The Evolutionary Emergence route to Artificial Intelligence

Alastair Channon

Degree: MSc in Knowledge-Based Systems 1995/96
SCHOOL OF COGNITIVE AND COMPUTING SCIENCES
UNIVERSITY OF SUSSEX
Supervisor: Inman Harvey
Submitted: 2 September 1996

(Minor revisions October 1996)

Abstract

The artificial evolution of intelligence is discussed with respect to current methods. An argument for withdrawal of the traditional ‘fitness function’ in genetic algorithms is given on the grounds that this would better enable the emergence of intelligence, necessary because we cannot specify what intelligence is. A modular developmental system is constructed to aid the evolution of neural structures and a simple virtual world with many of the properties believed beneficial is set up to test these ideas. Resulting emergent properties are given, along with a brief discussion.

Keywords: Artificial Intelligence, Emergence, Genetic Algorithms, Artificial Life, Neural Networks, Development, Modularity, Fractals, Lindenmayer Systems, Recurrence.

Acknowledgments

Thanks to my supervisor Inman Harvey for his encouragement in the area and comments on this project.

CONTENTS

Figures

1 Introduction

1.1 Aims

This project is directed towards my long-term goal of developing artificial intelligences (AIs) that are capable of more than just a small set of tasks and can grasp profoundly new situations on their own. My interest is in intelligence such as ours and ants' rather than dedicated chess computers or current expert systems.

The project was aimed at the heart of the problem: forming a general system that would adapt to behave in an intelligent way within an environment, without being given any information about how to behave. The behaviour would have to ***emerge*** from the combination of the system and its environment.

I planned to achieve this by adapting genetic algorithms (GAs) from their conventional form towards natural evolution. I aimed to achieve emergence as in natural evolution, via coexistence of similarly-capable systems, rather than GAs' usual operation of function optimisation.

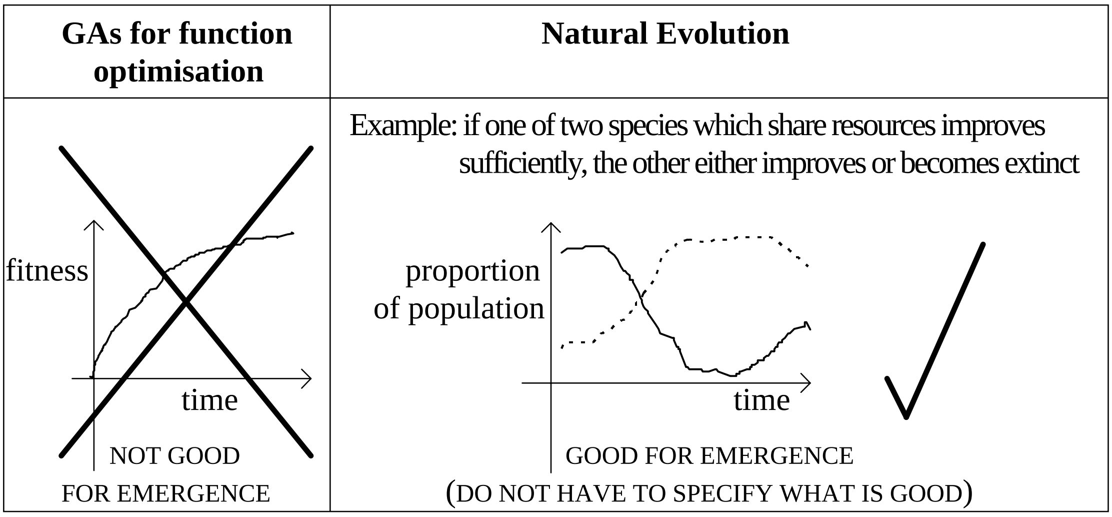
GAs for function optimisation
Natural Evolution
fitness
time
NOT GOOD
FOR EMERGENCE
Example: if one of two species which share resources improves sufficiently, the other either improves or becomes extinct
proportion of population
time
GOOD FOR EMERGENCE
(DO NOT HAVE TO SPECIFY WHAT IS GOOD)

Figure 1: The difference between GAs for function optimisation and natural evolution

In summary, the end results aimed for were demonstrable ***emergent*** properties, arising from the interactions between the items being evolved (within their environment), including some that could be considered ***intelligent***.

1.2 Rationale and Outline of the Dissertation

A genetic algorithm has to have objects to evolve. The choice of a suitable class of objects was made along the same lines as the choice of method used (GA): base it on something that has already worked. Natural evolution has evolved biological systems that run on massively parallel, low speed computation with a low number of processing steps. Artificial neural networks (ANNs) are an attempt to produce systems that work in a similar way to biological systems and so are well suited to this project.

Part I gives the primary background to this project: genetic algorithms (chapter 2) and artificial neural networks (chapter 3), including the type of ANNs used in this project. Parts II and III build on this, providing further background relating to GAs and ANNs respectively.

Part II discusses emergence, especially evolutionary emergence. An example of evolutionary emergence in a GA is given.

Part III discusses development and modularity in ANNs. It is explained why this is important to this work. This can be seen as the third example of taking note of what has worked in natural evolution, although this is not the only reasoning given. Consideration and coding of developmental modularity constituted approximately half of the work undertaken on this project.

By the end of part III, the reader may well have settled on the central ideas of the project as given in part IV. The words ‘calls for’ were chosen carefully in the chapter headings of part IV; they do not mean ‘needs’ but rather ‘would be very well served by’ , or ‘should urge us to use’.

One topic that will not be discussed there is ‘situation within a world’. Rod Brooks [1991a,b], one of the main figures in non-traditional AI, puts forward a strong argument that incremental development (including evolution) must take place within the world that the objects are to operate in. This is to avoid the problem traditional AI often has of there being a gap between the objects and the world. So, for example, some researchers argue that robots that are to operate in the real world must be evolved (or at least evaluated) in the real world. However, in this project the ANNs are only ever to operate in a virtual world and so there should be no concern about them being evolved in that virtual world. It is not a simulation and so suffers none of the problems that occur when trying to use a simulation to evolve robots for the real world. Where the virtual world differs from our world (however greatly), there is no unfortunate error. There is simply a difference.

The post-contemplation work begins in part V, with details of the developmental modularity system designed for this project. Part VI describes the main experiment (a non-conventional GA) itself, which uses the developmental modularity system of part V. This was designed to allow the emergence of properties that could be considered intelligent, as was the aim of the project. Conclusions are given in the final part (part VII).

I GENETIC ALGORITHMS AND ARTIFICIAL NEURAL NETWORKS

2 Genetic Algorithms

2.1 Conventional GAs

Conventional genetic algorithms (GAs) are search algorithms inspired by natural evolution. They perform (fairly) well at a wide range of difficult optimisation problems, which is currently what most GAs are used for. John Holland [1992,1993] developed the basis of the genetic algorithm (suited to evolution by both mating and mutation) in the early 1960s.

