analogue components is large and complex (as discussed in the second paragraph of this section).
- software simulations are slow and inaccurate, particularly as physical limits in the simulation are approached;

[3] This term was apparently first coined by Miller and Downing (2002).

- software simulations have design limits built into them;
- real analogue circuits need to be "built", i.e. physically assembled and tested with physical inputs;

4.1.1 Reconfigurable Analogue Hardware

In the last decade, switched-capacitor based integrated circuits (ICs) have appeared that offer a research platform that overcomes some of the problems of working with traditional analogue electronics. Configurable Analogue ICs, sometimes called Field Programmable Analogue Arrays (FPAAs),[4] implement an analogue circuit by downloading a configuration bitstream onto the IC. This makes the implementation of analogue circuits fast enough to consider testing circuits in hardware. To date, other than the work at Edinburgh by Hamilton and colleagues Hamilton et al. (1998), at Heidleberg in Germany by Meier's group Schemmel et al. (2002), the work done at JPL by Berenson et al. (2005) mentioned earlier and some work at MIT by Aggarwal et al. (2006), reconfigurable analogue devices appear to have been little used in research. This is partly due to the very small capacity of the early devices (several op-amps is typical) and partly due to the widespread shift to digital signal processing. Other than the work at JPL that uses evolutionary computation to allow a device to continue working by exploiting the physics of the platform even after the platform has been damaged (extreme heat was used in their experiments), the work using analogue devices is relatively conventional in that it either uses evolutionary algorithms to perform circuit searches Hamilton et al. (1998) or searches to find good parameter adjustments to PID loops Aggarwal et al. (2006). However to date, no one has taken advantage of reconfigurable analogue devices in a way that combines evolutionary search and reconfiguration to discover circuits, but also uses gene expression to say when the device should be reconfigured.[5]

Such a system would have some nice properties, as shown in Fig. 4.1. The genome would in effect be acting as a control system: it would contain both the configuration of the circuit components and the rules of reconfiguration. As changing environmental signals are fed into the system, the circuit responds by reconfiguring to the desired output. The fitness of each reconfiguration could be assessed by testing the analogue output, and the sum of fitnesses

[4] Alternative names for these ICs include Field Programmable Transistor Arrays (FPTA, see Schemmel et al. (2002) for an example) or Dynamically Programmable Analogue Signal Processors (dpASP).

[5] See Mattiussi (2005) for similar ideas but using gene expression to adjust components in an evolved circuit. However this work was not evolution *in materio* (SPICE simulation was used) and the circuit configurations were otherwise static.

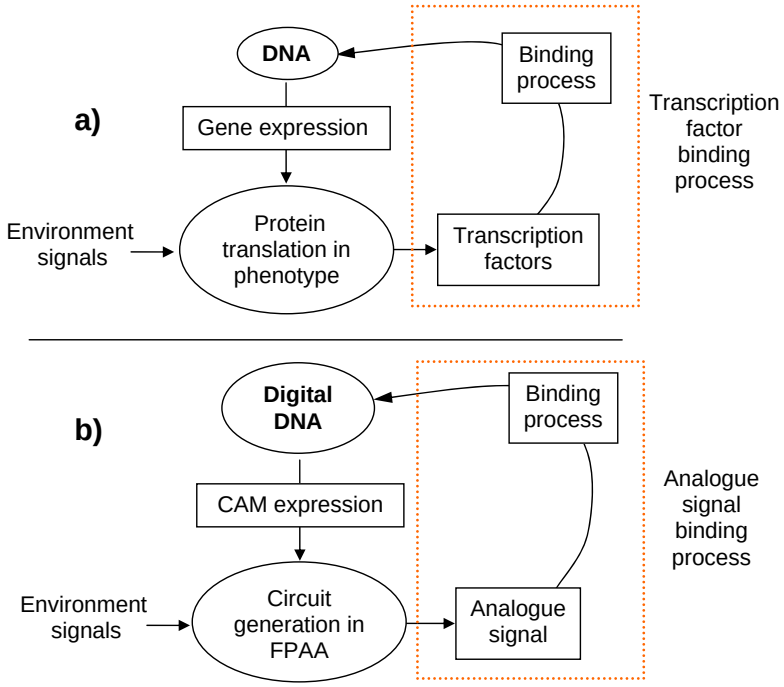


Figure 4.1: Diagram (a) shows natural gene expression where expression is affected by the presence of transcription factors in the cell. (b) A conceptual architecture with reconfigurable analogue hardware as the search domain. In contrast to Fig. 3.1, circuit components (configurable analogue modules, or CAMs) are expressed and used to form a circuit on an FPAA. The changing output of the circuit is used to trigger the next reconfiguration (Clegg et al., 2007)

used to estimate the overall fitness of the phenotype. The analogue output would need to be digitised to allow binding to the digital DNA. But by using the circuit output to provide the context for gene expression, we have a system that not only searches for solutions in given contexts, but one that reconfigures to new solutions when the context changes.

Having discussed our conceptual architecture, the nature of the search domain and the advantages of reconfigurable analogue devices over analogue circuit simulation, we can investigate some hardware platforms. We need to ensure that any device we choose has a sufficiently rich mixture of configurable functionality to make evolutionary search worthwhile. The platform needs to have an accessible API, so that circuit creation can be automated and driven from an evolutionary harness. There are also cost and time constraints for the project. For example, custom-made hardware such as that used by JPL is not an option open to us, and even creating our own development board for an

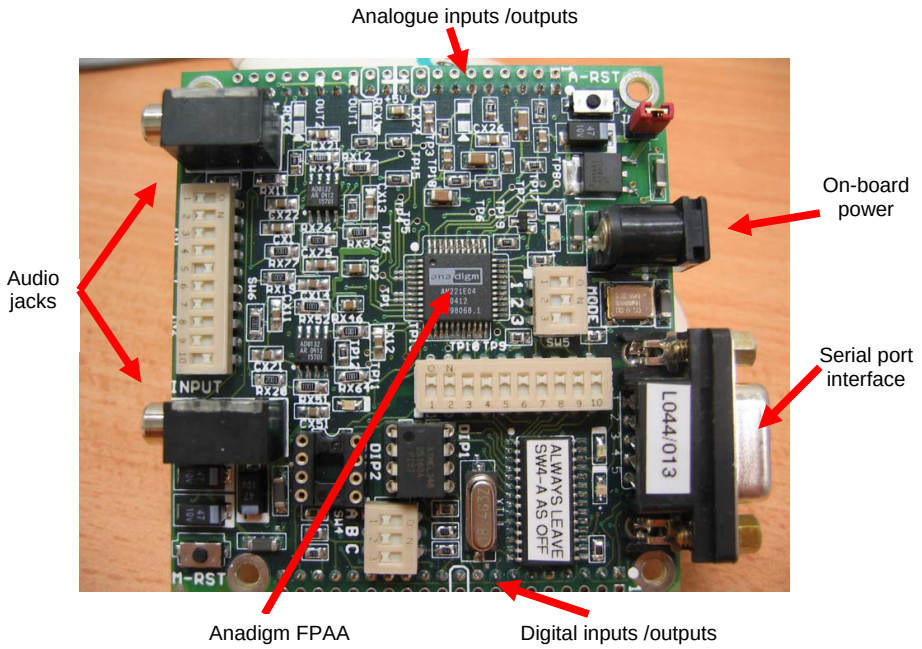


Figure 4.2: The Servenger PAM board as delivered in March 2006. It supports 7 analogue inputs and 2 outputs, and includes on-board power supply, EEPROM to store configuration files and an RS-232 serial interface. The Anadigm AN221E04 FPAA is located centrally and is about 1cm^2 in size.

FPAA would incur time and cost that would be best avoided.

4.1.2 An overview of FPAA technology

In the last decade, a number of silicon-based platforms have been proposed for evolvable hardware (Keymeulen et al., 2004; Stefatos et al., 2006; Stoica et al., 2007). Some of these (e.g. FPTA, FPTA2) have been funded by DARPA or NASA, and as such are not available to researchers outside the United States. However a few reconfigurable analogue devices, including some hybrid forms containing a mix of FPGA-type logic and op-amps Hamilton et al. (1998), are becoming more widely available.[6] After assessing price and the "lead time to delivery" from some manufacturers, it was decided to purchase a commercially available FPAA from Anadigm, pre-mounted on a development board (the Servenger PAM, see Fig. 4.2). Having the FPAA on a development board meant we could start investigating the hardware API immediately, as hard-

[6]NB. "Off-the-shelf" availability does not mean that no export licence from the US is required. Jacyl Technology's AX16 board with 4 FPAAs and an FPGA controller does require a licence. Export licences take between 6–9 months to obtain.

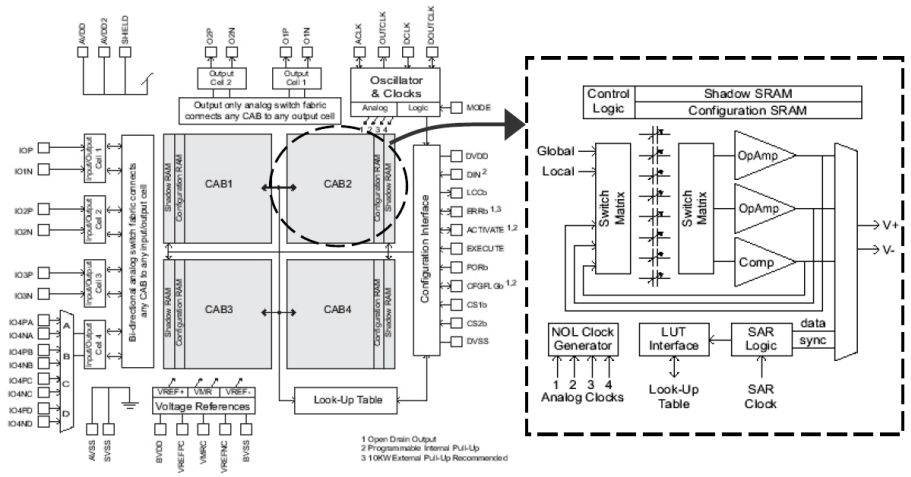


Figure 4.3: Anadigm's FPAA architecture (left), with the view inside a CAM shown on the right.

Image taken from Berenson et al. (2005)

ware interfaces such as on-board power, serial port connection and analogue connection pins were already made.

The Anadigm AN221E04 FPAA is capable of implementing *dynamically reconfigurable* analogue circuits. Its architecture is split into 4 CABs (Configurable Analogue Blocks). Each CAB can support one or more Configurable Analogue Modules (CAMs), the internals of which are shown in Fig. 4.3. Anadigm supplies a range of about 40 CAMs with various configuration options. CAMs provide high-level functional modules that contain circuits functioning as filters, multipliers, integrators, differentiators and so on. Table 4.1 gives a selection of typical CAMs that could be considered for an application conditioning the output of a sensor, such as a photodiode.

CAMs are configured by setting specific options, floating point parameters and clock speeds using the AnadigmDesigner2 software.[7] The software allows analogue circuit engineers to design and test circuits using a "drag-and-drop" graphical user interface. The software can also be controlled by an API "wrapper" referencing the `ad2.exe` file. The build up of circuits and the reconfiguration process can then be controlled by an external application. The low-level configuration commands (i.e. individual elements of the bitstream) are not provided by the API. However, circuits can be saved as AHF (Ascii Hex Format) configuration files, which can be downloaded onto the FPAA as needed. For dynamic reconfiguration during run-time, each circuit configuration file must

[7]This software is free to download from `http://www.anadigm.com`.

CAM function	Options (XOR)	Parameters (floating point)
Differential Comparator	Compare to (Signal ground, Dual input or Variable reference) Input sampling (phase 1 or 2) Output polarity (inverted or not) Hysteresis (0, 10,20,40mV) Output synchronization (none, phase 1 or 2) Synchronisation clock edge (rising or falling)	**VR Reference voltage level** **(8 bit programmable)**
Inverting Differentiator	Output hold (on or off) Input Sampling (phase 1 or 2)	**K, Differentiation Constant (us)**
Divider	Input sample and hold (off or on –input)	**D, Division Factor**
DC Voltage Source	Polarity Positive or negative (3.5volt differential)	
Half Cycle Inverting Gain Stage (optional Hold)	Input sampling (phase 1 or 2)	**G, Gain**
Half Cycle Sum/Difference Stage	Output Phase (1 or 2) Input 1 polarity (inverting or not) Input 2 polarity (inverting or not) Input 3 (Off or On) Input 3 polarity(inverting or not) Input 4 (Off or On) Input 4 polarity(inverting or not)	**G1, Gain** **G2, Gain** **G3, Gain (optional)** **G4, Gain (optional)**
Sum/Difference Stage with Low Pass Filter	Input changes on (Phase 1 or Phase 2) Input 1 polarity (inverting or not) Input 2 polarity (inverting or not) Input 3 (Off or On) Input 3 polarity(inverting or not)	**Fo, Corner Frequency** **G1, Gain** **G2, Gain** **G3, Gain (optional)**
Transimpedance Amplifier	Output Phase (phase 1 or 2)	**Z, Transimpedance (V/uA)**
Square Root	None	**None**
Multiplier	**Input Sample and hold, (Off or input sample and held)**	**M, Multiplication Factor**

Table 4.1: Selection of CAMs that could be used to condition the signal output of a photo-diode or similar sensor (taken from AnadigmDesigner2 software listings).

be saved in advance and then all possible transitions between the configurations determined. Anadigm's proprietary software works out the minimal bit differences required to reconfigure the FPAA between one configuration and the next, so that very fast updates are possible. Unfortunately, we were unable to make use of this feature and relied on a "hard reset" of the FPAA to reconfigure it (see issues, §6.3.1).

In terms of our conceptual architecture (Fig. 4.1), the feature of most interest on the Anadigm FPAA is the availability of high-level functional modules, whose behaviour can change according to their context of deployment (i.e. their inputs / outputs) and be configured by setting options or parameters. These two features allow us to model phenotype exploration by gene expression (permitting gene reuse in different contexts) and genetic exploration by

evolutionary mutation (fine tuning of functional behaviour). To try and make the links between our conceptual architecture and the proposed system architecture more obvious to the reader, the next section briefly recaps the biological mechanisms that control gene reuse during development, and suggests ways we could incorporate an abstract representation of these features into our system architecture.

4.2 Linking evolutionary developmental biology to our system architecture

The purpose of this section is to justify why we have abstracted a small part of the developmental process relating to gene reuse and show we can link elements of our conceptual model of that process to parts of the the hardware architecture of our platform.

We have seen how evolution is able to reuse genes as a by-product of the developmental process (§2.5). By controlling context-specific gene expression, developmental processes can create repetitive morphological features such as bone vertebrae, leaf venation, bristles and limb buds. It was argued in §2.5.1 that the majority of change in morphological form comes not from direct mutation of the gene itself as commonly supposed, but from mutation that causes the context of expression to change. An example is the Distal-less gene. This gene, important for limb bud development, evolved a new "switch" that led to it being expressed on a butterfly wing, where instead of a limb bud the gene makes a spot of colour (see Fig. 2.11 in §2.5.4).

It is vital to capture this context-specific nature of gene reuse in our architecture. Notwithstanding that low-level configuration bitstream mutation is unavailable to us on the Anadigm FPAA[8] (as it had been to Thompson (1997)), using CAMs directly in our representation frees us from the problem of having to evolve and protect high-level functional elements capable of reuse. The formation and protection of re-usable building blocks in evolutionary computation is a recognised and long-standing problem (Walker and Miller, 2008). Having a ready-made selection of CAM "primitives" as building blocks means that we can focus on making their reuse have *context-specific functionality*. Although coming across it long after selecting the Anadigm FPAA, it is interesting to see how closely the following quote from Thompson matches the platform architecture and backs our developmental approach:

> "The fine-grained control over the hardware that is required im-

[8] Directly mutating some of the bits on bitstream configuration files and attempting to download them was tried, but the FPAA failed to recognise any of the attempts made.

> plies a vast search-space for a system of any complexity, so techniques need to be developed to cope with this. One possibility is the use of developmental genetic encoding schemes for genetic algorithms, which allow the evolution of re-usable building blocks ... permitting fine-grained tuning of the building blocks, but yet reducing the search space to systems built out of them." (Thompson et al., 1999)

§2.9.1 argued that most work in the field of evolutionary computation has focused on the *performance* of population based, stochastic search. While performance is interesting from a computational perspective, it does not help address the issues of scalability or reuse. Nor does it help move attention towards new *types* of dynamic, flexible solutions that evolution can find for us. Most evolutionary computation aims at a phenotype representation containing a single, static solution. Neither the inclusion of redundancy in the genome (Miller and Smith, 2006), nor ideas of neutral variation to allow "island hopping" in the search landscape (Thompson, 2002), extend the genome representation to contain multiple phenotypes. Instead, ideas about redundancy have tended to remain at the level of unused genetic material in the genome. However, biological organisms contain an additional form of redundancy: genetic responses that will only be expressed under certain conditions. Often in evidence during development, this flexibility gives organisms room to manoeuvre in a variety of environmental conditions. By tying the expression of genes to particular contexts, a genome can contain *multiple phenotype responses*, including some that might be redundant. Recent developmental work has appeared in a broadly similar vein to this, where Tufte and Haddow model "phenotype plasticity" (Tufte and Haddow, 2007).

When faced with the breadth of examples from biology, it is difficult to see how to map the huge variety of proteins found in the cell nucleus to a digital representation. The problem of representing the richness entailed by different protein identities has been investigated by Bentley (2004b) using fractal expression. As we want to explore *in materio* evolution on an FPAA, we chose not to follow either this approach or one such as Tufte (2006) or Miller (2004), where cell neighbourhoods are modelled. Instead, the specific context provided by different protein transcription factors in the cell nucleus is modelled by transforming the analogue output of the FPAA into a complex digital context, that can trigger gene expression as part of a feedback process (see Fig. 4.1).

Having the FPAA output on its own represent the different contexts for gene expression would not give the evolutionary process the sort of flexibility available during biological development. §2.5.4 described how transcription

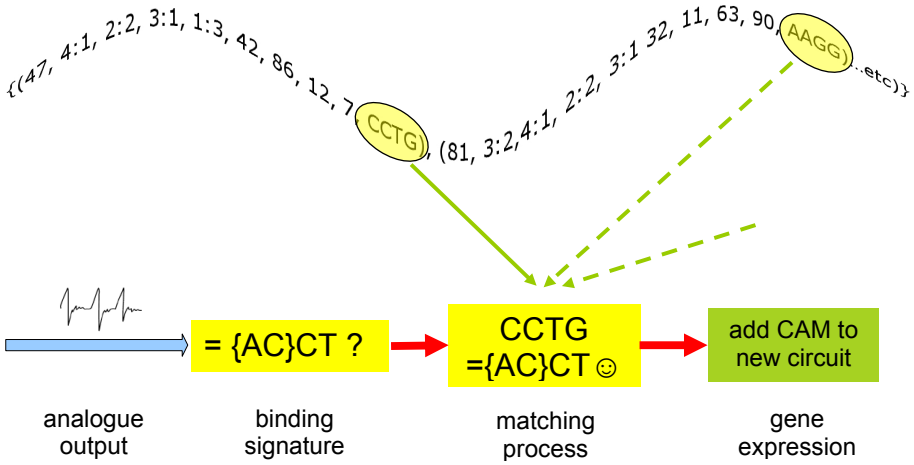


Figure 4.4: Binding works by transforming the output signal of the circuit into a "binding signature". This binding signature is checked against the binding site of each gene. If a match is found, the gene is added to the circuit configuration.

factors bind to stretches of DNA material. The presence of "wildcards" at some transcription factor binding sites widens the opportunities for gene expression. Likewise, the absence of wildcards in some signatures (e.g. the heart development gene, *tinman*) suggests that some genes have a very restricted range of expression.

In order to provide similar flexibility for evolution to exploit, we propose that the transform of the FPAA output maps onto a signature scheme containing wildcard positions. There are various ways of doing this. For example, a Fourier transform shows the amount of power in a signal at different frequencies. Signals containing a mixture of frequencies could be suitably binned to give the desired signature length, with the area of a bin corresponding to one of the 4 nucleic acids. While such a scheme would work, it was noted in discussions with colleagues[9] that Fourier transforms had limitations in terms of the amount of information they could extract from a signal, and a more attractive scheme could be devised that would let wildcard positions appear and allow evolution to manipulate them. Our final binding scheme therefore adopted a wavelet transform instead (§5.5.1 gives further details), with a grid superimposed on the transform to give a binding signature of the desired length (§ 5.5.2).

Fig. 4.4 gives an overview of how the matching process works to trigger gene expression. The analogue output of the circuit is used to create a binding signature. Each gene's binding site (shown by the ovals on the genome rep-

[9]Tim Clarke, Dept. of Electronics, University of York.

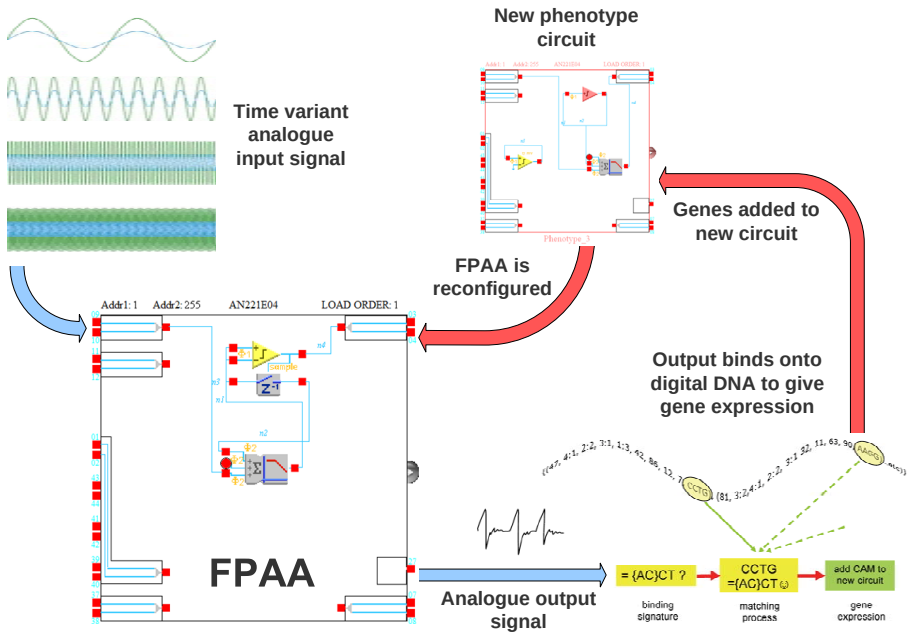


Figure 4.5: The system architecture, showing how circuit output "binds" onto parts of a digital genome, triggering gene expression which manifests as a new circuit configuration.

resentation string) is checked against the binding signature. In the diagram, although the binding sites on the genes are 4 bases long, the signature they are checked against is only 3 bases, therefore only the first 3 bases of a binding site are used. The curly braces in the signature indicated that the first position is a wildcard consisting of A or C. Thus for the example shown, the first gene's binding site would match with this binding signature. When a match is found, that gene is added to the set of genes that are expressed. The configuration represented by these genes is then used to make a new circuit. The matching process is part of the wider reconfiguration process of the chip, as shown in Fig. 4.5 (where Fig. 4.4 is embedded). The signal inputs shown in Fig. 4.5 represent 4 distinct stages of signal input. The varying input (actually an increase in the frequency of the signal) means that the signal output subsequently changes, leading to new genes being expressed and the reconfiguration loop starting again.

4.3 Summary of system architecture

This chapter has listed motives for taking evolutionary search into a design domain containing rich physics and gave reasons for selecting analogue electronics as that search domain. §4.1 argued that searching within virtual environments inevitably means that large areas of the physical design space are not considered. §4.1.1 outlined the advantages of reconfigurable analogue devices and how the platform architecture of the Anadigm FPAA was a good fit to our conceptual requirements. Finally, we discussed how creating a context to trigger gene expression from the analogue output of an FPAA allows us to evolve multiple phenotype solutions, as each phenotype stage comes from the expression of a subset of genes in the genome (§4.2). The binding process that triggers reconfiguration will use a wildcard scheme similar to that found in the binding sites of biological transcription factors, in the hope that evolution can manipulate the appearance of wildcards in a binding signature to its advantage. The next chapter looks at issues that arose during the implementation of our system.

Chapter 5

Implementation

We have covered the biology behind our conceptual architecture and shown why the Anadigm FPAA platform is suitable for a developmental approach, allowing evolutionary exploration of hardware-based analogue circuits, high-level gene reuse and the possibility of using gene expression to control reconfiguration (§4.2 and §4.3). We now cover how we implemented system. The implementation provides one important feature related to our conceptual requirements listed in §3.1, in that our mapping from the genome to a phenotype stage creates a context-specific functionality for the reuse of genes. Context-specific functionality is an important feature of gene reuse and one that means relatively few, very useful genes are responsible for a wide variety of the functional morphology found in natural organisms (see §2.5, §2.10.1 and §4.1.2). The implementation of context-specific gene reuse was fortunately relatively easy to achieve on the platform we have chosen.

This chapter describes in detail how the process of genome encoding and decoding is done, the implementation of the binding signature generation and binding site matching processes, problems relating to inter-process communication and the difficulties that were encountered using the Anadigm API. We begin with an overview of the system components.

5.1 Overview

At the heart of our architecture is an Anadigm FPAA on which we wish to test evolved circuits, as shown in Fig. 4.5. Around that are components that generate analogue signal input to the FPAA and transform its output. The main system components are shown in Fig. 5.1.

Most of our system runs on a single PC containing a National Instruments Data Acquisition (DAQ) card to get the analogue signals in and out of the

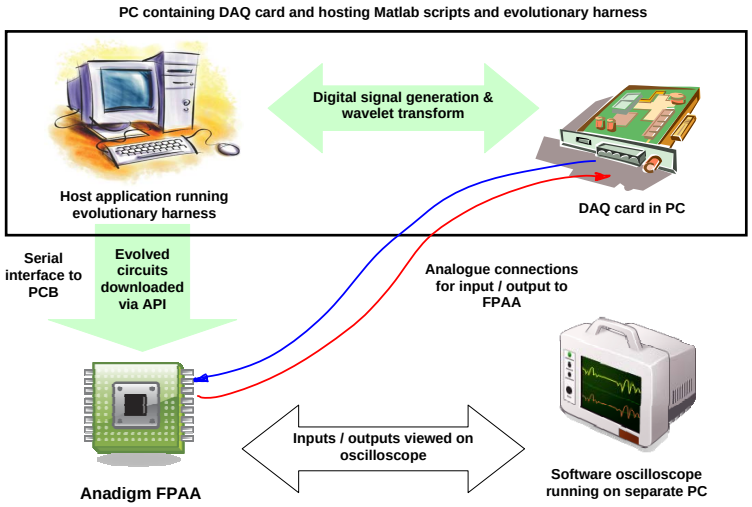


Figure 5.1: Main components of system architecture. Thick black border indicates the PC containing the software API and drivers, the evolutionary harness and the DAQ card.

FPAA. The PC runs the evolutionary process that creates the genome population using the CAM library supplied by Anadigm. This application references the Anadigm API to save phenotype circuit configuration files for each genome. It also references Mathwork's Matlab application, so that a wavelet script transforming the FPAA output can be started and stopped from within the application. The main application was coded in Visual Basic under the .Net framework, using Microsoft's Visual Studio. This environment has excellent support for referencing and accessing APIs under Windows and much better support for hardware interfaces than in the past. Due to the heavy processor requirements of signal generation, sampling and transform, the analogue signal inputs and outputs that were generated and sampled by the DAQ software drivers on the host PC were verified against measurements taken with a software-based oscilloscope running on a separate PC (running the oscilloscope and doing the signal generation / transforms was too much for one PC).

Each genome has a set of possible circuits formed from the combinations of its genes. One gene contains all the information relating to the configuration of a CAM "primitive", that is, a CAM with one of its options selected. A gene encoding also includes input and output connections for the CAM and the gene's binding site.

The evolutionary harness generates every possible circuit that could be made from the subsets of genes in a genome. Each circuit is saved as an ASCII

Hex file (*.AHF) and in Anadigm's graphical format (*.ad2) (see §4.1.2) so that phenotypes can be later viewed. Using the DAQ card drivers, the application begins generating the analogue input signal via the DAQ card to the FPAA. The FPAA output is sampled at regular intervals by a Matlab script (see §6.2.2 and Fig. 6.1). After a short time to allow the signal to settle, the transform of the output (the "binding signature") is matched against the binding sites of "genes" or CAMs (see §5.5.2). CAMs that match will form the next circuit, and the corresponding configuration file is downloaded onto the FPAA. The output signal is again given time to settle, before the new circuit is tested for fitness. Reconfiguration can take place at specific points or on an almost continuous basis.[1] Each input stage can be viewed as a *developmental stage* that the phenotype must be tested against to judge its fitness (see §6.2.2). The sum of all fitnesses gives the final fitness of the genome.

We begin this description of the system implementation by looking at the CAM representations in the genome and the problems of downloading invalid circuit configurations (§5.2). This is followed by a short introduction to Cartesian Genetic Programming (CGP, see §2.8.1), the representation used to describe the circuit connections (§5.2.4). We then step-through the process to decode a genome during the binding process (§5.5.2). This is followed by a closer look at the binding signature generation and matching algorithm (§5.5.2), and the chapter ends with some caveats for anyone planning to use the Anadigm API for similar work and a discussion of the problems encountered with Matlab.

5.2 Encoding the genome

5.2.1 Requirements

Circuits are formed by connecting together CAMs. In order to represent a circuit, we need to hold the configuration of each CAM and how they are connected to each other. We will consider each CAM (or basic configuration option of a CAM) as a gene in the genome. Due to the limited capacity of the Anadigm FPAA, genomes are restricted in length to just a few genes.[2] The mutually exclusive options of each CAM are each given an identification (CAM ID) and can become a gene. Most CAMs can make up to five or six genes this way. With each gene are details of its connections to other CAMs and the parameters that determine its behaviour. Finally, each gene has a binding site associated with

[1] This is dependent on processor speed, but see also the discussion in §6.3.1 on the effects of signal disruption of frequent reconfiguration.

[2] This is a hardware limitation. If we "daisy-chain" together some FPAAs, the genome length could be increased. See §7.4 for issues relating to this.

it, which determines *when* that gene can be expressed. The essential parts of the representation are shown below:

CAM ID	Connections	Parameters	Binding Site

Figure 5.2: A gene in the genome.

The mutation operator can operate on any of the fields. The details of each of the fields and how they are encoded is given below and an example genome is shown in Fig. 5.3. We begin by looking at a typical CAM and its configuration options.

5.2.2 CAM IDs and parameters

A selected list of CAMs showing a representative range of the functionality available has already been given in Table 4.1. We now look in more detail at a typical CAM to consider how the configuration options can be represented, and in particular how conflicts between configuration settings can be avoided.

The options and associated parameter ranges of the **BiQuadLinear Filter CAM** supplied by Anadigm are shown in Table 5.1 which was obtained by assessing the different values available to this particular CAM using the AnadigmDesigner2 software. The table does not show all the options available (e.g. resource usage, low-corner frequency) for reasons of space. A configuration for the BiQuadLinear Filter must lie somewhere along a row of this table if it is to be realised in a valid circuit. Each filter type represents a mutually exclusive option with an associated set of parameters. Initially, each gene in a genome has its CAM ID, connections, parameters and binding site given random values within certain ranges. Although it would be easiest to allow gene values to be freely mutated without regard to allowed parameter ranges, if a parameter is set outside an acceptable value, that CAM may not be instantiated in the circuit.

Unfortunately, Table 5.1 does not show the effect of changing a parameter value on the CAM, so that the ranges of other parameters are forced to accommodate it. For example, selecting the high pass option removes the options of setting the input sampling phase, polarity and resource usage. This means that in order to set the CAM to this filter configuration, one needs to set the filter type, before setting the other parameters related to it. As the user changes the configuration dialogue box, other parameter ranges are also adjusted automatically. For example, the internal clock speed of this CAM can be set to one of

Filter Type	Clock Speed Hz	Input Sampling	CAM Parameters			
			Corner Frequency	**Gain**	**Quality Factor**	**DC Gain**
Low Band	16K	Yes	32-1600	0.4-100	0.24-70	n\a
	4K	Yes	8-400	0.1-100	0.06-70	n\a
	2K	Yes	4-200	0.05-100	0.05-70	n\a
	0.25K	Yes	0.2-25	0.01-20	0.05-70	n\a
High Band	16K	No	32-1600	0.01-1.88	0.05-100	n\a
	4K	No	8-400	0.01-7.5	0.05-100	n\a
	2K	No	4-200	0.01-15	0.08-100	n\a
	0.25K	No	0.5-25	0.01-8.84	0.4-100	n\a
Band Pass	16K	No	32-1600	0.1-56	0.0125-46.9	n\a
	4K	No	8-400	0.01-14.1	0.05-100	n\a
	2K	No	4-200	0.01-7.07	0.1-100	n\a
	0.25K	No	0.5-25	0.01-1.41	0.5-100	n\a
Band Stop	16K	No	32-1600	0.4-100	0.24-70	0.01-1.88
	4K	No	8-400	0.01-7.5	0.06-70	0.1-100
	2K	No	4-200	0.01-15	0.12-70	0.05-100
	0.25K	No	0.5-25	0.01-11.8	0.6-70	0.01-20

Table 5.1: There are a wide range of parameter settings for the BiQuadLinear Filter CAM. However, many of them are affected by different clock speeds. Parameters such as corner frequency are relatively consistent, other like Gain vary more widely. As explained in the text, each of the parameter settings may be linked to another parameter value, so changing one may change the range available to another.

five values. If the clock speed exceeds the corner frequency parameter, that parameter becomes highlighted in red.

