

Interpreting a Genotype-Phenotype Map with Rich Representations in XMLGE

Saoirse Amarteifio

Department of Computer Science and Information Systems
University Limerick

A thesis submitted to the University of Limerick in the
College of Informatics and Electronics for the degree of Master of
Science

June 2005

*Dedicated to the memory of John O’Gorman who set me on this path and to
Michelle who walked it with me.*

Contents

Acknowledgements	9
Abstract	11
1 Introduction	13
1.1 Overview	13
1.2 Contribution	16
1.3 Thesis Layout	19
2 Grammatical Evolution	21
2.1 Grammatical Evolution	21
2.1.1 Overview	22
2.1.2 Ripple Trees and Crossover	25
2.1.3 Application and Variations	25
3 Selected Themes in Evolutionary Algorithms	29
3.1 Introduction	29
3.2 Manipulation of Representations	31
3.2.1 The Evolution of The Genetic Code	35

3.3	Model Building and Co-Evolution	37
3.4	Developmental Methods	40
3.4.1	Neutrality	40
3.4.2	Gene Expression	44
3.4.3	Artificial Embryogeny	49
3.5	Summary	55
4	XMLGE	57
4.1	Introduction	57
4.2	XML	59
4.3	XML and Evolutionary Algorithms	61
4.3.1	The GE mapping using XSLT	69
4.3.2	Evaluating Programs	74
4.4	Overview	75
4.5	Evolving a Dynamical System	79
4.5.1	Dynamical Particle System	80
4.5.2	Swarm Evolution	81
4.5.3	Experimental Setup	83
4.5.4	A Case for Remote Evaluation	88
4.5.5	Results	90
4.5.6	Discussion	91
4.6	Conclusions	92
5	Coevolving Antibodies for Selfish Genes	95
5.1	Principles	97
5.1.1	An Immune System Overview	97

5.1.2	The Emergence of Meaning	101
5.1.3	Artificial Immune Systems	103
5.1.4	Dual Processes	104
5.2	Methods	105
5.2.1	Using Grammatical Evolution for Feature Detection	106
5.2.2	Immune System Model	111
5.3	Experiments and Results	116
5.3.1	Quartic symbolic regression with small populations	123
5.4	Discussion and Conclusions	125
6	Codon Compression and Delayed Expression	129
6.1	Introduction	130
6.2	Algorithm Description	133
6.3	Experiment Results	137
6.4	Discussion	140
6.5	Chapter Conclusions	142
7	Conclusions	145
7.1	Lessons and Future Work	145
7.1.1	Extending the Dual Process Algorithm	148
7.1.2	Exploiting Compression	149
7.1.3	The Next Generation of XMLGE	149
7.1.4	Towards Efficient Dynamical Systems Evolution	149
	Publications	151
	Bibliography	152

Acknowledgements

I am grateful to internet friends and strangers for ideas, time and kindness. And more directly to those within the University of Limerick. Thank you to this extended research community which can processes ideas in astonishingly quick time. To close friends for CPU time or therapy. To my parents for the need, the will and the opportunity to learn.

To my supervisor Michael, evidently a master of bringing order out of chaos. I say this because he has allowed (my own) and fueled (with his own) chaotic, day-to-day (microscopic) ramblings and questions. However he did this while providing the (macroscopic) order parameters that reduced research entropy. Thank you for making this exercise, at the edge of chaos, coherent.

There are those that would thank God and those who would thank self-organizing dynamics for creating a complex world and for giving only sufficient understanding to ask questions.

Abstract

A novel XML implementation of Grammatical Evolution is developed. This has a number of interesting features such as the use of XSLT for genetic operators and the use of reflection to build an object tree from an XML expression tree. This framework is designed to be used for remote or local evaluation of evolved program structures and provides a number of abstraction layers for program evaluation and evolution. A dynamical swarm system is evolved as a special-case function induction problem to illustrate the application of XMLGE. Particle behaviours are evolved to optimize colony performance.

A *dual process evolutionary algorithm* based on the immune system using rich representations is developed. A dual process feature detection and feature integration model is described and the performance shown on benchmark GP problems. An adaptive feature detection method uses coevolving XPath antibodies to take selective interest in primary structures. Grammars are used to generate reciprocal binding structures (antibodies) given any primary domain grammar.

A codon compression algorithm is developed which shows performance improvements on symbolic regression and multiplexer problems. The algorithm is based on questions about the information content of a genome. This also exploits information from the rich representation of XMLGE.

Chapter 1

Introduction

As a little girl, Agnes used to go for walks with her father, and once she asked him if he believed in God. Father answered, "I believe in the Creator's computer". The answer was so peculiar to the child that she remembered it."

- Milan Kundera 'Immortality'

1.1 Overview

Interpreting a Genotype-Phenotype Map refers to the problem of encoding and extracting information from a representation. To achieve scalability in adaptive evolutionary systems, the artificial genome may not benefit from being considered as a literal encoding or intentional encoding to be mapped directly onto the phenotype. A biological system is almost entirely context-orientated. Abstraction-orientated and algorithmic-centric evolutionary computing contests biological dogma, which is data and event driven. Within this thesis, a number of rich representations are developed.

Evolutionary algorithms work by providing a structure that can be perturbed and evaluating those structures based on a fitness function. The fitness function is the extent of the environment which allows the algorithm/the structures to adapt or fit. There are exceptions where multi-objective fitness functions or coevolution are used to create an enriched environment. Yet one important aspect of biological complexity is the richness of elements which give rise to the enabling physics and chemistry exploited by natural organizations.

In nature, a complex system's environment tends to be more complex than the system itself. This allows for the system to make redundant observations in its environment and utilize signal chords rather than discrete events in adapting - this is a pervasive and important property.

The evolutionary mechanism in principle is ideal for exploiting events and inducing functional models when not operating in a vacuum. One finds that this issue is largely overlooked as one sees a need to compensate by explicitly considering rich representations. In consequence, an XML implementation of Grammatical Evolution is developed as a framework for investigating rich encodings. Information inherent in Grammatical Evolution mappings is retained as we consider exploiting it for search space modeling.

Grammatical evolution, on which the current work is based, exploits an explicit genotype-phenotype map which is formalized by a grammar. This mapping occupies an important position in evolutionary computing. Genotypes and phenotypes are independent structures which have different requirements so to speak. Some representations such as tree-based GP couple the survival of both phenotype and genotype by applying operators directly to the phenotype. It seems reasonable to decouple

these artifacts to allow for efficient perturbation and reuse of genotype structures. This mapping is our point of focus as we use rich representations to track and exploit information from this event.

Complex systems in nature have a number of invariant principles such as consistent embodiment and structural coupling with their environment. Adaptive systems have evolved to fit their environment and predict events. Considering adaptive systems as cognitive and perceptual, we consider how an evolutionary algorithm might employ features of perceptual systems to be more adaptive. In chapter 5, an algorithm is developed based on such questions. In the evolutionary computation literature, problems of perception are addressed predominately from the model building and modularization perspectives. As such, we construct a model builder and feature detection method based on the immune system, just one example of a cognitive, perceptual system. As the immune system 'perceives' via the mass effect of individual antibody-antigen binding events, so to might an evolutionary algorithm 'perceive' by evolving a library of feature detectors. This 'perceptual system' *interprets* and *generates* genomes. Thus, this first contribution to the *interpretation of a genotype-phenotype map* considers the genome to be a rich environment in which a second coevolving gene interpreter can detect regularities. Meaning is said to evolve as the genome is both the input and the output of this adaptive perceptual system.

The genome coevolves with an interpreter and implies rather than stores information in this context. It contains blocks of information which suggest 'variations' - these variations are blocks that may or may not be accommodated. Blocks are considered non-functional and have low information content. If they are accommodated, the blocks are assimilated and the genome evolves. This principle is reminiscent of William James' 'Great Man'.

In 1902 William James introduced his Great Man. This Great Man is an anomaly to his contemporaries. His ideas in retrospect can be considered beneficial. However society must accept him; if it does, both society and he benefit. If not, then presumably he was never great. William James was interested in scales of selection and similarities between perceptual systems at different scales. Evolution might be seen to be doing what we can appreciate more clearly at the scale of the Great Man.