GAs evolve an initially random population of solutions (to whatever the problem is) by picking which individuals (solutions) will live on and/or mate into the next generation. To do this they evaluate the individuals' 'fitnesses' via some procedure relevant to the problem. The fitter individuals reproduce, replacing less-fit individuals which perish. Then the cycle starts over again, starting with the resulting population from the last generation. In most GAs, the population size remains constant. There are two very common operators used at reproduction: crossover and mutation.

Evolution is greatly speeded up when mating is used to combine the fit solutions, rather than just reproducing an individual from another one. Simple (single-point) crossover, whereby an individual inherits its code (chromosome) up to a random cut point from one parent and from the other parent after the cut point achieves this. This is biologically faithful as animal chromosomes cross over in this way when two gametes (sperm and egg in humans) meet to form a zygote (which forms the embryo).

During reproduction a gene may randomly change, with low probability. This is called mutation and is also a biologically faithful idea. In observed biological systems, many mutations are neutral, that is they have no (or negligible) effect on the phenotype. This is not the case in most GAs as most GAs are used for some sort of function optimisation.

Crossover and mutation are the genetic operators used in this project. Other biologically faithful genetic operators exist. For example inversion increases linkage (reduces the likelihood that crossover will separate genes that need to occur together) and adds non-destructive noise that helps crossover to escape local optima. Another example is transposition, which increases duplication. These operators are not used in this project.

There are many minor variations on the conventional GA. Many different selection methods are used, such as rank-based (top n% survive to next generation) and fitness-proportional (number of offspring is proportional to fitness). The population may be distributed, individuals reproducing with nearby individuals, their children being born nearby. Chapter 15 describes a (basically) conventional distributed GA using a non-generational selection method, which I wrote to test the developmental modularity system before writing the main experiment. Should more information on conventional GAs be required, Goldberg [1989] is a good introductory text.

2.2 Variable length genotypes (required for emergence)

Most conventional GAs set out to solve a well-defined optimisation problem. For such problems the individuals (solutions) are generally encoded on a fixed-length genotype, often directly (using a bijection between the solution set and the chromosome set). But we wish to go beyond using GAs to solve A problem, towards using them for systems that do well in general, so giving rise to intelligent behaviour. GAs with fixed-length genotypes cannot be used to evolve increasingly impressive systems as natural evolution has done, so we need to use variable-length genotypes.

The use of variable-length genotypes in GAs is not a new idea. The subject of how the lengths should change has been addressed by Inman Harvey's Species Adaptation Genetic Algorithm (SAGA) theory [1992a,b,c]. He argues (and demonstrates) that the changes in genotype length should take place much slower than crossover’s fast mixing of chromosomes. The population should be nearly converged, evolving as species; mutation rates should be low enough that they do not disperse the species (in genotype space) or hinder the assimilation, by crossover, of good mutations into the species. The idea of species is not engineered in, but rather a result of this theory. A new species comes about when a progenitorial species splits into separate ones. A species becomes extinct when all its members die.

In slightly more detail, taking into account gene duplication and genotype to phenotype systems (ontogenesis), Harvey is arguing that the complexity of the information contained within the genomes of a species should change slowly, relative to the assimilation of advantageous mutations.

2.3 Genetic Programming

Genetic Programming (GP) is the application of GAs to evolving programs. The best known advocate of GP is Koza [1990,1992]. Individuals are commonly LISP S-expressions and so the genotypes have a tree structure rather than the more normal linear string. Crossover swaps complete sub-trees, so always producing syntactically correct children. If the sub-trees swapped are of different sizes then the child will (probably) not have the same genotype size as either parent.

Rodney Brooks proposed, in [Brooks 1992], an extension of Koza’s ideas with the aim of evolving robots. However, his Behavioural Language (BL) in effect forms blueprints rather than ‘recipes’ for the robots’ networks. This is highly objectionable on grounds that will be given in part III.

Frédéric Gruau [1996] uses GP to evolve his ‘cellular programming language’ code to develop artificial neural networks. See section 7.1 for a discussion of this.

3 Artificial Neural Networks

Artificial Neural Networks (ANNs) are an attempt to produce systems that work in a similar way to biological nervous systems. Biological nervous systems are built from neurons which are very much slower than electronic components; the power of animals' brains comes from massive parallel processing. The standard introduction to ANNs is provided by Rumelhart, McClelland and the Parallel Distributed Processing Research Group at the University of California [Rumelhart et al. 1992].

3.1 Conventional ANNs

Conventional ANNs contain a number of simple processing units called nodes. Each node has an output, which may be propagated to other nodes via a weighted link, and a number of inputs from other nodes. The inputs to a node are each multiplied by the weight of the relevant link and then summed to produce the total input to the node. The node's output is then a function of this total input (including any external inputs), any memory term (such as a proportion of the previous output) and any node constants, such as threshold. For example, the following sigmoid output function is often used:

$$\text{node output} = \frac{1}{1+e^{-(\text{total node input - node threshold})}} .$$

The structure of most ANNs has traditionally been layered, with all of a node's links being to nodes in higher layers; there a no cycles in the network. Without loss of generality, we can talk of an 'input layer' and an 'output layer', with all intermediate layers being 'hidden layers'. Learning in these ANNs is most commonly via supervised 'backpropagation': to train the network, example input-output pairs are presented in turn; for each pair, the networks' outputs are calculated by forward-propagating the inputs (calculating nodes' outputs) through the hidden layers to the output layer; gradient descent is used on the length of the error vector (actual outputs - example outputs) to adjust the weights just before the output layer; then the errors are back-propagated to the layer below by summing the weighted errors from nodes-output-to and gradient descent applied to adjust the next layer of weights; this is continued until the weights just above the input nodes have been adjusted; then the process is repeated with the next training pair.

Other sorts of 'neural networks' are also common. For example radial basis function (RBF) networks are similar to the above, but use (commonly) a Gaussian output function. So each node has two variables (field centre and field width) which gradient descent is applied to as well as link weights. Self-organising maps such as Kohonen networks are often bundled under the same heading as conventional neural networks. Most of these other network classes are, however, further removed from the biological nervous system background of ANNs, and are less suitable models for work such as this project.

3.2 Recurrent ANNs

Removing the feed-forward (layered) constraint of conventional ANNs results in recurrent networks. That is networks with link-cycles in them. Recurrent networks can have internal state sustained over time and demonstrate rich intrinsic dynamics. This makes them attractive for use in adaptive behaviour work. Evidence from neuroscience is also on their side, showing that most biological neural networks are recurrent.

Whilst recurrent ANNs can be very hard to study and construct manually, artificial evolution should not have any problem using them. Indeed, there seems to be little reason to constrain the evolution to feed-forward networks, especially when aiming for autonomous agents that are to act as complex dynamical systems that work within a time frame.

3.3 The ANNs used in this project

The ANNs used in this project are recurrent networks of nodes as used successfully (hence their choice here) by Inman Harvey, Dave Cliff and Phil Husbands in their evolutionary robotics work [Cliff et al. 1992,1993,1996; Harvey et al. 1992]. They evolved recurrent networks of these nodes for visual navigation tasks in simple environments.