The flexibility embedded in the graphical interface is difficult to reproduce as a flat text representation. Rather than have all possible options available within a CAM representation, it is easier to treat the mutually exclusive option as separate genes. Thus the BiQuadLinear Filter CAM would have four main variants (Low Pass, High Pass, Band Pass and Band Stop), with further sub-variants (related to input sampling phase type, polarity and resource usage), with each variant having additional floating point parameters (such as corner frequency or gain) that further alter its behaviour. In the case of the BiQuad-Linear Filter, this gives a total of nine "primitive genes", whose behaviour can be fine tuned by mutation.

With this scheme, there are just over two hundred CAM "primitives". [3]

[3]The actual number is larger. However, for reasons involving the design of the API it easier to

Within many primitive configurations, a large range of behaviour is possible as most CAMs take floating point parameter values. We encode a maximum of four parameters, with those that are unused being ignored. To get round the dynamic nature of parameter range adjustments, a parameter encoding is generated as a percentage value, which is applied as a function of the parameter range after the configured CAM has been added to the FPAA (as it is possible that the presence of other CAMs on the chip can alter the available parameter range). Unfortunately, as each CAM has to be added to a circuit before its parameter adjustments can be checked, it makes circuit generation too slow to be performed dynamically during run-time (it takes around twenty seconds to generate the circuits for a genome with three genes). Instead, each circuit configuration is saved in advance as an *.AHF file, which can be then downloaded as needed. It should be noted that generating the circuit configuration files is made slower by the error handling required to avoid connection conflicts or bad parameter values. As an example, to generate circuits for a population of one hundred genomes results in fourteen hundred files being written to disk, taking around thirty minutes (see also §5.6.1).

Having described representing CAM configuration, the next section looks at the technique we use to specify the connections between CAMs.

5.2.3 Connecting CAMs together

A circuit made from connected CAMs can be viewed as a network of connected nodes. In order to maintain the connection details between CAMs, the representation needs to be able to hold the maximum number of input connections a CAM could have. This means the representation will have a large degree of redundancy, as few CAMs have high numbers of inputs. Inevitably, many mutations to the genome may then have no effect on the phenotype stages they represent. Network models often use feed-forward, directed graphs to represent information flow. Electrical circuits, however, can have feedback loops. These form an important part of analogue circuit design.

However, the purpose of this thesis is not to discover the most efficient evolutionary algorithm or representation. What is important for us is to see whether our conceptual architecture can bring something new to evolutionary computation. For our representation, we needed an established technique, one that had both proven results as a search algorithm and one in which it would be easy to represent a network of functional nodes. Cartesian Genetic Programming (CGP) fulfils these requirements. In addition, its originator Dr. Julian

build up the CAM library omitting some of the option configurations.

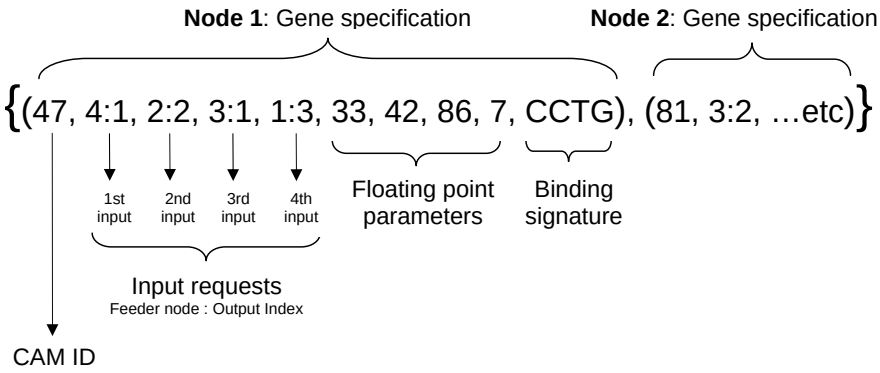


Figure 5.3: A simple CGP representation that caters for the maximum number of inputs and parameters that are available to a CAM. If a CAM has fewer inputs or parameters, the remaining values are not used (they may still be subject to mutation, however, see §5.2.4).

Miller,[4] kindly offered his help with any implementation or performance issues we might encounter.[5] Importantly for us, the method had already been used on an *in materio* evolvable hardware project (Harding and Miller, 2004, 2003) — the experience of which we were later grateful for when noise in our prototype made mutation rate and fitness assessment difficult. For a reminder of the background to CGP, we refer the reader to §2.8.1. We now give a short description of how we adapted it to implement our requirements.

5.2.4 Encoding circuit connections in the genome

CGP represents nodes connected in a feed-forward, directed graph. Our version is adapted to allow feed-back loops, as these form an interesting and important part of analogue circuitry. Our integer-based representation is shown in Fig. 5.3 and explained in detail below. While in theory a genome can contain any number of genes, the eventual size of a circuit depends on how many genes are being expressed at that moment. Fig. 5.4 shows the process of decoding the genome, from being fully specified (all genes expressed) to a circuit containing just a couple of CAMs.

The genome representation is the part subject to mutation and its length is defined by the number of nodes. As in CGP, the number of inputs to each node is fixed. In our case, as the maximum number of inputs on any of the pre-configured CAMs supplied by Anadigm is five,[6] we use this and later trim

[4] Dept. of Electronics, University of York.

[5] Particular thanks must also go to Dr. James Walker, with whom I had several discussions about possible representations for circuit configuration.

[6] NB. Fig. 5.4 shows only 4 inputs for clarity.

the excess inputs as needed. All five inputs are required to be satisfied under CGP. An input request generates a feeder node ID and the output index of that node. Thus in the example given in Fig. 5.3, the first input of Node 1 (CAM 47) requests the first output of Node 4. The second input requests the second output of Node 2, and so on. When this Node 1 is translated into a CAM, its variant of the BiLinear Filter CAM (CAM 47) turns out to have only two inputs, so the remaining input requests are ignored. The encoding links each input to an output (either from another node or itself). It is possible that one output on a node is linked many times and another not at all. In this case, the unlinked output is simply left "open". However, the process ensures that every input is connected and no inputs are left open.

Once the directed graph has been generated, it is necessary to translate the node into the specifications of the corresponding CAM. This means that most nodes have their inputs trimmed to the number on the corresponding CAM. For example, in Fig. 5.4, Node 2 has four inputs and one output, but the CAM corresponding to its ID (CAM 81) has two inputs and two outputs. Both of Node 2's inputs (shown in the specification as 3:2, 3:1, etc) are linked to the first and second outputs of Node 3. However, CAM 116 that Node 3 represents, has only one output, therefore this output is used to feed *both* of Node 2's inputs.[7] For Node 2's outputs on the other hand, only Node 1 makes a request for the second output. No other node requests input from the first output of Node 2, and so that output is left "open" (unconnected). The process of re-aligning inputs and outputs in the specification to the actual values on the CAMs means that although all inputs are connected, about 20% outputs are likely to be left unconnected.

5.2.5 Binding Sites

The final part of a gene specification is the binding site for each CAM. This is a string representing the four nucleic acids in DNA (e.g. AAC, TGT, etc). As a single FPAA can fit around four to seven CAMs depending on their resource usage, the binding site length is generally proportional to the expected expression rates, i.e. choosing a very short binding site might result in the expression of too many CAMs for most contexts. A future project could allow evolution to choose the binding site length, but in our current prototype this is fixed. §5.5.2 describes how binding sites are matched to the binding signature and covers in detail the generation of the binding signature itself.

[7] This is one example of an "arbitrary" design decision in the wiring algorithm. There are several others, in particular those relating to errors raised by the Anadigm API when attempting incompatible connections (see §5.3 and §5.6.1). If you wish to check specific details, Appendix §A.2 lists the Visual Basic code for the wiring algorithm.

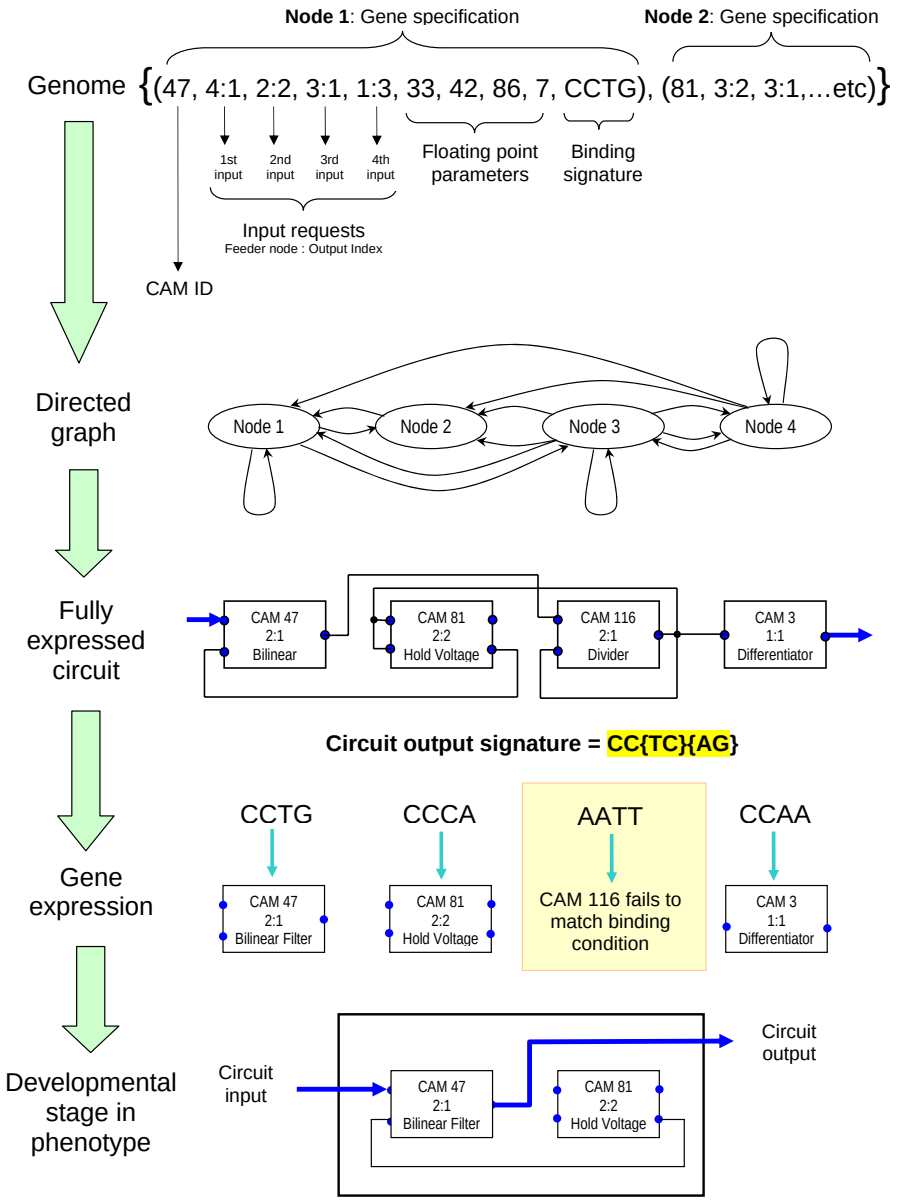


Figure 5.4: Genome to phenotype: decoding a genome into a developmental stage of the circuit phenotype. Circuit output is fed back to the genome which reacts by expressing those CAMs that match the binding signature, allowing the genome to respond to changing conditions by expressing a new phenotype stage. See §5.3 for full description of how a subset expressed from the genome is connected together.

5.3 Expressing a subset of genes

The previous sections detailing the genome have described how CAM configurations are represented, how a directed graph satisfying 5 input requests from each node is generated and how that directed graph is then modified to match the actual number of input requests that the CAMs can make.

The next part of the process involves the expression of genes from the genome. The directed graph contains the connection details for a fully expressed genotype (all CAMs expressed). However, there are many circuits that can be created from a subset of CAMs in the genome. Each subset is turned into a phenotype circuit configuration using the wiring connections of the fully expressed circuit. As each subset is generated, many with requested connections from missing CAMs, the circuit wiring specification from the genome no longer holds good for all inputs. For example, Fig. 5.4 shows a binding condition in which Node 3 (CAM 116) fails to be expressed. This leaves Nodes 1, 2 and 4 to create a circuit configuration. Unfortunately, Node 1 has only one output specified as being connected to Node 3. As Node 3 has not been expressed in this phenotype, the circuit is effectively split in two.

In cases such as these, the wiring algorithm defaults to having the first CAM take the circuit input, regardless of whether that means much of the circuit is unused. The wiring algorithm then traces through the circuit to the first "open" output that will enable the circuit to have both an input and output. In this case, the first open output it finds is on the first CAM, and so the other CAMs (Nodes 2 and 4) fail to have an impact on the circuit. The final phenotype circuit translation is shown with only Node 1 (CAM 47) making up the circuit. Node 2 (CAM 81) has been left in as a connection can still be made to the first input. Although no input goes to Node 2, the connection may still affect the phenotype fitness and we do not judge how "sensible" a circuit is from the viewpoint of conventional design.

Appendix §A.2 lists the code for the wiring algorithm and shows how each circuit is first connected following the full genome specification, ignoring nodes that have not been expressed. If this is unsuccessful, a search is attempted for suitable circuit inputs and outputs. It should be noted that despite the strategies employed in the algorithm (including localised error handling to avoid dropping out of object collection loops), it is sometimes not possible to connect an input or an output to the circuit, resulting in a dud circuit. However, we discovered in our experiments that evolution still made use of such circuits (see §6.7 for examples and discussion).

The decision to use an algorithm to try and create some sort of valid input and output to a circuit was taken after it was realised how few good circuits

Part of Gene	Mutation Rate
CAM ID	1/19 = 5.2%
Input Requests	10/19 = 52.6%
CAM Parameters	4/19 = 21%
Binding Site	4/19 = 21%

Table 5.2: Probabilities for gene configuration fields being mutated.

were created if the circuits were employed exactly as they were specified but with missing CAMs. Even with the wiring algorithm fixing up inputs and outputs, there a large number of circuits that cannot work due to broken connections. The wiring algorithm acts the same way during the construction of every circuit phenotype, so that although it admittedly "interferes" with the specification selected by the evolutionary harness, it does so in a consistent fashion and might be considered in the same light as restrictions on protein formations that occur in cell nucleus due to the cell chemistry or physics (see also the discussion at the end of §5.6.1).

5.4 Evolving the genome

When the initial population is generated, the gene configurations are created by applying random numbers (within specific ranges) to each field of a gene. For example, for CAM ID, a random number between 0–209 is generated. For an input request from another node, a number between 0 and the total number of genes in the genome is generated. CAM parameters and binding sites are generated the same way. After a genome has been selected as the fittest of its generation, it is cloned and mutated.[8] One mutation per gene is carried out. The fields are stored internally in an array and each field has an equal chance of being mutated. As some parts of a gene have more fields than others (e.g. input requests), there is a greater chance of these fields being mutated. CAM IDs have the least chance. However, where a mutation falls does not have the same impact from gene to gene, and clearly some mutations (such as CAM ID) will have greater impact than one on a redundant field (such as an unused input or parameter setting). Table 5.2 gives the likelihood of a mutation falling on the parts of a gene configuration.

Maintaining a fully expressed circuit wiring specification means that in the case that all CAMs in a genome are expressed, all inputs will be satisfied. How-

[8]This is assuming a 1+4 evolutionary strategy. See §6.5 for details on population sizes and mutation policy in the experiments.

ever, as there is a "patching up" process for circumstances in which full expression does not occur, it allows us to investigate how evolutionary progress is affected by the interaction between the context of expression and the binding site mutation of individual CAMs. If a CAM has connections specified as coming from another CAM and that CAM is never expressed in the same context, evolution must mutate their binding sites to allow both CAMs to be expressed simultaneously if those connections are to be realised in a higher fitness for that phenotype stage. However, a later mutation to a binding site may have a greater effect than simply allowing the expression of the missing CAM in that context. The new CAM may mean connections are possible that change the behaviour of the original CAM. This is particularly important for our conceptual architecture, as the expression of the same gene in different contexts should not result in predetermined circuit behaviour. Rather like the reuse of genes in different developmental contexts, under our scheme, different binding signatures are likely to result in the same CAM behaving differently, as it will be deployed in different circumstances.

This highly non-linear mapping from genome to phenotype gives several benefits. For example, it allows us to investigate phenotypic *stages* of development, as a phenotype may get high fitness in one context, low in another, and we can investigate what that means for phenotype exploration. We can assign progressively difficult tasks for contexts, and look to see what "useful" genes are conserved to allow the genome to "bootstrap" over generations. We can also alter fitness criteria for a particular context after a genome has already achieved good fitness for other contexts, and see what impact this has on the re-organisation of the genome. These interesting aspects of the architecture are discussed further in §7.4.

5.5 Search Domain feedback and response

To access the complexity and richness in the design domain of analogue hardware, we need some means of processing the output of that environment digitally so that it can interact with evolutionary processes on the host application. The first part of this is decoding the genome into phenotype circuits. The second part is the binding process that allows the genome to receive feedback to say when to express certain genes from the phenotype's functional domain of analogue signals. Translating the physical complexity of the analogue output signal from the FPAA into a digital form is done via a wavelet transform. As explained in §4.2, alternative schemes for this transform were considered, the primary other possibility being a simple Fourier transform (FT). While FTs are

fast and easy to implement, they extract a limited amount of information from a signal. Plus our original plans had been to use this architecture on devices such as remote sensors that needed to adapt to their environments. In such applications, the signals coming from the sensors are not stationary waves such as sine waves, but are more akin to changing levels of DC voltages. Changes in this type of signal would be best picked up by a transform process such as a wavelet transform, rather than an FT. Unfortunately, we were unable to test our FPAA system using real world sensor signals, and so the extra effort to implement the wavelet transforms did not provide us with substantial benefits. The process of matching the circuit output (i.e. the "context") to a set of a binding sites on the genome is done separately and is covered in §5.5.2.

5.5.1 An overview of wavelets

Wavelets grew out of the requirements of a group investigating new techniques for locating oil from the return of impulses applied to the ground (Hubbard, 1998). The return signal was cluttered. To process it properly required the ability to resolve simultaneously in time and frequency. The Fourier Transform (Soliman and Srinath, 1998) defined in Eq. 5.1 provides excellent resolution in the frequency domain, but time domain information is lost. Truncation of the signal produces conflicting artifacts (spectral spreading and leakage) related to the length and shape of the time window over which the signal is expressed.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{5.1}$$

Where i is the imaginary unit from Euler's formula[9], $\omega = 2\pi f$ (angular frequency) and t is time. There are methods for improving this trade-off between frequency and time. One such method is the Short Time Fourier Transform (STFT) Poularikas (1995), given in Eq. 5.2:

$$STFT_f(\omega, \tau, \gamma) = \int_{-\infty}^{\infty} f(t)\gamma(t-\tau)e^{-i\omega t} dt \tag{5.2}$$

The STFT gives the frequency components within different windows defined by the windowing function $\gamma(t-\tau)$, where τ is the time index. This function allows the frequency distribution within a time band to be measured. Whilst this is an improvement on the time-frequency resolution of the Fourier transform, it is still limited because of the window size. The smaller a window is, the better a high frequency component can be located in time. However, with smaller windows few cycles of low frequency components will be

[9]See explanation of this and $-i\omega t$ on Wikipedia at http://en.wikipedia.org/wiki/Wavelet.

observed and so there is a loss of resolution (Hubbard, 1998). Wavelets are designed to remove these artifacts. We can use them to analyse the output from our analogue circuit without distortion. The continuous wavelet transform, $W_f(a, \tau, \psi)$ where a is the wavelet scale, effectively "cuts up data or functions or operators into different frequency components, and then studies each component with a resolution matched to its scale" (Daubechies, 1992):

$$W_f(a, \tau, \psi) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\psi\left(\frac{t-\tau}{a}\right) dt \tag{5.3}$$

The wavelet transform provides a time-frequency localisation that is not available using the STFT. By calculating the correlation between the desired signal and a wavelet function then scaling, dilating and shifting the wavelet function before repeating the process, an accurate representation of the frequency variability with time can be built up. The transform repeatedly compares the similarity between the wavelet and the signal under inspection at different dilations and shifting positions.

In general, the energy contained within a signal, the average of the square of the signal, will be preserved by the wavelet transform and so the original signal can be recovered. The function $\psi(t)$, the "mother wavelet", is unscaled, undilated and unshifted. The dilation is inversely linked to the frequency band that the wavelet will detect. At a high dilation the wavelet is stretched over a wide time period and so is more suited to low frequency signals. At low dilations the wavelet is compressed into a smaller time frame and so will pick up high frequency signals. The term, a, in the wavelet transform is the dilation. In addition the a factor ensures that the energy in the transform is normalised (Poularikas, 1995).

There is a wide range of wavelets that have been created and each has varied properties. When performing the wavelet transform a critical factor in obtaining an adequate result is that the appropriate wavelet function is selected. As our circuit output was based on sine wave input, we used the "Morlet" wavelet (Daubechies, 1992). The next section walks through the binding process, starting with the wavelet transform, the Frobenius normalisation[10] and finally conversion to a binding signature. The signature is then matched to binding sites on the genome to create the next circuit reconfiguration.

[10]The Frobenius norm is a submultiplicative matrix norm based on an inner product on the space of all matrices. We use the Mathworks implementation for calculating this norm on the wavelet coefficient matrix.

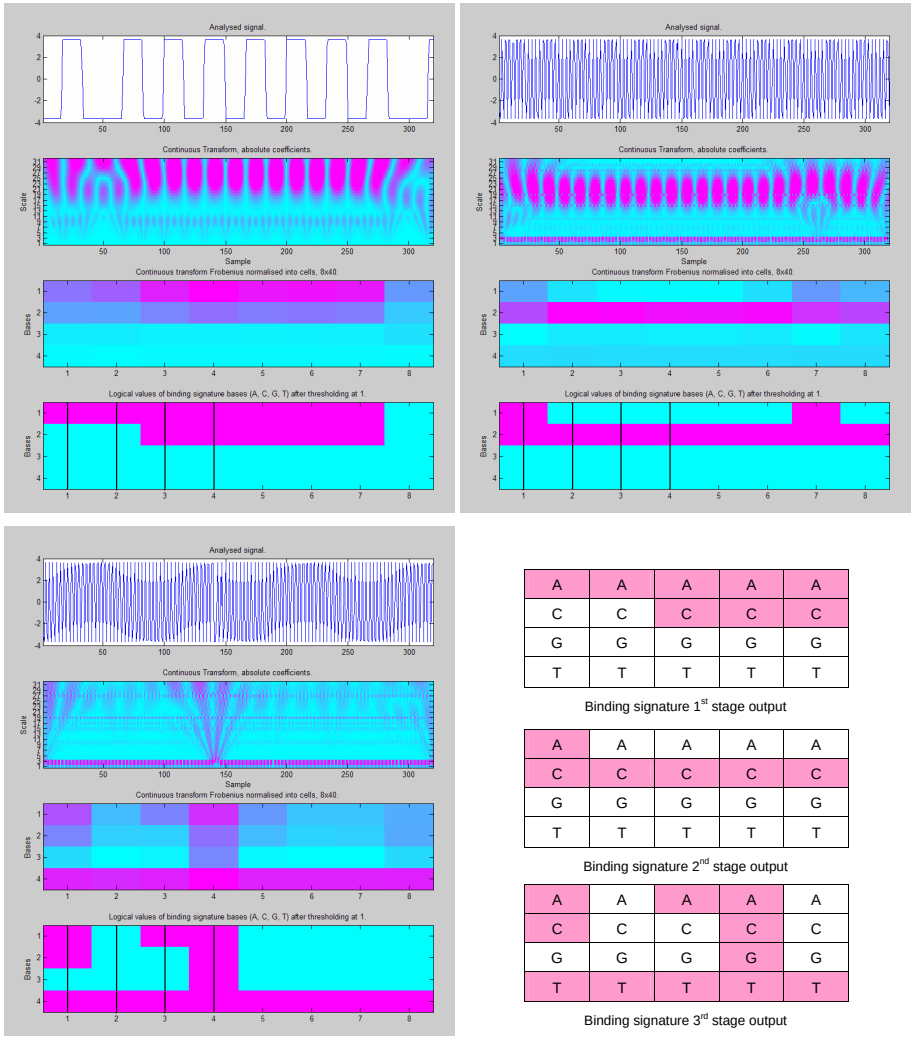


Figure 5.5: Three different circuit output behaviours analysed as wavelets using Mathworks' Wavelet Toolbox. In each picture, the top plot is the output signal of the circuit, below that is continuous wavelet transform, followed by a grid formed by averaging the intensity of the correlation values (wavelet coefficients) into cells using the Frobenius norm. Finally the same cells are compared against a binary threshold mask. The dark vertical bars in the last plot of each figure show binding signature positions (in this case, we are only reading the first four positions as the signature length=4). The nucleic acid values (A, C, G, or T) are read across from the Y-axis. More than one value on a vertical bar gives wildcards in that position. The bottom right picture shows each binding signature in detail. The three outputs give the following binding signatures, with wildcards shown in curly braces:

1^{st} output — AA{AC}{AC}{AC}
2^{nd} output — {AC}CCCC
3^{rd} output — {ACT}T{AT}{ACGT}T

5.5.2 The binding process

We take the wavelet transform of the FPAA analogue output and create the binding signature using the three stage process shown in each of grey boxes in Fig. 5.5. Each box shows circuit output for a sine wave input (1kHz, 5kHz and 10kHz respectively), each of which generates a different output and therefore a different binding signature. Each box shows the sampled analogue output of the FPAA on the top plot (see lines 10–48 in the Matlab script A.1). Below this is a continuous wavelet transform. This plots time (shifting) along the x-axis, frequency (dilation) along the y-axis, and the degree of correlation between the dilated and shifted wavelet and the signal at that time as an intensity value. The continuous wavelet transform was performed by a function called twice a second (see Appendix, lines 53–82 in A.1). This was as rapidly as our PC hosting the application could run a continuous transform (see problems with lag in §5.6.2).

On top of the continuous wavelet transform plot, we overlay a grid representing the 4 bases (A,C, T, G) as four rows (see lines 99–135 in listing A.1). The grid works almost as a truth table, in that we inspect the Frobenius norm of the correlation values (wavelet coefficients) in a grid cell (third plot from top) to see if it falls above a threshold or not. If the answer is positive, that square equates to that base being present at that position in the signature (bottom plot). The grid is read column by column. More than one value in a column means that this signature has a wildcard at that position. If no square in a column is above the threshold, then that position is ignored. As many positions are read as required to match a signature. The signature length can therefore be extended to meet the number of CAMs in the genome as needed. The Matlab script used to generate the wavelet and Fourier transforms is given in Appendix §A.1, and output logs of the entire process of creating circuits and testing the phenotype expressions of a generation can be viewed in Appendix §A.3.

Matching the circuit signature to binding sites

The matching algorithm simply compares the two strings; the binding signature and the binding site. If a CAM binding site matches the signature, it is added to the list of CAMs that will make up the next circuit configuration. The algorithm always compares all binding sites in the genome to the binding signature. This means that although the wiring algorithm generates all possible circuits from subsets of the genome's CAMs, the binding sites of some CAMs can mean that some subsets will never be expressed. For example a genome such as that shown in Fig. 5.6 may have three genes, two of which have the same binding site. Even though a genome like this has seven possi-

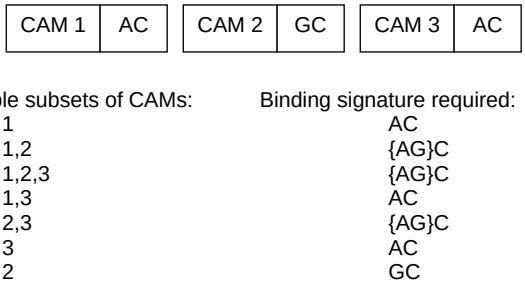


Figure 5.6: Simple three gene genome with binding sites, showing possible circuit subsets and the binding signature that would be required to express them.

ble circuits,[11] some subsets will never be expressed, as the binding signature they require is part of a bigger subset. In example given, both CAM 1 and CAM 3 require AC as a binding signature to be expressed. As the algorithm matches all CAMs where possible, if AC is the binding signature, both CAM 1 and CAM 3 get added to the list to make up the next circuit, and neither can ever be expressed as a circuit on their own. This effect is even more dramatic if the signature were to contain the wildcards AG in the first position, in which case every subset matches, but only one will be expressed (123), as this is the largest.

Even from such a small example, it is clear that the presence of wildcards has a major impact on which circuits are expressed. Generally, the presence of wildcards favours larger circuits, because more genes will be expressed. This is not to say that it has an effect on the fitness of the phenotype being tested, but it might have an impact if hardware measurements were being included as part of the fitness tests, such as power or resource usage. The effect of wildcards has been difficult to assess in the time period of the project. It would certainly be interesting to explore if the presence of wildcards in signatures narrows the problem of "subset explosion" as the length of genome increases (see §7.4, for further discussion).

5.6 Implementation Issues

The final section in this chapter looks at some implementation difficulties of using hardware, such as the Anadigm FPAA, as a platform on which to conduct evolutionary search. It also covers some of the issues with application lag,

[11] All possible subsets $= 2^n - 1$, where n is number of genes. One is subtracted as the empty set is ignored.

memory leaks, signal sampling and wavelet transform we had with Matlab.

5.6.1 The Anadigm API

Despite the Anadigm design libraries containing high-level functional blocks that fitted well with our conceptual requirements, the random mutation driving the search gave rise to many ineligible circuit configurations that caused unforeseen problems, often mid-way through an evolutionary run. The Anadigm API was created to allow automated circuit generation by third party software, and as such, offers flexible and well-thought out interface. The API is built using the .Net platform, and so its executable can be referenced on projects using a variety of languages (C++ , C#, VisualBasic and Javascript). Although the means of accessing the API differs in each language, its underlying interface is the same. If requested, documentation is available from Anadigm for the Automation API at no cost, although the company makes clear that it will not support applications using the Automation API, neither does it guarantee the accuracy of the documentation.

One quickly gets the flavour of this, as it is clear the documentation provides code snippets that were written for an earlier version of the software. The Automation API comprises objects and collections (groups of objects). After instantiating an object, one can view its members using the "dot notation" and Intellisense™ feature in Visual Studio. This makes for very quick exploration of nested object interfaces, even though some object methods may have no associated documentation.

The use of collections is one aspect of the API that can cause problems. These are in effect dynamic arrays, and as such access to particular objects in a collection is never guaranteed if one references that object by its index value in the collection. Instead, one is forced to loop through the collection using Microsoft's "for each *object* in collection ... next *object*" construct, checking each in turn. While there are sort and find interfaces to collections that can speed up access to a degree, in large collections the procedure of looping through each object can seem an unnecessary overhead. This is particularly true when attempting to wire up incompatible CAM inputs or outputs. For example, as a CAM contact can be either an input or output type, one needs to instantiate the CAM, then its Contacts collection, then each member of that collection, before the type of a contact can be checked (it is not possible to assume the first object in the contacts collection will be the first or main input connection). Thus the wiring algorithm quickly gets complicated. For example, as each CAM's Contacts collection has to be looped through to find its input contact type, one needs a second set of loops to check each of the CAMs to which it connects to

try and instantiate the connection.

This "looping within loops" means that error handling needs to be very localised. Being thrown out of loop, for example, because of an attempt to wire incompatible contacts together, means that you lose the chance to check the remaining contacts. The only way round this is to have errors caught within each loop so that they are not raised to the subroutine level. The code for the wiring algorithm is given in Appendix §A.2 and shows the process of looping through object collections and the localised error handling required to ensure the remainders of collections are still searched after errors are raised on unacceptable connections.