An important issue when interpreting the artificial genome is that of complexity; what information is stored or compressed in the genome? As we consider the issue of complexity, we see it that useful variants survive and are somehow compressed in the genome. In chapter 6 we consider a compression algorithm that stresses the agreement of alleles so as to 'interpret' what information is in a genome. Very often allele information is lost due to position dependence and loss of contexts, while this same 'loss' allows exploration. The proposed algorithm may be one of the closest algorithms in the EA literature to evolve solutions without exploring variation via wild 'block shifting and repositioning', although it does use a random read cursor. Evolution seems to proceed through *codon compression*, which is the crystallization of the exploration *delayed expression* allows. Position is preserved while variation is made possible using delayed expression.

1.2 Contribution

The thesis considers a number of information processing perspectives in evolutionary algorithms using a novel XML implementation of Grammatical Evolution. Two

new algorithms are presented (i) a dual process evolutionary algorithm and (ii) a compression algorithm. Contributions are summarized below;

- The rich representation approach considers tagging information and exploiting the genotype-phenotype map. This allows visualization of the relationship between genotype and phenotype. This information can also be used to improve search space modeling. This can be seen in both high level algorithms mentioned above and enables many of the other contributions discussed.
- An XML based implementation of Grammatical Evolution is developed which uses XSLT in a novel way to implement genetic operators. A pattern for using reflection is introduced for instantiating object trees from XML trees. This adds another degree of program language independence to Grammatical Evolution.
- The XML framework can be easily distributed, allowing arbitrary 'smart clients' to participate in performing experiments and perform new roles at runtime. This has been implemented using XML Web Services.
- A method for generating XPath-based antibodies from grammars is developed. A mapping grammar allows any grammar to generate a complementary grammar in a domain-neutral manner. XPath queries are regular expressions which have a selective interest in primary structures. This *lock and key* relationship between an XPath query and a reciprocal XML structure is an interesting analogy to many biological mechanisms and may be conceived as a promising method for bootstrapping 'perception' and meaning. This application of grammars allows for the specification of reciprocal structures at a high level of representation with implied semantics.

- An adaptive feature detection mechanism uses associative XPath templates to select degenerate codon sequences with above average fitness. This aims to overcome some of the limitations and drawbacks of module encapsulation. The GE mapping is exploited in order to detect features at the phenotype yet store them as degenerate sequences that have no explicit context.
- A dual process model considers evolution to be a perceptual system which has (i) selective interest (feature detection) and (ii) integration capabilities. The dual process algorithm is an implementation of an artificial immune systems (AIS).
- A variable length 'chaining operator' allows fragments to be stacked by fitness in the genome. This is an effective part of the dual process algorithm and appears to make important contributions to algorithm performance.
- A compression algorithm demonstrates how rich representations can be used to significantly improve performance on symbolic regression and make small improvement on multiplexer. Perhaps more importantly, the algorithm provides a vehicle for exploring how evolution reduces uncertainty of its environments and as such increases the complexity of the genome. We outline trends in compression and divergence during a number of tests.
- An approach to evolving dynamical systems on remote machines using the XMLGE framework is explained. A novel method for evolving synergy in multi-particle systems uses a global entropy measure to optimize swarm behaviour.

1.3 Thesis Layout

In the next chapter, Grammatical Evolution is presented and we discuss some of its key features and applications. The XML implementation of Grammatical Evolution developed for the current study is described later in chapter 4.

In chapter 3, we discuss a number of themes in evolutionary computing. We consider how the genome has been perceived since the early days of the genetic algorithm.

In chapters 5 and 6 the dual process algorithm and compression algorithms are described.

Chapter 7 suggests conclusions and future work.

Chapter 2

Grammatical Evolution

The word "computer" was peculiar and so was the word "Creator", for father would never say "God" but always "Creator" as if he wanted to limit God's significance to his engineering activity... So she asked father if he ever prayed. He said "That would be like praying to edison when the light bulb burns out."

- Milan Kundera 'Immortality'

2.1 Grammatical Evolution

Grammatical Evolution (GE) [O'Neill and Ryan, 2003] is an evolutionary automatic programming mechanism that can evolve programs in arbitrary languages. It is of the Genetic Programming (GP) family; the general application of evolutionary computing to automatic program generation. The following sections provide an overview of GE while looking at a number of specific GE characteristics and some of its applications.

2.1.1 Overview

GE is characterized as a GP derivative by its use of a linear genome which is read sequentially and mapped to a phenotype structure using a Backus Naur Form (BNF) grammar to dictate legal phenotypes. The linear genome can be perturbed in a number of ways typically using the simple genetic algorithm as a search mechanism. Particle Swarm Optimization [Kennedy and Eberhart, 2001] has also been used with GE [O'Neill and Brabazon, 2004].

The mapping is, in the abstract, analogous to the biological DNA-RNA-Protein mapping. The genome can be represented as binary [O'Neill and Ryan, 2003], integer, or real valued numbers [O'Neill and Brabazon, 2004]. If the genotype is a binary string, a *transcription* phase maps 8 bit codons to codon integer values. Following this, is the *translation* phase; as the genome is read, its codon values are used in a function to select a production rule from the BNF grammar. The BNF grammar, an example of which is shown below, defines a set of terminal and non-terminal symbols and their corresponding productions.

Typically, the function used to select productions is *the codon value modulus the number of grammar choices*, which yields a legal production. This production is 'expressed' in the phenotype thus changing the translation context i.e. the grammar context. Continuing the mapping, subsequent codons are used in the appropriate grammar context. The variable length genome can be wrapped up to a specified threshold allowing the genome to be reused a number of times. Consider the following example. A possible symbolic regression BNF grammar is shown below. Three symbols and their production choices are shown; <expression>, <operator> and <variable>.

```
<expr> ::= <expr><op><expr>
          | <var>
<op>   ::= + | - | * | /
<var>  ::= x | y | z | 1.0 | 2.0
```

A codon sequence will be read sequentially in the grammar context in order to develop the phenotype. 1) Starting at the grammar's start symbol `<expr>`, the first codon value modulus the number of production choices (i.e. 2) will give the production choice between either `<expr><op><expr>` or `<var>`. 2) If `<var>` was chosen, the mapping would continue to read the next codon in order to choose a terminal value and then stop. If on the other hand `<expr><op><expr>` was chosen, the mapping would continue to expand the left most `<expr>` as in step 1. In the event that there are no more codons (and the wrapping threshold as been reached) for either the expansion of non terminals or writing of terminals, the individual can be considered invalid and given the lowest possible score. Alternatively a repair strategy can be adopted to maintain valid genomes [Hemberg and O'Reilly, 2002].

Studies evaluating wrapping show that the wrapping operator is more important to Santa Fe Ant than for symbolic regression. However in both cases, the number of individuals being wrapped decreases over a run [O'Neill and Ryan, 2003]. This may suggest that the wrapping operators is something like a fallback option used while genetic material is being condensed. It would seem preferable not to add dependencies to the same codons.

The mapping has a number of interesting consequences. The first is with respect to neutrality which is discussed in more detail in the next chapter. Briefly, it is the observation [Kimura, 1983] that many genotypes map to the same phenotypes and that many phenotypes differ by single point mutations. Neutrality is not simply

interesting as an analogy but as a possible means to add robustness to evolutionary algorithms. While it may not be accurate to discuss the GE mapping as neutral at one scale, it is true to say that the genome values at bit levels and codon mapping levels may be mapped or mutated in a degenerate fashion. In biology, many codon values map to the same amino acids. For example, mutating the last position in a codon triplet does not always code a different amino acid*. Studies in GE shown that degeneracy in the genetic code, mapping between 4,6,8,12 and 16 bit codes and codon values has positive effect on the maintenance of diversity and success rates [O'Neill and Ryan, 2003].