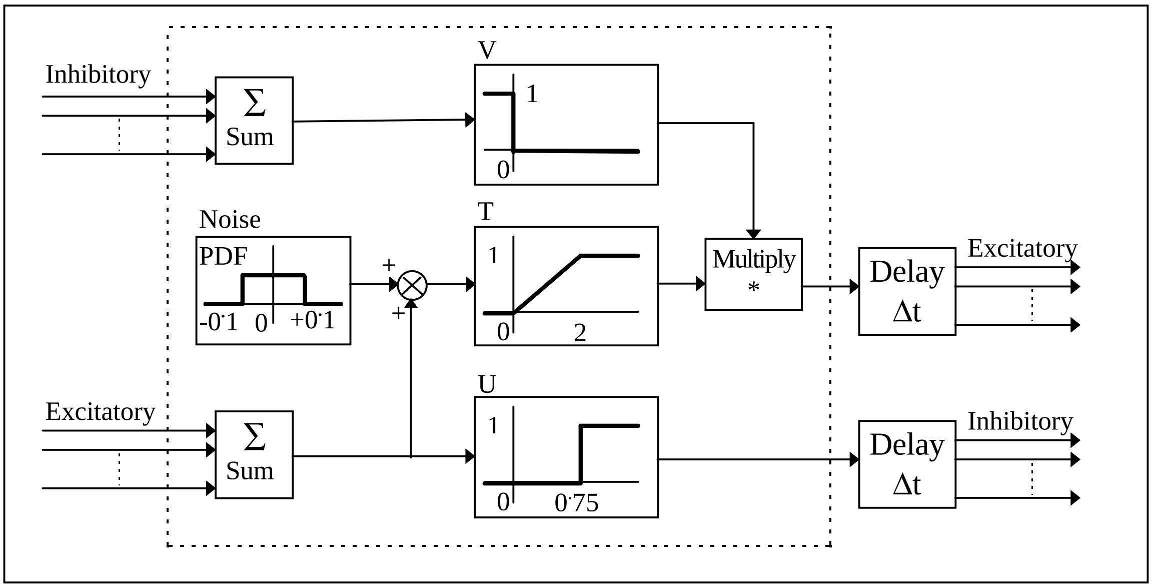

Figure 2: Schematic block diagram of the neurons used in this project: from [Cliff et al. 1993]

For details, please see [Cliff et al. 1993].

All links have weight 1 for this project; no lifetime learning was used. This was to avoid the criticism that learning was the main factor, rather than the evolution, as can be leveled at related work (see chapter 8.2). However, the importance of lifetime learning is recognised in chapter 21 (Suggestions for further work).

II EVOLUTIONARY EMERGENCE

4 Emergence

Emergence is relevant to artificial intelligence (AI) because it has become apparent, through AI work to date, that we do not understand enough about intelligence to be able to program it into a machine. Therefore AI must aim either to increase our understanding about intelligence to a level such that we can program it into a device, or to build a device which outperforms the specifications that we give it. The first approach is, presumably, the one being taken by researchers in the field of traditional AI. The second is a newer approach and the approach taken in this project.

Emergence relates to unexpected, unprogrammed, behaviours. However this is not the best way to define emergence, because it depends on the predictive ability of the observer and demands a once-only instance of emergence. Steels [1994] gives temperature and pressure as examples of emergent properties that would not be classified as emergent by such a definition. He uses emergence to refer to ongoing processes which produce results requiring vocabulary not previously involved in the description of the system's inner components. This is the meaning used throughout this project.

5 Evolutionary Emergence

Conventional genetic algorithms (GAs) use problem-specific evaluation functions. Natural evolution has no (explicit) evaluation functions. Individuals simply live until they die, reproducing, or not, during this time. As a result of organism-environment interactions, certain behaviours fare better than others. This is how individuals are `evaluated` and how the non-random cumulative selection works without any long-term goal. It is also why new abilities can emerge.

Aiming at such emergence in GAs seems to be the most promising route to building a device which outperforms the specifications that we give it (see above). I therefore see evolutionary emergence as the foremost hope for the development of artificial intelligences of the human/ ant brain variety (as opposed to the chess computer variety).

5.1 An Example: ‘Farmers and Nomads’

Sannier and Goodman [1987] used a GA to evolve genomes within an artificial world. Each individual (genome) was given the ability to detect the conditions in its internal and immediate external environments. The population exists within a two dimensional toroidal world containing ‘food’. An individual’s ‘strength’, which is taken (deducted) from its parents’ at birth, increases on consumption of food and decrease in each time step (and upon reproduction). To reproduce, an individual’s strength must be above a threshold; an individual dies if its strength drops below a lower threshold.

The genomes encode rules which allow them to move in one of eight directions (N,NE,E,SE,...) with conditional program branching based on whether or not there is food in any of the eight neighbouring locations.

In the experiment reported, food was restricted to two ‘farm’ areas, spaced apart in the toroidal world. The level of food in a farm varied periodically; when one farm was having its ‘summer’ the other would be having its ‘winter’. Also, a farm’s potential is lower the more it was either over-consumed or neglected (under-consumed) during the previous period.

Out of the evolution emerged two classes of individual: ‘farmers’ who staid put in one of the farms, their farm populations rising and falling with the ‘seasons’, and ‘nomads’ who circled the world, moving in such a way that they passed through both farms during their summers. The nomad population would increase as it went through a farm and decrease as it moved through the area without food. Groups of individuals of each class were extracted from the total population and tested in the absence of the other class. Whilst farmers could survive without nomads, it was found that nomads needed the farmers so that the farms would not be neglected between visits.

The (relevant) important thing in this work is the emergence of the two classes of individual. Never was it specified that they should come about. The evolution resulted in them simply because they do better than other solutions within the environment. The only information given, above the genetic algorithm and external environment, was the possible actions (moves) and conditional (food in neighbourhood?). It could be said that the system outperformed the specifications that were given it.

III DEVELOPMENT, MODULARITY AND FRACTALS IN ANNS

6 Why this is relevant

The brain is modular at several levels. For example the cerebral cortex contains macro modules of hundreds of minicolumns, each a module containing approximately one hundred neurons. There are known examples of neural structures that serve a purpose different from their original use, for example [Stork et al. 1991]. When one considers animals as a whole, it is clear that most properties evolved from ancestral properties which had slightly different functions. For example, if we could trace far enough back up our evolutionary tree, we would not expect ears to have suddenly appeared on an individual which was in a situation where ears would be useful. Similarly then, we can expect all (or at least most) neural structures to be descended from neural structures which once had a different use.

Evidence from gene-theory tells us that genes are like a recipe, not a blueprint. In any one cell, at any one stage of development, only a tiny proportion of the genes will be in use. Further, the effect that a gene then has depends upon what the cell can affect - that is, what the cell's neighbours are.