Even so, there are some errors that are not handled correctly inside the Anadigm API, against which special provision has to be made. These errors get raised to the highest level they can and may even require the routine to be aborted, causing that phenotype to fail to be downloaded onto the FPAA.[12] The API also provides different levels of warning. Warnings can generally be ignored, as they seem to relate to accurate simulation of the circuit within AnadigmDesigner2. As these often occur when parameter ranges on the CAM's options are exceeded, either because of incorrect clock speed or some other parameter setting has reduced the original parameter range, it is still worth trying to download a parameter value that has raised the warning. Sometimes it may lie just outside the declared parameter range for simulation, but is still acceptable to the hardware.

Where errors prevent a connection being made, a decision has to be taken in software whether to abort the attempt at creating a circuit or to continue. An example is attempting to assign the FPAA input. Initially the algorithm tries the first input of the first CAM, as would have been specified on a fully expressed phenotype. If this fails, it tries to see if there is an "open" input connection on the same CAM it can use. If this fails, it looks for the next "open" input connect in the circuit. Finally, it tries to connect to the last CAM in the circuit, even if its input is already connected. If this fails the circuit is undoubtedly a dud as no input signal can enter the circuit. However, it is still saved as a configuration — as there are cases where even dud circuits can score high fitnesses (see §6.7). But what is important to note is that the connection attempts are made as a result of errors being raised by the Anadigm software. These connection attempts were not part of the genome specification. The same situation occurs trying to connect the circuit output. So despite trying to avoid the limits that a software simulation of hardware would impose, the example

[12] One instance involved a modal error message box being raised from the API. This could not be handled from the main application, and halted all execution during an evolutionary run until a user clicked "OK" to remove it. However, after raising the issue with Anadigm, they graciously sent a specially compiled version with the modal error message removed.

illustrates how error handling affects circuit implementation and consequently the design exploration by phenotypes. However, a more positive way to view the wiring decisions is that they act as a kind of arbitrary "physics" or "biochemistry" that governs some connection rules: the rules are consistent, they always act in the same way and they add a layer of complexity above that given by gene expression, much as real physics and chemistry act to control protein behaviour.

5.6.2 Matlab scripts and processes

One unfortunate result of choosing a wavelet transform of the output signal on which to base our binding signature was that there are relatively few wavelet libraries that we could use. In particular, the DAQ card drivers that would have integrated well with the main application under the .Net framework did not offer wavelet transform as part of their signal processing package.

Originally we intended to use Matlab's Signal Acquisition Toolbox to both generate the input signals to the FPAA via the DAQ card drivers, and to sample the output. It was argued that Matlab would be much more flexible with regard to signal generation than the DAQ card drivers. What was not apparent was the amount of processing and memory Matlab would require in order to perform continuous signal sampling and wavelet transform.

The initial design had a Matlab m-file script running in parallel to the main application, which both generated sine wave input to the FPAA and and sampled the output signal to perform a Fourier transform. The input was divided into three stages: each stage having a different frequency. The Fourier transform was used as part of the fitness criteria in later tests (to establish the amount of power in the output signal). To do this, we needed to write the fitness scores and binding signatures to disk midway through an input stage. The main application read the files and summed the results to create a fitness score for the genome. However, it became clear that the burden of performing signal generation, signal sampling and subsequent Fourier and wavelet transform was too much for a single processor. In addition, Matlab itself seemed to suffer a memory leak, causing a processing lag which led to out of date fitness scores being written to disk and being picked up by the main application. As our gene expression is a dynamic reconfiguration processes, the two applications needed to run synchronously to work. The script was sent to Mathworks for suggestions on how to cure the lag, however the problem was never resolved and the only solution seemed to be to use the DAQ card drivers to generate the signal input from within the main application.[13]

[13]This proved an advantage, as it was possible to control very closely when signal generation

We continued to use Matlab to sample the output to perform the Fourier analysis and wavelet transform. The principal advantage to this is that Matlab's native data format is a 2-dimensional array or "matrix", therefore it proved trivial to compare the wavelet transform to a thresholded binary mask to create the binding signature. Again this was written to disk and read by the main application. As the output was sampled and the signature written every half second, there were rare occasions when file access clashed between the two applications. Rather than write extensive error checking to work out at what point in a test this occurred, we simply restart the test for the phenotype (see listing of application output, in particular line 528 in A.3).

The memory leak in Matlab continued to cause us problems on long evolutionary runs, to the extent it would eventually crash the machine as it used up all the available RAM.[14] In order to avoid this, it was decided to start and stop the Matlab process from within the main application. The Matlab executable was instantiated as an application object within Visual Basic, which meant it could be shut down and restarted as required. Unfortunately, the Matlab application object is instantiated as a DCOM server object under Windows, so that the `quit()` method on the Matlab object does not kill the process completely. It remains in RAM and is used next time the application object is started in the main application. This meant the memory leak continued, even with `quit()` being called on the object. Our only recourse, after several weeks working with Mathworks to try to cure the problem, was to explicitly kill the process using Windows system calls and restart it. Again, this slowed down the testing of phenotypes as a new instance of Matlab would have to be loaded into RAM before signal generation and testing could start.

However, the memory leak was the last major hurdle to clear and meant we were finally able to execute evolutionary runs lasting two to three days (the amount of time required to evolve 50–60 generations).

5.7 Summary

This chapter covered the implementation of system architecture. It related in detail those aspects of the implementation that provided parts important to the conceptual architecture, such as the context-sensitive reuse of genes / CAMs under different conditions of expression. It described the main components of the system architecture, how the software elements were built and how decisions relating to connecting CAMs together and the choice of wavelet trans-

would start and stop, and to re-do any test where fitness scores failed to be written to disk in time.

[14] Mathworks technical support were unable to provide a fix for this. The full script used is given in Appendix A.1 (Matlab ver. 2006b).

forms were motivated and implemented. It finally records, as a caveat, some of the issues we had trying to use the Anadigm API provided by the AnadigmDesigner2 software under Microsoft's .net platform and Visual Basic. We also describe the performance problems of continuous wavelet transform using Matlab's Wavelet Toolbox. The next chapter gives the details and results of the experiments we ran using the system described.

Chapter 6

Experiments

The previous chapter discussed the implementation of the conceptual architecture. Once glitches and memory leaks in the software had been ironed out and the system was stable enough to run for several days, we were able to conduct preliminary experiments. This chapter covers the design of the experiments, aspects of search space coverage and the problems of hardware noise. It gives the encouraging results of tests against a hypothesis looking at evolved phenotype behaviour through several developmental stages. The prototype demonstrated that the evolved mechanism can control progressive developmental stages (canalisation) or provide homoeostatic control in the application.

6.1 Overview

6.1.1 Aims

The broad aim of these experiments is to test the hypothesis of the thesis (§3.3). However, the experiments themselves focus on a specific hypothesis relating to the performance of evolved phenotypes in different environments. As this is too specific for the wider scope of the thesis hypothesis, we use this section to show how the system exhibits the desired characteristics of our conceptual architecture (Fig. 3.1). The experiments themselves are detailed in §6.6. To the best of my knowledge, the experiments are entirely novel. I have not found similar work related to assigning fitness at intermediary stages of the development of a phenotype, or work that uses gene expression to move from one developmental stage to the next, where those stages and the movement between them have been configured by a single evolutionary search.

Earlier chapters premised that a developmental approach could help evo-

lutionary computation overcome difficulties relating to *scale* and *reuse*. This led to the design of our conceptual model and the implementation of a prototype system; one of few in evolutionary computation that tie the evolution of gene expression to phenotype exploration. Given the results of our prototype, there are encouraging signs that the exploration of developmental stages could be expanded over longer time spans (i.e. the execution time of the phenotype would be extended, allowing more developmental stages to be passed through). While there are questions relating to the frequency of reconfiguration over very short time intervals (§6.8.1), provided the system is given time to settle, there should be no restriction to the number of developmental stages a phenotype can be assessed against, bearing in mind the increase in time to reach a solution. Although it was not possible to conduct experiments on our prototype to prove it (due to the limited capacity of the FPAA), the ability to extend the number of developmental stages a phenotype adapts to indicates that the system has a degree of *scalability*.[1]

The second characteristic of the architecture we want to highlight is the *reuse* of genes in different contexts. The previous chapter showed how the advantages of a non-linear mapping from genotype to phenotype could provide *contexts* for CAM expression. The context of a CAM's expression means that the same CAM is likely to provide different circuit functionality when expressed with other groups of CAMs, despite retaining its evolved configuration in either group. These two factors controlling CAM behaviour (evolutionary configuration and context-dependent functionality) correlate well with the flexibility inherent to biological gene reuse and its context-dependent functional morphology.

Gene expression allows developmental stages in the phenotype to explore dynamic environments. As environments change, different genes are expressed or inhibited allowing the phenotype to adjust its mode of exploration. While we wished to evolve a genome capable of demonstrating dynamic response to changing input, we also wanted to show how a single genome could evolve *a set of adaptive behaviours* to different environments. This interesting aspect of phenotype plasticity is discussed in the experiment results given in §6.7.

Even at the onset of experimentation, several unknowns remained in the experiment design. For example, how often should gene expression be allowed to reconfigure a phenotype? If we divide phenotype lifespan into multiple stages, how should a phenotype be tested for fitness? The latter question is one we will return to in our final chapter. However, for now, we take a simple, if naive, analogy from biology to justify our method.

[1] Increasing the number of developmental stages is only one aspect of scalability, see also §5.5.2 and § 7.4 on potential difficulties relating to scale.

In biological terms, the only measure of fitness applied to phenotypes is successful reproduction. However, there are clearly selection pressures applied to the developmental stages of a number of animals (for example, the specialist camouflages of caterpillars, the eggs of ground-nesting birds, even the behaviour of many animal young). The degree these developmental stages contribute to the overall fitness of the organism is obviously difficult to determine. But we feel high performance in these stages must contribute to the phenotype fitness to some degree, and used this as the basis of our approach. The method of assigning fitness is described in detail in §6.2.2.

Method *The behavioural requirements of the task are broken up into developmental stages, to which we apply selection pressure by awarding fitness scores to each stage. The sum of fitness scores give the overall phenotype fitness.*

Not unexpectedly, as the number of stages (or environments) a phenotype must adapt to goes up, the time to reach a solution increases. But it appears from our experiments that rather than an exponential increase in the total size of search space that one might expect, the increase is linear, as one is in essence examining repeated instances of the same domain. Secondly, and perhaps most interestingly, not all developmental stages need be weighted equally in terms of their overall fitness contribution. If desired, stages can have their fitness scores weighted so that the most important behaviour is targeted at the expense of other stages. This ability to guide evolution was not one we were fully able to explore, but is discussed speculatively in §7.4.

6.1.2 Summary

Our experiments intend to illustrate how:

- a single genome can evolve to meet different fitness requirements at different developmental stages;
- gene reuse can provide different functionality when the context of expression changes;
- a genome can respond dynamically to changing conditions.

The next section looks at the types of task that our system could perform to fulfil these aims, followed by a description of the task chosen. We detail how initial experiments (§6.3) had issues relating to noise (§6.3.1) and address the question of effective search space coverage (§6.3.4), including binding site (§6.3.4) and phenotype variability (§6.3.4). We then examine the evolutionary strategy used in the experiments (§6.4) and the experiment hypothesis (§6.6). A discussion of the results concludes the chapter (§6.7).

6.2 Tasks

6.2.1 Task requirements

The requirements of the system architecture (shown in Fig. 4.1) suggest the task requirements should include:

a) a series of changing analogue inputs;

b) a series of signal processing tasks that Anadigm CAMs on the FPAA can perform;

c) a means of measuring the fitness of circuits.

While this would seem to offer a broad scope for potential tasks, we are limited by the signal processing possible on this size of FPAA and the time / difficulty of constructing a reliable test environment. Unfortunately this prevented us from carrying out our initial proposals to look at the real-world signal conditioning of sensor output (such as a photo-diode responding to changing light — see §4.1.2). However, we were able to set up a task for the prototype using computer generated analogue input which fulfilled the above criteria. The advantage to having input signals created by the host application was that we had precise control over how an input signal would change and could more easily track sources of noise in our system. As the DAQ card came with drivers for generating analogue sine wave output, it was decided to use a series of different frequencies as input to the FPAA. While it could be argued that wavelet transforms may not be the most efficient method to analyse stationary waves (wavelets are better at detecting *changes* in signals), it was easy enough to generate frequencies sufficiently far apart for the transform to clearly distinguish binding signatures.

Having decided on the FPAA input, we looked for a task which would be easy to gauge in fitness tests. It was important that the task could be achieved in a number of different ways so that the evolutionary runs were not attempting to find rare solutions in a large search space. Having more than one solution also meant that successive runs could be compared for aspects such as novelty and robustness, and the fitness tests perhaps tweaked to encourage solutions in those directions.

The task chosen was to maximise the power of the output in some input stages and to minimise it in others. The task was split into three (i.e. requiring three fitness assessments): maximise the amount of power in the signal in the middle stage, minimise it in the other two (see §6.2.2). However, even with this simple task and set of input stages, assessing the fitness of the phenotypes proved surprisingly deceptive (§6.3).

6.2.2 Task description

The prototype has set two primary objectives:

1. to evolve reliable reconfiguration;

2. to configure the FPAA in stages according to different fitness criteria.

No requirement was set regarding the time to reach solutions to the task or the resources used. Three different sine waves were selected as input stages (1kHz, 5kHz and 10kHz), each input being generated for a fixed duration. The fitness criteria were to minimise power output from the chip on the 1kHz and 10kHz stages, and maximise power output during the 5kHz stage.

Each phenotype starts against the output of a 'bare wire' (i.e. a straight connection between input and output), so that for the search to move anywhere, a phenotype must reconfigure the FPAA. Not reconfiguring the FPAA incurs a penalty (see below). A signal frequency is input for 5 seconds to allow the system to settle, after which the wavelet transform is converted to a binding signature and checked against each gene binding site. Each match is added to a list of genes that will be expressed to make up the next circuit. Reconfiguration takes place, while input continues for a further 5 seconds, allowing the system to settle again. Finally, a Fourier transform gives a power reading for that circuit which is converted into a fitness score. The input is then changed to the next frequency step and the process repeated, or a new test started. Fig. 6.1 shows how the system evaluates an input stage. Appendix A.3 (lines 357 onwards) show the output logs as the FPAA is reconfigured after binding during input sequences.

The fitness scores are based on a formula that moves the selection process in the direction of circuits that maximise power in the middle frequency band. To the best of my knowledge, this work is the first that assigns intermediate developmental stages a fitness score, and so the derivation of the fitness formula was arrived at through practical consideration and experiments:

$$Fitness = (B + R) - (A + C) \tag{6.1}$$

where B, A and C are the power readings at mid, lower and upper frequencies respectively. R is the reconfiguration bonus. As mentioned previously, the phenotype must initially respond to the output of a bare wire that passes the input signal through unaltered (except for a slight reduction in voltage). If no reconfiguration occurs, the power readings return negative fitness values. Due to noise in the system and a result of elitism in a 4+1 evolutionary scheme, phenotypes that do nothing (i.e. fail to reconfigure) can get selected above those

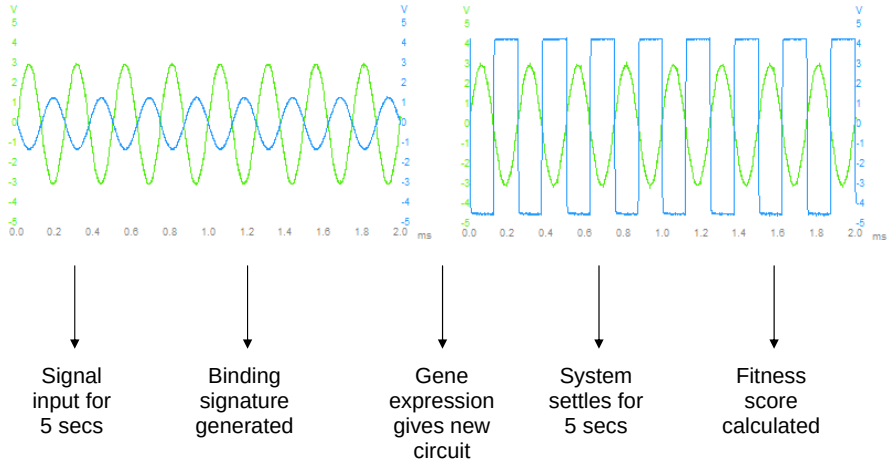


Figure 6.1: Sequence of binding and fitness evaluation for an input stage (output of FPAA is blue / darker line).

that reconfigure badly if power readings are used exclusively as the basis of selection. As this imposes an unacceptable delay for the search process to 'get started', we penalise any phenotype that fails to reconfigure from the initial setting. Conversely phenotypes that reconfigure — even to a bad circuit — have their fitness scores augmented by a small bonus (usually less than 10% of final fitness score). On runs for random sequence frequency steps, more reconfigurations are required so bonuses are accordingly reduced (see §6.7). The addition of a reconfiguration bonus gives some indication of the problems that are typically encountered when creating fitness measures. Fitness can be notoriously difficult to nail down, and if care is not taken, evolution will blindly follow a path your experiments may not have intended.

6.3 Preliminary Experiments

Building a system such as ours requires us to use the Anadigm API and FPAA in ways it was never intended. In a similar fashion, the use of Matlab to run signal sampling from which we calculate continuous wavelet and Fourier transforms is also unusual. We encountered problems both building the system (as described in §5.6) and during our evolutionary runs. We now look at the initial experiments that helped us overcome issues we had with noise, search space coverage and hysteresis.

6.3.1 Noise

There are three main areas where noise affects the assessment of phenotype fitness. The first was noise during wavelet transform. The second was minor inconsistencies in the Fourier transform readings for phenotype fitness. The third source of noise was related to fitness score fluctuations that came from signal disruption, caused either by hard re-sets of the FPAA during reconfiguration or by stepping up the input frequency. In addition to noise, the final part of this section also looks at the problem of hysteresis — an unexpected phenomena linked to particular CAMs at the moment of deployment.

Noise during wavelet transform

During wavelet transform, wavelet coefficients measure similarity of the wavelet to a signal as it is time shifted at different scales. Part of our binding process converts wavelet coefficients into a four-row grid of binary values determined by a threshold (§ 5.5.2). The more noise present in a signal, particularly a very weak signal, the more likely it will have coefficients scaled over the threshold. This translates into columns with more than one value — giving "wildcards" for that binding position — so that more genes are expressed as a result (see §5.5.2 for an explanation).

During evolutionary runs, no attempt is made to ensure gene expressions make valid circuits. It is possible that circuits may be broken (no output is produced) or circuit output may be severely restricted or otherwise made noisy. In these cases, as coefficients are scaled, the noise introduced into the binding process results in *wildcards being produced for all positions*. An unexpected effect of this is that a particular circuit can be expressed by a "lucky" binding signature (i.e. one brought about by random noise, due to a previous stage's poor circuit). Although this new circuit may get a high fitness score, the binding signature that caused the circuit to be expressed might never be repeated. Elitism results in the evolutionary process being 'conned' into keeping and mutating the poor solution for many generations. The mutations never make successive phenotypes improve above the lucky fitness score of the original phenotype, as the binding that allowed the good circuit to be downloaded occurred by chance, leaving the evolutionary process stranded on a false peak of high fitness.

This raises the question of whether a 4+1 strategy with elitism is suitable for noisy environments, as fitness assessments can be led astray by random fluctuations.[2] Crossover, or similar strategies that gain variety from a remainder of the population, might avoid this pitfall. However, given our motivation for

[2]Pauline Haddow has mentioned to me that her group in Trondheim had similar problems with 4+1 in noisy environments.

using small populations (we needed a fast method to create a generation, due to the amount of time needed to create the circuit configuration files) and our stated aim of not wanting to demonstrate an optimal search technique (§5.2.3), we decided to continue with the 4+1 strategy but resorted to taking the median of several tests per phenotype. The method is described in §6.3.3. There is evolutionary computation research commenting on evolving solutions in noisy environments, but what we resorted to here was a simple practical measure to get past the problems mentioned. We do not compare our approach with others or claim it was a good solution to the problems — it was simply an pragmatic measure we took that enabled us to continue with our experiments.

Noise in Fourier transforms for fitness calculations

A second source of noise in the system comes from the Fourier transforms at each frequency step. These readings are susceptible to fluctuations due to variations in heat and interference from surrounding electrical and computer appliances. Power readings of the same circuit configuration will therefore always vary from one test to another. The fluctuations are small but can vary by as much as 40% (particularly if a test run starts with the equipment "cold"), although the figure is generally closer to 10% depending on the CAMs involved. The effects of this variation was reduced by testing each phenotype several times. After initial runs, we were able to improve accuracy by taking the average of three readings close to the frequency of interest. This was initially defined as a band of 200Hz centred either side of the frequency being measured, but was later reduced to 100Hz to give greater accuracy. However, it appears impossible to completely eradicate variation in the Fourier transform and the fitness scores of phenotypes tested several times in succession will always vary to some degree. But generally speaking, the variation witnessed was not greater than 10% of the final fitness score (see A.3 for examples of variability).

Signal disruption

The third source of noise is caused by signal disruption and settling after a reconfiguration done using a "hard reset" on the FPAA. As mentioned briefly in §4.1.2, dynamic reconfiguration without disrupting the signal was not possible under our scheme, as it would have meant calculating all possible circuit state transitions and generating the configuration "bit differences" that would allow the very fast updates required for dynamic reconfiguration. Instead, our reconfiguration method was to send a "hard reset" to the FPAA and then to download the configuration bitstream. This gives a "jolt" in the signal output,

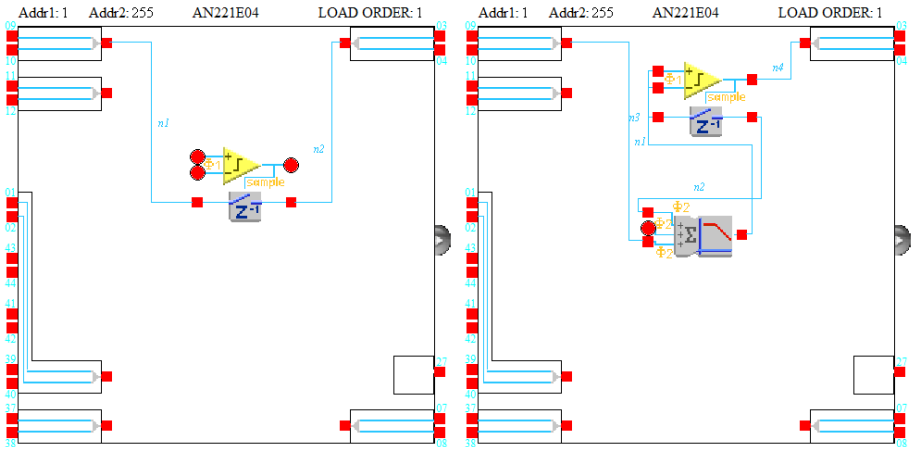


Figure 6.2: The circuits deployed by an unreliable champion phenotype. Left circuit (HoldVoltageControlled) was downloaded at 1kHz, the circuit on the right (with SumFilter added) was downloaded at 5kHz and 10kHz.

part of which is sometimes picked up in the sampling period for the wavelet transform. There is a similar, smaller "jolt" as each frequency increase step occurs on the input, as the signal generation is briefly halted and restarted at the new frequency. In either of these cases, when the binding process is given a free hand to reconfigure rapidly, or the frequency steps are small and numerous, evolution struggled to find genomes with high levels of fitness. Some speculative assessments for this are given in §6.8. In order to minimise the effects of signal disruption, input stages were increased significantly in length to give the system time to settle. Fig. 6.1 shows how the signal transforms and measurements were made, with time to settle in each case after a reconfiguration of the FPAA.

6.3.2 Hysteresis

Aside from these examples, a more serious source of fluctuations in our phenotype fitness was discovered after the first week of preliminary runs. Despite the degree of expected noise, some champion phenotypes would not reproduce anything close to the scores they achieved during the evolutionary run. For example, a champion phenotype that had scored 4130 at the midpoint of one run, was re-tested five times at the end of the run. It scored successively -1067, -1036, 300, 4096, -1993 (the total range for fitness scores is generally around -2000 to 4500). On closer inspection, it was noticed that despite the huge variation in fitness scores, the phenotype was reconfiguring using the same circuits at each frequency step, therefore the variation could not be due to noise

in the binding signature. Two circuits were being used by the phenotype (see Fig. 6.2) . The tests were repeated with the circuits loaded separately to observe their behaviour at each step. Neither circuit produced the high power rating in any of the frequency steps. In fact the circuits seemed inert and produced no output at all. Each of the circuits was then downloaded while input signals were being put through the chip. At first nothing happened, but then at random, the switch in one circuit latched producing a high power output. Once downloaded the circuit reversed the switch at high frequency so that no output signal was produced at that frequency and no reconfiguration was required to produce a high fitness score. The reason for this behaviour lies in the fact that CAMs such as those shown in Fig. 6.2 exhibit hysteresis. Certain CAM behaviour depends on the input to a comparator (GainSwitch, GainPolarity, etc.). If the two inputs of the comparator are connected together then even small amounts of noise are going to lead to the CAM behaviour 'flipping' according to the coincident moment the circuit is instantiated with circuit input. The eventual state of circuits such as these is therefore impossible to predict.[3]

6.3.3 Tackling noise and hysteresis

Rather than exclude CAMs that exhibit hysteresis from the pool of "primitives" that evolution could select from, and to try and overcome the effects of noise in the other areas described above, we decided on a general strategy to test each phenotype five times, and to take the median result for our fitness scores. This strategy, despite increasing the length of each run fivefold, had the desired effect of reducing the worst variability due to noise or unexpected CAM behaviour. The results can be seen in Fig 6.3, where the heavy lines are the champion phenotype fitness scores, while the light lines are the best of a generation fitness score. Runs that tested each phenotype once show best of generation scores varying considerably from the current champion. Taking the median of five tests gives best of generation scores much closer to the current champion, and produces final solutions that are more stable (in terms of their behaviour) and which show earlier gains in fitness. The decision about how many tests per phenotype should be run were based on practical time constraints. More tests would have given us greater accuracy, but we felt the results using five tests per phenotype were good enough (i.e. stable solutions were found in a reasonable amount to time) to allow us to proceed.

[3]Our thanks to Dave Lovell of Anadigm for this explanation.

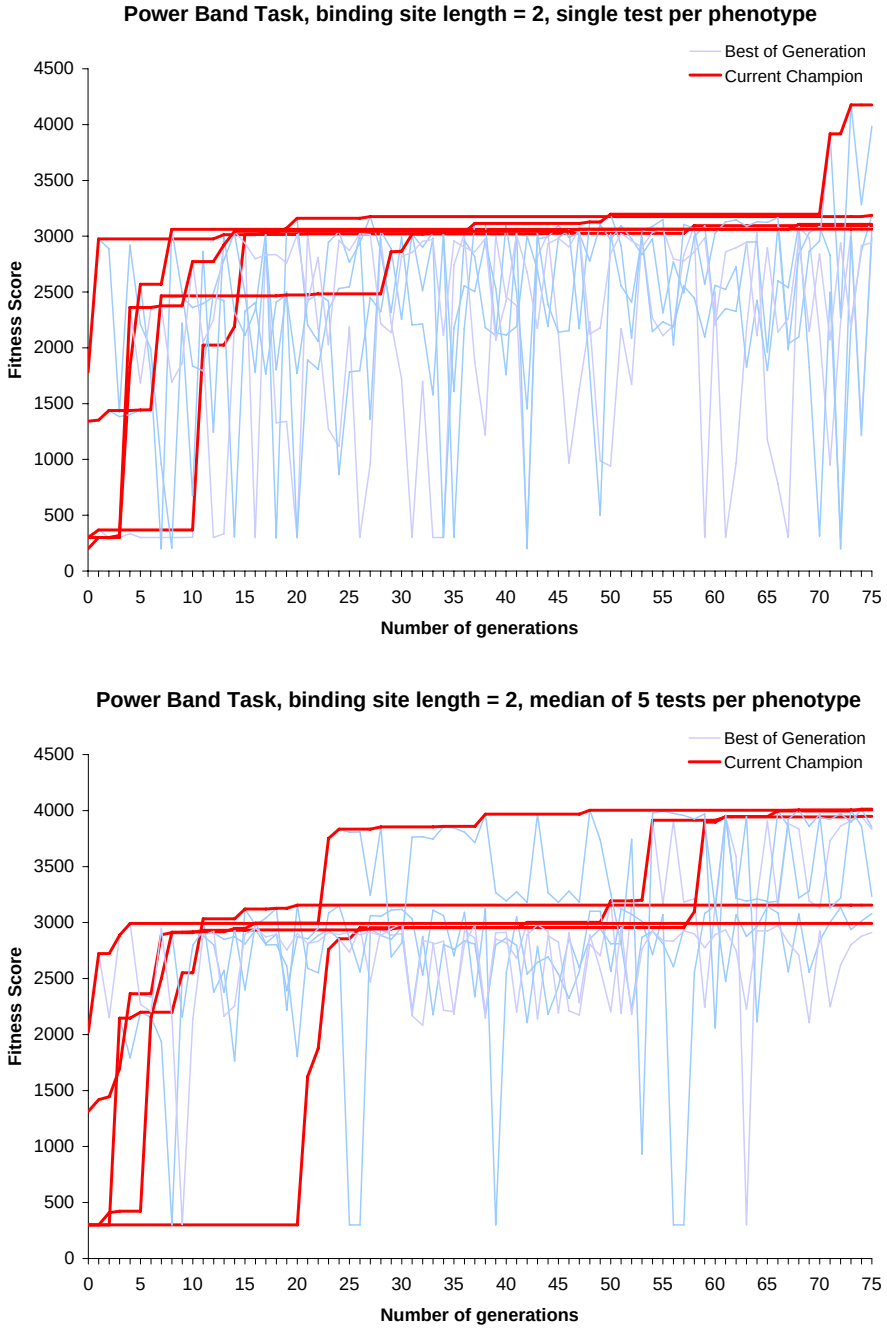


Figure 6.3: Graphs showing five typical evolutionary runs using a binding site of length of two: one test per phenotype (top) and median of five tests per phenotype (bottom). The charts show the reduction in noise (shown by the pale lines) when running five tests per phenotype. The closer the best of each generation score (shown by the pale lines) is to the current champion score (bold lines), the more stable and reliable are the phenotypes.

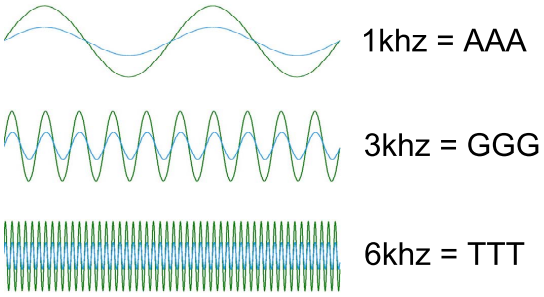


Figure 6.4: Input frequencies and their binding signature (length =3) outputs on bare-wire circuits.

6.3.4 Search space coverage

In order to check that a search algorithm is effectively covering a search space, tests need to be done to verify that the stochastic process is producing the degree of variability required in the population. In our case, there are two main factors with regard to phenotype exploration.

1. Mutation within the gene configuration;
2. Mutation on the gene binding site.

These are listed separately due to the fact that unless a gene is expressed at some point in development, it can have no impact on the phenotype, regardless of its gene configuration. The mutation rates for gene configuration were given earlier in §5.4. However, with regard to the mutation on binding sites, it is instructive to look at the chances of binding occurring given random generation of a binding string.

Binding site variation

Fig. 6.4 shows the binding signatures generated (length = 3) if a bare wire is attached from the circuit input to its output at the stated input frequencies. In order for a gene to be expressed in the initial input stage, it must have a binding site corresponding to AAA. If no reconfiguration occurs during the first stage, then it will need to have a gene with a binding site matching GGG and so on. Once reconfiguration has taken place, the remaining bare wire signatures are irrelevant as the output is then determined by the current circuit configuration.

Table 6.1 shows the results of a randomly generated population of 100 genomes, each with three genes and each gene having a binding site of three

No. of bases matching in binding site	Binding signature			Total
	AAA	GGG	TTT	
All 3	7 (7)	0 (0)	9 (9)	~5.3%
1st 2	18 (18)	6 (5)	32 (22)	~18.6%
1st	77 (58)	70 (52)	86 (63)	~78%

Table 6.1: Results from a sample of 100 genomes, showing the percentage of randomly generated binding sites matching the initial bare-wire configuration given in Fig. 6.4. Total percentages calculated to one decimal place and represent total out of 300 genomes (100 genomes for each column).

bases. The first figure in a column represents the number of genes that were found to match the signature (out of 100 genomes), the figure in brackets gives the number of genomes (as a genome may contain more than one gene with a binding site matching AAA / GGG / etc.). The first row in the table gives the figures for genes that matched all 3 bases of the signature (e.g. AAA). The second row gives the figures for binding sites matching the first 2 bases of a signature (e.g. AA*) and the last row gives the figures matching just the initial base (A**). Based on these figures, the final column gives the approximate percentage of genes that would be expressed given binding signatures of 3, 2 and 1 base in length (the total percentage is therefore out of 300 genomes, and calculated to one decimal place).