The degeneracy and *intrinsic polymorphism* in the GE mapping allow not only for neutrality but a degree of scalability and material reuse. Overlapping genes observed in nature [Lewin, 1999] demonstrate how the same genetic material can be used a number of times to code different functions. This is also exploited by GE for example in its use of wrapping and also implicitly in separate mapping contexts i.e. the same sequence in a different individual, in a different context. In GE, a sequence of codons is read and used in terms of a 'changing' grammar context, where the context changes as the codons are applied to choose productions. In theory, special codon sequences could code completely different phenotype structures in different parts of a program. Thus there is degeneracy and the possibility of reuse at a sequence level as well as the code level. This sequence level property is referred to as *intrinsic polymorphism* and the downstream changes are said to *ripple*. It is interesting to ask how much redundancy can be packed into sequences in practice and how this is effected by grammar complexity? Degenerate sequences may well promote diversity. A mutation in an upstream codon, modifies the significance of all downstream codons.

*Crick's wobble hypothesis

This means that the significance of genomes can change radically. The flip side is that the explicit information about a feature can be easily lost. However it seems this type of 'information' loss may not be as detrimental as it might first seem. In biology and a number of artificial abstractions including GE and floating representations [Wu and Lindsay, 1996], genetic material can be used through alternate reading frames. In [Wu and Lindsay, 1996], this is considered at a 'building block' level while in GE, implicit features or sequences are mapped in given grammar contexts.

2.1.2 Ripple Trees and Crossover

GE genotype-phenotype mapping can be visualized as mapping a genome onto a derivation tree working on the left most non-terminal; constructing a tree in pre-order. As such the earlier part of the genome creates what is referred to as the *spine* of a tree which is a depth first segment. A crossover event will exchange the reciprocal of this, leaving the spine with vacant leaf nodes called *ripple sites*. A donated sequence from a crossover event will be used to 'rebuild' the tree along ripple sites in pre-order. This behaviour is called *ripple crossover* [Keijzer et al., 2001]. For example see the derivation tree in Figure 2.1, which has been constructed in pre-order. The line drawn through the tree indicates where the ripple sites lie.

2.1.3 Application and Variations

We finally look briefly at recent variants of Grammatical Evolution such as Meta-Grammar GA [O'Neill and Brabazon, 2005], π GE [O'Neill et al., 2004], (GE)² [O'Neill and Ryan, 2004]. Other variants such as the position-independent GAUGE and Chorus are discussed in [O'Neill and Ryan, 2003] while Grammatical Swarm

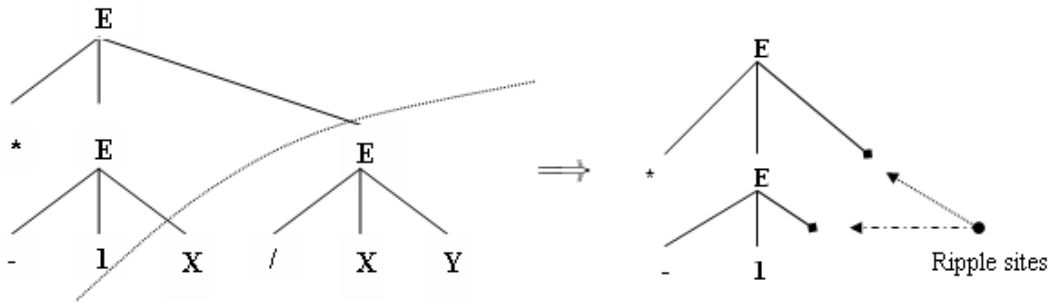


Figure 2.1: Derivation tree and separation at ripple crossover. During crossover, ripple sites are exposed and the donor genome segment rebuilds the tree along these sites.

which uses particle swarm optimization as the *evolutionary engine* is described in [O’Neill and Brabazon, 2004].

π GE is a position independent variation of Grammatical Evolution similar to the GAUGE and CHORUS systems discussed in [O’Neill and Ryan, 2003]. In this case, the order of derivation sequence steps applied to non-terminals is not predefined as in the standard GE ordering from left to right. Instead, the mod function, already familiar to GE, is used for a second time to choose which non-terminal will be developed. In other words, a symbol production is chosen using the mod function based on the number of production choices. This may yield a production with a number of non-terminals such as `<expr><op><expr>` from the grammar above. In GE, these would have been developed from left to right. However in π GE a second application of the mod rule will determine which non-terminals to develop first. This is said to provide a degree of position independence which shows positive performance improvements over GE on a number of benchmark problems. Position independent representations are discussed further in the following chapter.

Grammatical Evolution by Grammatical Evolution (GE)² [O'Neill and Ryan, 2004] is in principle a method to evolve the genetic code similar to previous work on the subject [Keller and Banzhaf, 1999]. This is discussed in the next chapter. In practice (GE)² coevolves the grammar that is used in constructing primary structures. The primary grammar is referred to as the *solution grammar* while the grammar grammar is a meta-grammar. The meta-grammar is considered a *universal grammar*, a term adopted from linguistics. The solution grammar is *dependent on* the universal grammar. By adapting the genetic code in this way i.e. the means by which a primary genome is mapped via an evolved solution grammar, the (GE)² system can be seen to be adaptive and suited to dynamic problems. The evolved grammars can be related to dynamically defined functions as the grammars can store fit *composite* productions. This is distinct from the most basic grammars which must be elemental to allow all expression types. (GE)² might well be considered a *dual process selection algorithm* like the algorithm developed in chapter 5.

Following (GE)², the meta-Grammar Genetic Algorithm builds on this use of grammars in an application to modularity and reuse and shows clear performance advantages on both static and dynamic problems [O'Neill and Brabazon, 2005]. The cited study describes a number of interesting grammar complexity developments. The starting point is a grammar (GE) which specifies a fixed length 8 bit sequence production where each bit can take a binary value. The next level describes a higher order grammar (GE+BB) where all blocks of multiples of two, which can make up for example the fixed length 8 bit sequence, are given. This aims to exploit and reuse useful bit combinations. Further developments allow for the use of meta-grammar to allow for the evolution of grammars as introduced in (GE)² [O'Neill and Ryan, 2004] and exploiting multiple binary sequence modules of different sizes.

Chapter 3

Selected Themes in Evolutionary Algorithms

Agnes Thought to herself: the Creator loaded a detailed program into the computer and went away... In his place, there is a program that is ceaselessly running in his absence, without anyone being able to change anything whatever.

- Milan Kundera 'Immortality'

3.1 Introduction

This chapter reviews evolutionary algorithm methodologies that aim to overcome the scalability problems in the basic evolutionary algorithm abstraction. While the review is not exhaustive, it provides an overview of a number of relevant issues.

The simple genetic algorithm [Holland, 1992, Goldberg, 1989, De Jong, 1975] in theory works by combining partial solutions into solutions of higher order. However,

using simple mappings and simple operators it performs poorly on difficult problems, problems where there is interaction (crosstalk) between variables or features [Goldberg et al., 1993b]. In such instances, it is difficult to evaluate mutually exclusive features within reasonable time. Consequently these algorithms will converge to a local minimum when applied to difficult problems. A number of methods that have tried to resolve this issue are reviewed.

1) Early methods manipulated the representation so as to reduce the likelihood that building blocks, once found, would be disrupted [D. E. Goldberg and Korb, 1989, Harik, 1997, Wu and Lindsay, 1996].

2) Later methods attached additional processes to the evolutionary algorithm to model the search space. These attempted to increase the likelihood that the algorithm would maintain and successfully recombine features. These include probabilistic modeling techniques [Pelikan et al., 1999, Pelikan and Mühlenbein, 1999, Baluja, 1994], [Pelikan and Mühlenbein, 1998, Baluja and Caruana, 1995] and explicit feature detection [Wu and Stringer, 2004, Garibay et al., 2003, De Jong and Oates, 2002].

3) Artificial Embryogenies begin to consider the third scale of evolutionary dynamics discussed in the next chapter. They are referred to as *developmental* in that phenotype development has an explicit ontogenetic or embryonic phase [Banzhaf et al., 2003, Bentley and Kumar, 1999, Bongard and Pfeifer, 2001],[Federici and Roggen, 2004]. Early so-called developmental methods [Banzhaf, 1994] exploited explicit genotype phenotype mappings as is also the case with GE [O'Neill and Ryan, 2003]. We refer to this methodology simply as *gene-expression based*. Lately gene expression based methods have begun to consider alternative mechanisms of gene expression [Ryan et al., 2002, Wu and Garibay, 2002, Lones, 2004]. As gene expression based methods become combinatorial [Kuo et al., 2004] the differences between gene ex-

pression and embryogenesis become blurred in practice. We look briefly at generative encodings. For an early example, Kitano [Kitano, 1990] 'grew' neural networks rather than allow a genome to encode its architecture. As such the genome does not fully encode the phenotype but describes how it develops.