The above two paragraphs are related: For a type of module to be used for a novel function (and then to continue to evolve from there), and yet still perform its current one, either an extra module must have been created or there must have been one 'spare'. Either way, a duplication system is required. This could be either by gene duplication or as part of a developmental process - a recipe. Gene duplication can be rejected as a sole source of neural module duplication, because our genes do not have the capacity to store all specific connections without a modular coding [Boers and Kuiper 1992]. Therefore, we come to the conclusion that for the effective evolution of neural structures, a developmental process is required.

7 An overview of modular developmental strategies for ANNs

There are currently three main approaches to the modular development of ANNs: cellular encoding, cellular automata and Lindenmayer systems. The first two are examined and argued against in the first two sections of this chapter. The third section introduces Lindenmayer systems and argues for them in this field; the next chapter gives a more full introduction to them. The points made in chapter 6 should be kept in mind whilst reading this chapter.

7.1 Frédéric Gruau’s Cellular Encoding

Frédéric Gruau [1996] uses genetic programming (GP - see section 2.3) to evolve his ‘cellular programming language’ code to develop modular artificial neural networks. The programs used are trees of graph-rewrite rules whose main points are cell division and iteration. Like many practitioners of GP, he considers (has been heard by the author to say that) evolutionary algorithms can be used only as a tool in the design process.

The crucial thing required for this project that is missing from Gruau’s approach is exactly what is missing from GP. Modularity can only come from either gene duplication (see objections in chapter 6) or iteration. But iteration is not a powerful enough modular developmental backbone. Consider, for example, the cerebral cortex’s macro modules of hundreds of minicolumns mentioned in chapter 6. These are complicated structures that cannot be generated with a ‘repeat one hundred times: minicolumn’ style rule. There is variation between macro modules.

So in GP, we are reduced to gene duplication for all but simple iterative structures. What is required is a rule of the sort ‘follow (rules X)’ where (rules X) is a marker for (pointer to) rules encoded elsewhere on the genotype. But this would be difficult to incorporate into GP. A more sensible route would be to use a system which was designed to handle such rules. The systems in sections 7.2 and 7.3 are both such systems.

7.2 Cellular Automata

The use of conventional cellular automata (CA) for the construction of artificial neural networks can be found in many books on CA. However, the results are always poor networks. The interest here seems to be more in the area of neuron growth than the development of the network. Whilst in principle I cannot see any objection to evolving CAs for ANN development, the amount of work involved was clearly too great for this project and appeared to offer no advantage over the system chosen: Lindenmayer Systems (next section), which are related to cellular automata.

CAM-Brain [de Garis 1993] is a project to ‘implement a cellular automata based artificial brain with a billion neurons by 2001, which grows/ evolves at (nano-) electronic speeds’ (from the abstract). Whilst the resulting technology may be of use in this field, there are two main problems with the CAM-Brain project as it is, in relation to this project. The first is that not enough consideration has been given to what will be required to evolve intelligent behaviour. This single brain without any suitable way of interacting with other intelligences at (nano-) electronic speeds will be of little use for the evolution of intelligence. This could be overcome by using hundreds or thousands of these devices (possibly inside one machine). The second, more serious problem with the current approach is that it involves feeding ‘signal cells’ along ‘trails’ (contained by ‘sheath cells’). The signal cells would be determined by the genotype. This amounts to a program and, without serious thought of how to incorporate a ‘follow (rules X)’ style rule, receives the same criticism and rejection (for work such as this project) as Gruau’s Cellular Encoding.

7.3 Lindenmayer Systems

Lindenmayer systems (L-systems), which are introduced fully in the next chapter, can be seen as a relation to cellular automata, possibly a parent of (although not historically). They need not use the restriction that a cell always stays in the same position (so with the same neighbouring positions) and commonly are not restricted to a regular grid arrangement.

For this project, L-systems offer all the advantages of CAs, and more, and none of the disadvantages. Also work on the evolution of L-systems as modular development strategies for ANNs has already been undertaken with some success (see the next chapter). I felt that they were the clear choice.

8 Lindenmayer Systems

As discussed in chapter 6, gene-theory has shown us that in nature genes are used as a recipe for each cell to follow. The development of each cell is determined by the relevant genes, which are determined by the cell's immediate environment. All cells use the same set of rules, the genes. This principle is related to that of fractals [Mandelbrot 1982].

Lindenmayer Systems [Lindenmayer 1968] (L-systems) were developed to model the biological growth of plants. They are a class of fractals which apply rules ('production rules') in parallel to the cells of their subject. Thus a specified 'axiom' subject (typically one or two cells) develops by repeated re-application of these rules. The use of context-sensitive production rules allows the rule used for a step in a cell's development to be determined by the cell's immediate environment. A context-sensitive production rule commonly takes the following form:

$$L < P > R \rightarrow S$$

where	L = left-context	P = predecessor	(old cell)
	R = right-context	S = successor	(replacement cell or cells)

Figure 3: A context-sensitive L-system production rule template

The most specific production rule that matches a cell's situation is applied.

L-systems with more than two (left and right) contexts could exist but I have not as yet seen any, probably because no related application requires more than two contexts.

8.1 Kitano’s Graph-Generating Grammar for evolving ANN connectivity matrices

Kitano [1990] used a very simple form of L-system to evolve $2^k\mathrm{x}2^k$ connectivity matrices. The rules had no contexts and took the form:

$$P \rightarrow \begin{matrix}(AB)\\(CD)\end{matrix}$$

The rules were encoded in blocks of five characters (eg. PABCD) on the genotype. The number of rules on the genotype was variable. Development started with the axiom character matrix ‘S’, which was programmed to always be the first character on the genotype. After each developmental step, the matrix would have doubled in both width and height. The final rules used would have lower case characters in the successor. These would be from {a,b,c,...,p}, which were constants representing the 16 possible 4-bit binary numbers to use in the connectivity matrix.

Kitano demonstrated better results than direct encoding when evolving simple ANNs (such as XOR and simple encoders) using training by backpropagation. He also showed that the number of rules could be small.

One criticism of Kitano’s work must be that the resulting network architectures are just fully connected clumps of unconnected nodes; backpropagation is still doing most of the work.

8.2 Boers’ and Kuiper’s L-systems for evolving ANNs

Boers and Kuiper [1992] used L-systems to evolve modular feedforward network architectures that were evaluated after training by backpropagation. If two modules were set as connected, then they had links from every ‘output node’ of the first module to every ‘input node’ of the second. A module’s ‘input nodes’ are those that do not receive any input link from a node within the module. Similarly a module’s ‘output nodes’ are those that do not send any output link to a node within the module.

A fixed length alphabet was used for the rules. Letters A to H represented the possible node characters, numbers 0 to 5 the possible ‘skip’ sizes and the characters ‘[‘ and ‘]’ grouped nodes and modules into larger modules. A skip number N after a module indicates that output links are to be made to the module N modules further down the current network string. The restricted alphabet restricted the possible network architectures but still produced some good results.

The left and right contexts are lists of nodes that the predecessor should be connected to. The predecessor could contain skips and modules and had to contain at least one letter. The successor could also contain skips and modules but could be empty.