While not an exact match, we can see that the stochastically generated numbers broadly tally when we calculate the probabilities. If we take the probability p of base A being generated in the binding site of a gene as:

$$p = \frac{1}{4} \tag{6.2}$$

Given the generation of each successive base is independent of the previous result, the probability of a binding site being AA is p^2, and for AAA is p^3. So we can write:

$$p_a = \left(\frac{1}{4}\right)^n \tag{6.3}$$

where p_a is the probability of the binding site being all As and n is the length of binding site. This means that the probability of a binding site not being all As is:

$$\neg p_a = 1 - p_a$$

$$= 1 - \left(\frac{1}{4}\right)^n \tag{6.4}$$

To work out the probability that at least one gene g in a genome contains a binding site of all As, we can subtract the probability that no gene in a genome has a binding site of all As. As we know the probability of a binding site not being all As is Eq. (6.4) for one gene, the probability of at least one gene in a genome of more than one gene is that probability raised to power of g, where g is the number of genes in the genome:

$$p_g = 1 - (\neg p_a)^g \tag{6.5}$$

Substituting in Eq. (6.4) gives the general formula:

$$p_g = 1 - \left(1 - \left(\frac{1}{4}\right)^n\right)^g \tag{6.6}$$

where n is the length of binding site and g is the number of genes. Thus for binding sites of length two and length three, and for genomes of three genes, the probabilities of having bindings sites with all As in at least one gene is:

$$\begin{aligned} AAA &= 1 - \left(1 - \left(\frac{1}{4}\right)^3\right)^3 \\ &= 1 - \left(\frac{63}{64}\right)^3 \\ &= 0.057 \\ &= 5.7\% \end{aligned} \tag{6.7}$$

$$\begin{aligned} AA &= 1 - \left(1 - \left(\frac{1}{4}\right)^2\right)^3 \\ &= 1 - \left(\frac{15}{16}\right)^3 \\ &= 0.177 \\ &= 17.7\% \end{aligned} \tag{6.8}$$

These are sufficiently close to the figures shown in Table 6.1 not to cause concern, i.e. the margin of difference over 300 genomes is under 1%, therefore we can conclude the degree of variability in the binding signatures is working as it should — see statistical test for significance later (§ 6.7). However, it is worth noting the low probability of generating all three bases of the binding signatures for any of the initial bare-wire stages, particularly as we use very small

populations for a 4+1 strategy (i.e. 5% of four genomes is generally zero!). This realisation led us to "seed" initial binding site generation with a higher proportion of the expected bases of an initial stage (see §6.5 for details). Of course, merely generating a binding site that would allow a gene to be expressed does not mean that the circuit will be useful or even valid, but without an initial expression of some sort, no progress can be made.

Phenotype variability

One concern of using high-level functional modules as "primitives" for evolution to "fine tune", was that the level of functional granularity might be too coarse or too fine, and this might have a detrimental effect on the search for novel circuits. As explained in §5.2.1, the primitives are generated from mutually exclusive CAM options, and there may even be sub-variants of those. While it is difficult to make a general statement about the degree of difference in functionality between one option on a CAM and another, it became clear after some initial runs that CAM IDs in successful genomes were remarkably stable across many generations.

There are various possibilities for this. One is that many CAM primitives perform broadly similar functions on the type of input we were using (sine waves of different frequencies). So mutation to another CAM primitive would be unlikely to significantly raise fitness. In a similar vein, CAM ID may not be as critical to fitness as might be supposed and that binding sites may play a more prominent role. Another possibility was that the frequency of CAM ID mutation was too low (1/19 per gene) to have an impact on small populations and relatively short evolutionary runs. This latter possibility was one we needed to ensure was not impinging on the phenotype's ability to explore the search domain, and so we correlated some runs showing levels of fitness over generations against a plot showing the CAM ID mutations in the "best of each generation" genomes over the same period.

Fig. 6.5 shows a typical example. The top plot shows the variation of CAM ID mutation from the best of each generation over 75 generations. Each genome had three genes. The lower plot shows the level of fitness achieved by the best of each generation and the current champion by this run over the same period. Vertical bars have been placed over both plots at points of significant jumps in fitness. The top plot gives some idea of how stable CAM IDs are, despite mutations occurring in the best of each generation. The mutations to CAM ID seem to show relatively good search space coverage. This is shown by the degree of coverage of each of the CAM ID lines. When a CAM ID has not been mutated in a generation, the line remains flat. If one follows a particular

gene, such as G2, it can be seen how its CAM ID goes through a period of turbulence between generations 23–27, which corresponds to the period where the steepest increase in fitness levels was achieved. So it seems reasonable to assume that mutations to this gene's CAM ID during that period had significant impact. But G1 and G3 do not show similar correlations.[4] In fact, the second major jump in fitness levels does not correspond to a change in CAM IDs of any of the genes, suggesting that the increase in levels of fitness at this later stage of evolution was caused by "fine tuning" gene configurations. However, it must be noted that the persistence of CAM IDs in successful phenotypes over many generations is made more puzzling by the variety of phenotype circuits that achieve high levels of fitness. For example in the case of some preliminary experiments where very long evolutionary runs were performed, CAM ID mutation can clearly be seen to correlate with jumps in fitness (see Fig. C.1 and C.2). Given that the task chosen is relatively easy, there should be many ways for phenotypes to solve it (for example, compare phenotype variety in Fig. 6.9, 6.10 and B.1). One might expect therefore, that an occasional CAM ID mutation would occur late in a run that would raise the fitness score, even if only slightly. This does not seem to happen, and the causes remains an area of investigation for us. However, we think the cause is not down to a lack of CAM ID mutation, as the plots do show the best of generation mutations to CAM ID giving good coverage of the search space.

Redundancy and neutral variation

Another area of concern relating to search space coverage was the degree of redundancy in the genome specification. For example, the number of CAMs with 5 inputs is very low, therefore many mutations that fall on a gene's connection specification are never realised in the phenotype. While this was a deliberate design decision to encourage neutral variation (and similar to Walker and Miller (2008))[5] there was the possibility that such "wasteful" mutation was hindering the discovery of high fitness levels. To ensure this was not the case, we adopted a variable mutation scheme based on population size.[6] This scheme increases the amount of mutation for progressive members of the population. For example, the first member of the population to be mutated has one mutation per gene, the second member two mutations per gene and so on. The

[4] However, G1 and G3 may not have been expressed (mutation to a CAM ID has no impact unless that gene is expressed). Unfortunately it proved too difficult to extract from the log files which genes were expressed during the tests for each generation, and the analysis is weakened as a result of not knowing this information.

[5] Neutral variation occurs when mutation causes the genome to be changed without altering its fitness, see discussion in Thompson (2002).

[6] I am indebted to Dr. Simon Harding (EFPL, Lausanne) for this suggestion.

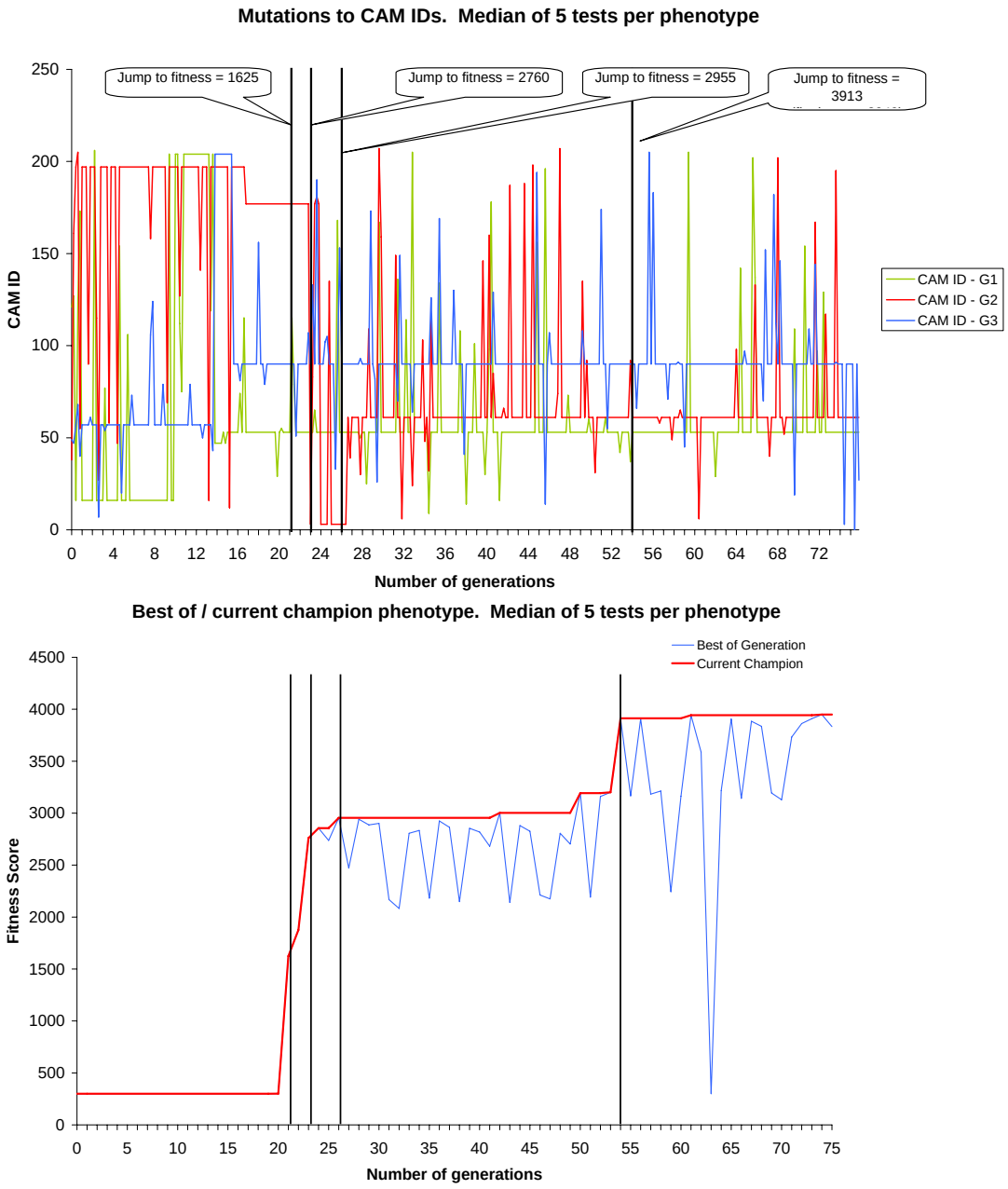


Figure 6.5: Median of five tests per phenotype using a binding site length of two: Top chart shows CAM ID mutations in best of generation (3 genes per genome). Bottom chart shows fitness scores for best of generation and current champion. The vertical bars show significant jumps in fitness.

scheme allows both fine levels of adjustment (small amounts of mutation) and greater variability (large degrees of mutation). The mutation logs of several runs seem to show that the scheme provides a means to move the population forward early on with large mutations, and to move it incrementally in later stages (see §6.5).

6.4 The Evolutionary Harness

There is a vast literature debating population sizes, mutation rates and evolutionary strategies. Unfortunately, much of this work is done on well studied types of search landscape (for example, work investigating the performance of evolutionary algorithms designing 1– or 2–bit adders). Such detailed knowledge of the search space is rare in evolvable hardware, as the presence of real-world physics can make the search domain extremely complex or noisy, and quite likely beyond the ability of human assessment. Given that we are not in a position to assess the nature of our search space and make an informed decision over aspects such as population size or mutation rates, we must proceed with an experimental approach. However, it is worth reiterating that the purpose of this thesis is not to test the efficacy of particular search algorithms in finding a solution. Providing an evolutionary strategy finds satisfactory results, it will meet our purpose. How quickly it achieved it or how many resources it took are not the object of inquiry for this thesis.

We have left it to others to demonstrate that CGP is an effective search algorithm (Walker and Miller, 2008). In keeping with previous work on CGP, we chose a 4+1 evolutionary strategy (Miller and Smith, 2006; Walker and Miller, 2007a,b). In this scheme, the best of a previous generation is cloned and kept, the remaining 4 copies are mutated and the selection process repeated. The 4+1 scheme has a number of advantages:

- easy to implement (small populations, no crossover);
- quick turnover of populations make it easier to view lack of progress;
- time and memory needed to generate a population is low.

These characteristics were helpful during our debugging phases of building the prototype. As many errors were raised during the fitness assessment of phenotypes (including errors from circuit creation, fitness function design and signal sampling), it was important to keep population sizes low so we could begin re-testing as quickly as possible. In addition to this, even with low numbers of genes per phenotype, several files need to be written to disk (e.g. for a three gene genome, seven circuit configuration files are produced, each with

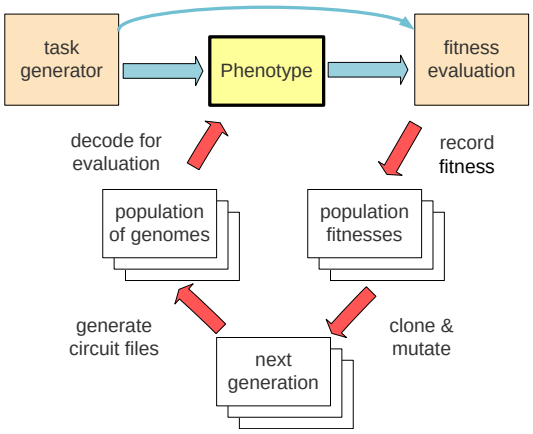


Figure 6.6: Evolutionary harness.

an associated graphical format file, giving 14 files in all). As the processor and memory requirements of the system ran the PC to around 95%–100% of its capacity during the circuit generation phase, it was important to keep populations small so that other applications (such as Matlab performing the wavelet and Fourier transforms) were not struggling for resources. Fig. 6.6 shows how the evolutionary harness integrated into the system architecture.

6.5 Evolutionary parameters

Tests were performed over 75 generations using a 4+1 evolutionary scheme. Initial tests tried up to 500 generations, but improvement was rare after about 50 or 60 generations. This may be because the tests were too easy for the phenotypes to solve, as it became clear that there were many ways of achieving similar levels of high fitness. Another possibility is that relatively few mutations were required to reach the higher levels of fitness once an initial solution had been found.

In addition to reconfiguration bonuses, further "bootstrapping" assistance was provided by allowing genes to have "seeded" binding sites. Thus for a bare wire configuration (i.e. a single wire from input to output), the binding signature produces a set of signatures that run through the frequencies (if no reconfiguration occurs) from AA, CC/GG (the middle band could be either), to TT. Having binding sites of length two gave a good chance of matches being found, but to speed up the search we seeded initial generations with binding site bases known for signatures with a bare wire configuration at the initial frequency step. For example, the first generation in a step-up specialist envi-

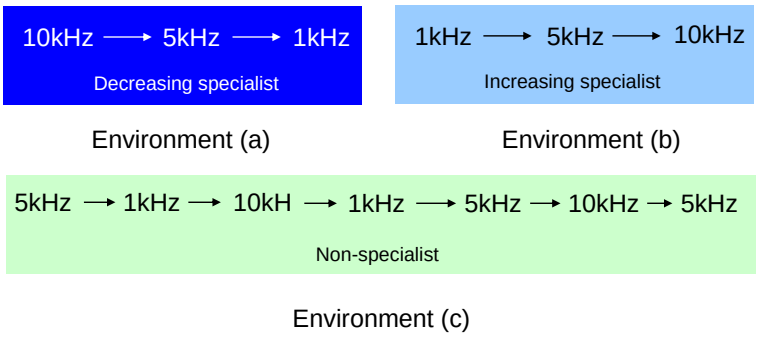


Figure 6.7: The 3 phenotype environments.

ronment might have all binding sites seeded with A or C bases, as the bare wire output signature for 1kHz is generally AAA (only first two positions used).

6.6 Main experiment

Having convinced ourselves that some of the issues relating to noise, hysteresis and search space coverage had been reduced to an acceptable level, we were able to propose an experiment to look at the evolution of phenotype development. As part of our interest in gene expression during developmental stages, we set up a hypothesis to test not only if phenotypes could be adaptable, but also whether having such adaptability incurs a fitness cost.

Experiment hypothesis —*Phenotypes that evolve to cope in an unpredictable environment perform less well than phenotypes that evolve as specialists when both phenotypes are placed in a predictable environment.*

To test this, the three frequency steps were considered developmental stages during which different behaviour was required from the phenotype. In specialist environments, the steps in frequency either increased or decreased. In non-specialist environments, all possible transitions occurred. Fig. 6.7 shows input-stage transitions for specialists (a) and (b), and non-specialists (c).

In all environments the fitness test remained the same: maximise power in the 5kHz band, minimise it elsewhere. As a phenotype uses circuit output to trigger a reconfiguration, the current configuration is crucial to how the phenotype configures the next step. For example, a specialist phenotype may have configured to a high pass filter with gain suitable for high fitness during a 5kHz stage. On stepping up to 10kHz, this filter produces a binding signature that the phenotype can use to configure to the next circuit. However, if the same step occurs as part of a sequence that runs from 10kHz to 5kHz and

back to 10kHz, the previously unknown step from 10kHz to 5kHz may cause the phenotype to configure the FPAA to another circuit. The step from 5kHz to 10kHz now results in a different output signature and the previously used genes for the good circuit no longer match the new binding signature. Specialists evolving in predictable environments can therefore both profit and suffer from previously encountered steps in a non-specialist environment. To counter potential bias from phenotypes encountering a recognised sequence of inputs, the non-specialist environment starts with a bare wire configuration at 5kHz (as opposed to 10kHz or 1kHz) and introduces steps that will not have been previously met by specialist phenotypes (such 1kHz to 10khz).

6.7 Main results

Each of the specialist and non-specialist phenotypes were evolved over 75 generations, using the median of five tests per phenotype. The champion phenotype was then tested in environments it had not evolved in. Each test in the new environment used the average of two results (each result being the median of five fitness scores, as during the evolutionary runs). This it was hoped would remove any effects of the new circuits being tested on the FPAA "cold" without the previous state having an impact (perhaps because of some circuits making the chip hotter than others), as in evolutionary runs similar circuits are tested repeatedly and the FPAA generally runs at the same temperature. The results, shown in Fig. 6.8, show poor capability for specialists in their 'opposite' specialist environment (where the frequency steps are reversed) and in the non-specialist environment. The reason seems largely the result of a strategy employed by phenotypes in the final stage of specialist environments. Circuits that reconfigure to 'broken' last circuits (at 1kHz or 10kHz stages, for example see Fig. 6.9 and Fig. B.1) help their fitness scores and have nothing to lose from reconfiguring to a circuit that no longer uses circuit input or one that produces no output at all. But the same phenotypes suffered badly if they needed to recover from that stage in the non-specialist environment, and generally failed to reconfigure.

The runs for non-specialist phenotypes appear to show a slightly reduced fitness in a decreasing environment compared to the specialists of that environment. Taking the median of the specialist (decreasing) runs in their own environment as S and the median for the non-specialists in a decreasing environment as NS, the median fitness score for the two sets of runs show a slight difference:

$$S = 3156$$

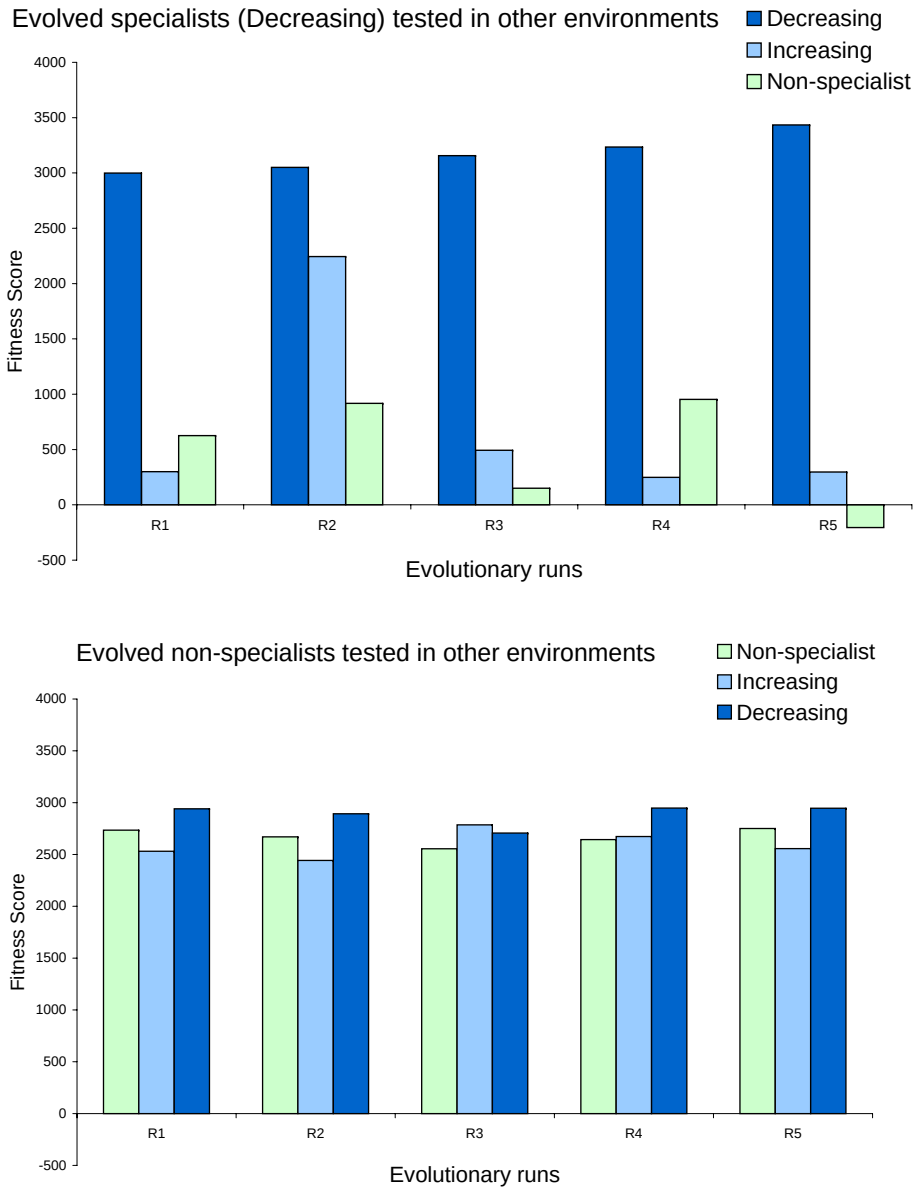


Figure 6.8: Results of specialists (top) and non-specialists (bottom) tested across all environments. Dark bars show phenotype scores in their own environment.

$$NS = 2940$$

To find out if this is statistically significant, we can set up a null hypothesis to say whether distributions of the two sets have the same median for their runs (s and ns):

$$h_0 : s = ns \tag{6.9}$$

As the distribution of either set of runs is unknown, we use a non-parametric significance test — the rank sum test (Mann-Whitney-Wilcoxon, 1947) — to determine if h_0 is true. The Matlab `ranksum` function, `[p,h] = ranksum(s, ns)`, performs a two-sided rank sum test of the null hypothesis h_0 for the data in `s` and `ns`. The p-value of the test (i.e. the probability that the data in `s` and `ns` are independent samples from identical distributions) is returned in `p`. `h=1` indicates a rejection of h_0 at the 5% significance level. For the two sets of evolutionary runs we get:

```
s = [2999,3050,3156,3235,3434]
ns = [2940,2892,2706,2948,2946]
[p,h] = ranksum(s, ns)
p = 0.0079
h = 1
```

Thus refuting h_0 with a confidence level over 99%. This would seem to back the hypothesis given in §6.6, however it should be borne in mind that reconfiguring to manage seven frequency steps is more difficult than reconfiguring for three, and the added difficulty may have led to lower fitnesses. This conclusion has some backing from test results for non-specialists in specialist environments. In all cases, the non-specialists performed well; not as well as specialist phenotypes, but their scores were higher than the scores they achieved in the environment they evolved in, leading us to suspect that the specialist environments were easier.

Examples of champion phenotypes are shown in Fig. 6.9– 6.12.[7] Given the simplicity of the task and the size and complexity of the search space, it was to be expected that the phenotypes would discover a variety of ways to gain high fitness scores, and on the surface it appears that no two champion phenotypes are identical. However, there are CAMs that appear in many of the successful phenotypes. For example, nearly all successful phenotypes included a Gain-Switch of some type that was deployed at the 5kHz input stage (Appendix B has several instances) where maximum power output was required. Similarly,

[7] Appendix B has further examples.

where the specialists could get away with "dying" (i.e. downloading a broken circuit on the final input stage), they would adopt this to boost their fitness scores. Naturally, a broken circuit is the best way to minimise the power from a circuit, and although all such broken circuits get the same fitness score in terms of our fitness criteria, none of these final "broken" circuits have the same configuration. Finally, the redundant parts of the genome (containing genes that were never expressed) showed a wide variety of CAMs, indicating that flexibility lay in the wings should the evolutionary conditions change.

Examination of the reconfiguration patterns show that the non-specialists were able to go back and forth using the same good circuits in both forms of specialist environment, demonstrating robust homoeostasis. This suggests that this mechanism of control could be suitable for in-situ evolution in localised environments, where the broad range of inputs (and required response) is known (for example, diurnal patterns in data received by an external sensor), but where inputs might have local peculiarities. Similarly, the specialists demonstrate very high fitnesses can be achieved over predictably changing environments. It is interesting to speculate what applications this could be put to, particularly with regard to the evolution of phenotypes that "expire" as a means to gain the highest possible fitness during their last developmental stage. Single-use hardware might seem uncommon, but there are examples in military or space applications.

6.8 Further investigations

6.8.1 More frequent reconfigurations

Rather than limit the genome to just 3 reconfigurations (one per input stage), we tried increasing the potential number of reconfigurations by increasing the tests for binding to 10. The input frequency was stepped up by 1kHz, with each step lasting for 5 seconds. The task was the same as for the specialist (increasing) environment shown previously. That is to say, no account was taken of the intermediate reconfigurations between tests for fitness at 1kHz, 5kHz and 10kHz. The results for 5 runs are shown in Fig. 6.13. Despite taking the median of 5 tests per phenotype as before, it seems that the increase in noise described in §6.3.1 from more frequent interruptions to signal input (and output, from reconfigurations) prevented the genome from achieving high levels of fitness. In particular, although an increase in fitness levels can be seen, no genomes achieved the levels of fitness of phenotypes in tests where less disruption to the signal occurred.

This may be down to the system requiring more time to settle after a re-

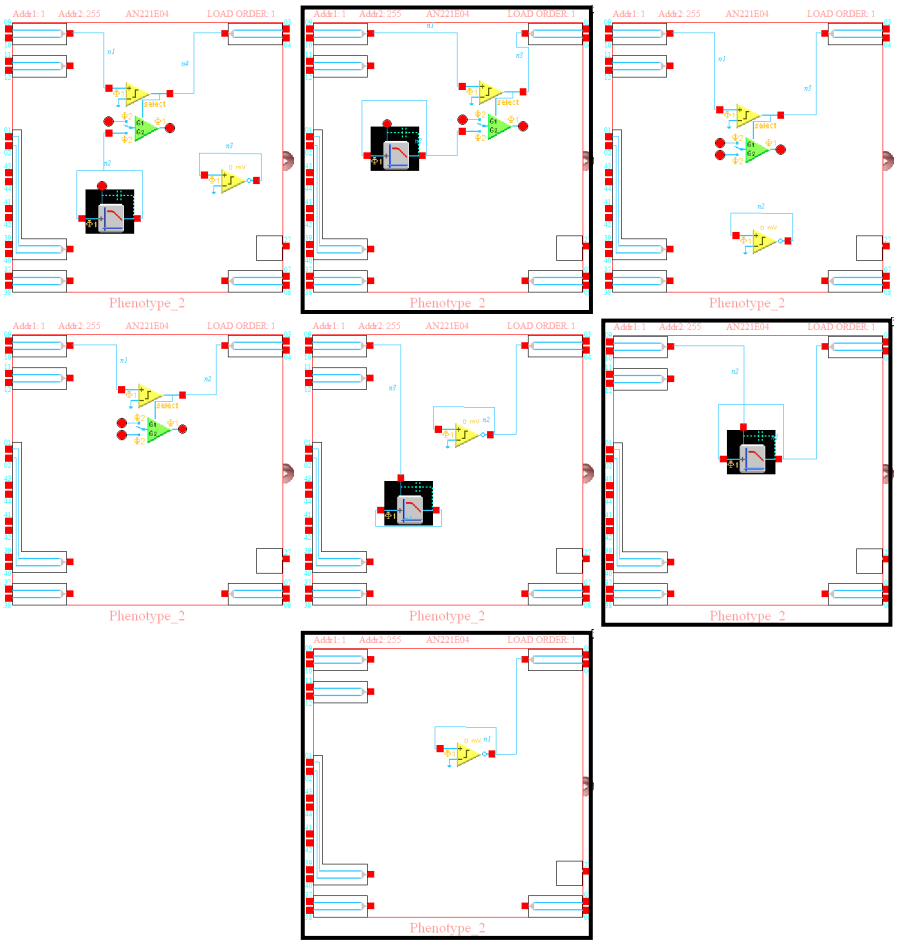


Figure 6.9: Example of champion specialist (decreasing) phenotype, showing the 7 possible circuit expressions. The first circuit shows the fully specified genome containing FilterLowFreqBilinear, GainSwitch and Comparator CAMs. This phenotype scored 3434 and deployed 3 circuits (no. 6 at 10kHz, no. 2 at 5kHz and no. 7 at 1kHz, shown with additional borders) Note "dead" circuit in the final configuration!

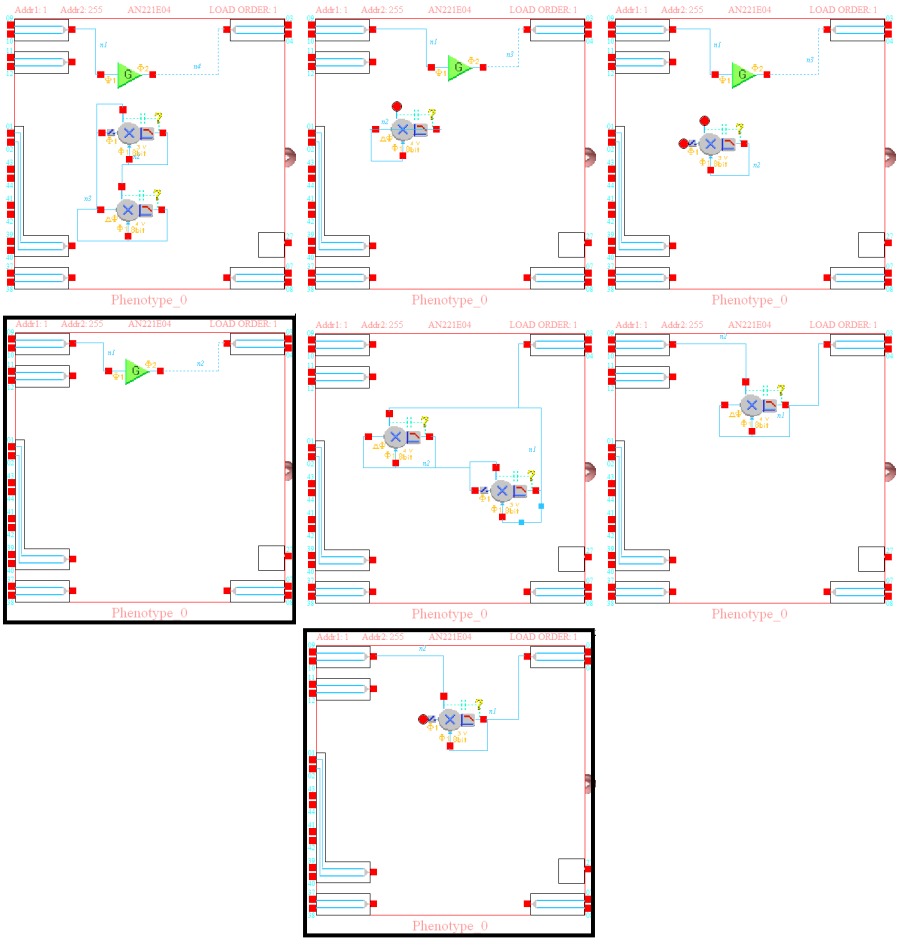


Figure 6.10: Example of champion specialist (decreasing) phenotype, showing the 7 possible circuit expressions. The first circuit shows the fully specified genome containing GainHalf and two MultiplierFilterLowFreq CAMs. Notice how on the full specification the two multipliers are connected to each other, but are not connected to either circuit input or output. However, the wiring algorithm is able to connect them up when they are expressed as single CAMs. One side effect of this is that the first 4 expressions of this phenotype are probably functionally equivalent. This phenotype scored 3050 and deployed 2 circuits (no. 7 at 10kHz and 5kHz, and to no. 4 at 1kHz, shown with additional borders).