Each of these three developments tell us something about the artificial genome. First, attempts were made to protect its integrity. Then, it was used to encode an external model as opposed to being used in the simple and untamed recombination-based method of the simple GA. Finally a developmental approach put emphasis on the development itself rather than allowing the genome to entirely specify the phenotype. Some methods which could be called generative, use the genome to guide development transitions.

3.2 Manipulation of Representations

There have been efforts to both engineer good representations and efforts to evolve good representations. These try to respect linkage or learn linkages, allowing for building blocks to be preserved with higher probability.

One of the first of these approaches categorized among the competent GAs as 'perturbation based'* such as the messy GA [D. E. Goldberg and Korb, 1989] or later the faster variant [Goldberg et al., 1993a]. The messy GA is an iterative algorithm where each iteration consists of a primordial phase: where building block identification takes place; and a juxtapositional phase: where building blocks are assembled into complete solutions. During each iteration, the algorithm attempts to identify and assemble higher level building blocks. Candidate solutions are represented by variable

*While these early perturbation based algorithms were among the first to be called competent, later competent GAs used other methods such as probabilistic model building.

length chromosomes consisting of <locus, allele> pairs which allows a non-fixed i.e. *floating* representation of alleles. Variable length genomes can be over or under specified; where a locus is associated with more than one allele, the one which is defined nearest the left of the chromosome will be used during the solution's evaluation. If a solution has no alleles defined for some of its gene loci, values are taken from a template chromosome. The best solution found within the juxtapositional phase of a particular iteration is used as the template chromosome for the next iteration. The aim of the algorithm is for the template chromosome to contain optimal building blocks up to the current level and form the basis for the discovery of higher level building blocks in subsequent iterations. The messy GA offers superior performance to simple GAs on problems with high levels of epistasis. It has been suggested that this is largely due to position independence [D. E. Goldberg and Korb, 1989]. However Pollack and Watson suggest that the messyGA has many unique features and it is difficult to suggest the contribution of these. They argue that the feature that most distinguished it is *partial commitment* where individuals commit to specifying only a subset of the entire gene set. The Incremental Commitment GA is a similar algorithm that does not use floating representations and they report good performance on hierarchically consistent building block problems with high epistasis and random genetic linkage [Watson and Pollack, 1999].

The Linkage Learning Genetic Algorithm [Harik, 1997, Harik and Goldberg, 1997], adopts the floating representation of the messy GA. Chromosomes are arranged as a ring. Both the position at which interpretation starts and the ordering of the alleles determines how the chromosome is interpreted. Different starting locations can lead to a chromosome being interpreted in different ways. Non-coding regions may appear on chromosomes which are not evaluated if genes correspond to the same al-

lele. Because reading can begin at any point, the expression of alleles is probabilistic [Harik and Goldberg, 1997]. Harik uses the term extended probabilistic expression (EPE-n) to suggest the number of similar alleles that can be represented on the same chromosome. Crossover evolves copying a contiguous sequence of `<locus,allele>` pairs from a donor chromosome and then grafting it into a recipient chromosome, removing existing copies of the newly grafted `<locus,allele>` pairs from the recipient chromosome thus changing both its interpretation and the linkage between its gene loci. A second child is produced as the donor and recipient roles are swapped. When selection rate is appropriate, crossover brings genes that constitute building blocks closer together as linkages are learned. Harik observes the *mixing problem* [Goldberg et al., 1993b], that a linkage learning mechanism must evolve linkage faster than the population converges. Storing and probabilistically selecting from multiple alleles wins this race against selection [Harik, 1997]

Annie Wu and colleagues studied the significance of non-coding DNA and floating representations in genetic algorithms [Lindsay and Wu, 1996, Wu and Lindsay, 1996, Wu and De Jong, 1999]. The use of floating representations is considered at the level of building blocks. Within this floating representation, there are bits that identify building blocks and those that encode the value of the block. For example, 8 bits might be used, the first four might identify the block and the remaining four bits the value. This leads to degenerate coding mechanisms inspired by overlapping genes theory which recognized that there may be insufficient genetic material in a genome to map a phenotype therefore genetic material is reused. This type of reuse is exploited by GE as discussed in the previous section. Similarly, in the floating representation, over-specification, under-specification and overlapping can occur due to existence of block defining codes. Studies comparing fixed and

non-fixed representations showed interesting results for abstract problems and on symbolic regression. The average population fitness increase faster and levels off at a higher value using floating representations and diversity was higher. Interestingly while 60% of building blocks survived from one generation to the next, 97% of those that did not survive, did not survive due to not being selected - while only 3% did not survive due to disruption [Wu and De Jong, 1999]. This strong link between selection and building blocks survival leads the authors to suggest that operators play only a small role in building block disruption. Thus it is possible to safely increase construction of building blocks by increasing operator's activity and absorb the disruptive effects through appropriate tuning of selection pressure [Wu and Lindsay, 1996, Lindsay and Wu, 1996]. Further work strained the analysis by reducing the genetic material available to the representations, including negative blocks and randomizing non-coding segments. Results showed that degeneracy could be exploited in restricted genotypes. Diversity and robustness to change could be achieved through degeneracy as blocks were preserved, presumably dissociated from selective pressure. In particular randomising non-code segments (which may arise due to broken blocks), which could be detrimental due to loss of memory and reduce chance of block rediscovery was in fact advantageous as exploration outweighed the potential disadvantage [Wu and Lindsay, 1996, Lindsay and Wu, 1996]. There was an increase in the recombination of building blocks. Such work suggests that populations should maintain as many useful blocks as possible (even if they are not always expressed) and aim to express them probabilistically (perhaps optimally).

3.2.1 The Evolution of The Genetic Code

During translation, codons from mRNA are translated into amino acids which are linked to a growing protein. tRNA (transfer RNA) assists this process by associating an amino acid to a codon(s). 1) Each tRNA molecule has a three base anticodon that binds to the matching mRNA codon using complementary matching. 2) It also carries an amino acid identifier. A tRNA molecule is joined to an amino acid to form aminoacyl-tRNA. This joining is managed by an enzyme simply called aminoacyl-tRNA synthetase (ATS), which recognizes the amino acid identifier and joins an appropriate amino acid. 2) Ribosomes move along the mRNA molecule and allow tRNA molecules to match each mRNA codon and add the associated amino acid (from aminoacyl-tRNA complex) to a growing chain. The natural genetic code has evolved as part of the fit organism. This important process is often overlooked in evolutionary algorithms..

Fundamentally, a genetic code determines the meaning of coding DNA and the fitness landscape. In this sense, previous work on floating representations by Wu et al [Wu and Lindsay, 1996, Lindsay and Wu, 1996] as discussed above might be considered as an approach to evolving the genetic code in that the meaning of genetic material is open to evolution. Other research considered appropriate codings to present fitness landscape that favoured navigation. One example of this is gray encoding [Caruana and Schaffer, 1998]

Attempts to evolve the genetic code are based on gene expression based methods, the first of which was [Banzhaf, 1994]. More recent work such as the grammar based (GE)² [O'Neill and Ryan, 2004] and Chemical Genetic Algorithm (CGA) [Suzuki and Sawai, 2003] also use explicit genotype-phenotype mappings and address

the evolution of genetic codes. Theoretical studies model how a primitive genetic code may have originated [Bedian, 2001, Weberndorfer et al., 2003].

In [Banzhaf, 1994] a table of mappings from codon values to phenotype values represent the genetic code. The system evolves to build redundancy around useful phenotype mappings and discards less useful phenotype mappings. Similar work has been carried out with grammars [O'Neill and Ryan, 2004], supporting the earlier findings of [Banzhaf, 1994] and showing promising results in dynamic environments.