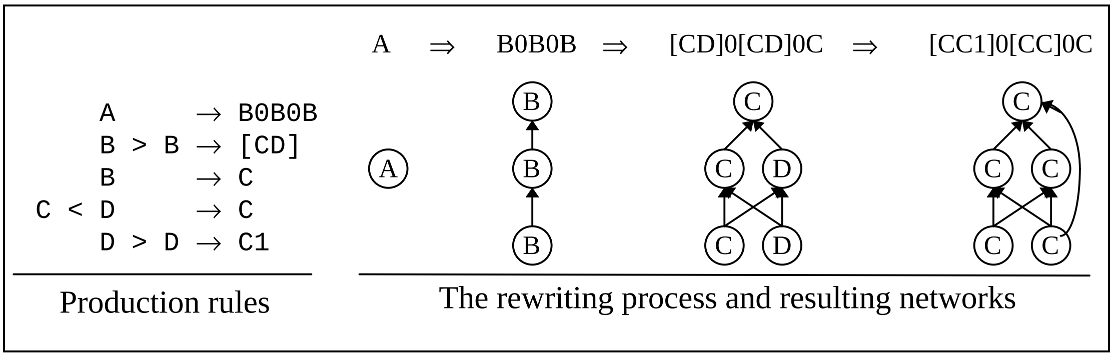
A → B0B0B
B > B → [CD]
B → C
C < D → C
D > D → C1
Production rules
A ⇒ B0B0B ⇒ [CD]0[CD]0C ⇒ [CC1]0[CC]0C
A
B
B
B
C
C
D
C
D
C
C
C
C
C
The rewriting process and resulting networks

Figure 4: An example L-system development: from [Boers et al. 1993]

The genetic encoding (or rather decoding) involved starting from each of the first five bits of the genotype and reading the string six bits at a time. A translation table (based loosely on the genetic code of RNA) was used to translate each six bits into a character from the rule-set's alphabet and "*". Each minimal substring containing five *s encodes a production rule (*L*P*R*S*); invalid production rules are thrown out. Then the decoding is repeated, using the genotype in reverse.

The evolution of production rules used a conventional GA, with fixed-length genomes initially randomised. The axiom "A" was always used. A limit of 6 rewrite passes over the network string was used (and required by most genomes). One of the main fitness functions used was testing on a simple character recognition problem: distinguishing between 3*3 pixel 'T's and 'C's rotated and embed within a 4*4 pixel array.

One criticism of Boers' and Kuiper's work must be that the resulting network architectures are just fully connected clumps of unconnected nodes; backpropagation is still doing most of the work. It is possible, from their results, that the genetic algorithm is doing little more than constructing layers of nodes with the provision for links that skip intermediate layers.

Another criticism is that when the resulting rules are run, it becomes clear that they generate many more nodes than are used. These extra nodes serve a function as modules to be skipped but it would surely be better if variations in skip size were set in the skip sizes than by inserting redundant nodes that make up most of the 'network'; that is, the (skip size) representation that was intended is not the one being used and so should either be removed or changed.

IV CENTRAL IDEAS OF THE PROJECT

This part (part IV) of the project sets out the key conclusions reached during study of the area, as set out in parts I to III, before the design of the main experiment was started.

9 Artificial Intelligence calls for Evolutionary Emergence

There are two ways in which we can proceed towards an artificial intelligence of the human/ ant (as opposed to computer chess) variety:

1) try to understand enough about intelligence to be able to program it in

2) try to build a device which outperforms the specifications we give it, that is intelligent.

The first is the approach of traditional artificial intelligence. I believe that much progress, in terms of useful software with intelligent aspects, will be made in this way. But I agree with many others in the AI community that we cannot conceivably go all the way with this approach. Therefore we are left with option 2.

Evolutionary Emergence, from genetic algorithms, seems to be the most promising route to building a device which outperforms the specifications that we give it.

10 Evolutionary Emergence calls for the dismissal of Explicit Fitness functions

Natural evolution has no (explicit) evaluation functions. Individuals simply live until they die, reproducing, or not, during this time. As a result of organism-environment interactions, including interactions between similarly-capable organisms, certain behaviours fare better than others. This is how individuals are `evaluated` and how the non-random cumulative selection works without any long-term goal. It is also why new abilities can emerge.

If we want to achieve this level of emergence in our artificial evolution, then we must stop treating evolution as a search for the optimum organism with evaluation according to some guess at what that involves.

11 Evolutionary Emergence of AI calls for Developmental Modularity

The evolutionary emergence of (better than very low-level) intelligence requires new neural structures. We can expect all (or at least most) neural structures to be descended from neural structures which once had a different (non-trivial) use.

For a neural structure to be used in a novel way (possibly after further evolution) and yet still perform its current role(s), a duplication system is required. This could be either gene duplication or part of a modular developmental process.

Gene duplication can be rejected as a sole source of neural structure duplication, because the capacity required to store all specific connections in a large network without a modular coding is infeasible.

Therefore, we come to the conclusion that for the effective evolutionary emergence of intelligence, a modular developmental process is called for.

12 Developmental Modularity calls for L-systems

A modular development strategy should be able to use a module in a large number of specific situations (rather than just in an iterative fashion).

It makes intrinsic sense that the development of part of an organism should depend partly on its local environment, including itself, and not on remote parts of the environment.

Lindenmayer systems can have both of these fundamental properties. Indeed, using the most general notion of an L-system, it is possible that all systems with these properties are examples of L-systems.

V THE LINDENMAYER SYSTEMS USED

13 Details of the Lindenmayer Systems used

The aim here was to construct L-systems suited to evolving the class of recurrent ANNs outlined in section 3.3. Firstly I shall give the system used: the axiom network was always two nodes, the first with character (label) ‘0’ having an excitatory output to the second, which had character ‘1’. The production rules take the following form:

Outline: **L < P > R → S , bits for links details**

Detail: $\mathbf{L_t,L_c < P > R_t,R_c \rightarrow S_r,S_n, \quad b_1,b_2,b_3,b_4,b_5,b_6}$

L_t,L_c = Left context = required input link: L_t = link type (0=inhibitory,1=excitatory)
L_c = initial bits of other node’s character
P = Predecessor (initial bits of node’s character)
R_t,R_c = Right context = required output link: R_t = link type (0=inhibitory,1=excitatory)
R_c = initial bits of other node’s character
S_r = Successor 1: replacement node’s character (replacement node inherits all predecessor’s links)
S_n = Successor 2: new node’s character (new node inherits all predecessor’s links, and then:)
(b_1,b_2) = reverse types [inhibitory/excitatory] of (input,output) links on ‘new’ successor [0=no,1=yes]
(b_3,b_4) = form (inhibitory,excitatory) link from ‘replacement’ successor to ‘new’ successor "
(b_5,b_6) = form (inhibitory,excitatory) link from ‘new’ successor to ‘replacement’ successor "

Figure 5: Template of the Lindenmayer system production rules used in this project

Some of the limitations of this template are intentional. I had to avoid the architectures resulting from the L-systems used by Boers and Kuiper (criticised in section 8.2 as being just fully connected clumps of unconnected nodes for backpropagation to train), which would have been complete failures considering the class of nodes being used. Therefore, successors were restricted to be two nodes (a replacement and a new node), one node (either a replacement or a new node) or no nodes (the predecessor node is destroyed) and the use of skip distances to specify new links was replaced by the links-detail bits, allowing only fairly straightforward cell (node with links) divisions.