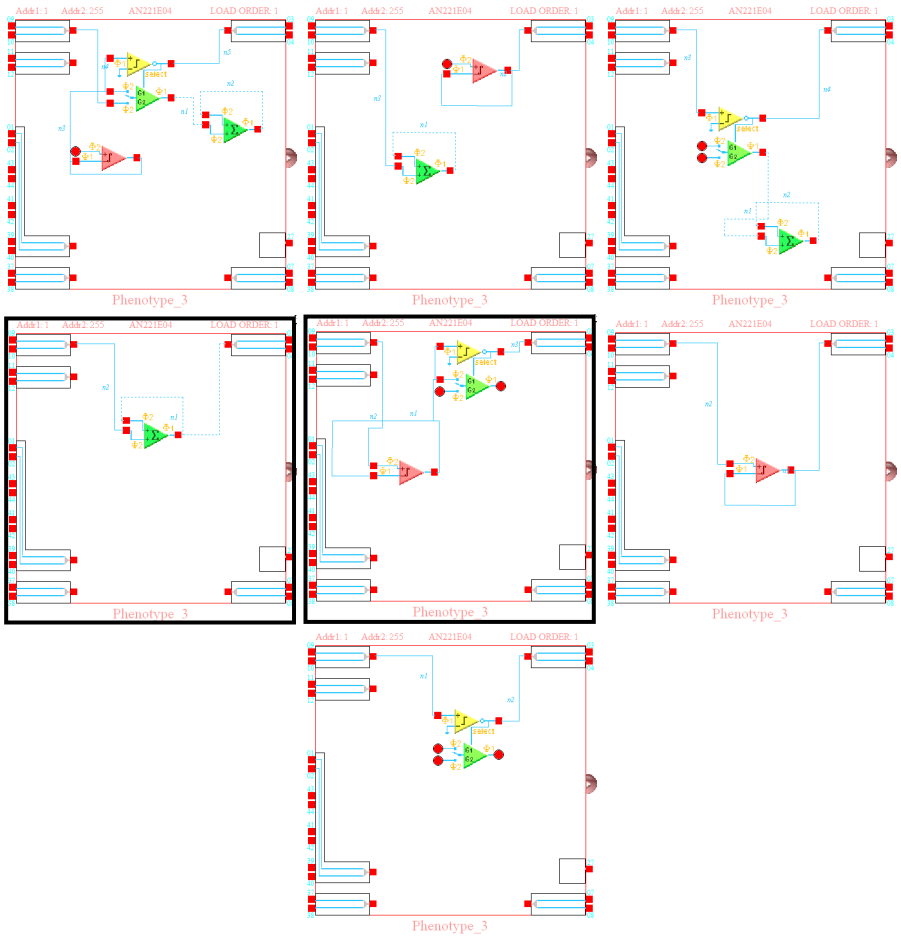


Figure 6.11: Example of champion non-specialist phenotype, showing the 7 possible circuit expressions. The first circuit shows the fully specified genome containing SumIntegrator, GainSwitch and SumDiff CAMs. This phenotype scored 2555 and deployed 2 circuits (no. 4 at 1kHz and no. 5 at 5kHz, shown with additional borders). What is interesting here is that 1Khz expression is apparently non-functional (shown by the dotted connection lines), but it passed enough current to still tell the phenotype when to reconfigure but not enough to give it a poor fitness in terms of its power reading.

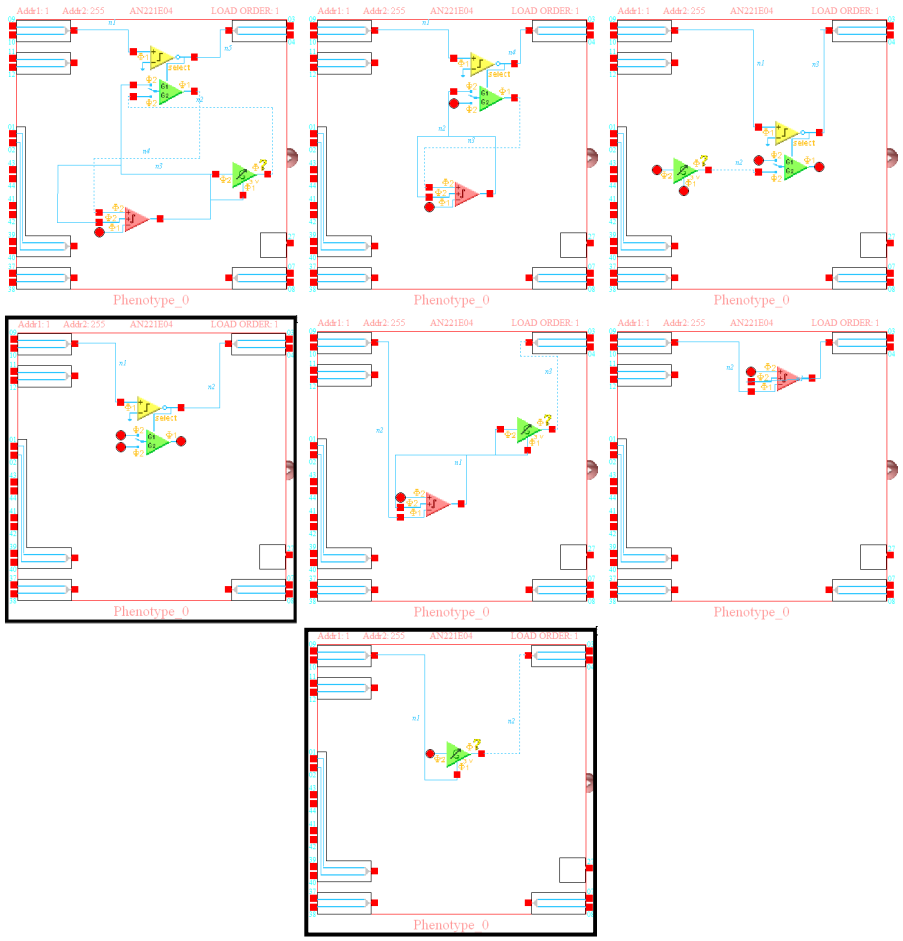


Figure 6.12: Example of champion non-specialist phenotype, showing the 7 possible circuit expressions. The first circuit shows the fully specified genome containing SumIntegrator, GainSwitch and GainVoltageControlled CAMs. This phenotype scored 2644 and deployed 2 circuits (no. 4 at 5kHz and no. 7 at 1kHz, shown with additional borders). Again note the poor but sufficient circuit deployed for the 1kHz stage.

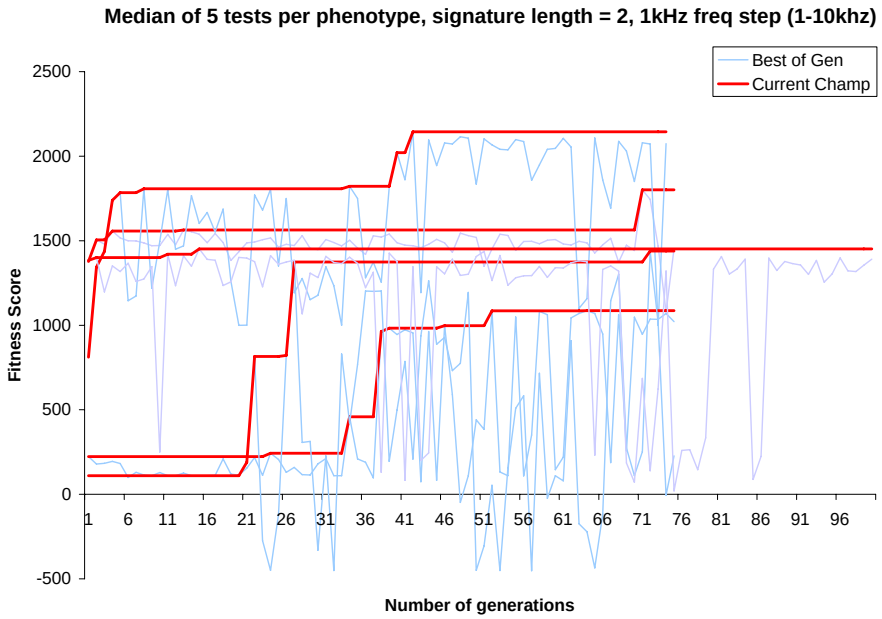


Figure 6.13: Additional Results: Chart shows the effect of reconfiguring the FPAA at 1kHz frequency steps (5 tests per phenotype, genome with gene binding sites of length 2).

configuration, as the evolutionary process can be led astray by false fitness assessments such as those described in §6.3.1. However, the increase of noise may also have an impact on the relationship between binding site length and the generation of wildcards. This relationship is discussed further below, and more speculatively in § 7.4.

6.8.2 Increasing binding site length

Prior to the experiment described in §6.6, we experimented with various binding site lengths.[8] Initially binding sites of three bases were generated per gene, and the performance of genes containing sites of two bases were compared against these, both in the three step frequency tests (see Fig. 6.14) and in the ten step version. It was found that increasing the site length neither improved the final level of fitness achieved nor the time taken to reach that level of fitness. Due to a perception that genes containing binding sites of length three were slower to achieve comparable levels of fitness, tests were carried out up to 500 generations, but with no significant improvements on runs that stopped after 75–100 generations.

However, it must be said that the length of binding site and the effect of

[8]NB. This was also before using the median of five tests per phenotype.

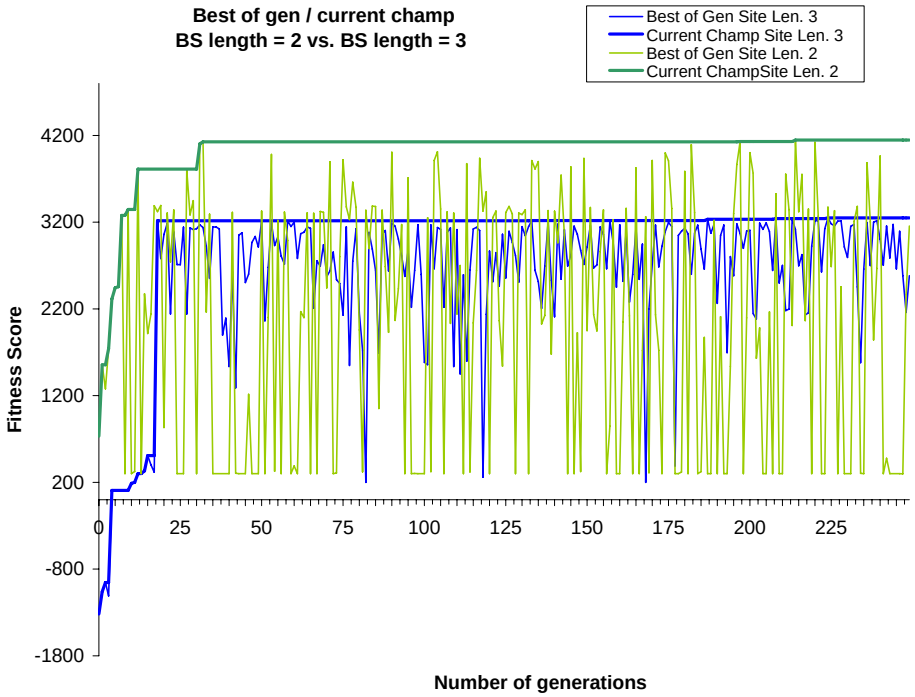


Figure 6.14: Additional Results: Chart shows a comparison between best of generation and current champions of genomes with gene binding sites of length two and three.

wildcards on gene expression and genome fitness is deceptive. It might be the case that in noisier environments, genomes containing more genes with longer gene binding sites could result in more stable behaviour. Unfortunately the size of our FPAA meant we were limited to genomes containing a maximum of three genes, as a fully expressed genome containing more genes than this could exceed the capacity of the FPAA (depending which CAMs were involved). As a result, we were unable to test this out reliably, particularly as noisier output signals tend to produce more wildcards in a signature, resulting in bigger circuits being configured (see §5.5.2). As the FPAA was very limited in its resources, larger circuits exceeded the available resources and the unpredictable nature of gene expression during an evolutionary run made longer genomes with longer binding sites impractical to test. This would be an interesting area to explore further, as suggested in §7.4.

6.9 Summary

The experiments in this chapter covered the evolution of an analogue circuit (or filter) that would maximise the power output of a sine wave input at 5kHz. To do this, the circuits needed to reconfigure from a circuit that minimised power output at 1kHz. The circuits also need to reconfigure after the 5kHz stage to minimise the power output of a 10kHz signal. The fact that evolution could discover changes in the input and respond to them demonstrates that the system is capable of reconfiguring to adaptive filters as needed. Considering our FPAA platform as a "reconfigurable device", this part of the work confirms the broad hypothesis of the thesis. During these early experiments, many issues with noise and the effects of random variation were dealt with, and gave us confidence to proceed with the second set of experiments that tackle the more specific hypothesis given in §6.6. These experiments demonstrated that there is indeed a cost associated with being generalist, in that generalists perform more poorly in a predictable environment that a specialist. However, the specialists achieve their high levels of fitness in predictable environments by using an irreversible configuration. This means that in generalist environments they cannot recover once they reach what they believe is a final stage and so suffer poor fitness scores as result. In all, over 30 experiments were carried out, over a period of almost four months. The preliminary experiments took around 24 hours each to complete 75 generations. The second set of experiments, using median fitness scores described earlier, would take up to three days each to complete.

As we have seen, the prototype has successfully managed to give us some interesting results: it has evolved specialist and non-specialist solutions with small populations over only a limited number of generations. What we have shown in a broader sense is that search can by self-guided by a mechanism such as gene expression across several search domains. Each domain can have a different fitness criteria against which the reconfiguration of the phenotypic stage is judged against. However, as mentioned in §6.8, limitations with the prototype left several questions unanswered and there are several areas where it would be interesting to run more experiments and to take the work further. Broader questions relating to other aspects of development, such as the embodiment of some form growth, are beyond the scope of this work. But we feel our system gives the possibility of creating a self-governing mechanism, discovered by evolutionary search, that could form the basis of cell differentiation and hence be incorporated into a model of growth. The final chapter of this thesis looks at some of these questions and how we might best approach them, given the encouraging results of the prototype.

Chapter 7

Conclusions

7.1 Context of interpretation

The motivation for bio-inspired computation is given variously as ranging from curiosity about how "natural" algorithms might work to admiration for biological characteristics such as robustness, adaptability and ingenuity. My own motivation was similarly born out of admiration and curiosity. I wanted to see if we could build a system that captured some of the qualities that organisms use to adapt and respond to their environments.

Justifying this excursion into bio-inspired paradigms was made easier due to the somewhat limited success of evolutionary computation. While not diminishing its achievements and breadth of application, doubts remain among practitioners that unless evolutionary computation demonstrates the scalable reuse of modular functionality, the promises to take automated design search into areas too difficult for human engineers will start to ring hollow in applications for research funding. The evolutionary computation community is all too well aware of how that fate befell the artificial intelligence (AI) community in the early 1970s, when critical books from Minsky, Dreyfus and others (Dreyfus, 1992; Minsky, 1988) devastated the field and caused funding for AI research to virtually disappear (Crevier, 1993). When one looks at what happened to the AI community, it is clear that hopes had been raised artificially high by extravagant claims about the sort of problems AI would solve. Meeting those claims proved elusive, and scepticism set in among those responsible for research funding. As a community we cannot let this happen to evolutionary computation.

We may be approaching a juncture: either we move away from the seemingly endless stream of papers claiming further optimisation of well-studied evolutionary algorithms, or we will face increasing criticism from those within

our own discipline that evolutionary computation has progressed little beyond the tasks and applications first identified in the 1960s. This concern is not intended to denigrate the work on performance — from a computational perspective, the work is justified. However, it is difficult to see how improving performance is going to help us with scalability. There is no evidence that suggests better performance is going to unlock the key to successful reuse or the ability to tackle larger, more complex problems.

By the same token, as we seek ways of meeting scalability we should be circumspect with our claims about what our techniques can deliver. Given the source of our inspiration, it not surprising that we turn again to nature for ideas. Everywhere around us is evidence of how marvellously the eukaryotic cell scales, adapts, evolves. Does the secret of evolutionary search finding solutions to large, complex problems lie within the cell? According to Wolpert (2003), there can be no doubt it does. The answer for computer scientists is less certain. Despite their obvious success, biological systems are fantastically complicated, while our abstractions of those systems and processes can be superficial. What to abstract, and why, has become a hot topic of conversation in evolutionary computation conferences and workshops. The plundering of the latest findings in biology by computer science can sometimes seem akin to the "land grab" of the American mid-west in the 1870s: the first to incorporate the latest findings into their model, becomes the first to lay claim to having introduced those ideas into the field! But careless borrowing and subjective abstraction of biological processes could backfire on us.

It may be time to take stock of what we mean by scalability and discuss the sorts of solution discovery we can sensibly aspire to with evolutionary computation. We may have to accept that the way biology scales and reuses genes is so far removed from the way humans engineer that we will never make evolutionary computation design cars, hospitals or even a house. This not to say we cannot evolve complex control systems, highly adapted to specific circumstances. But it may depend more on what we are controlling and the application environment than we have previously realised.

Development offers us a way to adapt and respond to dynamic environments, yet keep the advantages of evolutionary search. This is already a scaling up of what can be tackled by evolutionary computation. We are now well versed in how to evolve a genome that represents a good solution for a static environment and there are many papers telling us how to do it quickly. Gene expression offers us something different: the ability to extract solution subsets contained within a single, evolved control structure. Perhaps the challenge for us is to ignore the requests for evolutionary computation to design big, complex applications, and instead think of an application realm where small,

autonomous, embedded devices require highly localised configuration. Where devices that operate across a range of context-specific conditions would be too expensive or time consuming to configure individually. Instead of scaling up, perhaps the future of evolutionary computation is to think small, and think many.

7.2 Summary of work presented

The arguments put forward in this thesis make claims that evolutionary computation has failed to keep pace with developments in evolutionary biology. The canonical form of an evolutionary algorithm is itself testimony to a somewhat outdated view from the 1970s that declared genes alone were responsible for evolutionary design. Developmental biology has increasingly challenged this view, with evidence demonstrating the degree to which gene expression is governed by developmental processes. Those processes act to constrain, conserve, reuse and modify the opportunities for gene expression. Without developmental processes dictating patterns of expression, the variations of repetitive functional morphology we see around us would be impossible. There is nothing in a segment of DNA material that says when or where it will be used, or reused, except in the context of where that material will be expressed. DNA is an entirely passive instruction set, which would be meaningless were it not for its context-specific interpretation by proteins.

It is hard to overstate the importance of this for evolutionary computation. If you want your evolutionary designs to reuse useful genes, you need to introduce something that determines when and where they will be reused. You need an environment that gives contextual flexibility to reuse, so that reuse in different contexts can mean different things.

The system presented here met the requirements of the conceptual architecture in Chapter 3. However, capturing those conceptual needs does not mean that our system offers a solution to all the issues of scalability and reuse. While we still hope that these can be met by combining developmental approaches with evolutionary computation, our prototype is not intended as a blueprint for scaling up evolutionary search to large, complex applications. The prototype successfully demonstrates context-sensitive gene reuse, in a form that allows gene expression to guide evolutionary search across several phenotypic stages of development. Comparing this achievement with the work of others in the field is difficult: the majority of work in evolutionary computation focuses on the optimisation of a search strategy, while the little work there is on phenotype development has largely been led by researchers interested in char-

acteristics such as stability or robustness (such as Pauline Haddow's group in Trondheim and Tyrrell's group at York). Neither is the work here comparable to evolution *in materio*, where evolution is able to directly access the physical properties of the platform to create novel circuits (such as the work at JPL, and researchers such as Thompson and Miller). We believe the ideas behind our system are novel, even if the techniques we used to build our system are relatively well established. The evolutionary search algorithm and representation (4+1 and CGP) are widely known and studied, and our work is not intended to demonstrate an improvement in performance through their adaptation. Similarly, the use of wavelets for signal analysis is already almost two decades old and the use of feedback as a mechanism of control in analogue systems has been around even longer. These aspects of our prototype are not new and are not presented as such. The novelty in our approach is the creation of a form of evolutionary search that relies on gene expression to guide itself across several search domains. By using different fitness criteria at each domain (which we choose to call a phenotypic stage in development), the results from the system demonstrate a proof of the hypothesis: evolutionary search can control successive stages of phenotype configuration through the use of gene expression.

Our system set out to demonstrate that gene expression, tied to a specific context of deployment, is a first step towards enabling gene reuse. Evolution is free to manipulate the rules that configure a gene and dictate its context of expression. But in addition to this, phenotypes need to be able to respond to changes in their environment, and so we tied the control of gene expression to the signal output of the FPAA. Feedback from the phenotype as it explores developmental environments allows evolution to select on the basis of phenotype performance in specific contexts. It remains for engineers to determine what weighting they wish to give certain contexts in the hope of coaxing evolution down certain paths of control.

The FPAA in our system can be best thought of as analogous to a cell nucleus, within which the presence of transcription factors (the context of expression provided by the signal output) determines which genes are expressed from the genome at a particular stage of development. The resulting configuration from the gene expression can then be assessed for fitness. The genome representation, and the inclusion of gene expression tied in a feedback loop to the genome, gives the system some novel features. While the prototype has not been tested on a wide range of tasks, the prototype implementation and results that were obtained demonstrate that:

- configurable, high-level modular functionality is suitable for representation as genes in a genome;

- the translation from genome to phenotype should be sufficiently rich so that reuse of genes in different contexts can give different functionality;
- gene expression allows solutions to be discovered using a subset of genes from the genome;
- gene expression can say when to deploy solutions in changing environments;
- a genome can be evolved that expresses phenotypes adapted to multiple developmental stages;
- redundancy is possible from genes that are not expressed and in parts of the genome not realisable in the phenotype;
- by extracting rich information from the search domain and tying it to gene expression, we have an automated mechanism of reconfiguration that can guide genomes across successive search domains that humans would find difficult to explore.

Pleasing though this list of features may be, the construction of the prototype and the results we have obtained raised many questions. The next section examines some of the design decisions made with regard to the platform and system implementation.

7.3 A Review of Design Decisions

The conceptual requirements were motivated by the list of key points given at the end of Chapter 2 (§2.10.1). The decisions that were then taken, regarding the choice of platform and implementation of the system architecture, reflected the need to meet those requirements. While those requirements were met, we acknowledge that there were many difficulties encountered during the coding and testing stages of the project, and that some of these were due to our choice of the Anadigm FPAA as a platform on which to conduct evolutionary search.

Firstly, it must be noted that the Anadigm FPAA we used was an "off the shelf" component. Its manufacturers never intended it for use in evolutionary computation. To use it as such, required help from the company's engineers and we are grateful to them for the assistance they were able to give (see § 5.6.1). Without their help, it is doubtful this project would have been completed. However, Anadigm make a point of saying that the Automation API they provide is not supported and neither will they supply details relating to the generation of the bitstream used to configure the FPAA. Researchers wishing to make use of the platform for further work should bear in mind that

some of the most interesting technology related to this FPAA is patented by Anadigm and remains inaccessible and proprietary. For example, the ability to do dynamic reconfiguration during runtime (without a "cold reset" of the FPAA) was a key characteristic behind our decision to go ahead with the platform. Unfortunately, on closer inspection, it turned out not to be possible to employ this mechanism. This had implications for the experiments, in particular, it meant we were unable to experiment with "continuous" reconfiguration between fitness test points. As we had to employ a cold reset of the FPAA before downloading a configuration bitstream, the signal output would stop and then start again with the new configuration. If many configurations occurred close together, the stopping and starting of the signal output created too much noise for the evolutionary search to be successful. This resulted in us having to employ three stages during which reconfiguration could occur, and these stages required long periods (several seconds each) between them to allow any disruption in the signal output caused by the cold reset to have settled before fitness assessment could be made.

Compromises such as these are no doubt common when experimental hardware is being used for the first time. During the implementation of our system, some aspects relating to circuit representation and search were hampered by constraints imposed by the platform. It could be argued that implementing the system in software would have reduced many of the problems we encountered, and with the benefit of hindsight that may be the case. However, there were a number of positive benefits that Anadigm's API gave us with the respect to the platform. For example, the context-specific reuse of genes was achieved by virtue of having to wire up the CAMs according to the genome specification. CAMs missing in a particular context of expression meant that a CAM in one context could receive or output an entirely different signal to the same CAM in another context. If this feature had not been present in our system, a context-sensitive scheme of gene reuse would need to have been designed and implemented somehow.

The question of whether a system such as ours could have been more easily built using software simulation should also be addressed bearing in mind of our principal motivation for selecting an FPAA: we wanted to employ our form of evolutionary search and reconfiguration in an area of design where the complexity that exists comes from certain physical effects that are difficult to simulate. That is what makes simulation in the area of analogue circuit design difficult. Admittedly software does exist, such as SPICE, which gives accurate simulation (within limits) of analogue circuit components and is widely used in circuit design. As previously mentioned, SPICE already has some association with search techniques such as evolutionary computation (Koza et al. (2004);

Mattiussi (2005)). However, any work with SPICE deals with simulated, rather than real world, physics. The software simulation of a circuit is not a guarantee how that circuit will perform when it is built in hardware. Our choice of a reconfigurable analogue platform was in part motivated by the understanding that evolution would be allowed to react to actual circuits as they were implemented on the chip, rather than in simulation. It was hoped that evolution would manipulate the signal processing *in materio* and that this might give a better chance for novel circuits to be discovered. While this decision was well-motivated, it turned out that there were more disadvantages to dealing with real world, noisy signals on hardware than the hope of novel circuit discovery could justify. However, it should be accepted that it was never our intention to discover previously unknown forms of circuits, making use of some difficult-to-understand physical properties of the platform. Like many modern FPGAs, low-level access to the configuration bitstream is not possible on the Anadigm FPAA. As a consequence, the opportunity of using the physical characteristics of the platform to create entirely novel circuits is restricted. But this criticism does not mean that the search space evolution has access to is not large and difficult, neither does it mean that a software simulation of a similar environment would have given better results or an easier implementation. However, for an exploratory project such as ours, the additional difficulties of working in hardware probably outweighed the advantages given by the Anadigm API and the CAM architecture. Despite being a good fit with our conceptual architecture, the reality of working with limited support on novel hardware was often frustrating.

Some the biggest software difficulties we faced were as a result of choosing a signal transform based on wavelets. The processing requirements of continuous wavelet transform are considerable; it is not a technique designed for rapid signal analysis. Neither is it particularly suited to picking up changes to a stationary wave, such as steps in frequency. Our thinking behind the adoption of wavelets was that if we were to use our system as a means to evolve and adapt (by reconfiguration) devices such as wireless sensors, those systems would be feeding analogue input signals that were of a type that would be ideally suited to wavelet transform. However, the work involved to implement the wavelet transform and the processing problems we subsequently incurred delayed our project by several weeks. Even when the hurdles with Matlab were overcome, we were left with a system that was perhaps overly susceptible to noise. Matlab's implementation of the continuous wavelet transform results in weak signals being scaled appropriately. This coupled with our "wildcard" binding scheme meant that bindings were easily triggered by noisy signals, which had the effect of misleading the evolutionary search by producing randomly good

fitness scores. In hindsight, another scheme to perform signal transform could have been adopted (such as Fourier transform) that would have been easier to implement and given the benefit of a lower processing load.

Most issues related to our choice of platform are down to problems of processing noisy signals. Here it is relatively easy to make judgements about the wisdom of adopting analogue signal processing as a design area, or using hardware components as our "reconfigurable unit" or cell nucleus. But given the novelty of the work presented, it is much harder to assess aspects of the system implementation such as the binding scheme. Indeed, this was one area that we were unable to explore in the time available and its influence on the search process remains obscure. Taking our schema design from biological inspiration was not unreasonable, as our approach wanted to investigate the potential for gene expression to guide the search process in as natural a way as possible. However, the process of matching binding sites to a binding signature that contains wildcards can mean that there are subsets of genes that will never be expressed (§5.5.2). This reduction in search space coverage was difficult to quantify, as evolution was able to choose the circuit to deploy at any stage and therefore affect the presence of wildcards in the signature. This, coupled with the effect of noise and the increase in wildcards it produced, gives an element of uncertainty when we come to describing how the system works. The feedback loop that triggers gene expression is certainly an intriguing part of the system, but future work should investigate how this mechanism might help or hinder the search process, and to do that, the system would need to be implemented in a way that would avoid the problems of noise and excessive wildcard generation (i.e. a software only environment). The next section looks at this and other aspects of taking the work forward.

7.4 Future work

Perhaps the single biggest difficulty facing the Anadigm FPAA, both in terms of garnering interest from industry and other academics, and in terms of testing the platform, is its limited size. Having each CAM primitive represented as a gene means that even short genomes risk exceeding the capacity of the FPAA. This is partly because there is no way of knowing whether phenotype testing during an evolutionary run will mean that, a) full expression of the genome will occur, or b) the CAMs that get expressed will use more resources than usual. These two factors mean we have had to keep the genome length to no more than 3 genes, even though the FPAA could generally support between 4–7 CAMs. Even so, a genome of 3 genes produces 7 possible phenotype ex-

pressions, and that number quickly escalates as genome length increases. What would be a short genome by most evolutionary computation standards of 16–20 genes, for us would generate upwards of 65,000 to over a million possible phenotype expressions, having implications for writing to disk and the time required to generate and search through this number of files.[1]

As the total number of phenotype expressions rises, doubts also begin to surface over the degree of specificity possible with our binding site scheme. Fortunately, as Carroll has illustrated, binding signatures (even without wildcards) generate huge numbers of combinations (Carroll, 2006). But that does not answer how we extract rich information from the search domain that can effectively access all those combinations if needed. We have already shown that the presence of wildcards can have consequences affecting subsets of gene expression (§5.5.2). An unforeseen effect of wildcards is that they reduce access to specificity still further. But there may not be sufficient richness in the many application environments for binding signatures to search upwards of a million different gene combinations. For some applications this may not be a problem, as solutions to the task might be relatively easy to find, but until we can formulate some experiments that can test these assertions, it is difficult say with confidence whether this aspect of the binding signature scheme is a weakness or not.

Our binding signature allows the domain space to be searched using a transform that is further processed to remove much of the "richness" of information it could convey. This raises the question of how effectively such a scheme works. It should be pointed out that the signal to trigger a gene expression may not need to be "rich" in information (cf. "weak linkage" in Kirschner, 2005). Unfortunately we have no way of knowing whether evolution is making use of the size of combinatorial space for binding signatures. So having a complex scheme of signal transform and encoding may not be giving advantages to the search process. What we do know is that biological genes exist with transcription factors using highly specific binding sites, while others make use of a more flexible wildcard scheme. But to my knowledge, the relationship between the number of wildcards in the binding site of a transcription factor that affects a gene and the frequency with which that gene is translated in different contexts has not been demonstrated. With our prototype we could gain only an inkling of what this relationship might be for an architecture such as ours: it needs much more testing and experiment design to verify.

§2.5.2 discusses how some Hox-gene clusters show very strong conservation across all species of animal. One of our earliest and unfulfilled aims was to

[1] It should be borne in mind that the total number of circuit configuration files would be double this number. See §5.2.2

investigate if genes that are important for early developmental stages would be conserved if the requirements for later stages are changed. This will be a fascinating area to study for the future and one which could give us real insight into how the conservation of useful genes affects features that develop in later evolution. One of the difficulties of setting up an experiment to test this is the development of modular functionality that can be "tweaked" and traced through an evolutionary run. It is hard to imagine we could come up with an example to match the development of the mammalian middle ear given on page 32, as the changes in functional morphology led to separate emergent functionalities between species. But tracing the development of such functional features through evolution (and perhaps even trying to guide them) is an exciting challenge for those who want to use developmental processes in evolutionary computation.