Suzuki and Sawai's Chemical Genetic Algorithm (CGA) [Suzuki and Sawai, 2003] which was later extended to Chemical Genetic Programming [Piaseczny et al., 2004] also considers this issue. Once again, the goal is to optimize the genotype phenotype mapping as it can not be assumed that a human-chosen hard-coded method is efficient. In the CGA [Suzuki and Sawai, 2003] both codes in DNA and code translations coevolve. The algorithm imitates a biological cell that includes a DNA string, a set of aminoacyl-tRNAs, a set of tRNAs and a set of indexed amino acids. Artificial embryogenies, discussed below, also exploit enriched cell models. As DNA is always exploited in the context of a cell, genetic information on DNA has no meaning without the construction of such intracellular units [Bedian, 2001]. In CGA, molecular units involved in translation are exchanged between cells in addition to DNA exchange. In this way, the appropriate mapping evolves to an optimum [Suzuki and Sawai, 2003]. Suzuki and Sawai report superior performance to the simple GA on deceptive problems.

3.3 Model Building and Co-Evolution

Feature detection is any bias in an interpretation system which leads to the 'recognition' of fundamental units. We review past work that has considered model building, feature preservation and co-evolution.

Wu and Stringer [Wu and Stringer, 2002, Wu and Stringer, 2004] use a chunking operator to identify and preserve building blocks or useful rules. Important rules are identified and used to update a communal memory. Individuals in the population benefit from all or portions of the shared rules to augment their own rules i.e. an individual evolved rules and rule 'pointers'. Chunking refers to creation of single units or 'chunks' of information out of multiple smaller related bits of information. This is based on human tendencies to process information in this way, significantly increasing the amount of information they can manage. Their GA also works on variable length genomes [Wu and Stringer, 2004].

[Wu and Stringer, 2002] try to find a way for GAs to modify and form chunks of epistatic building blocks and applied the method to the control of simulated Micro Air Vehicles (MAVs). Base (primordial) chromosomes are evolved while periodically the evolutionary process is halted for chunking to create a shared memory of memory chromosomes. Each individual has a base chromosome and memory chromosome. These are augmented prior to fitness evaluation. A memory mechanism identifies, preserves and allows choice (selective access) of chunks. Chunk elements are updatable and replaceable (when better found). Gene salience is measured as gene frequency in accordance with [Goldberg et al., 1993b]. They updated the memory periodically and found that every 10 generations showed best results.

Experiments on MAXSUM and MAV experiments showed the GAs ability to exploit good rule sets and avoid bad rule sets. They also found that MAV rules evolved were both fit and compact, as the chunking GA was able to reduce the size of base chromosomes over time. They refer to this effect as *winnowing* [Wu and Stringer, 2004]. GA chunking performed much better than standard GA in terms of average fitness and equivalent in terms of best fitness. They propose separate memories for decomposable problems.

In justifying the chunk library, Wu and Stringer allude to the fact that memory is important to dynamic environments where a simple evolutionary algorithm may normally lose possibly useful features that are temporarily unfit [Wu and Stringer, 2004, Wu and Stringer, 2002].

With regard to this point, another very interesting study was carried out in [Hasegawa and Iba, 2004]. Like our model described in chapter 5, this GP model also exploits an immune system-like library of 'antibodies'. It is widely appreciated within the artificial immune systems (AIS) community that an important feature of AIS is the maintenance of diversity to improve performance in dynamic environments. [Hasegawa and Iba, 2004] demonstrate how this feature can be used in multimodal problems. They note that in traditional GP it is almost impossible to evolve two solutions simultaneously while in the Multimodal Search Genetic Programming mechanism they achieved success 6 out of 10 times [Hasegawa and Iba, 2004].

Algorithms focusing on modularity report scalability for classes of problems with regularities [O'Neill and Brabazon, 2005, De Jong and Oates, 2002, De Jong, 2003, Garibay et al., 2003, Watson and Pollock, 2003] and describe under what circumstances modularity may or may not be beneficial [Garibay et al., 2004]. Related to this, algorithms using niching or explicit recognition of hierarchy consider the de-

construction and reconstruction of problems, see for example [De Jong et al., 2004, De Jong et al., 2005].

One approach to modularization has been to explicitly encapsulate and reuse modules for example ADFs [Koza, 1994]. This allows algorithms to start matching in terms of combinations of properly encapsulated modules, boundaries of which are respected during crossover. However, [De Jong, 2003] observes the lack of theory on guiding development and evaluation of modules and suggests the use of *Pareto-coevolution*. Using this method, a population of modules and a second population of assemblies or solutions are coevolved. Evolutionary Multi-Objective Optimization [Schaffer, 1985] is used to evaluate modules against multiple criteria. Satisfying multiple objectives can achieve a tighter modeling in determining when it is suitable to encapsulate a module [De Jong and Oates, 2002, Garibay et al., 2004]. Pareto-Coevolution is capable of exploiting structure in certain large search problems by gradually consolidating learned information. While modules are fixed templates, in principle any type of detectable pattern can be considered. This suggests that a challenge for research is developing means to identify patterns [De Jong, 2003]. [Garibay et al., 2003] also recognize both issues of modular exploitation and automatic discovery yet only address the former.

Perhaps the most successful model building approaches are those based on probabilistic model building [Pelikan and Mühlenbein, 1998, Pelikan and Mühlenbein, 1999], [Pelikan et al., 1999, Goldberg, 2002, Pelikan, 2005] most notably Hierarchical Bayesian Optimization Algorithm (HBOA) [Pelikan, 2005], which is a hybrid of hierarchical composition (niching) and multivariate probabilistic modeling originally used in BOA [Pelikan et al., 1999]. Lately, probabilistic model building has been adopted in GP [Sastry and Goldberg, 2003] and [Shan et al., 2004], where [Shan et al., 2004] uses a

stochastic context-free grammar. This is used as an online model of the problem and is sampled to generate a new generation.

Probabilistic model building algorithms are classified as *competent genetic algorithms*. These differ largely on the methods used to estimate the distribution of variables or the requirement for prior information. A population of genomes are evaluated and joint distribution/joint density functions are used to estimate the distribution which is sampled when generating the new population. The simplest probabilistic methods estimate the occurrence probability for single variables, the earliest of which was population based incremental learning PBIL [Baluja, 1994]. Later methods estimated the joint distributions of two or more variables. For a complete review refer to [Pelikan, 2005, Pelikan et al., 1999].

3.4 Developmental Methods

The following subsections review gene expression primarily in the context of neutrality. Neutrality represents a perspective on evolutionary dynamics that is coupled to other themes such as evolvability and weak interactions in gene networks. Having looked at properties of gene expression we close by considering the emerging class of artificial embryogenies.

3.4.1 Neutrality

A key contribution of gene expression based methods is their ability to promote diversity and stability in mapping to the phenotype. From studies in RNA and protein molecules, clusters of phenotypes are known to be connected by single point mutations. These are called neutral networks. This allows genetic change to be made

while maintaining an existing phenotype and can reduce the chance of being trapped in suboptimal regions. A considerable fraction of all mutations are said to be neutral, having no causal link to the survival or reproduction of individuals [Kimura, 1983]. This view of evolution is not without controversy. Developmental algorithms like GE exploit neutral redundancy through abstract modeling of the biological genetic code where 64 codons map to only 20 amino acids. The use of redundant introns in GAs and GP has also been studied to enable redundancy. However, another level of neutrality is evident in genetic regulatory networks and protein folding, which some artificial embryogenies aim to exploit.

While there are a number of perspectives on evolvability, one perspective is the ability of random variations to sometimes produce improvements over the parent [Wagner and Altenberg, 1996]. Following this perspective, [Ebner et al., 2001] evaluated the differences between five genotype-phenotype mappings based on their ability to reach phenotypes and stability on random neutral walks [Huynen, 1996]. The first is a standard direct encoding. Second, a voting mechanism allows a number of genes to vote on the expression of a phenotype feature. It exhibits redundancy in that mutations may not affect the outcome. Note it behaves like a direct encoding in its one-one genotype-phenotype mapping. A third, cursor based method, uses separate commands to move a cursor and to write or clear bits. Two other mappings are based on a non-uniform one dimensional cellular automata [Wolfram, 1983] and random boolean networks [Kauffman, 1993]. The neutral walk is carried out as follows; A random phenotype is mapped from a genotype. From this genotype, single point mutations are applied. If a new phenotype is found, termed innovation, it is logged. The neutral neighbours are also logged. One of these neutral neighbours is chosen at

random and the procedure is repeated until no further innovations are possible. The boolean network showed significant improvements over the other mappings. This is said to provide a good balance between randomness (randomly scattering mapping into phenotypes around genotype space) and structure (maintenance of neutral networks). This work demonstrates the intuitive effect of redundant mappings on rugged or adaptive landscapes [Ebner et al., 2001].