The left and right contexts were given link types so that they could suitably handle (distinguish) inhibitory and excitatory links.

To extend the idea that the most specific rule available is used, contexts and predecessors need only match the start of a node’s character. (‘Characters’ are bit-strings of variable length.) Thus short characters in a rule allow it to match many nodes but more specific rules will be used instead when applicable. This approach is well suited to an incremental construction of production rules, such as their evolution.

Development (the running of the production rules) begins with the axiom network (above) of two nodes joined by an excitatory link. The node the link is from is marked as the network’s first input node (it has no inputs from nodes within the network) and the other node is marked as the network’s first output node (it has no outputs to nodes within the network).

Whenever a new node is created, it is set as the next network input node if it has no inputs, the next network output node if it has no outputs and neither if it has either. Once a network input or output has been allocated, it is fixed. If that node is later deleted, then a new node cannot take its place; the network input or output would simply always be 0 on that input/ output. This method of allocation of network inputs and outputs ensures that the addition or removal of a network input/ output node at a later stage of development will not wreck the numbering of previously set network inputs and outputs.

14 Genetic Encoding of the Production Rules

The genetic encoding (or rather decoding) is loosely similar to that used by Boers and Kuiper. For every bit of the genotype, an attempt is made to read a rule that starts on that bit. A valid rule is one that starts with ‘11’ and has enough bits after it to complete a rule; the number of bits varies because the node-characters can be of any length.

To read a rule, the system uses the idea of ‘segments’. A segment is a bit string with its odd-numbered bits (1st,3rd,5th,...) all 0. Thus the reading of a segment is as follows: read the current bit; if it is a 1 then stop; else read the next bit - this is the next information bit of the segment; now start over, keeping track of the information bits of the segment. Note that a segment can be empty (have 0 information bits).

The full procedure to (try to) read a rule is to read a segment for each of the left context, predecessor, right context, successor 1 (replacement node) and successor 2 (new node). Then, if possible, the six ‘links details’ bits are read. Only if all this is achieved before the end of the genotype is a rule created.

For a context, the first information bit of a segment is made the link type bit and the any remaining bits are made the node character specification.

```
GENOTYPE: 1 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0
Decoding:     +++ < > _1_ ->_1_ * _0_ * 0 1 1 1 0 0
                +++ < _1_ > _1_ ->_0_ * _1_ * 1 0 0 0 0 0

Rules:              <     >  1 ,      →     1 ,0     0 ,1 ,1  ,1 ,0 ,0
                    <  1  >  1 ,      →     0 ,1     1 ,0 ,0  ,0 ,0 ,0
(Template:   L_t,L_c <  P  >  R_t,R_c  →     S_r,S_n,  b_1,b_2,b_3,b_4,b_5,b_6)
```

Figure 6: Example of the genome decoding method used in this project

In fact, this genotype encodes a valid XOR network - see appendix 3.

15 Initial Experiment to verify the system so far

A fairly conventional distributed genetic algorithm was used to evolve these L-systems (producing ANNs as above). Fitness was calculated as -1 * (sum of errors upon testing on XOR with inputs 00,01,10,11). Thus the worst possible fitness (using the ANNs of this project) was -4, the best 0 and a network that produces no output has fitness -2.

The GA used a population of size just 64 distributed over a 3-dimensional torus of length 4; each position on the torus always being occupied. An individual is selected at random, then 2 of its neighbours. The fitter neighbour is selected as the other parent and the weaker neighbour is replaced by the child (produced using variable length crossover and mutation, described in chapter 17). Further details are not important as any conventional GA would have produced similar results.

After the genotype lengths have increased enough to form random rules (see chapter 17), an optimum network (that is rules encoding an XOR network) is soon found. See appendix 3 for the output of a sample run of this experiment. The code for the experiment is in appendix 4.

It is worth noticing that after a solution has been found, the GA goes on to find solutions with more rules, which do not decrease the fitness of the individual.

VI THE MAIN EXPERIMENT

16 Details of the Individuals being evolved

The individuals being evolved are recurrent ANNs of the type described in section 3.3. Their genotypes are decoded into L-system production rules, as described in part V.

Individuals also have their position in the (artificial) world and the direction they are facing associated with them.

The first five network outputs have pre-programmed functions associated with them:
network output 1: try to reproduce with an organism directly in front;
network output 2: try to kill an organism directly in front;
network output 3: turn anti-clockwise by ninety degrees;
network output 4: turn clockwise by ninety degrees;
network output 5: try to move one space forward.

See the next chapter for details of the functions and network input and output.

17 Outline of the Genetic Algorithm used

The main point to notice about the following description of the GA is that no (explicit) fitness function is used. The hope was to see emergent intelligent (and other) behaviours, such as individuals only trying to mate with others of the same species, only killing those of other species and acting in groups where beneficial.

The genetic algorithm (GA) operates on a population distributed over a two-dimensional torus. Population sizes used in tests were normally in the hundreds, sometimes thousands.

Initialisation

Every space in the world has an individual with genotype '0' put in it.

The Main Loop

The population is passed over in cycles. In each cycle, every individual alive at the start of the cycle will be processed once. The order in which the individuals are processed is randomised. This random cycle method prevents some of the emergent properties that were found to dominate the evolution (and so prevent more interesting emergent properties) in runs using a simpler method.

To process an individual, first its network inputs are set equal to the network outputs of the individual in front of it. This will depend on the direction the individual is facing. To be more specific: for each network input node, if the individual in front has a corresponding network output node (see morphogenesis details) then the network input value is copied from there, otherwise the network input value is set to zero.

The activities of the individual's nodes are now updated, with just one synchronous step. Thus the fastest possible response to an input would involve a link directly from the input node to an output node. The more links involved (including any recurrent loops), the more time cycles required.

Depending on the (excitatory) outputs of the ANN, the individual now carries out the appropriate pre-programmed actions:

If there is a number 1 network output node producing sufficient excitatory output,
  the individual tries to reproduce with an organism directly in front of it;
If there is a number 2 network output node producing sufficient excitatory output,
  the individual tries to kill an organism directly in front of it;
If there is a number 3 network output node producing sufficient excitatory output,
  the individual turns anti-clockwise by ninety degrees;
If there is a number 4 network output node producing sufficient excitatory output,
  the individual turns clockwise by ninety degrees;
If there is a number 5 network output node producing sufficient excitatory output,
  the individual tries to move one space forward.

Note that, for example, there need not be a number 1 network output node for there to be a number 2 - see morphogenesis section.

Any further network outputs do not produce any pre-programmed action, although they are copied to other individual's network inputs at the same time as the first five network outputs.