It may be the case that to answer these questions, we will have to derive a new architecture that is composed of software elements only. Hardware is difficult and expensive to scale, and a software only system would give a much greater range of tasks we could tackle. The elements of feedback, gene expression and output transform can still form part of the system, although it may be debatable how much "richness" can be extracted from virtual environments as part of the feedback process, and the arguments for trying to build the sort of complex transform and binding scheme seen in our architecture might be weakened as a result. But whatever system is built to take this work further, I believe the future looks bright for the increasing use of ideas from developmental biology in evolutionary computation. Evolutionary algorithms have performed usefully as search-based optimisation tools for several decades. The next step is to look at processing tasks as a series of developmental stages in which we seek solutions, using gene expression as the means of exploration and feedback as the trigger for phenotype reconfiguration.

Appendix A

Program code and scripts

This appendix lists the Matlab script called by the main application as explained in §5.6.2. It also gives some output from the logging function in the application to show how each CAM configuration could be identified from the logs and a trace made to see which elements had been altered by mutation (see §5.2.2). Finally, we show example code from the main application (the wiring algorithm) to illustrate some of the issues related to error handling and how the Anadigm API is accessed. Note for this code sample, the FPAA has already had its CAMs downloaded so that the any parameters exceeding the allowable ranges have been altered. The application retrieves these values if needed, and in the case of the wiring algorithm, tests connections *in situ* (see §5.6.1).

A.1 Matlab script for wavelet and Fourier transform

Listing A.1: Matlab script, started and stopped from main application for each phenotype, and used for signal sampling, wavelet and Fourier transforms.

```
function ProofOfConceptTask

%% First check if we are still accessing the DAQ I/O drivers.  If we
%% are, then make sure we stop scanning the card.
if (~isempty(daqfind))
    stop(daqfind)
end

% set up main output from chip for wavelet transform
AI = analoginput('nidaq','Dev1');
% add two channels for analogue input.  One to determine WHEN to write
% fitness values to files (based on freq of input into chip), the other
% to measure the actual VALUES to write to the files (based on output of
% chip).  These correspond to AI inputs AI0 (pin 68) and AI1 (pin 33) on
% the terminal block.
addchannel(AI,0:1,{'output_from_chip','input_into_chip'});
AI.Channel.InputRange = [-10 10];
set(AI,'InputType', 'SingleEnded');
set(AI,'SampleRate', 50000);
set(AI,'SamplesPerTrigger', inf);
set(AI,'TimerPeriod', 0.5);

bsize = (AI.SampleRate)*(AI.TimerPeriod);
Fs = get(AI,'SampleRate');

% maxRate = daqhwinfo(AI, 'MaxSampleRate')
```

```
% set(AI, 'SampleRate', 50000);
% set(AI, 'SamplesAcquiredFcnCount', 10000);
% bsize = 10000;
% set(AI,'SampleRate', 50000);
% set(AI, 'SamplesAcquiredFcn', {@plot_output_of_chip, bsize, Fs});

set(AI,'TimerFcn',{@plot_output_of_chip, bsize, Fs})

% plot config stuff
% fighandle = 1;
% ScreenSize is a four-element vector: [left, bottom, width, height]:
% scrsz = get(0,'ScreenSize');
% Figure position. This property specifies the size and location
% on the screen of the figure window. Specify the position rectangle
% with a four-element vector of the formrect = [left, bottom, width,
% height]
% figure('Position',[scrsz(3)/2 scrsz(2) scrsz(3)/2 scrsz(4)])
% figure('Position', scrsz)
% % stop screen flicker
% set(gcf,'doublebuffer','on')

start(AI);

%%%% END OF PROGRAM %%%%

%%%% FUNCTIONS %%%%
function plot_output_of_chip(obj, event, bsize, Fs)

% 2 channels in data
data = zeros(bsize, 2);
data = getdata(obj, bsize);
% now take first SamplesToPlot of that
SamplesToPlot = 320;
% define our variables to create the ASCII binding signature
Signature = 4;
SigDim = SamplesToPlot/Signature;
SigDimV = ones(1,Signature)*SigDim;
Scales = 32;
Bases = 4;
BaseDim = Scales/Bases;
BaseDimV = ones(1,Bases)*BaseDim;
% get output from chip in 1st channel
samples = data(1:SamplesToPlot, 1);

% % show analysed signal
% subplot(411)
% plot(samples);
% title('Analysed signal.');
% set(gca,'Xlim',[1 SamplesToPlot],'Ylim',[-4 4]);
% set(gca,'Xlim',[1 SamplesToPlot]);
%
% % Perform continuous wavelet transform by mexh at all integer
% % scales from 1 to Scales.
% subplot(412)
%  coefficients = cwt(samples, 1:Scales, 'morl', 'plot');
coefficients = cwt(samples, 1:Scales, 'morl');
% title('Continuous Transform, absolute coefficients.')
% ylabel('Scale')
% xlabel('Sample')
% set(gca,'Zlim',[0 4]);

% Divide the data matrix into a series of cells.  We need the 4 bases (A,
% C, G, T) as rows, with the signature length as the number of columns
% (SigDim).  Thus scales gives us the total rows in the matrix, and
% SamplesToPlot is the total number of columns in the matrix.  The cells
% are used to create a new matrix.  Each cell is used to create a
% Frobenius-norm of the values in the matrix and that average is used in
% the new matrix to create the final binding signatures.
% e.g.  ccfs = Scales x SamplesToPlot
%       cells = BaseDim x SigDim
%       bindingSignature = prod(size(cells))

cells=mat2cell(coefficients, BaseDimV, SigDimV);

```

```
% create matrix of size we need
bindingSignature = zeros(size(cells));

% loop through cells, getting Frobenius-norm of each cell
% NB. The Frobenius-norm of matrixA, sqrt(sum(diag(A'*A))).
for s=1:numel(cells)
    bindingSignature(s)=norm(cells{s}, 'fro');
end

% horizontal flip of matrix to keep plots pretty
bindingSignature = flipud(bindingSignature);

threshold = max(max(bindingSignature)) / 2;

% if true, insert 1 into truth matrix
ThresholdedValues = bindingSignature > threshold;

% create mask of form
% AAAA
% CCCC
% GGGG
% TTTT
% NB. ASCII values: A=65, C=67, G=71, 84=T
% first create ones matrix, then convert to int to map to ASCII values
Sig = ones(Bases, Signature);
A = Sig(1,:)*65;
C = Sig(2,:)*67;
G = Sig(3,:)*71;
T = Sig(4,:)*84;

SigCell = cell(size(Sig));
SigCell(1,:) = cellstr(char(A)')';
SigCell(2,:) = cellstr(char(C)')';
SigCell(3,:) = cellstr(char(G)')';
SigCell(4,:) = cellstr(char(T)')';
% logical index using mask gives binding signature as vector of columns
% e.g.
%   AA
% C
%  G G
%  T
% would be C, GT, A, AG (gaps are zero values)
SigCell(~ThresholdedValues)={' '};

% write to dummy array
for i=1:Signature
    bs(i,:) = reshape(SigCell(:,i),1,4);
end

% subplot(413)
% imagesc(bindingSignature);
% title(['Continuous transform Frobenius normalised into cells, ',
%   num2str(BaseDim),'x', num2str(SigDim),'.'])
% colormap(cool(128));
% ylabel('Bases')
% set(gca,'Ylim',[0.5 4.5]);
%
%
% subplot(414)
% imagesc(ThresholdedValues);
% title(['Logical values of binding signature bases (A, C, G, T) after
%   thresholding at ', num2str(1), '.'])
% ylabel('Bases')
% set(gca,'Ylim',[0.5 4.5]);

% write binding signature of chip output to file.
try
    dlmwrite('D:\BindingSignature.txt', bs,'delimiter','')
catch
    pause(0.25)
    dlmwrite('D:\BindingSignature.txt', bs,'delimiter','')
end

% calculate freq getting 2nd input channel as signal input to chip
```

```
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
% NB. data channels = {'output_from_chip','input_into_chip'});
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
% Work out freq input into chip
[f,mag] = daqdocfft(data(:,2),Fs,bsize);

% Find the maximum value of the input signal, use it to say when to
% perform a fitness test.  Test Points currently defined at 1K, 5K and
% 10K. NB. ymax = max amplitude
[ymax,maxindex] = max(mag);
maxfreq = f(maxindex);
%plot(f,mag)
band = 100;

% test point freqs
LFTP = 1000;
MFTP = 5000;
HFTP = 10000;

 if LFTP - band < maxfreq && maxfreq < LFTP + band
     fitnessScore = calculateFitnessScores(data(:,1), bsize, Fs);
     try
         dlmwrite('D:\LowFreqMaxPower.txt',fitnessScore)
    catch
         pause(0.25)
         dlmwrite('D:\LowFreqMaxPower.txt',fitnessScore)
    end
     %disp([num2str(maxfreq),'-   ', num2str(fitnessScore)])

 elseif MFTP - band < maxfreq && maxfreq < MFTP + band
    fitnessScore = calculateFitnessScores(data(:,1), bsize, Fs);
    try
         dlmwrite('D:\MidFreqMaxPower.txt',fitnessScore )
    catch
         pause(0.25)
         dlmwrite('D:\MidFreqMaxPower.txt',fitnessScore )
    end
    %disp([num2str(maxfreq),'-  ', num2str(fitnessScore)])

 elseif HFTP - band < maxfreq && maxfreq < HFTP + band
    fitnessScore = calculateFitnessScores(data(:,1), bsize, Fs);
    try
         dlmwrite('D:\HighFreqMaxPower.txt',fitnessScore)
    catch
         pause(0.25)
         dlmwrite('D:\HighFreqMaxPower.txt',fitnessScore)
    end
    %disp([num2str(maxfreq),'-  ', num2str(fitnessScore)])
 end

clear fitnessScore

function fitnessScores = calculateFitnessScores(output, bsize, Fs)
% fft of chip output
 xfft = abs(fft(output));
 index = find(xfft == 0);
 xfft(index) = 1e-17;
% mag = 20*log10(xfft);
 mag = (xfft);
%  floor rounds numbers to inf
 mag = mag(1:floor(bsize/2));
 f = (0:length(mag)-1)*Fs/bsize;
 f = f(:);
% [ymax,maxindex]= max(mag);
% maxfreq = f(maxindex);

%[f,mag] = daqdocfft(output,Fs,bsize);

% need to scale the x-axis as it is scaled against blocksize and freq
% (mag is bsize/2 long)

band = int16(100/max(f)*length(mag));

% test point freqs (in Kilohertz)
```

```
249 TP1 = 1000;
250 TP2 = 5000;
251 TP3 = 10000;
252
253 LFTP = int16(TP1/max(f)*length(mag));
254 MFTP = int16(TP2/max(f)*length(mag));
255 HFTP = int16(TP3/max(f)*length(mag));
256
257 a = mean(mag(LFTP - band:LFTP + band));
258 b = mean(mag(MFTP - band:MFTP + band));
259 c = mean(mag(HFTP - band:HFTP + band));
260
261
262 fitnessScores = [a,b,c];
```

A.2 Wiring algorithm code

Listing A.2: Visual Basic code for wiring algorithm, showing Anadigm API, the use of loops through object collections and the required localised error handling within each loop (see §5.3 and §5.6.1 for full description of wiring algorithm and error handling issues). The application had 3178 lines of code.

```
1  Private Sub WireUpExpressedCAMs(ByVal ExpressedPhenotype As List(Of Node), ByVal phenotypeIndex
       As Integer)
2          'Wires up what has been expressed of a phenotype, i.e. if only
3          '2 CAMs of a 5 node genotype have been expressed in the
4          'phenotype, the code tries to wire them up according to the
5          'rules held in the genome.  This is likely to mean CAMs are
6          'left floating or only connected to themselves, or that the
7          'circuit has no input or output. To solve this, the wiring
8          'algorithm tries to see if the expression has left any
9          'dangling outputs or inputs. If so, it tries to wire these to
10         'the input or output as required.  If this fails, then the
11         'algorithm makes a decision to wire up circuit input using the
12         'first cam with a free input.  It then traces to the first
13         'free output for circuit output. Some errors occur, as a CAM
14         'can change its numbers of inputs or outputs according option
15         'chosen. Hence much local error trapping, as the code should
16         'drop through to the next option.  However, this is causing a
17         'lot of errors, and we get modal error messages raised
18         'occasionally - might need a fix for this if we start to do
19         'long runs overnight.
20         Dim node As New Node
21         Dim input As New Input
22         Dim output As New Output
23         Dim currentCAM As AnadigmDesigner2.ICam
24         Dim param As AnadigmDesigner2.ICamParameter
25         Dim contact As AnadigmDesigner2.IContact
26         Dim inputContact As String = ""
27         Dim outputContact As String = ""
28         Dim errorMessage As String = ""
29         Dim ConnectToCamID As Integer = 0
30         Dim CircuitOutputConnected As Boolean = False
31         Dim CircuitInputConnected As Boolean = False
32         Dim n As Integer = 0
33         Dim diff As Double = 0.0
34         Dim ConnectingNode As New Node
35
36         node = New Node
37         input = New Input
38         output = New Output
39
40         Try
41         For Each node In ExpressedPhenotype
42                 ' node type and id identifies the cam on this chip
43                 currentCAM = currentchip.Cams(node.CamType & "_NodeID:" & node.NodeID)
```

```
44                ' now loop through the wiring specification in the
45                ' node.
46                For Each input In node.InputConnections
47                        If input.ConnectingNodeIndex = CircuitInputFeed Then
48                                'if circuitinputfeed is detected,
49                                'connect this cam to circuit input
50                                inputContact = CAMArray(node.CamID).InputConnections(0)
51                                outputContact = currentchip.Name & "\InputCell1\Out"
52                                'check if contacts are actually
53                                'there?!
54                                Try
55                                        currentCAM.Contacts(inputContact).ConnectWire(
                                              outputContact)
56                                        CircuitInputConnected = True
57                                Catch ex As Exception
58                                        errorMessage = vbCrLf & ex.Message
59                                End Try
60                        Else
61                                'connect remaining inputs (but don't
62                                'bother trying to connect to nodes
63                                'that haven't been expressed)
64                                ConnectingNode = New Node
65                                ConnectingNode = population(phenotypeIndex).Item(input.
                                      ConnectingNode)
66                                Try
67                                        If ExpressedPhenotype.Contains(ConnectingNode) Then
68                                                ConnectToCamID = ConnectingNode.CamID
69                                                inputContact = CAMArray(node.CamID).
                                                      InputConnections(input.MyIndex)
70                                                outputContact = currentchip.Name & "\\"
71                                                outputContact = outputContact & currentchip.Cams
                                                      (ConnectingNode.CamType & "_NodeID:" &
                                                      ConnectingNode.NodeID).Name & "\\"
72                                                outputContact = outputContact & CAMArray(
                                                      ConnectToCamID).OutputConnections(input.
                                                      ConnectingNodeIndex)
73                                                currentCAM.Contacts(inputContact).ConnectWire(
                                                      outputContact)
74                                        End If
75                                Catch ex As Exception
76                                        errorMessage = vbCrLf & ex.Message
77                                End Try
78                        End If
79                Next input
80                'try to set parameter settings on currentCAM to those
81                'held in phenotype node.Phenotype(holds) these as a
82                'percentage, so need to find range of acceptable
83                'values on CAM.  Obviously, these ranges change as
84                'values are applied (i.e. one setting affects another)
85                'so some errors are likely to be raised here. Only 4
86                'parameters are currently configured for each node.
87                'If a CAM has more than this, an error will be raised.
88                n = 0
89                For Each param In currentCAM.Parameters
90                        diff = param.UpperLimit - param.LowerLimit
91                        param.Value = node.Parameters(n) / 100 * diff
92                        n += 1
93                Next param
94        Next node
95
96        'if no input to the circuit exists, first try and find a
97        'dangling input.  if that fails, connect circuit input to last
98        'cam's inputs (see above)
99        If CircuitInputConnected = False Then
100               outputContact = currentchip.Name & "\InputCell1\Out"
101               'first try to see if there are dangling inputs after
102               'expression of cams
103               For Each node In ExpressedPhenotype
104                       input.MyIndex = FirstDanglingInput(node)
105                       If input.MyIndex >= 0 Then
106                               'we have a dangling input on this
107                               'node, so connect to inputcell1
108                               currentCAM = currentchip.Cams(node.CamType & "_NodeID:" _
109                                                             & node.NodeID)
```

```
110                                 inputContact = CAMArray(node.CamID).InputConnections(input.
                                        MyIndex)
111                                 Try
112                                         currentCAM.Contacts(inputContact).ConnectWire(
                                                outputContact)
113                                         CircuitInputConnected = True
114                                 Exit For
115                                 Catch ex As Exception
116                                         errorMessage = currentCAM.Name & "-input:" &
                                                inputContact & "->" & outputContact & vbCrLf & ex.
                                                Message
117                                 End Try
118                         End If
119                 Next node
120                 If CircuitInputConnected = False Then
121                         'no dangling input found, so connect first
122                         'input of first cam to be expressed we have to
123                         'check to see if an input is already
124                         'connected, as sometimes it doesn't allow
125                         'contact to same equipotential
126                         currentCAM = currentchip.Cams(0)
127                         For i As Integer = 0 To currentCAM.Contacts.Count - 1
128                                 contact = currentCAM.Contacts.Item(i)
129                                 If contact.Type = AdContactType.adInputContact Then
130                                         If contact.IsConnected(contact) = False Then
131                                                 Try
132                                                         currentCAM.Contacts(contact.Name).
                                                                ConnectWire(outputContact)
133                                                         CircuitInputConnected = True
134                                                 Exit For
135                                                 Catch ex As Exception
136                                                         errorMessage = currentCAM.Name & "-input
                                                                :" & contact.Name & "->" &
                                                                outputContact & vbCrLf & ex.Message
137                                                 End Try
138                                         End If
139                                 End If
140                         Next i
141                         'OK, all inputs on first cam were connected, so now try last cam.
142                         'If this fails, this phenotype really is a dud! (though if
143                         'expressedPhenotype.count = 1 it will be the same)
144                         If CircuitInputConnected = False Then
145                                 currentCAM = currentchip.Cams(ExpressedPhenotype.Count - 1)
146                                 For i As Integer = 0 To currentCAM.Contacts.Count - 1
147                                         contact = currentCAM.Contacts.Item(i)
148                                         If contact.Type = AdContactType.adInputContact Then
149                                                 If contact.IsConnected(contact) = False Then
150                                                         Try
151                                                                 currentCAM.Contacts(contact.Name
                                                                        ).ConnectWire(outputContact)
152                                                                 CircuitInputConnected = True
153                                                         Exit For
154                                                         Catch ex As Exception
155                                                                 errorMessage = currentCAM.Name &
                                                                        "-input:" & contact.Name &
                                                                        "->" & outputContact &
                                                                        vbCrLf & ex.Message
156                                                         End Try
157                                                 End If
158                                         End If
159                                 Next i
160                         End If
161                 End If
162         End If
163
164         '!!!CIRCUIT OUTPUT!!!!  connect first free CAM output to
165         'circuit output.  This may still end up with a broken circuit
166         'as collections do not guarantee order in which objects are
167         'accessed i.e. last CAM in collection may come first, and that
168         'CAM may be unconnected to others.  NB. no checks are made to
169         'see if this CAM has an input - we may need to do this later
170         '(see below comments about mesh object).
171         For Each currentCAM In currentchip.Cams
172                 inputContact = currentchip.Name & "\OutputCell1\In"
```

```
173                 For i As Integer = 0 To currentCAM.Contacts.Count - 1
174                         contact = currentCAM.Contacts.Item(i)
175                         If contact.Type = AdContactType.adOutputContact And _
176                            contact.IsConnected(contact) = False Then
177                                 Try
178                                         currentCAM.Contacts(contact.Name).ConnectWire(
                                                inputContact)
179                                         CircuitOutputConnected = True
180                                 Exit For
181                                 Catch ex As Exception
182                                         errorMessage = currentCAM.Name & "-output:" & contact.
                                                Name & "->" & inputContact & vbCrLf & ex.Message
183                                 End Try
184                         End If
185                 Next
186         Next currentCAM
187
188         'no free output found, so connect last output of last cam to
189         'be expressed
190         If CircuitOutputConnected = False Then
191                 currentCAM = currentchip.Cams(ExpressedPhenotype.Count - 1)
192                 inputContact = currentchip.Name & "\OutputCell1\In"
193                 For i As Integer = 0 To currentCAM.Contacts.Count - 1
194                         contact = currentCAM.Contacts.Item(i)
195                         If contact.Type = AdContactType.adOutputContact Then
196                                 Try
197                                         currentCAM.Contacts(contact.Name).ConnectWire(
                                                inputContact)
198                                         CircuitOutputConnected = True
199                                 Exit For
200                                 Catch ex As Exception
201                                         errorMessage = currentCAM.Name & "-output:" & contact.
                                                Name & "->" & inputContact & vbCrLf & ex.Message
202                                  End Try
203                         End If
204                 Next
205         End If
206
207         'rats!  one of those cases where the last cam can't be
208         'connected to an output e.g. DC voltage cam, so try to connect
209         'first cam's output.  NB. if only one cam was expressed, this
210         'will be the same as above and still fail.
211         If CircuitOutputConnected = False Then
212                 currentCAM = currentchip.Cams(0)
213                 inputContact = currentchip.Name & "\OutputCell1\In"
214                 For i As Integer = 0 To currentCAM.Contacts.Count - 1
215                         contact = currentCAM.Contacts.Item(i)
216                         If contact.Type = AdContactType.adOutputContact Then
217                                 Try
218                                         currentCAM.Contacts(contact.Name).ConnectWire(
                                                inputContact)
219                                         CircuitOutputConnected = True
220                                 Exit For
221                                 Catch ex As Exception
222                                         errorMessage = currentCAM.Name & "-output:" & contact.
                                                Name & "->" & inputContact & vbCrLf & ex.Message
223                                 End Try
224                         End If
225                 Next
226         End If
227
228         'check circuit is reasonably connected?  We need to loop
229         'through the mesh object here, via all the output contacts and
230         'see if each on is connected - if we continue until we reach
231         'the input of the circuit.  Thus circuit is good.  Need to do
232         'this, because broken circuits give errant wavelet transforms
233         'and can cause a "freak" binding to occur which gets a high
234         'fitness value and is later impossible to reproduce and
235         'completely confuses evolution.
236         Catch ex As Exception
237                 File.AppendAllText(logFilePath, vbCrLf & " Wiring failed, dud circuit - " & ex.
                        Message)
238         End Try
239
```

```
240      'debug stuff
241      If CircuitInputConnected = False Then
242           'ad2.Visible = True
243           File.AppendAllText(logFilePath, vbCrLf & "This circuit has no input.")
244           File.AppendAllText(logFilePath, errorMessage)
245      End If
246      If CircuitOutputConnected = False Then
247           Try
248                 File.AppendAllText(logFilePath, vbCrLf & "This circuit does not produce an
                        output.")
249                 File.AppendAllText(logFilePath, errorMessage)
250                 'inputContact = currentchip.Name & "\InputCell1\Out"
251                 'outputContact = currentchip.Name & "\OutputCell1\In"
252                 ''all else has failed - attempt bare wire as no ouput is possible
253                 'for this circuit??
254                 ''screwing up the run, but...
255                 'currentchip.IOCells(0).Contacts(inputContact).ConnectWire(outputContact)
256                 'ad2.Visible = True
257            Catch ex As Exception
258                 File.AppendAllText(logFilePath, errorMessage &" Bare wire failed - " & ex.
                        Message)
259            End Try
260      End If
261   Exit Sub
262   ''''''''''''''''''''
263 End Sub
```