Volkert and Conrad's *dual dynamics* model makes an important distinction in terms of informational impact between strong interactions (individually responsible for modification e.g. control molecules in biology) and weak interactions (gradual, indistinguishable contribution to behavior modulation e.g ion concentrations) [Volkert and Conrad, 1997, Volkert and Conrad, 1998, Volkert, 2003]. The relationship of the dual dynamics model to neutrality may not be immediately clear as weak influences are not widely discussed in the GP literature. We note that there are many mappings in natural developmental systems, while the genetic code is the most familiar to the GP community. Therefore, we use this as a reference point. It is well known that many codons map to the same amino acid and that there are redundant bases in some codons. The independent influence of each base could strongly or weakly influence the amino acid outcome.

Volkert's work evaluated behavior space coverage and accessibility to neutral and near-neutral networks [Volkert, 2003]. Information obtained from such analysis was correlated with performance observed on evolutionary tasks. By adding weak interactions, Volkert distinguishes the dual dynamics model from non-uniform cellular automata similar to NK Boolean Networks [Kauffman, 1993] or cellular automata which use discrete influences. Multiple strong inputs are translated using a random

function in a lookup table and are then integrated with a summed influence from weak interactions to give a local state transition. Varying the influence of strong and weak intention via tuning of parameters allowed Volkert to evaluate the differences between entirely strong, entirely weak and dual networks [Volkert, 2003]. Volkert's hypothesis is that a system which uses a coupling function to dictates a collective weak influence function to *linearly modulate* the outcome of a strong influence function, will demonstrate increased evolvability. Linear modulatory coupling enforces a structure-function relationship that enhances mutation buffering. Importantly, Volkert distinguishes between linear modulation and logic based modulation. Results from behavioral space coverage experiments showed that for some behaviours there are a larger number of neutral sets of linear modulation networks and strong function networks than for logic based networks. Similar tests investigating the number of neutral offspring from single mutations further suggested the importance of these distinctions [Volkert, 2003].

Neutrality is a widely studied theme in promoting evolvability in artificial systems. Many other important issues such as Volkert's study of weak interaction, modularity, compartmentalization, and extra dimensional bypass have been considered by Michael Conrad [Conrad, 1990] and are explored in a special issue on Evolvability in Biosystems (69) 2003. These issues are hugely important to artificial evolutionary systems but sadly can not all be covered here.

3.4.2 Gene Expression

Interest in representations leads naturally to discussion of gene expression based approaches. These methods make the genotype-phenotype mapping (GPM) explicit and aim to overcome the limitations of more direct mappings. One clear advantage of these approaches is the separation of genotype and phenotype survival thresholds.

While the methods discussed here are sometimes referred to as developmental, we prefer not to use this term, which is reserved instead for embodied, stateful mappings with ontogenetic processes. We term these methods *gene expression based* in that they consider alternate ways of expressing genetic material in order to map the phenotype. One of the earliest of these methods which borrows from Kimura's theory of neutrality follows from [Banzhaf, 1994]. This mapping is said to be critical to search progress as the larger the [genotype] space that is mapped to good phenotypes, the better the performance [Keller and Banzhaf, 1999][†].

A number of gene expression methods consider 'gene concentrations' to allow for weak interactions between alleles [Azad and Ryan, 2003, Wu and Garibay, 2002]. Chorus [Ryan et al., 2002] is based on GE [O'Neill and Ryan, 2003]. However, in Chorus a codon only causes a grammar rule to be fired if it refers to a transition which is currently applicable and if there are no other relevant rules *waiting* to be expressed. Waiting rules are determined by concentration levels; If a codon maps to a rule which cannot currently be applied, the rule's entry in a *concentration table* is incremented. In this way, a particular gene may have an immediate or a delayed effect and position is not as strongly coupled to its role. However Chorus does not perform as well as GE.

[†]The type/rate of change on the phenotype with respect the the type/rate of change on the genotype is important to the evolvability of a system and navigation through the search space.

The proportional GA [Wu and Garibay, 2002] determines expression by the *proportion* of its corresponding symbols within a chromosome. Such weak interaction is consistent with combinatorial gene expression in nature. This algorithm is shown to be as good as a standard GA on many problems and significantly better on certain problems.

Michael Lones' Enzyme Genetic Programming [Lones, 2004] is based on careful consideration of biological principles. Enzyme GP is characterized by deriving program structure from component connection choices rather than explicitly specifying structure. Components are given meaning via *shape* using *activity* or *functionality models* [Lones and Tyrrell, 2004a], the later of which is relevant to variable length GP. These models capture the usage of Genetic Programming components in terms of their inputs and outputs and allow the interaction between components to be evolved in a position independent manner. These descriptions allow components to be described with variable precision, hierarchically in terms of their sub components and independent of the program structure they are in. The details of functionality [Lones and Tyrrell, 2004a] are beyond the scope of this discussion. *Implicit context* is enabled in Enzyme GP through the use of functionality measures that describe a component's purpose which allow components to fulfil roles with variable specificity [Lones and Tyrrell, 2004a]. Lones and Tyrell [Lones and Tyrrell, 2004b] explore the importance of this type of implicit context in biological systems where useful variation is exploited while inappropriate change can be ignored. This notion of context is also relevant in principle to degenerate coding in GE.

A number of studies on artificial chemistries for GP [Banzhaf and Lasarczyk, 2004] and gene regulatory networks [Kuo et al., 2004] for GP represent an ideology that is also important to the current research. As we discuss gene regulatory networks,

perspectives spill into the realm of artificial embryogenies, as discussed in the next section (for example of using regulatory networks in an artificial ontology refer to [Bongard and Pfeifer, 2001]). However let us continue to discuss this trend of exploiting gene expression in Genetic Programming here.

The shortcomings of Genetic Programming and requirements to cope with complexity are well characterized in [Banzhaf and Miller, 2004]. As in GA, GP has scaling problems. Development and regulatory gene networks are becoming widely accepted as a promising approach to the complexity issue. As such it is necessary to understand the marriage between self-organization and selection or variation. Kaufmann must be cited as a hugely important contribution to this line of thought. However Banzhaf and Miller take up this investigation in the context of Genetic Programming [Banzhaf and Miller, 2004]. Self organization can lead to complex structures. What role does the genome play? A perspective is summarized in [Banzhaf and Miller, 2004] as

The biochemical network of interactions between substances is the underlying substrate of a system of control built upon the effect of additional substances (signaling substances), that is itself produced by genes. The system, is highly combinatorial in that many of the biochemical (maintenance) substances can interact with each other and with the signal substances. It is through a control of the expression of the where and when of the signals that genes exert their control on the underlying network.

Banzhaf and Miller refer to *heterochrony* as the control of the onset, rate and offset of a gene's expression. Following this they observe that the external influences in the cell can determine this type of regulation i.e. when genes are to be expressed. We

add, a complex, multi particle system creates a non-computable spatial state. The regulation problem can be considered to be a functional induction problem where each particle in the combinatorial system must appropriately determine the *when* and to what *extent* its simple behaviours are expressed. This becomes interesting when they react to the environment they are modifying. In general, the major mechanisms of developmental biology are summarized in [Banzhaf, 2003] as heterochrony, the use of spatial patterns and social interaction.

[Banzhaf and Miller, 2004] Outline 'nature's recipes' for exploiting information during development, many of which are widely appreciated within the emerging field of Artificial Embryogeny.

- The importance of embodiment - the natural regularities of the environment in which a system develops
- Phenotypes are open system in steady states not in static equilibrium. We add that this is also important for composition as entropy 'leaks' between 'hierarchies' thus providing the rules of composition.
- They observe the incremental development of higher order complexity
- Development proceeds by way of communication between social systems
- Successive developmental states are recursively defined of preceding blueprints. They point to Lindenmayer Systems. We observe that fractal forms are pervasive in both living and non-living forms and have been greatly studied in chaos theory [Peitgen et al., 2004]. Also, morphogenesis has special significance in spatially extended systems [Bonabeau, 1997] which is not observed in L-Systems.