An individual can only reproduce with or kill an organism directly in front of it if there is one there. It can only move one space forward if there is nothing there. If reproduction occurs, then the child is born in the space beyond the individual being mated with if it is empty, otherwise the child replaces the individual being mated with.

Reproduction involves crossover and mutation followed by morphogenesis:

Crossover

The parent chromosomes (genotypes) are randomly ordered, so that either might form the beginning of the child's chromosome. A cut point is chosen somewhere along the first parent chromosome and the corresponding cut point is set on the second. There is a programmed in twenty five percent probability that the second chromosome's cut point will be offset by one gene, either upstream or downstream, from the first chromosome's cut point. Strict crossover is enforced: the first cut point must be after the first gene and the cut point for the second parent chromosome is set to just before its last gene if it was after it. This is what provides the main force for initial increase in genotype length (before any L-system rules are formed) but becomes negligible once long genotype lengths have been achieved.

The child chromosome is copied from these two sections of the parents' chromosomes.

Mutation

One gene (one bit) is picked at random on the genotype, and flipped.

Morphogenesis

When (before) an organism is born, it undergoes morphogenesis. This is currently set as four morphogenesis steps. Each morphogenesis step is a synchronous pass over the ANN's nodes, using the most specific rule from its L-systems for each node. The initial direction the organism is facing is set at random (from north, east, south, west).

Whilst I would have liked to use more morphogenesis steps, and indeed this does not seem too slow the program down much, I found it very difficult to understand what was happening in a recurrent ANN of more than 32 nodes (as often occurs with 4 steps: 2 nodes from the axiom network repeatedly dividing 4 times results in 32 nodes) as it was - see appendix 1. Similarly the only problem with using development during an individual's life (that is not all at birth) was that this also made it harder to understand what was going on.

18 Implementation

The world, a two dimensional torus, is shown in the main window as a square, along with the number of passes over the population that have been performed (the ‘time’). An individual is shown as an isosceles triangle in the world, the direction it is facing being indicated by the orientation of the triangle. See appendix 1. Most runs used a world of ‘side-length’ 20, which thus contains up to 400 individuals. Worlds with thousands of individuals were sometimes used, but these incurred correspondingly slow run times for a time-cycle, which is clearly of the order side-length squared.

In this way it is possible to view the externals of what is going on in the world. To view the internals of an individual, the user simply clicks on the individual to display it in a separate window. This has been set up to temporarily halt the main program, allowing the user to click on and view more individuals, in more windows, before the world changes. The main program continues once all the other windows have been closed.

A view of an individual includes its genotype, L-system rules and ANN, post-morphogenesis. Initially the ANN is displayed with input nodes near the bottom of the window, output nodes near the top and all others in the middle. To make the network clearer, the user can move the nodes around the window by dragging and dropping them. Picking up nodes is made easier by the use of a hidden grid, which all nodes and mouse-click-positions are snapped to. Links and labels are automatically updated as the nodes are dragged around the window.

Despite this, networks are still very difficult to understand - see appendix 1. The number of links in most interesting individuals made it hard enough to examine them on a good screen, where nodes can be repeatedly moved around, and impossible on a printout; hence only 1 is included in this dissertation.

Note: for documentation reasons, a right-click within either the world window or an individual-view window has been programmed to write a picture metafile, containing the instructions to draw the window’s contents, to the operating system’s clipboard.

19 Results

Careful and repeated examination of the world and the networks within it has revealed the following as a 'typical' run of the evolution:

[I cannot be sure that all of these happen every time but most are easy to spot and do seem to appear in at least most runs.]

1) Excitatory activity builds up in the axiom networks until it passes the decision threshold in some individuals, at which point reproduction (random) begins. This happens too fast to see at execution speed; a trace must be used to observe it.

2) Breading continues randomly until genotype lengths have increased to a length capable of containing a valid production rule.

3) Breeding is no longer totally random. Initial successful individuals include those that spin around, trying to reproduce all the time (spinning brings them into contact with more individuals) and those that spin around, killing all the time.

4) Networks that have links to the 'move forward' node appear in the population. Individuals that 'kill and move forward' all the time do well, killing off some of the spinner-killers of 3.

5) Combinations of the basic individuals appear. For example individuals that always try to move forward, turn some of the time, try to kill some of the time, trying to reproduce when they are not killing, do well.

7) Up till now, much of the activation in an individual's network has come in via its first few nodes, something common to most of the population from early on. Such individuals rely on noise when not near others, to keep their activations up in order to act. Now, however, killers are making the population fewer in number and there are many blank areas in the world. Around now, individuals with recurrent excitatory-link-loops within them appear. These loops in effect supply an activation store, allowing the individuals to be less random in their behaviour when in empty areas. I have observed this a few times.

8) Towards the end, I sometimes think (but am not sure) that some of the networks are using outputs and inputs above number 5 to recognise each other and determine their actions. However, such networks have so many links that I cannot be sure that this is really true. It certainly looks as though they can repeatedly bump into similar individuals without killing them and yet run around killing most of the other individuals in the world. Curiously, they do not seem to reproduce as much as they kill.

9) The end usually comes very rapidly, when very good killers annihilate most of the world and then themselves get killed by (I presume) a stationary rotator-killer. A few such stationary individuals remain but have no possibility of moving (because they do not have the relevant output node) and so no possibility of evolving.

VII CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

20 Conclusions

This project has been a ‘probable success’. That is, it appears that the arguments used have been proved correct by the main experiment; some of the observed behaviours could indeed be considered intelligent, if only at a very low level. However, there is too much uncertainty about the more impressive results, such as individuals using spare outputs and inputs to recognise each other. More time is required to verify the greater claims.

If nothing else, I believe that I have created a good base from where to continue, which was my personal aim on starting the project - I plan to continue with this and related work. Looking back, it is possible that I have attempted too much and so not dealt with some of the theoretical issues, such as the genetic encoding of production rules, in the depth they deserve.

21 Suggestions for Further Work

The first problem that I think should be tackled is that of how to examine the system in such a way as to be confident of what is going on within the ANNs.

Then I think that some of the theory should be ‘firmed up’. A fair amount of this work has been based on experimental results, both my own and other peoples. I believe that some of this, especially the genetic encoding of production rules, are open to a theoretical approach.

Further issues include making development take place over an individual’s lifetime and the incorporation of lifetime learning. I would also like to put the few remaining hard-specified parameters under evolutionary control. However, I see these issues as secondary.

BIBLIOGRAPHY

[Boers and Kuiper 1992] E.J.W. Boers and H. Kuiper. Biological metaphors and the design of modular artificial neural networks. Unpublished joint Master's thesis, departments of Computer Science and Experimental Psychology, Leiden University, The Netherlands.

[Boers et al. 1993] E.J.W. Boers, H. Kuiper, B.L.M.Happel and I.G.Sprinkhuizen-Kuyper. Designing modular artificial neural networks. Technical report, Leiden University.

[Boers et al. 1995] E.J.W. Boers, M.V. Borst and I.G. Sprinkhuizen-Kuyper. Evolving artificial neural networks using the "Baldwin Effect". In D.W. Pearson, N.C. Steele and R.F. Albrecht, editors, Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Alès, France, 333-336. Springer Verlag Wien New York.