A.3 Output logs of circuit creation and tests for a phenotype generation

Listing A.3: Output log from final generation of a specialist phenotype. Unfortunately the log prints longer lines than it is possible to display in some places. The output initially shows what has changed due to mutation from the clone of the current champion (as we use a 4+1 evolutionary scheme). This is followed by the full listing of this generation's genome values. Each of the genome specifications is listed, showing CAM option, parameter settings and binding site values. Finally the connected genome is listed (i.e. the full genome specifications after the wiring algorithm as adjusted input requests to match CAM inputs) and the circuit generation can begin. The phenotype tests are then shown, with each reconfiguration being logged at the frequency of input and the value of the binding signature that caused the reconfiguration. The median fitness of each phenotype is taken and the degree of variation between individual tests can be seen. The generation test ends by recording the best of this generation and noting the current champion score. Note too, the error on line 528, where Matlab fails to write signature file (probably due to its size in RAM getting too large), causing the application object for matlab to be killed, restarted and the test for that phenotype re-run.

```
1  Best for generation 73: 2360 (current champion: 2999)
2
3  Mutating gene 0 of genotype 0
4  Parameter value has changed from 12 to 55
5  Mutating gene 1 of genotype 0
6  Binding Signature has changed from 1 to 4
7  Mutating gene 2 of genotype 0
8  Input request has changed from -1 to 1
9  Mutating gene 0 of genotype 1
10 Input index has changed from -2 to 4
```

```
Mutating gene 0 of genotype 1
Binding Signature has changed from 1 to 3
Mutating gene 1 of genotype 1
Parameter value has changed from 4 to 99
Mutating gene 1 of genotype 1
Binding Signature has changed from 4 to 2
Mutating gene 2 of genotype 1
Binding Signature has changed from 1 to 3
Mutating gene 2 of genotype 1
Input index has changed from -1 to 3
Mutating gene 0 of genotype 2
Input request has changed from 1 to 2
Mutating gene 0 of genotype 2
Input request has changed from -1 to 1
Mutating gene 0 of genotype 2
Binding Signature has changed from 1 to 1
Mutating gene 1 of genotype 2
Binding Signature has changed from 3 to 4
Mutating gene 1 of genotype 2
Parameter value has changed from 40 to 49
Mutating gene 1 of genotype 2
Input index has changed from -1 to 5
Mutating gene 2 of genotype 2
Input index has changed from -1 to 5
Mutating gene 2 of genotype 2
Input index has changed from 0 to 3
Mutating gene 2 of genotype 2
Input index has changed from -1 to 3
Mutating gene 0 of genotype 3
Input request has changed from -1 to 2
Mutating gene 0 of genotype 3
Binding Signature has changed from 1 to 1
Mutating gene 0 of genotype 3
Parameter value has changed from 26 to 73
Mutating gene 0 of genotype 3
Input request has changed from -1 to 0
Mutating gene 1 of genotype 3
Parameter value has changed from 30 to 5
Mutating gene 1 of genotype 3
Input request has changed from -1 to 1
Mutating gene 1 of genotype 3
Input request has changed from 0 to 1
Mutating gene 1 of genotype 3
Binding Signature has changed from 3 to 4
Mutating gene 2 of genotype 3
Input index has changed from -1 to 1
Mutating gene 2 of genotype 3
Input request has changed from 2 to 2
Mutating gene 2 of genotype 3
Binding Signature has changed from 1 to 4
Mutating gene 2 of genotype 3
Binding Signature has changed from 4 to 4
Mutating gene 0 of genotype 4
Input index has changed from 1 to 1
Mutating gene 0 of genotype 4
Binding Signature has changed from 1 to 4
Mutating gene 0 of genotype 4
Input request has changed from 1 to 1
Mutating gene 0 of genotype 4
Binding Signature has changed from 2 to 4
Mutating gene 0 of genotype 4
CAM ID has changed from 89 to 24
Mutating gene 1 of genotype 4
Input request has changed from -1 to 1
Mutating gene 1 of genotype 4
Input index has changed from -1 to 1
Mutating gene 1 of genotype 4
Binding Signature has changed from 4 to 4
Mutating gene 1 of genotype 4
Input index has changed from 1 to 2
Mutating gene 1 of genotype 4
Input request has changed from 0 to 0
Mutating gene 2 of genotype 4
Input request has changed from 2 to 0
```

```
Mutating gene 2 of genotype 4
Input index has changed from -1 to 5
Mutating gene 2 of genotype 4
Binding Signature has changed from 1 to 3
Mutating gene 2 of genotype 4
Input request has changed from -1 to 1
Mutating gene 2 of genotype 4
Parameter value has changed from 43 to 3
Mutated Genome 0: 89,-2,-2,0,1,1,0,-1,-1,-1,-1,30,55,26,37,2,4,1,1,
     38,0,0,0,1,-1,-1,-1,-1,-1,-1,4,22,40,30,3,4,3,4,
     23,2,0,1,-1,-1,-1,-1,-1,-1,-1,43,23,16,56,1,1,1,3,
Mutated Genome 1: 89,-2,4,0,1,1,0,-1,-1,-1,-1,30,12,26,37,2,4,1,3,
     38,0,0,0,1,-1,-1,-1,-1,-1,-1,99,22,40,30,3,2,3,1,
     23,2,0,-1,-1,-1,-1,-1,-1,-1,3,43,23,16,56,1,1,3,3,
Mutated Genome 2: 89,-2,-2,0,1,2,0,-1,-1,1,-1,30,12,26,37,2,4,1,1,
     38,0,0,0,1,-1,5,-1,-1,-1,-1,4,22,49,30,4,4,3,1,
     23,2,3,-1,5,-1,3,-1,-1,-1,-1,43,23,16,56,1,1,1,3,
Mutated Genome 3: 89,-2,-2,0,1,1,0,2,-1,0,-1,30,12,73,37,2,4,1,1,
     38,0,0,1,1,1,-1,-1,-1,-1,-1,4,22,40,5,3,4,4,1,
     23,2,0,-1,-1,-1,-1,-1,1,-1,-1,43,23,16,56,1,1,4,3,
Mutated Genome 4: 24,0,3,2,2,1,1,2,4,0,3,30,12,26,37,4,4,1,4,
     38,0,0,0,2,-1,-1,-1,-1,1,1,4,22,40,30,3,4,3,1,
     23,0,0,-1,-1,1,5,-1,-1,-1,-1,3,23,16,56,1,3,1,3,

Fully Expressed Genome: 0
Number of nodes: 3
-----------------------------------
0. CAM ID: 89
Inputs (input index/inputting node/inputting node's output index):
Circuit Input, 1:0:1, 2:1:0,
Outputs (output index/connecting node/connecting node's input index):
1:0:1, 0:1:0, 1:1:1,  Parameters: 30,55,26,37,

CAM: HoldVoltageControlled
Dual Input
{89, 3:2,  []}

Binding signature: CT

1. CAM ID: 38
Inputs (input index/inputting node/inputting node's output index):
0:0:0, 1:0:1,
Outputs (output index/connecting node/connecting node's input index):
0:0:2,  Parameters: 4,22,40,30,

CAM: FilterLowFreqBilinear
External Cap Value
{38, 2:1,  [0.489 (0-100000), 1 (0.1-10), 0.8 (0.8-0.8), ]}

Binding signature: GT

2. CAM ID: 23
Inputs (input index/inputting node/inputting node's output index):
0:2:0,
Outputs (output index/connecting node/connecting node's input index):
0:2:0,  Parameters: 43,23,16,56,

CAM: FilterBilinear
Low Corner Frequency
{23, 1:1,  [20 (20-20), 80.1 (8-400), 1 (0.05-10), 0.25 (0.025-5), ]}

Binding signature: AA

Total dangling outputs: 0


Fully Expressed Genome: 1
Number of nodes: 3
-----------------------------------
0. CAM ID: 89
Inputs (input index/inputting node/inputting node's output index):
Circuit Input, 1:0:1, 2:1:0,
Outputs (output index/connecting node/connecting node's input index):
1:0:1, 0:1:0, 1:1:1,  Parameters: 30,12,26,37,
```

```
149
150 CAM: HoldVoltageControlled
151 Dual Input
152 {89, 3:2,  []}
153
154 Binding signature: CT
155
156 1. CAM ID: 38
157 Inputs (input index/inputting node/inputting node's output index):
158 0:0:0, 1:0:1,
159 Outputs (output index/connecting node/connecting node's input index):
160 0:0:2,  Parameters: 99,22,40,30,
161
162 CAM: FilterLowFreqBilinear
163 External Cap Value
164 {38, 2:1,  [0.489 (0-100000), 1 (0.1-10), 0.8 (0.8-0.8), ]}
165
166 Binding signature: GC
167
168 2. CAM ID: 23
169 Inputs (input index/inputting node/inputting node's output index):
170 0:2:0,
171 Outputs (output index/connecting node/connecting node's input index):
172 0:2:0,  Parameters: 43,23,16,56,
173
174 CAM: FilterBilinear
175 Low Corner Frequency
176 {23, 1:1,  [20 (20-20), 80.1 (8-400), 1 (0.05-10), 0.25 (0.025-5), ]}
177
178 Binding signature: AA
179
180 Total dangling outputs: 0
181
182
183 Fully Expressed Genome: 2
184 Number of nodes: 3
185 ----------------------------------
186 0. CAM ID: 89
187 Inputs (input index/inputting node/inputting node's output index):
188 Circuit Input, 1:0:1, 2:2:0,
189 Outputs (output index/connecting node/connecting node's input index):
190 1:0:1, 0:1:0, 1:1:1,  Parameters: 30,12,26,37,
191
192 CAM: HoldVoltageControlled
193 Dual Input
194 {89, 3:2,  []}
195
196 Binding signature: CT
197
198 1. CAM ID: 38
199 Inputs (input index/inputting node/inputting node's output index):
200 0:0:0, 1:0:1,
201 Outputs (output index/connecting node/connecting node's input index):
202 Dangling output warning! Index: 0,  Parameters: 4,22,49,30,
203
204 CAM: FilterLowFreqBilinear
205 External Cap Value
206 {38, 2:1,  [0.489 (0-100000), 1 (0.1-10), 0.8 (0.8-0.8), ]}
207
208 Binding signature: TT
209
210 2. CAM ID: 23
211 Inputs (input index/inputting node/inputting node's output index):
212 0:2:0,
213 Outputs (output index/connecting node/connecting node's input index):
214 0:0:2, 0:2:0,  Parameters: 43,23,16,56,
215
216 CAM: FilterBilinear
217 Low Corner Frequency
218 {23, 1:1,  [20 (20-20), 80.1 (8-400), 1 (0.05-10), 0.25 (0.025-5), ]}
219
220 Binding signature: AA
221
222 Total dangling outputs: 1
```

```
223
224
225 Fully Expressed Genome: 3
226 Number of nodes: 3
227 ------------------------------------
228 0. CAM ID: 89
229 Inputs (input index/inputting node/inputting node's output index):
230 Circuit Input, 1:0:1, 2:1:0,
231 Outputs (output index/connecting node/connecting node's input index):
232 1:0:1, 0:1:0,  Parameters: 30,12,73,37,
233
234 CAM: HoldVoltageControlled
235 Dual Input
236 {89, 3:2,  []}
237
238 Binding signature: CT
239
240 1. CAM ID: 38
241 Inputs (input index/inputting node/inputting node's output index):
242 0:0:0, 1:1:0,
243 Outputs (output index/connecting node/connecting node's input index):
244 0:0:2, 0:1:1,  Parameters: 4,22,40,5,
245
246 CAM: FilterLowFreqBilinear
247 External Cap Value
248 {38, 2:1,  [0.489 (0-100000), 1 (0.1-10), 0.8 (0.8-0.8), ]}
249
250 Binding signature: GT
251
252 2. CAM ID: 23
253 Inputs (input index/inputting node/inputting node's output index):
254 0:2:0,
255 Outputs (output index/connecting node/connecting node's input index):
256 0:2:0,  Parameters: 43,23,16,56,
257
258 CAM: FilterBilinear
259 Low Corner Frequency
260 {23, 1:1,  [20 (20-20), 80.1 (8-400), 1 (0.05-10), 0.25 (0.025-5), ]}
261
262 Binding signature: AA
263
264 Total dangling outputs: 0
265
266
267 Fully Expressed Genome: 4
268 Number of nodes: 3
269 ------------------------------------
270 0. CAM ID: 24
271 Inputs (input index/inputting node/inputting node's output index):
272 Circuit Input,
273 Outputs (output index/connecting node/connecting node's input index):
274 0:1:0, 0:1:1, 0:2:0,  Parameters: 30,12,26,37,
275
276 CAM: FilterBiquad
277 Low Pass
278 {24, 1:1,  [40 (8-400), 1 (0.1-100), 0.707 (0.06-70), ]}
279
280 Binding signature: TT
281
282 1. CAM ID: 38
283 Inputs (input index/inputting node/inputting node's output index):
284 0:0:0, 1:0:0,
285 Outputs (output index/connecting node/connecting node's input index):
286 Dangling output warning! Index: 0,  Parameters: 4,22,40,30,
287
288 CAM: FilterLowFreqBilinear
289 External Cap Value
290 {38, 2:1,  [0.489 (0-100000), 1 (0.1-10), 0.8 (0.8-0.8), ]}
291
292 Binding signature: GT
293
294 2. CAM ID: 23
295 Inputs (input index/inputting node/inputting node's output index):
296 0:0:0,
```

```
297 Outputs (output index/connecting node/connecting node's input index):
298 Dangling output warning! Index: 0,  Parameters: 3,23,16,56,
299
300 CAM: FilterBilinear
301 Low Corner Frequency
302 {23, 1:1,  [20 (20-20), 80.1 (8-400), 1 (0.05-10), 0.25 (0.025-5), ]}
303
304 Binding signature: AG
305
306 Total dangling outputs: 2
307
308
309 Connected Genome: 0: 89,-2,-2,0,1,1,0,-1,-1,-1,-1,30,55,26,37,2,4,1,1,
        38,0,0,0,1,-1,-1,-1,-1,-1,-1,4,22,40,30,3,4,3,4,
        23,2,0,-1,-1,-1,-1,-1,-1,-1,-1,43,23,16,56,1,1,1,3,
310 Connected Genome: 1: 89,-2,-2,0,1,1,0,-1,-1,-1,-1,30,12,26,37,2,4,1,3,
        38,0,0,0,1,-1,-1,-1,-1,-1,-1,99,22,40,30,3,2,3,1,
        23,2,0,-1,-1,-1,-1,-1,-1,-1,-1,43,23,16,56,1,1,3,3,
311 Connected Genome: 2: 89,-2,-2,0,1,2,0,-1,-1,-1,-1,30,12,26,37,2,4,1,1,
        38,0,0,0,1,-1,-1,-1,-1,-1,-1,4,22,49,30,4,4,3,1,
        23,2,0,-1,-1,-1,-1,-1,-1,-1,-1,43,23,16,56,1,1,1,3,
312 Connected Genome: 3: 89,-2,-2,0,1,1,0,-1,-1,-1,-1,30,12,73,37,2,4,1,1,
        38,0,0,1,0,-1,-1,-1,-1,-1,-1,4,22,40,5,3,4,4,1,
        23,2,0,-1,-1,-1,-1,-1,-1,-1,-1,43,23,16,56,1,1,4,3,
313 Connected Genome: 4: 24,-2,-2,-1,-1,-1,-1,-1,-1,-1,-1,30,12,26,37,4,4,1,4,
        38,0,0,0,0,-1,-1,-1,-1,-1,-1,4,22,40,30,3,4,3,1,
        23,0,0,-1,-1,-1,-1,-1,-1,-1,-1,3,23,16,56,1,3,1,3,
314
315 Creating circuit 0 of phenotype 0
316 Creating circuit 1 of phenotype 0
317 Creating circuit 2 of phenotype 0
318 Creating circuit 3 of phenotype 0
319 Creating circuit 4 of phenotype 0
320 Creating circuit 5 of phenotype 0
321 Creating circuit 6 of phenotype 0
322 This circuit has no input.
323 Creating circuit 0 of phenotype 1
324 Creating circuit 1 of phenotype 1
325 Creating circuit 2 of phenotype 1
326 Creating circuit 3 of phenotype 1
327 Creating circuit 4 of phenotype 1
328 Creating circuit 5 of phenotype 1
329 Creating circuit 6 of phenotype 1
330 This circuit has no input.
331 Creating circuit 0 of phenotype 2
332 Creating circuit 1 of phenotype 2
333 Creating circuit 2 of phenotype 2
334 Creating circuit 3 of phenotype 2
335 Creating circuit 4 of phenotype 2
336 Creating circuit 5 of phenotype 2
337 Creating circuit 6 of phenotype 2
338 This circuit has no input.
339 Creating circuit 0 of phenotype 3
340 Creating circuit 1 of phenotype 3
341 Creating circuit 2 of phenotype 3
342 Creating circuit 3 of phenotype 3
343 Creating circuit 4 of phenotype 3
344 Creating circuit 5 of phenotype 3
345 Creating circuit 6 of phenotype 3
346 This circuit has no input.
347 Creating circuit 0 of phenotype 4
348 Creating circuit 1 of phenotype 4
349 Creating circuit 2 of phenotype 4
350 Creating circuit 3 of phenotype 4
351 Creating circuit 4 of phenotype 4
352 Creating circuit 5 of phenotype 4
353 Creating circuit 6 of phenotype 4
354 Generation 74
355  (current champion: 2999)
356
357 Now testing phenotype 0 of generation 74
358 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
359 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_0_circuit_5.ahf
```

```
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2327, HighFreq: 154
Fitness score: 2373
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2271, HighFreq: 154
Fitness score: 2317
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2330, HighFreq: 154
Fitness score: 2376
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2315, HighFreq: 153
Fitness score: 2362
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_0_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2320, HighFreq: 153
Fitness score: 2367
Median fitness score for this phenotype: 2367

Now testing phenotype 1 of generation 74
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_1_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 552, HighFreq: 154
Fitness score: 498
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_1_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 551, HighFreq: 154
Fitness score: 497
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_1_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 550, HighFreq: 153
Fitness score: 497
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_1_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 550, HighFreq: 154
Fitness score: 496
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_1_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 550, HighFreq: 152
```

```
Fitness score: 498
Median fitness score for this phenotype: 497

Now testing phenotype 2 of generation 74
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of TTTT found, reconfigured to circuit: 1 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2323, HighFreq: 701
Fitness score: 1922
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of TTTT found, reconfigured to circuit: 1 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 3039, HighFreq: 699
Fitness score: 2640
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of TTTT found, reconfigured to circuit: 1 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 3072, HighFreq: 709
Fitness score: 2663
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of TTTT found, reconfigured to circuit: 1 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2319, HighFreq: 669
Fitness score: 1950
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of TTTT found, reconfigured to circuit: 1 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_2_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 3061, HighFreq: 689
Fitness score: 2672
Median fitness score for this phenotype: 2640

Now testing phenotype 3 of generation 74
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
```

```
LowFreq: 0, MidFreq: 2358, HighFreq: 153
Fitness score: 2405
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2250, HighFreq: 724
Fitness score: 1726
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2321, HighFreq: 154
Fitness score: 2367
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2346, HighFreq: 153
Fitness score: 2393
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_5.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 1 at 5000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_3_circuit_6.ahf
Binding of AAAA found, reconfigured to circuit: 2 at 1000 kHz.
LowFreq: 0, MidFreq: 2287, HighFreq: 154
Fitness score: 2333
Median fitness score for this phenotype: 2367

Now testing phenotype 4 of generation 74
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_4_circuit_3.ahf
Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_4_circuit_1.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
LowFreq: 2627, MidFreq: 2962, HighFreq: 692
Fitness score: -157
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_4_circuit_3.ahf
Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_4_circuit_1.ahf
Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
LowFreq: 2628, MidFreq: 2992, HighFreq: 703
Fitness score: -139
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_4_circuit_3.ahf
Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
NB. At this point, Matlab fails to write the signature values to file, causing the main
    application to report an error.
No threshold signature file from Matlab?
Could not find file 'D:\Anadigm FPAA projects\Anadigm data files\LowFreqMaxPower.txt'. -
    repeating test for phenotype 4...
Retrieving the COM class factory for component with CLSID {A052DEB6-24BF-4425-B4AE-
    E8C55D264566} failed due to the following error: 800706ba. - repeating test for
    phenotype 4...
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
    phenotype_4_circuit_3.ahf
```

```
533 Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
534 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_1.ahf
535 Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
536 LowFreq: 2649, MidFreq: 3161, HighFreq: 640
537 Fitness score: 72
538 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
539 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_3.ahf
540 Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
541 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_1.ahf
542 Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
543 LowFreq: 2645, MidFreq: 2987, HighFreq: 645
544 Fitness score: -103
545 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
546 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_3.ahf
547 Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
548 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_1.ahf
549 Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
550 LowFreq: 2651, MidFreq: 2944, HighFreq: 714
551 Fitness score: -221
552 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
553 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_3.ahf
554 Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
555 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_1.ahf
556 Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
557 LowFreq: 2632, MidFreq: 2996, HighFreq: 700
558 Fitness score: -136
559 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\BareWire.ahf
560 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_3.ahf
561 Binding of TTTT found, reconfigured to circuit: 0 at 10000 kHz.
562 Reconfiguration file name: D:\Anadigm FPAA projects\Anadigm data files\
        phenotype_4_circuit_1.ahf
563 Binding of GTGTGTGT found, reconfigured to circuit: 01 at 5000 kHz.
564 LowFreq: 2634, MidFreq: 2942, HighFreq: 708
565 Fitness score: -200
566 Median fitness score for this phenotype: -136
567
568 Best for generation 74: 2640 (current champion: 2999)
```

Appendix B

Further examples of phenotypes

The purpose of this appendix is to show that the champion phenotypes in the experiments were able to find a variety of circuits that gave them high levels of fitness. Some CAMs seem to occur regularly in successful phenotypes, such as the GainSwitch CAM. The functional behaviour of this CAM reflects the requirements of the task, which ostensibly was to reconfigure the circuit on a change of input frequency. However, some phenotypes also employed circuits utilising CAMs such as GainSwitch or one of the bilinear filter CAMs, set at the correct corner frequency, so that the switch or filter would come pass through the signal (perhaps amplifying it in the process) at the right moment and no reconfiguration of the FPAA was required.

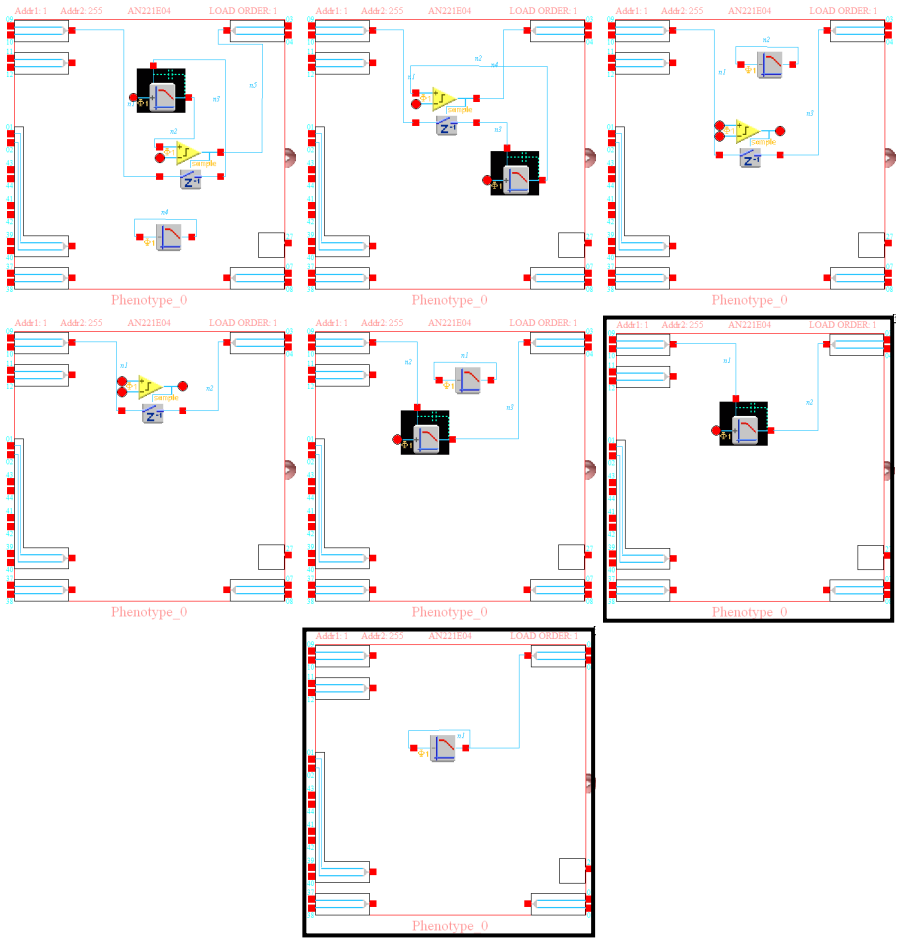


Figure B.1: Example of champion specialist phenotype (decreasing) demonstrating the wide variety of circuit configurations in the 7 possible gene expressions. The first circuit shows the fully specified genome (i.e. the result if all genes were expressed) containing FilterLowFreqBilinear, FilterBilinear and HoldVoltageControlled CAMs. Only 3 circuits could be deployed in the specialists environment, giving these genomes a large degree of redundancy. This phenotype scored 2999 and deployed just 2 circuits (no. 6 at 5kHz and no. 7 at 1kHz, shown with additional borders). The experiment is described in §6.6.

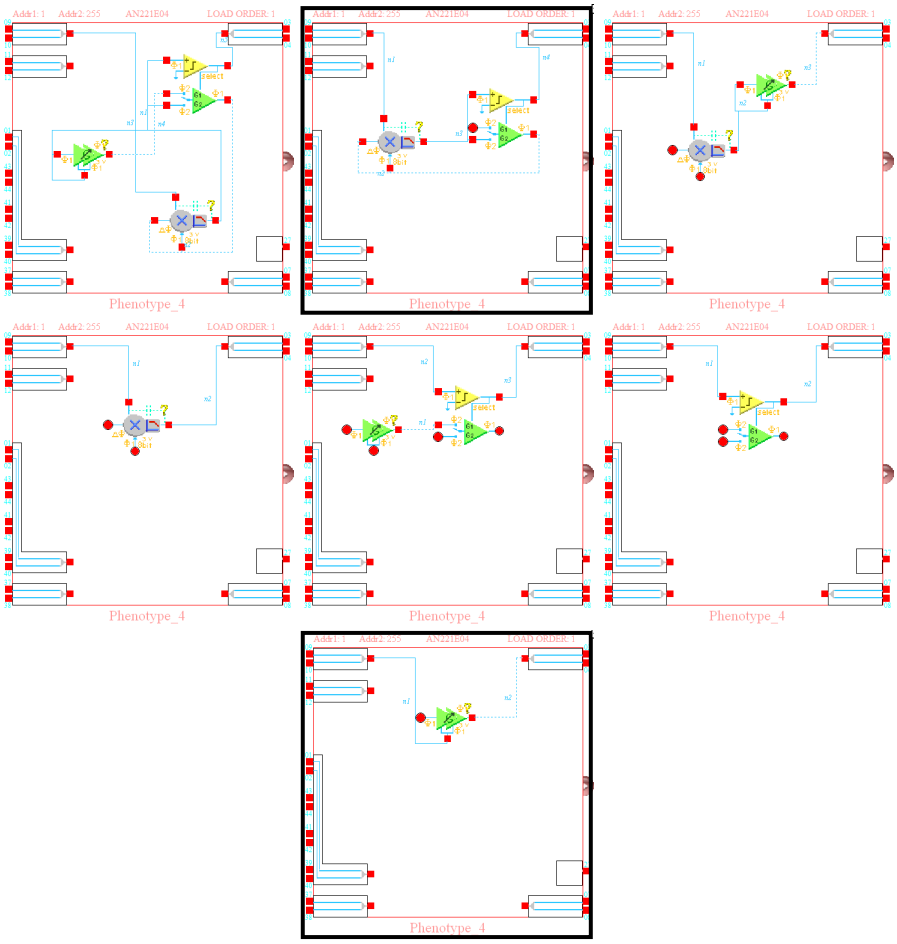


Figure B.2: Examples of champion specialist phenotype (decreasing), showing the 7 possible circuit expressions The first circuit shows the fully specified genome containing MultiplierFilterLowFreq, GainVoltageControlled and GainSwitch CAMs. This phenotype scored 3156 and deployed 2 circuits (no. 2 at 10kHz and no. 7 at 5kHz and 1kHz, shown with additional borders). The experiment is described in §6.6.

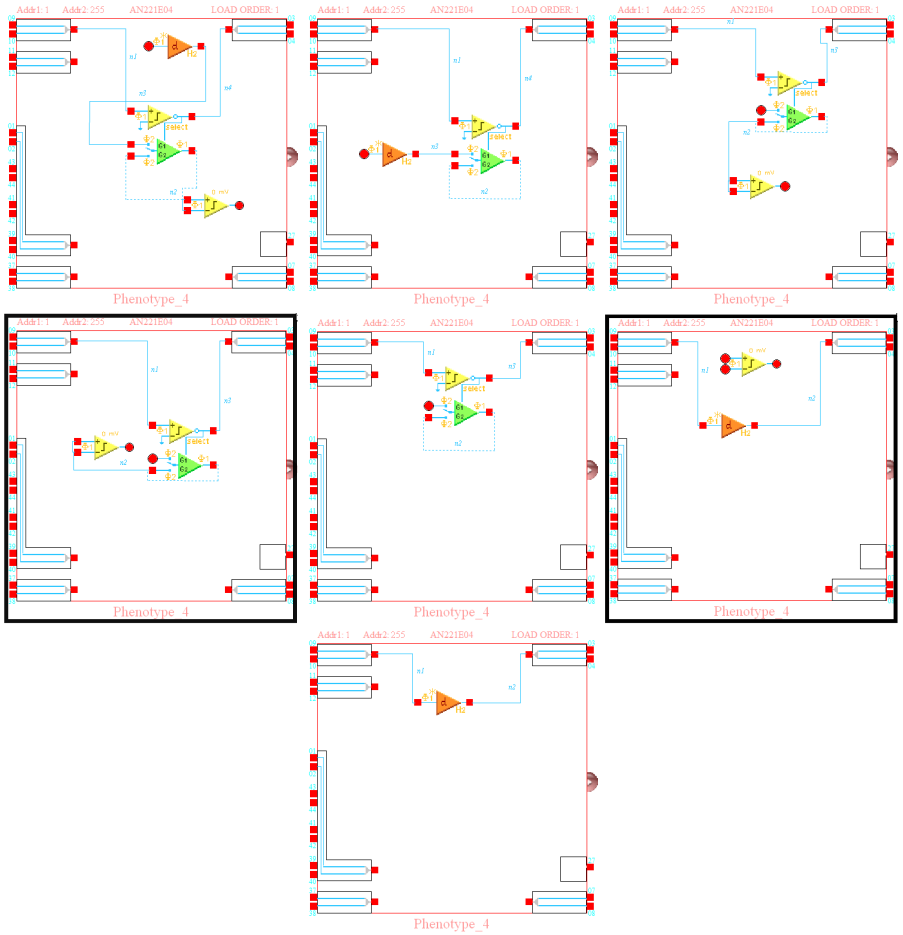


Figure B.3: Examples of champion non-specialist phenotype, showing the 7 possible circuit expressions. The first circuit shows the fully specified genome containing Differentiator, GainSwitch and Comparator CAMs. This phenotype scored 2735 and deployed 2 circuits (no. 4 at 5kHz and no. 6 at 1kHz, shown with additional borders). The experiment is described in §6.6.

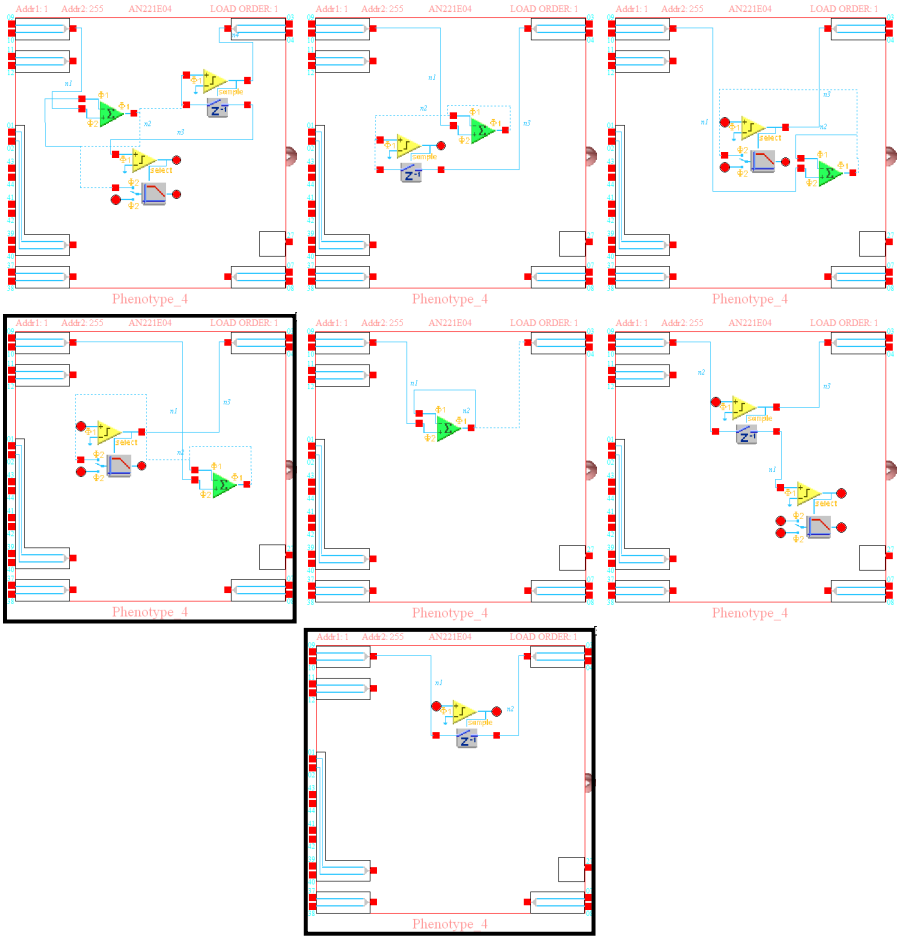


Figure B.4: Examples of champion non-specialist phenotype, showing the 7 possible circuit expressions. The first circuit shows the fully specified genome containing HoldVoltageControlled, GainSwitch and SumDiff CAMs. This phenotype scored 2670 and deployed 2 circuits (no. 4 at 1kHz and no. 7 at 5kHz, shown with additional borders). The experiment is described in §6.6.

Appendix C

Further results

This appendix lists some additional results to those given in §6.8.

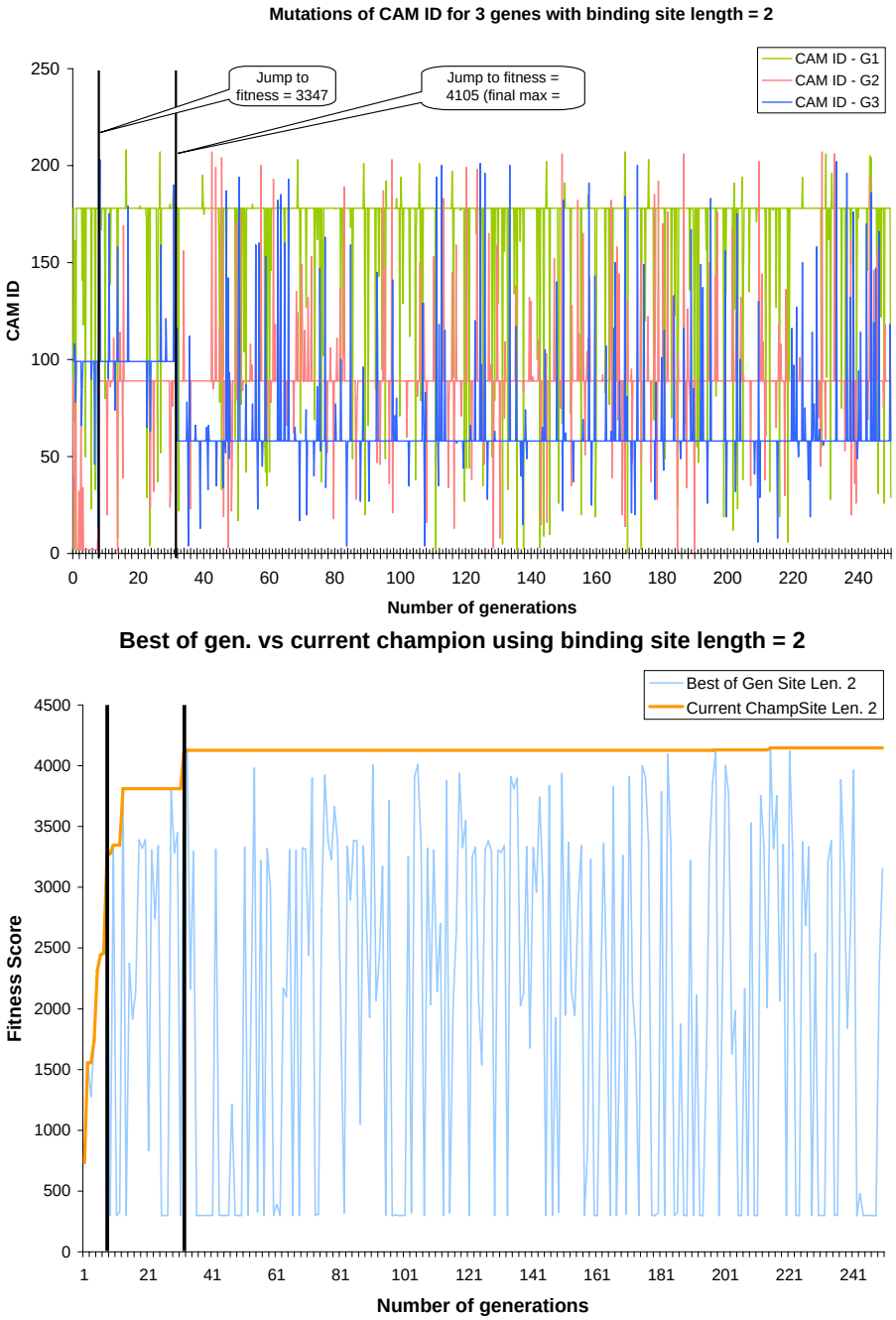


Figure C.1: Chart showing the stability of CAM ID to mutation over a long evolutionary run, with binding site lengths of 2, applying one test per phenotype. This run of 250 generations took almost 2 days to complete.

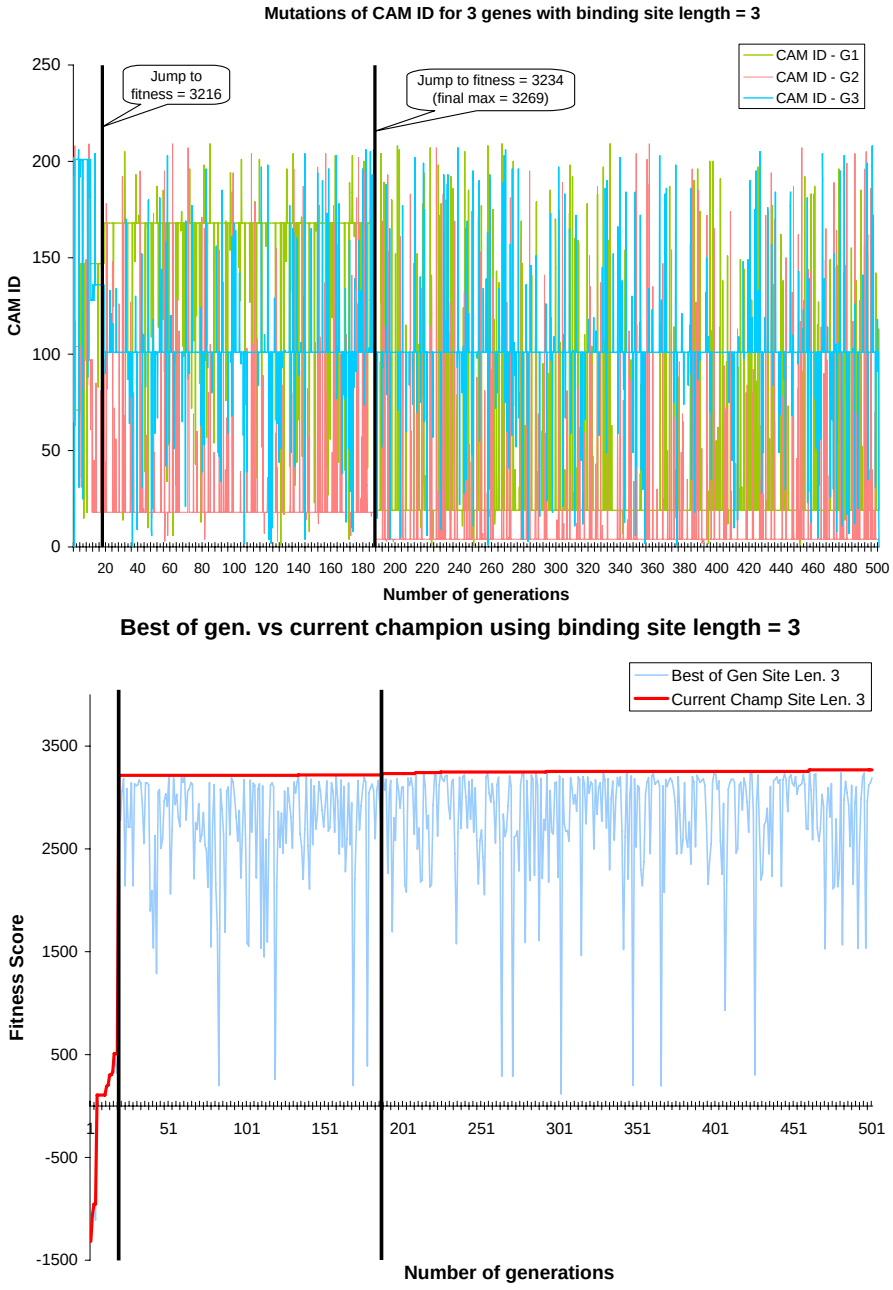


Figure C.2: Chart showing the stability of CAM ID to mutation over a long evolutionary run with binding site lengths of 3, applying one test per phenotype. Note reduction in the amount of noise (best of generation fitness scores) compared to previous chart showing binding site length =2. This run of 500 generations took 3.5 days (83.3 hours) to complete.