- Fitness evaluations are punctual i.e. feedback occurs at sub scales rather than on the entire organism.

Banzhaf and Miller outline a number of other important features, some of which are addressed by artificial embryogenies discussed below. The interested reader is advised to refer to [Banzhaf and Miller, 2004].

Following such insights, Banzhaf discusses general properties of artificial regulatory networks [Banzhaf, 2003]. ARNs are applied to evolutionary problems in [Kuo et al., 2004]

An ARN consists of a bit string representing a genome and mobile information carrying molecules or proteins which are equipped with bit patterns for interacting with the genome at regulatory sites located upstream from the genome. Attachment to these sites produces inhibiting or activation of the proteins on corresponding sites. These inhibitory/excitatory interactions constitute the ARN and are shown to have interesting dynamics in [Banzhaf, 2003]. [Kuo et al., 2004] extend the dynamical network model to add semantics to gene expression. This allows the model to be applied to simple function optimization. These models are very new and have yet to be developed and applied to a wider range of GP problems.

Another development along this line of thought is Genetic Programming base in [Artificial] Chemistries [Banzhaf and Lasarczyk, 2004]. These are similar in one respect to Enzyme GP in that the structure of a program is not given explicitly but derived from connection choices [Lones and Tyrrell, 2004a].

In summary, Genetic Programming over the years has gone through stages of reconsidering how genes are mapped to proteins. The cutting edge perspective is stepping into biological inspired processes of self-organizing development. It is im-

portant to remember that this history goes back further than questions about genetic codes and genotype-phenotype mappings in GP, to the fundamental issue of how information is encoded, preserved and expressed from the artificial genome.

3.4.3 Artificial Embryogeny

In many ways artificial embryogenies (or artificial ontogenies) are a special application of evolutionary algorithms, where the program itself is a developing structure such as an emergent cellular automata or multicellular state [Banzhaf et al., 2003, de Garis, 1992], morphologies/structures [Hornby and Pollack, 2001, Hogeweg, 2000], neural networks [Kitano, 1990, Gruau et al., 1996, Gruau, 1994] or gene networks [Eggenberger, 1997, Bongard and Pfeifer, 2001]. The interpretation of the term *developmental* may well depend on the level on interaction between the genetic and ontogenetic processes. For example in [Amarteifio and O'Neill, 2004] we evolved rules to control particles in spatially extended systems. The fitness of these systems depends on emergent, non-computable states in the agents world. However, once the agent is created, the genetic material has no further influence in developing agent behaviours. On the other hand, artificial embryogeny is defined by [Bentley and Kumar, 1999] as having

- Indirect correspondence between alleles and phenotypic effects. The genotype is now regarded a set of growing instructions, a recipe which defines how the phenotype will develop.
- Polygeny. Phenotypic traits are produced by multiple genes acting in combination

An algorithm such as GE applies a rule to a stream of codons in order to develop a phenotype. The developing phenotype and changing grammar context does influence further development; this might be thought of as *syntactically developmental*. In fact GE has many interesting qualities at this level including syntactic neutrality. However, is this type of control sufficient for evolving complex systems? Does it promote diversity and evolvability? Is there sufficient complexity in the genome (wrapped or unwrapped) to develop complex structures. Artificial Embryogeny or Artificial Ontogeny systems are, in principle, a class of emerging algorithms that aim to address these concerns. We review a number of these algorithms asking what they offer beyond syntactically developmental algorithms.

Developmental processes shift the role of the evolutionary algorithms to specify generative rules rather than entire phenotype structure putting emphasis on ontogeny. However as well as being a special application, artificial embryogenies demonstrate design principles for evolutionary algorithms in general, as seen for example in artificial regulatory networks [Banzhaf, 2003] above. Artificial self-organization and artificial evolutionary processes have become symbiotically coupled. On the one hand evolution can be improved through the use of developmental processes while developmental processes offer insight to general evolutionary theory - distinct from development.

At the heart of embryogenesis and morphogenesis is the spatial blueprint-temporal development coupling. Stages of development could be stateless or statefull. One may distinguish between ballistic and adaptive assembly plans [Rieffel and Pollack, 2004]. While adaptive plans incorporate feedback during development, ballistic do not. The only feedback is at the evolutionary scale. GE is an example of a ballistic assembly plan. Approaches which embody statefull stages are generally based on cell chemistries.

One of the earliest uses of the term *Artificial Embryogeny* in evolutionary algorithms may come from De Garis [de Garis, 1992] who is broadly interested in evolving very large intelligent network structures. De Garis used a genetic algorithm to specify cell behaviours in order to grow shapes. While some shapes emerge more naturally than others, De Garis noted the difficulty in evolving structures which depend on regulating contexts for behavioral expression, i.e. heterochrony [Banzhaf, 2003]. Lately work such as [Streichert et al., 2003] point to the important of *limited growth* for more advanced regulation. In any complex system including embryological processes and the immune system, in addition to growth processes like differentiation, processes such as apoptosis are equally important to sculpt a solution [Cohen, 2000].

A developmental mechanism as an indirect encoding aims to reduce the combinatorial explosion of allele-matching requirements to enable evolvability of large phenotypes. These encodings exploit gene reuse. Peter Eggenberger [Eggenberger, 1997] addresses the gene expression issue and offers an important perspective on this matter. He suggests that the complex gene expressing events and genotype-phenotype mappings in developmental mechanisms allow reduction of genetic information without losing complex behaviour. Presumably, this exploits large scale redundancy and may not necessarily apply at all to simple problems. He observes that the genotype will not necessarily grow as the number of cells increase. GE uses a degenerate coding scheme where a wrapped genome can be used several times. While this promotes gene reuse, it may not overcome the trade-off i.e. pleiotropy; as genes are reused, dependencies also increase. A gene reuse mechanism must be intelligent enough to overcome this. In GE this becomes the problems of finding one sequence of codons which, when mapped in the context of the grammar in 2 or more situations, will produce the favourable phenotypes at the location.

Rieffel and Pollock [Rieffel and Pollack, 2004] discuss development through indirect encodings/mappings (i.e ontogeny) in erroneous environments and the ability for these encodings to specify intermediate morphological elements. Development error is imposed through a world physics where an action and a result of an action are disassociated; an action is dependant on physics and may not have the 'intended' effect. This seems unnatural because an adaptive system relies on regularities and is like learning to walk when the ground periodically gives way under your feet. However it is plausible because regularities are not always obvious. Rieffel and Pollock's intermediate morphological elements or ontogenic scaffolding, are features that exist during development, assist development but do not necessarily appear in the solution. Such features could be considered, generally, as temporal rather than structural building blocks. Rieffel and Pollock address an interesting question; is there sufficient information to allow the evolutionary process to cope with stochastic environment? They seem to present the issue as a complication with developmental error and resultant fitness distributions, which results in a fitness assignment problem. This leads them to develop the following observation; it may be more informative to allow each genotype multiple stochastic developments. A genotype will then produce an entire distribution of phenotypes. Their answer to determine a mapping's yield is pareto optimisation, which requires some supervision, together with the use of multiple trials which is arguably quite similar to the use of test cases in a symbolic regression problem (i.e. quartic symbolic regression where $x = 1$ gives a noisy signal). They also observe distinct developmental phases where scaffolding is first created to facilitate solution construction and then removed. While this seems very interesting in terms of morphogenesis, it is unclear whether a teleology-imposing observation has been made. This and the fact that their physics makes scaffolding a more probable solu-

tion makes the relevance of these results seem unclear. While we are unclear about the findings, it is certainly a very provocative study. What information is driving the indirect mappings? How can evolutionary algorithms control one-to-many genotype-phenotype mappings? In this study this issue will be referred to as enabling canalized diversity.