[Brooks 1991a] R.A. Brooks. Intelligence without representation. Artificial Intelligence, 47: 139-159.

[Brooks 1991b] R.A. Brooks. Intelligence without reason. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91) 139-159.

[Brooks 1992] R.A. Brooks. Artificial life and real robots. In Proceedings of the First European Conference on Artificial Life. MIT Press/ Bradford Books, Cambridge, MA.

[Cariani 1991] P. Cariani. Emergence and Artificial Life. In C. G. Langton, J.D. Farmer, S. Rasmussen and C. Taylor, editors, Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, Vol. X, 775-797. Addison-Wesley.

[Clark 1994] A. Clark. Happy couplings: emergence and explanatory interlock. Department of Philosophy, Washington University in St. Louis.

[Cliff et al. 1992] D. Cliff, I. Harvey and P. Husbands. Incremental evolution of neural network architectures for adaptive behaviour. Technical report CSRP256, University of Sussex School of Cognitive and Computing Sciences.

[Cliff et al. 1993] D. Cliff, I. Harvey and P. Husbands. Explorations in evolutionary robotics. Adaptive Behaviour, 2(1): 73-110.

[Cliff et al. 1996] D. Cliff, I. Harvey and P. Husbands. Artificial evolution of visual control systems for robots. To appear in M. Srinivisan and S. Venkatesh, editors, From Living Eyes to Seeing Machines. Oxford University Press, in press 1996.

[Crutchfield 1994] J.P. Crutchfield. Is Anything Ever New? - Considering Emergence. In G. Cowan, D. Pines and D. Melzner, editors, Santa Fe Institute Studies in the Sciences of Complexity XIX. Addison-Wesley.

[Dawkins 1986] R. Dawkins. The Blind Watchmaker. Longman, Essex.

[de Garis 1993] H. de Garis. CAM-BRAIN. Report, Brain Builder Group, Evolutionary Systems Department, ATR Human Information Processing Research Laboratories, Kansai Science City, Kyoto, Japan.

[Elias 1992] J.G. Elias. Genetic Generation of Connection Patters for a Dynamic Artificial Neural Network. In L.D. Whitley and J.D. Schaffer, editors, Proceedings of COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, 38-54, IEEE Computer Society Press, Los Alamitos, CA.

[Goldberg 1989] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, Massachusetts, USA.

[Golberg et al. 1990] D.E. Goldberg, K. Deb, and B. Korb. An investigation of messy genetic algorithms. Technical report TCGA-90005, TCGA, The University of Alabama.

[Gruau 1992] F. Gruau. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In L.D. Whitley and J.D. Schaffer, editors, Proceedings of COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, 55-74, IEEE Computer Society Press, Los Alamitos, CA.

[Gruau 1996] F. Gruau. Artificial cellular development in optimization and compilation. Technical report, Psychology department, Stanford University, Palo Alto, CA.

[Harvey et al. 1992] I. Harvey, P. Husbands and D. Cliff. Issues in Evolutionary Robotics. Technical report CSRP219, School of Cognitive and Computing Sciences, University of Sussex. Also in J.A. Meyer, H. Roitblat and S. Wilson, editors, Proceedings of SAB92, the Second International Conference on Simulation of Adaptive Behaviour. MIT Press/ Bradford Books, Cambridge, MA, 1993.

[Harvey 1992a] I. Harvey. Species Adaptation Genetic Algorithms: A basis for a continuing SAGA. Technical report CSRP221, School of Cognitive and Computing Sciences, University of Sussex.

[Harvey 1992b] I. Harvey. Evolutionary Robotics and SAGA: The case for hill crawling and tournament selection. Technical report CSRP222, School of Cognitive and Computing Sciences, University of Sussex.

[Harvey 1992c] I. Harvey. The SAGA Cross: The mechanics of recombination for species with variable-length genotypes. Technical report CSRP223, School of Cognitive and Computing Sciences, University of Sussex.

[Hinton and Nowlan 1987] G.E. Hinton and S. Nowlan. How learning can guide evolution. Complex Systems, 1:495-502.

[Holland 1992] J.H. Holland. Genetic Algorithms. Scientific American, 267(1):44-50.

[Holland 1993] J.H. Holland. Adaptation in Natural and Artificial Systems. MIT Press.

[Kitano 1990] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. Complex Systems, 4, 461-476. Champaign, IL.

[Koza 1990] J.R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report STAN-CS-90-1314, Department of Computer Science, Stanford University

[Koza 1992] J.R. Koza. Genetic Programming. MIT Press/ Bradford Books, Cambridge MA.

[Lindenmayer 1968] A. Lindenmayer. Mathematical models for cellular interaction in development, parts I and II. Journal of theoretical biology, 18, 280-315.

[Mandelbrot 1982] B.B. Mandelbrot. The Fractal Geometry of Nature. Freeman, San Francisco.

[Maynard Smith1987] J. Maynard Smith. When Learning Guides Evolution. Nature, 329:761-762.

[Miller at al. 1989] G. Miller, P.M. Todd and S.U. Hegde. Designing Neural Networks using Genetic Algorithms. In J.D. Schaffer, editor, Proceedings of the third International Conference on Genetic Algorithms, 379-384, Kaufmann, San Mateo, CA, 1989.

[Mitchell 1993] M. Mitchell and S. Forrest. Genetic algorithms and artificial life. Working Paper 93-11-072, Santa Fe Institute.

[Murre 1992] J.M.J. Murre. Learning and Categorization in Modular Neural Networks. Harvester Wheatsheaf, 1992.

[Nolfi and Parisi 1995] S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks. Technical report 95-15, Department of Neural Systems and Artificial Life, Institute of Psychology, National Research Council, Rome, Italy.

[Peretto 1992] P. Peretto. An Introduction to the Modeling of Neural Networks. Cambridge University Press.

[Purves et al. 1992] D. Purves, D.R. Riddle and A-S LaMantia. Iterated patterns of brain circuitry (or how the cortex gets its spots). Trends In Neuroscience, pages 362-367, Vol. 15, No. 10, 1992.

[Rumelhart et al. 1992] D.E. Rumelhart, J.L. McClelland and the PDP Research Group. Editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol.1 (tenth printing), 194-281.

[Sannier and Goodman 1987] A.V. Sannier II and E.D. Goodman. Genetic learning procedures in distributed environments. Technical report, A.H. Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University.

[Smolensky 1992] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D.E. Rumelhart, J.L. McClelland and the PDP Research Group, editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol.1 (tenth printing), 194-281.

[Steels 1994] L. Steels. The Artificial Life roots of Artificial Intelligence. Cited in [Clark 1994]. Not read by author of this project.

[Stork et al. 1991] D.G. Stork, B. Jackson and S. Walker. ‘Non-optimality’ via pre-adaptation in simple neural systems. In C. G. Langton, J.D. Farmer, S. Rasmussen and C. Taylor, editors, Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, Vol. X, 409-429. Addison-Wesley.