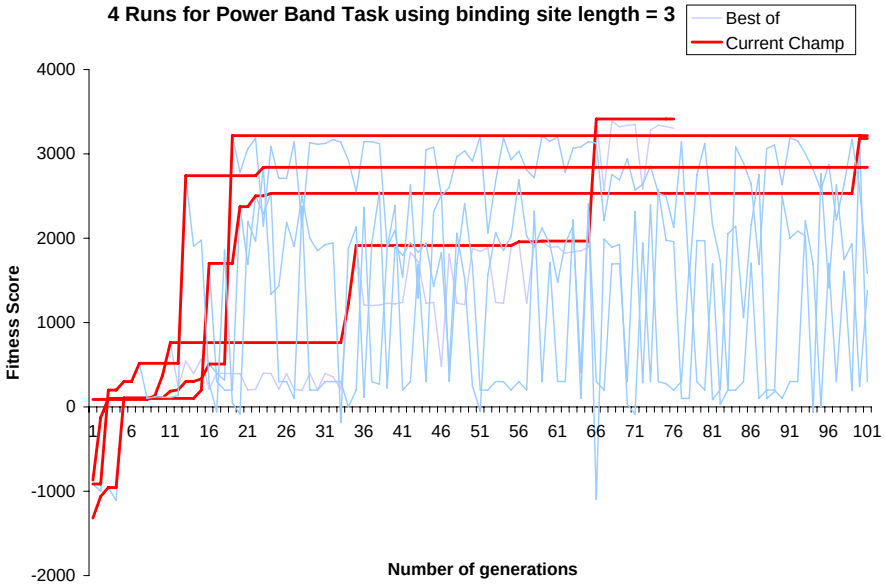


Figure C.3: Early runs investigating effects of increasing binding site length to 3. Chart shows best of generations against current champion phenotype. Despite relatively high fitnesses, large amounts of noise are present in each generation and the phenotypes were often unreliable. Depending on whether runs were mostly during the day or night, a run of 100 generations took around 16.5 hours to complete, so the 4 runs in total took 5 working days to finish.

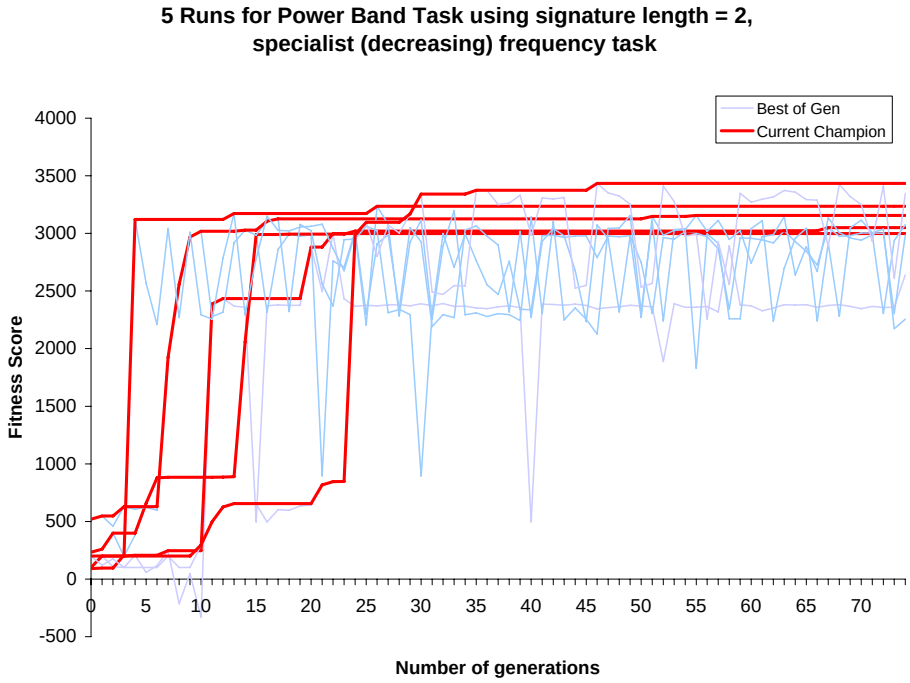


Figure C.4: Best of generation against current champion for specialist (decreasing) phenotype (see also Fig. 6.8). Note low levels of noise throughout all runs. Experiment is described in §6.6.

Bibliography

Aggarwal, V., M. Mao, and U.-M. O'Reilly (2006). A self-tuning analog proportional-integral-derivative (pid) controller. In *AHS*, pp. 12–19. IEEE Computer Society. 81

Alberts, B., A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter (2002, March). *Molecular Biology of the Cell: Fourth Edition*. Garland. 28

Attenborough, D. (1984). *The Living Planet*. BBC Worldwide Ltd.: BBC TV in association with Time Life Films. 12, 13, 14, 15

Bak, P. (1996). *How Nature Works: The Science of Self-Organised Criticality*. Copernicus Press. 46

Banzhaf, W. (1998). *Genetic Programming*. San Francisco: Morgan Kaufmann Publishers. 52

Banzhaf, W., G. Beslon, S. Christensen, J. Foster, F. Kepes, V. Lefort, J. F. Miller, M. RAdman, and J. J. Ramsden (2006, September). Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics 7*(9), 729–735. 11, 29, 72

Bentley, P. (2004a). The Garden Where Software Grows. *New Scientist 2437*. 10, 49

Bentley, P. J. (2004b). Fractal proteins. *Genetic Programming and Evolvable Machines 5*(1), 71–101. 87

Berenson, D., N. Estevez, and H. Lipson (2005). Hardware evolution of analog circuits for in-situ robotic fault-recovery. In *Evolvable Hardware*, pp. 12–19. IEEE Computer Society. 79, 81, 84

Bongard, J. C. and H. Lipson (2004). Automating genetic network inference with minimal physical experimentation using coevolution. See Deb et al. (2004), pp. 333–345. 49

Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics 6*(2), 81–101. 49

Brameier, M. (2003). *On linear genetic programming*. Ph. D. thesis, Dortmund. 52

Brameier, M. and W. Banzhaf (2003). Neutral variations cause bloat in linear GP. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa (Eds.), *Genetic Programming, Proceedings of EuroGP 2003*, Volume 2610 of *LNCS*, pp. 290–299. Springer-Verlag. 52

Bremermann, H. J. (1962). Optimization through evolution and recombination. In M. C. Yovitis and G. T. Jacobi (Eds.), *Self-Organizing Systems*, pp. 93–106. Washington, D.C.: Spartan Books. 53

Bremermann, H. J., M. Rogson, and S. Salaff (1966). Global properties of evolution processes. In H. H. Pattee, E. A. Edlsack, L. Fein, and A. B. Callahan (Eds.), *Natural Automata and Useful Simulations*, pp. 3–41. Washington D.C.: Spartan Books. 53

Cantu-Paz, E. (1998). Designing efficient master-slave parallel genetic algorithms. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 455. Morgan Kaufmann. 59

Cantu-Paz, E. and D. E. Goldberg (1997). Predicting speedups of ideal bounding cases of parallel genetic algorithms. In T. Bäck (Ed.), *Proc. of the Seventh Int. Conf. on Genetic Algorithms*, San Francisco, CA, pp. 113–120. Morgan Kaufmann. 59

Cardelli, L. (2005). Abstract machines of systems biology. *T. Comp. Sys. Biology 3737*, 145–168. 24

Carroll, S. (2006). *Endless Forms Most Beautiful: The New Science of Evo Devo and the Making of the Animal Kingdom*. Weidenfeld & Nicolson. 34, 35, 36, 37, 72, 153

Carroll, S. B., J. K. Grenier, and S. D. Weatherbee (2001). *From DNA to Diversity*. Blackwell. 29, 31, 33, 72

Clegg, K., S. Stepney, and T. Clarke (2007). Using feedback to regulate gene expression in a developmental control architecture. See Lipson (2007), pp. 966–973. 82

Coello, C. A. (2000). An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surv. 32*(2), 109–143. 60

Crevier, D. (1993). *Ai*. New York: BasicBooks. 145

Daida, J. (2004). Considering the roles of structure in problem solving by a computer. In U.-M. O'Reilly, T. Yu, R. L. Riolo, and B. Worzel (Eds.), *Genetic Programming Theory and Practice II*, Chapter 5, pp. 67–71. Kluwer. 52

Daida, J. and A. M. Hilss (2003). Identifying structural mechanisms in standard genetic programming. In E. e. a. Cantú-Paz (Ed.), *Genetic and Evolutionary Computation – GECCO-2003*, Volume 2724 of *LNCS*, Chicago, pp. 1639–1651. Springer-Verlag. 52

Daida, J., A. M. Hilss, D. J. Ward, and S. L. Long (2005). Visualizing tree structures in genetic programming. *Genetic Programming and Evolvable Machines 6*(1), 79–110. 52

Darwin, C. (1859). *On the Origin of Species. A facsimile of the First Edition*. Oxford University Press. 14, 21

Darwin, C. and T. Henry (1974). *Charles Darwin, Thomas Henry Huxley. Autobiographies*. Oxford University Press. (edited by Gavin de Beer). 14

Daubechies, I. (1992). *Ten Lectures on Wavelets*. Capital City Press, Montpelier, Vermont. 104

Dawkins, R. (1989). *The Selfish Gene* (2nd ed.). Oxford University Press. 18, 19

Dawkins, R. (1996). *Climbing Mount Improbable*. Norton. 21, 22

Dawkins, R. (1998). Darwin and Darwinism. In *Microsoft Encarta Encyclopedia 98*. Microsoft Corporation. 17, 18

Deb, K., R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell (Eds.) (2004). *GECCO 2004 — Genetic and Evolutionary Computation Conference 2004, Proceedings, Part I*, Volume 3102 of *Lecture Notes in Computer Science*. Springer. 185, 192

Dimutrescu, D., B. Lazzerine, l. Jain, and A. Dumitrescu (2000). *Evolutionary Computing*. CRC Press LLC. 53, 54, 55

Dreyfus, H. (1992). *What Computers Still Can't Do*. Cambridge: MIT Press. 145

Ferald, R. D. (2001). The eye in focus. *Karger Gazette 64*, 2–4. Available online: http://www.karger.com/gazette/index.htm. 21, 22

Fernández, F., G. Galeano, J. A. Gómez, and J. L. Guisado (2004). Control of bloat in genetic programming by means of the island model. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. T. A. Kabán, and H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature - PPSN VIII*, Volume 3242 of *LNCS*, pp. 260–269. Springer-Verlag. 52

Ferraro, P., C. Godin, and P. Prusinkiewicz (2005). Toward a quantification of self-similarity in plants. *Fractals 13(2)*, 91–109. 67

Fleischer, K. W. and A. H. Barr (1994). A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. *Artificial Life III*, 389–416. 67

Fogel, D. B. (Ed.) (1998). *Evolutionary Computation: the fossil record*. Piscataway, NJ: IEEE Press. 54, 60

Fogel, D. B. (2000). *Evolutionary Computing*. IEEE. 21, 38, 39, 50, 51, 53, 54, 55, 56, 57, 58, 66

Fogel, D. B. and A. Ghozeil (1997). A note on representations and variation operators. *IEEE Trans. on Evolutionary Computation 1*(2), 159–161. 57

Fogel, L. (1962). Autonomous automata. *Industrial Research 4*, 14–19. 50

Fogel, L. (1963). *Biotechnology: Concepts and Applications*. Prentice Hall. 49

Fonlupt, C. (2005). Book review: Genetic programming IV: Routine human-competitive machine intelligence. *Genetic Programming and Evolvable Machines 6*, 231–233. 10, 51

Fraser, A. S. (1957). Simulation of genetic systems by automtic digital computers I.–Introduction. *Australian Journal of Biological Sciences 10*, 484–491. 53

Fraser, A. S. (1968). The evolution of purposive behaviour. In H. von Foerster, J. D. White, L. J. Peterson, and J. K. Russell (Eds.), *Purposive Systems*, pp. 15–23. Spartan Books. 53

Friedburg, R. (1958). A learning machine. *IBM Journal of Research and Development 2*, 2–13. 49

Friedman, G. J. (1956). Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis, University of California, Los Angeles. 49

Fuhrer, M., H. W. Jensen, and P. Prusinkiewicz (2006). Modeling hairy plants. *Proceedings of Pacific Graphics 68*(4), 333–342. 69

Gavrilets, S. (1997). Evolution and speciation on holey adaptive landscapes. *Trends in Ecology and Evolution 12*, 307–312. 41

Gehring, W. and K. Ikeo (1999). Pax 6: Mastering eye morphogenesis and eye evolution. *Trends Genet 15*, 371–377. 21

Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley. 53, 57

Goldberg, D. (2002). *The Design of Innovation: Lessons from and for competant Genetic Algorithms*. Kluwer Academic Publishers. 53, 61, 63

Goodwin, B. (1994). *How the Leopard Changed Its Spots: The Evolution of Complexity*. Charles Scribner's Sons. 20

Gordon, T. and P. Bentley (2002). On evolvable hardware. 64, 67

Gordon, T. G. (2001). Book review of Hardware Evolution by Adrian Thompson. In *Genetic Programming and Evolvable Machines*, 2(4). Kluwer Academic Publishers. 72

Gould, S. J. (1989). *Wonderful Life: The Burgess Shale and the Nature of History*. Norton. 20

Haddow, P. C. and J. Hoye (2007). Achieving a simple development model for 3D shapes: are chemicals necessary? See Lipson (2007), pp. 1013–1020. 70

Haddow, P. C. and G. Tufte (2001). Bridging The Genotype-Phenotype Mapping For Digital FPGAs. In *Evolvable Hardware*, pp. 109–115. IEEE Computer Society. 70

Hamilton, A., K. Papathanasiou, M. Tamplin, and T. Brandtner (1998). Palmo: Field programmable analogue and mixed-signal vlsi for evolvable hardware. In *ICES '98: Proceedings of the Second International Conference on Evolvable Systems*, London, UK, pp. 335–344. Springer-Verlag. 81, 83

Hancock, J. T. (2003). The principles of cell signalling. See Kumar and Bentley (2003). 24

Harding, S. and J. Miller (2004). Evolution in materio: Initial experiments with liquid crystal. In *Evolvable Hardware*, pp. 298–. IEEE Computer Society. 72, 97

Harding, S. and J. F. Miller (2003). A scalable platform for intrinsic hardware and in materio evolution. In *Evolvable Hardware*, pp. 231–234. IEEE Computer Society. 65, 97

Harding, S., J. F. Miller, and W. Banzhaf (2007). Self-modifying cartesian genetic programming. See Lipson (2007), pp. 1021–1028. 53

Hogeweg, P. (2000a). Evolving mechanisms of morphogenesis: on the interplay between differential adhesion and cell differentiation. *Journal of Theoretical Biology 203.4*, 317–33. 67

Hogeweg, P. (2000b). Shapes in the shadow: Evolutionary dynamics of morphogensis. *Artificial Life 6*(1), 85–101. 67

Holland, J. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM 3*, 297–314. 53

Holland, J. (1975). *Adaptation in Natural and Artifical Systems* (1st ed.). University of Michigan Press. 56, 57

Holland, J. (1992). *Adaptation in Natural and Artifical Systems* (Fifth printing, 1st MIT ed.). MIT Press. 53

Hornby, G. S., M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata (1999). Autonomous Evolution of Gaits with the Sony Quadruped Robot. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Volume 2, pp. 1297–1304. Morgan Kaufmann. 68

Hornby, G. S., H. Lipson, and J. B. Pollack (2001). Evolution of generative design systems for modular physical robots. In *IEEE International Conference on Robotics and Automation*. 68

Hornby, G. S. and J. B. Pollack (2001a). The advantages of generative grammatical encodings for physical design. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pp. 600–607. IEEE Press. 68

Hornby, G. S. and J. B. Pollack (2001b). Evolving L-systems to generate virtual creatures. *Computers and Graphics 25*(6), 1041–1048. 68

Hornby, G. S. and J. B. Pollack (2002). Creating high-level components with a generative representation for body-brain evolution. *Artif. Life 8*(3), 223–246. 69

Howard, J. (2001). *Darwin. A Very Short Introduction*. Oxford University Press. 13, 14, 15, 16, 20

Hubbard, B. (1998). *The World According to Wavelets — The Story of a Mathematical Technique in the Making*. A K Peters, Natick, MA. 103, 104

Hutcheon, P. D. (1996). Fear ignorance – not sociobiology! *Humanist in Canada 9*, 12–14. 48

Jacob, C. (1999). Lindenmayer systems and growth program evolution. In T. S. Hussain (Ed.), *Advanced Grammar Techniques Within Genetic Programming and Evolutionary Computation*, Orlando, Florida, USA, pp. 76–79. 67, 68

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing 9*(1), 3–12. 10

Kang, L., Y. Liu, and S. Y. Zeng (Eds.) (2007). *Evolvable Systems: From Biology to Hardware, 7th International Conference, ICES 2007, Wuhan, China, September 21-23, 2007, Proceedings*, Volume 4684 of *Lecture Notes in Computer Science*. Springer. 194, 195

Kauffman, S. (1995). *At Home in the Universe: The search of the laws of self-organization and complexity*. Oxford University Press. 37, 45

Kauffman, S. (2000). *Investigations*. Oxford University Press. 40, 42, 43, 45

Kauffman, S. and S. Johnsen (1991). Coevolution at the edge of chaos: coupled fitness landscapes, poised states and coevolutionary avalanches. *Journal of Theoretical Biology 149*, 467–482. 45

Kauffman, S. and S. Levin (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology 128*, 11–45. 38, 41

Keane, A. M. (2000). An Intoduction to Evolutionary Computing in Design Search and Optimisation. In A. M. Keane (Ed.), *Theoretical Aspects of Evolutionary Computing*. Kluwer Academic. 60

Keymeulen, D., R. S. Zebulum, A. Stoica, V. Duong, and M. I. Ferguson (2004). Evolvable hardware for signal separation and noise cancellation using analog reconfigurable device. In *FPL*, Volume 3203 of *Lecture Notes in Computer Science*, pp. 270–278. Springer. 83

Kirschner, M. (2005). *The Plausibility of Life*. New Haven: Yale University Press. 153

Kosman, D., C. M. Mizutani, D. Lemons, W. G. Cox, W. McGinnis, and E. Bier (2004). Multiplex detection of rna expression in drosophila embryos. *Science 305*(5685), 846. 33

Koza, J. R. (1989). Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan (Ed.), *Proc. of the 11th Joint Conf. on Genetic Algorithms*, San Francisco, CA, pp. 786–774. Morgan Kaufmann Publishers. 57

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press. 51

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press. 51, 53

Koza, J. R., D. Andre, F. H. Bennett III, and M. Keane (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann. 51

Koza, J. R., L. Jones, M. Keane, and M. Streeter (2004). Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In *Genetic Programming Theory and Practice II*, Chapter 8, pp. 121–138. Kluwer. 51, 59, 80, 150

Koza, J. R., M. Keane, and M. Streeter (2003, Feb). Evolving inventions. *Scientific American*, 52–59. 59, 60

Koza, J. R., M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers. 10, 51

Kumar, S. and P. Bentley (Eds.) (2003). *On Growth, Form and Computers*. Elsevier. 23, 25, 28, 67, 72, 189, 196

Langdon, W. B. (2000). Quadratic bloat in genetic programming. In D. Whitley, D. E. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Las Vegas, Nevada, USA, pp. 451–458. Morgan Kaufmann. 52

Langdon, W. B. and W. Banzhaf (2000). Genetic programming bloat without semantics. In *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Volume 1917 of *LNCS*, Paris, France, pp. 201–210. Springer Verlag. 52

Langdon, W. B. and R. Poli (1997). Genetic programming bloat with dynamic fitness. Technical Report CSRP-97-29, University of Birmingham, School of Computer Science. 52

Lenat, D. (1983). The role of heuristic in learning by discovery: Three case studies. In R. Michalski, J. Carbonell, and T. Michell (Eds.), *Machine Learning*, pp. 243–306. Tioga Publishing. 50

Lewontin, R. (1974). *The Genetic Basis of Evolutionary Change*. Columbia University Press. 65, 66

Lindenmayer, A. and P. Prusinkiewicz (1989). Developmental models of multicellular organisms: A computer graphics perspective. In C. G. Langton (Ed.), *Artificial Life*, pp. 221–249. Addison-Wesley. 67

Lipson, H. (Ed.) (2007). *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*. ACM. 186, 188, 189, 196

Lones, M. A. (2003). *Enzyme Genetic Programming: Modelling Biological Evolvability in Genetic Programming*. Ph. D. thesis, The University of York, Heslington, York, YO10 5DD, UK. 53, 67

MacKay, D. (2003, Sep). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press. 56

Mann, H. B. and D. R. Whitney (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics 18*, 50–60. 135

Mattiussi, C. (2005). *Evolutionary Synthesis of Analog Networks*. Ph. D. thesis, EPFL, Lausanne. 81, 151

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer. 56

Miller, J. (2000). Review: First NASA DOD Workshop on Evolvable Hardware 1999. *Genetic Programming and Evolvable Machines 1*, 171–174. 10

Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *GECCO*, Volume 2, pp. 1135–1142. Morgan Kaufmann. 53

Miller, J. F. (2001, 9-11). What bloat? cartesian genetic programming on boolean problems. In E. D. Goodman (Ed.), *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, San Francisco, California, USA, pp. 295–302. 52, 53

Miller, J. F. (2004). Evolving a Self-Repairing, Self-Regulating, French Flag Organism. See Deb et al. (2004), pp. 129–139. 52, 69, 87

Miller, J. F. and K. Downing (2002). Evolution in materio: Looking beyond the silicon box. pp. 167–176. IEEE Computer Society. 65, 72, 80

Miller, J. F. and S. L. Smith (2006, April). Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation 10*(2), 167–174. 87, 130

Miller, J. F. and P. Thomson (2000). Cartesian Genetic Programming. In *Genetic Programming, Proceedings of EuroGP 2000*, Volume 1802 of *LNCS*, pp. 121–132. Springer. 53

Miller, J. F., P. Thomson, and T. Fogarty (1997). *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications*, Chapter Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. Wiley. 52

Minsky, M. (1988). *Perceptrons*. Cambridge: MIT Press. 145

Měch, R. and P. Prusinkiewicz (1996). Visual models of plants interacting with their environment. In *SIGGRAPH*, pp. 397–410. 68, 70

NASADoD:2000 (2000). *Proc. 2000 NASA/DoD workshop on Evolvable Hardware*. IEEE Computer Society. 195

O'Neill, M. and C. Ryan (2003). *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers. 52

Peisach, D., P. Gee, C. Kent, and Z. Xu (2003). The crystal structure of choline kinase reveals a eukaryotic protein kinase fold. *Structure 11*, 703–713. 27

Poularikas, A. (1995). *The Transforms and Applications Handbook*. CRC Press, Boca Raton, FL. 103, 104

Provine, W. B. (1986). *Sewall Wright and Evolutionary Biology*. University of Chicago Press. 38, 41

Prusinkiewicz, P. (2000). Simulation modeling of plants and plant ecosystems. *Communications of the ACM 43*(7), 84–93. 67

Prusinkiewicz, P. and A. Lindenmayer (1990). *The Algorithmic Beauty of Plants*. Springer-Verlag. 68

Rechenberg, I. (1963). *Evolutionsstrategie: Optimierung Technisher Systeme nach Prinspipien der Biologischen Evolution*. Frommann-Holzboog. 51

Rechenberg, I. (1964). Kybernetische Lösungsansteuerung einer experimentellen Forschungsaufgabe. Presented at the Annual Conference of the WGLR at Berlin in September 1964. 51

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzbog. 51

Robinson, G. and P. McIlroy (1995). Exploring some commercial applications of genetic programming. In T. C. Fogarty (Ed.), *Evolutionary Computing*, Number 993. Springer-Verlag. 59

Ruse, M. (1996). Are Pictures Really Necessary? The Case of Sewall Wright's Adaptive Landscapes. In B. Baigre (Ed.), *Picturing Knowledge: Historical and Philosphical Problems Concerning the Use of Art in Science*. University of Toronto. 38

Saunders, P. (1993). The organism as a dynamical system. In W. Stein and F. Varela (Eds.), *Thinking about Biology*, Volume III, pp. 41–63. Addison-Wesley. 46, 47

Schemmel, J., K. Meier, and M. Loose (2002). A scalable switched capacitor realization of the resistive fuse network. *Analog Integr. Circuits Signal Process. 32*(2), 135–148. 81

Schraudolph, N. N. and R. K. Belew (1992). Dynamic parameter encoding for genetic algorithms. *Machine Learning 9*, 9–21. 51, 57

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. Ph. D. thesis, Technische Universität Berlin, Berlin, Germany. 51

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Basel, Stuttgart: Birkhäuser. Volume 26 of Interdisciplinary Systems Research. 51

Schwefel, H.-P. (1981). *Numerical optimization of Computer models*. John Wiley & Sons, Ltd. 51

Sharman, K. C., A. I. Esparcia Alcazar, and Y. Li (1995). Evolving signal processing algorithms by genetic programming. In A. M. S. Zalzala (Ed.), *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, Volume 414, Sheffield, UK, pp. 473–480. IEE. 59

Shipman, R., M. Shackleton, and I. Harvey (2000). The use of neutral genotype-phenotype mappings for improved evolutionary search. *BT Technology Journal 18*(4), 103–111. 65

Skipper, R. (2002). The Heuristic Role of Sewall Wright's 1932 Adaptive Landscape Diagram. In *Proceedings Philosophy of Science Assoc. 18th Biennial Mtg - PSA 2002: PSA 2002 Symposia*. 38, 41, 42

Solé, R. and B. Goodwin (2000). *Signs of Life: how complexity pervades biology*. Basic Books. 37, 40, 42, 44, 45

Soliman, S. and M. Srinath (1998). *Continuous and Discrete Signals and Systems*. Prentice Hall International, Upper Saddle River NJ. 103

Solomonoff, R. (1966). Some recent work in artificial intelligence. *Proc. of the IEEE 54:12*, 1687–1697. 50

Stadler, P. (2002). Fitness landscapes. In M. Lässig and A. Valleriani (Eds.), *Biological Evolution and Statistical Physics*, pp. 187–207. Springer-Verlag. 41

Stanley, K. and R. Mikkulainen (2003). A taxonomy for artificial embryogeny. *Artificial Life 9*(2), 93–130. 70

Stefatos, E. F., T. Arslan, D. Keymeulen, and I. Ferguson (2006). Autonomous realization of boeing/jpl sensor electronics based on reconfigurable system-on-chip technology. In *ISVLSI*, pp. 85–90. IEEE Computer Society. 83

Stephens, C. (2003). interviewed by EvoNet, `http://www.evonet.org`. 59

Stoica, A., D. Keymeulen, R. S. Zebulum, M. Mojarradi, S. Katkoori, and T. Daud (2007). Adaptive and evolvable analog electronics for space applications. See Kang et al. (2007), pp. 379–390. 79, 83

Streeter, M., M. Keane, and J. Koza (2003). Routine human-competitive automatic synthesis using genetic programming of both the topology and sizing for five post-2000 patented analog and mixed analog-digital circuits. In *2003 Southwest Symposium on Mixed-Signal Design*, pp. 5–10. IEEE Circuits and Systems Society. 51

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In J. D. Schaffer (Ed.), *Proc. of the Third Int. Conf. on Genetic Algorithms*, San Mateo, CA, pp. 2–9. Morgan Kaufmann. 57

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In L. Davis (Ed.), *Handbook of Genetic Algorithms*, pp. 332–349. Van Nostrand Reinhold. 57

Thompson, A. (1996). Silicon evolution. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 444–452. MIT Press. 64, 72, 79

Thompson, A. (1997). An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics. In *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, pp. 390–405. Springer-Verlag. 63, 64, 86

Thompson, A. (2002). Notes on design through artificial evolution: Opportunities and algorithms. In I. C. Parmee (Ed.), *Adaptive computing in design and manufacture V*, pp. 17–26. Springer-Verlag. 61, 63, 87, 128

Thompson, A., P. Layzell, and R. S. Zebulum (1999). Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution. *IEEE Transactions on Evolutionary Computation 3*(3), 167–196. 64, 79, 87

Tomassini, M., L. Vanneschi, J. Cuendet, and F. Fernandez (2004). A new technique for dynamic size populations in genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, Portland, Oregon, pp. 486–493. IEEE Press. 52

Torrensen, J. (2000). Scalable Evolvable Hardware Applied to Road Image Recognition. See NASADoD:2000 (2000), pp. 151–160. 63

Tufte, G. (2006). Cellular Development: A Search for Functionality. *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2669–2676. 87

Tufte, G. and P. Haddow (2003). Building knowledge into developmental rules for circuit design. *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, 69–80. 67

Tufte, G. and P. C. Haddow (2007). Extending artificial development: Exploiting environmental information for the achievement of phenotypic plasticity. See Kang et al. (2007), pp. 297–308. 87

Turing, A. (1950). Computing machinery and intelligence. *Mind 49*, 433–460. 49

Turing, A. (1952). The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society 237*(641), 37–72. 71

Van Valen, L. (1973). A new evolutionary law. *Evolutionary Theory 1*, 1–30. 44

Vanneschi, L. (2004). *Theory and Practice for Efficient Genetic Programming*. Ph. D. thesis, Faculty of Sciences, University of Lausanne, Switzerland. 52

Vassilev, V. K., D. Job, and J. F. Miller (2000). Towards the automatic design of more efficient digital circuits. See NASADoD:2000 (2000), pp. 151–160. 63

Voigt, H.-M., W. Ebeling, I. Rechenberg, and H.-P. Schwefel (Eds.) (1996). *Parallel Problem Solving from Nature – PPSN IV*, Berlin. Springer. 51

Vose, M. D. (1999a). *The simple genetic algorithm: foundations and theory*. Cambridge, MA: MIT Press. 53

Vose, M. D. (1999b). What are genetic algorithms? a mathematical perspective. In L. D. Davis, K. De Jong, M. D. Vose, and L. D. Whitley (Eds.), *Evolutionary Algorithms*, pp. 251–276. Springer. 53

Waddington, C. (1957). *The Strategy Of The Genes*. George Allen & Unwin. 46

Walker, J. and J. F. Miller (2008). The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*. (in press). 53, 86, 128, 130

Walker, J. A. and J. F. Miller (2007a). Changing the genospace: Solving ga problems with cartesian genetic programming. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. Esparcia-Alcázar (Eds.), *EuroGP*, Volume 4445 of *Lecture Notes in Computer Science*, pp. 261–270. Springer. 53, 130

Walker, J. A. and J. F. Miller (2007b). Solving real-valued optimisation problems using cartesian genetic programming. See Lipson (2007), pp. 1724–1730. 53, 130

Wearing, H., M. Owen, and J. Sherratt (2000). Mathematical modelling of juxtacrine patterning. *Bulletin of Mathematical Biology 62*, 293–320. 24

Wolpert, D. and W. Macready (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM. 58

Wolpert, D. and W. Macready (1997). No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation 1*(1), 67–82. 58

Wolpert, L. (1998). *The Principles of Development*. Oxford University Press. 23, 25, 69

Wolpert, L. (2003). Relationships Between Development And Evolution. See Kumar and Bentley (2003), Chapter 2, pp. 47–62. 23, 26, 29, 30, 31, 32, 73, 146

Wolpert, L., R. Beddington, and T. Jessell (2002). *Principles of Development*. Oxford University Press. 35

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In G. J. Rawlins (Ed.), *Foundations of genetic algorithms*, pp. 205–218. Morgan Kaufmann. 57

Wright, A. H., M. D. Vose, and J. E. Rowe (2003). Implicit parallelism. In *GECCO*, Volume 2724 of *Lecture Notes in Computer Science*, pp. 1505–1517. Springer. 57

Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth Annual Congress of Genetics*, Volume 1, pp. 356–366. 38, 41

NB. Numbers in red after the source show the page numbers in this document where the source is cited (hyperlinked in electronic version).