Bentley describes the evolution of fractal proteins for providing a rich medium for evolution in order to improve evolvability [Bentley, 2004b, Bentley, 2004a]. The ideology behind this work is, in part, to produce representations with as few constraints and as much richness as possible. This is in keeping with biological design principles. In this model, the genome is extended to encode both structural and regulatory genes. The genome is embodied in a cell which has a cytoplasm that also hosts behaviours. Cells in turn exist in an extended environment with one or more cells and one or more fractal proteins. A fractal protein is defined as a subset of the Mandelbrot set. Because of the richness in fractal proteins, fractal chemistries naturally emerge. All fractal proteins are merged to calculate whether a gene should be activated. This weak linkage property has been discussed in previous sections. The space of fractal subsets is interesting; similar fractals can be near to each other, similar fractals can be found in many different places and the space is theoretically infinite. Through this representation, Bentley suggests the counterintuitive consequences; rather than make the search space as small as possible, the space is made close to infinite. However, the method is able to find solutions without difficulty proving that evolvability, not size of search space, is critical to evolution. Fractal GP approach demonstrate interesting developmental patterns in evolving gene regulatory networks. See results in [Bentley, 2004b]. Fractal protein model is extended to immune regulatory networks in [Bentley and Timmis, 2004]

Federici discusses the problem of applying artificial embryogenies to difficult problems. In ontogenesis genes might be thought of being degenerate both structurally and temporally. A gene that 'could' be mutated and not seem to be relevant to the solution, may be important during development. This type of dependency can result in creating deceptive fitness landscapes [Federici, 2004]. To address these issues, Artificial Neural Networks are used to encode growth space, diversity is regarded and development may happen in more than one Embryonic stage. The motivations for these decisions are interesting. ANNs are used because of their supposed suitability to modeling genetic regulatory networks and their affinity to neutral explorations [Federici, 2004]. The major contribution of this work is on *embryonic stages*. It has been observed that species that are evolutionary related, share early stages of embryonic development and differ in later stages! This observation leads Federici to suggest that subsequent modification of early stages could be catastrophic and reducing modification of early steps may assist evolvability. Diversity is important for many reasons but in the context of the previous consideration, it will allow exploration in different regions of the search space when returning to previous stages is impossible. This is achieved by evolving an ANN with one chromosome, preserving that chromosome and using a copy of that chromosome for the next developmental stage. The state of the ANN in the previous stage is used to initialize the next stage of development. By using chromosome copies in subsequent stages, the individual's ontogeny is conserved as evolution continues to operate incrementally. Results showed a positive effect on performance and good resistance to fault tolerance[Federici, 2004].

Above we have described just a few features of artificial embryogenies; embryonic stages, richness of environments, error tolerances and scaffolding in development, limited Growth and indirect mappings.

3.5 Summary

We have reviewed research in encodings in genomes, model building and modularity in genomes, and finally gene expression and developmental processes. Later chapters exploit the principles discussed in these sections in developing evolutionary models.

From the first section on representations we extracted salient features of interest such as the preservation of linkage and the use of degeneracy. In the subsequent section on model building, we recognized the importance of maintaining a memory of modular structures, assuming useful modules could be identified.

As we look at studies based on gene expression, it becomes clear that an appropriate genotype-phenotype mapping is required to ensure that the space of useful phenotypes is mapped. Perhaps the most ideal mapping is a combinatorial gene expression mechanism which allows for scalability of phenotypic complexity. This in turn begs the question about context of genetic material and how genes used redundantly and pleiotropically can be given meaning. Following discussions of neutrality and genetic codes, we reviewed a number of gene expression based mechanisms.

Finally we looked at a number of principles, sometimes found in artificial embryogenies, which are important to development. These biologically inspired principles may contribute to evolvability in artificial systems.

Chapter 4

XMLGE

...To load a Program into the computer: this does not mean that the future has been planned down to the last detail, that everything is written "up above". For example, the program did not specify that in 1815 a battle would be fought near Waterloo and that the French would be defeated, but only that man is aggressive by nature, that he is condemned to wage war, and that technical progress would make war more and more terrible.

- Milan Kundera ' Immortality '

4.1 Introduction

This chapter introduces the XML-based Grammatical Evolution framework which may be used for stand-alone and distributed Grammatical Evolution. Using XML for evolutionary computing is a novelty that offers advantages for analysis, collaborative research and system decomposition. Initially developed for use on a single-machine, inherent qualities of XML made distributing the process relatively painless. The pri-

mary purpose for using a distributed model has been to evaluate single GE genomes on remote clients and carry out evolution on one machine in situations where individual evaluation is expensive. However the parallelisation model can be extended to allow the selection process to be distributed also.

This chapter describes Grammatical Evolution in eXtensible Markup Language (XML) [W3C, 2005a]. XML is used to mark up all artificial genetic material and permits rich representations. The eXtenable Stylesheet Transformation Language (XSLT) [W3C, 2005b] is a declarative language used to write genetic operators. XML trees encode all information required to generate programs in a high level programming language. This allows program structure to be serialized and assists a distributed evaluation architecture. Many of the methods used are inherently inefficient because they are based on XML. However XMLGE has been developed to explore the use of rich representations.

The primary motivation for XMLGE is to abstract the evolutionary process, taking all evolutionary logic and placing it in XML. From a philosophical perspective, this is similar to the idea that organisms in nature consist of the same elements and processes and differ only through DNA. XMLGE uses core set of classes to load and manage XML, which dictates properties and processes expressed at runtime. From a practical standpoint this allows one to concentrate on evolutionary logic and share logic in a standard, persistent way. An extensible evolutionary process is defined through XML with well-defined phases. Ideally it should be possible to carry out any evolutionary experiment changing only evolutionary logic defined as XML or adding elements to the XMLGE environment written in high-level code for what we term the fitness environment.

We continue to section 4.2 to provide background information on XML. We introduce XML, XML Schema and XSLT. Following this we will discuss XML in more detail through its applications in XMLGE. We describe how these elements are used for evolutionary operators, evolved-program serialisation, communication and validation. We will take an overview of the XMLGE framework describing its usage.

4.2 XML

As evolutionary algorithms and their analysis are data intensive, we consider XML for their representation. XML is a W3C standard mark-up language and has been widely adopted across many industries. It is an extensible language allowing document authors to define their own elements, attributes and document structure.

The hierarchical structure of an XML document implies semantics in a similar fashion to a LISP S-Expression. In XMLGE, XML documents represent both genomes and S-Expressions. XML Genomes have a flat structure, a parent XML 'Genome' node and variable length list of 'Codon' nodes with integer values (we do not consider binary codons here). XML S-Expressions are hierarchical structures with each node level corresponding to a symbol taken from the GE grammar.

XML has a number of standard supporting languages including standards XML Schema [W3C, 2005e], XPath [W3C, 2005d] and XSLT [W3C, 2005b]. XML schema is normally used to define document types. It constrains and validates document instances. In XMLGE, XML schema corresponds to the BNF grammar. In addition to being far more expressive than BNF, because it is represented in XML, it can be parsed by XML tools such as XPath.

The XML-based grammar is very similar to the BNF counterpart, apart from the use of XML Schema keywords such as 'choice', 'sequence', 'enumeration', 'simpleType' and complexType. These elements are important in the mapping as they provide information such as whether the symbol is a terminal or non-terminal or if a choice is being made and hence if the GE rule must be applied to use a genome value. The natural recursive processing order of XSLT adheres to the way GE is naturally applied. This is an interesting use of XML Schema because XML schema normally validates an existing document instance (which it may also do here) but it now generates valid document instances too.

XPath is a regular expression language for matching node sets in XML trees. Working from a context node it is possible to select child nodes and parent nodes, ancestor and descendant nodes, sibling nodes and attributes. A query can filter by node names and positions and can specify a node that has a 'related' node or node set with certain constraints. For example, an XPath query could identify a child node, which has a child with a certain name, which has a certain attribute of a certain name and a certain value. XSLT is a language used to transform XML data. Taking a source document it uses a number of templates to match nodesets using XPath and describe result output using the context node's values. It works recursively in an XML tree using values from one source document to determine the output document. XSLT is commonly used to generate HTML content from XML sources.

XSLT stylesheets and XPath play an important role in XMLGE. XSLT is used to transform XML data based on evolutionary logic or to transform data into other forms for analysis or representation of data. In the first case, XSLT is used for example as a crossover operator or the GE transcription operator. In the later case, XML results can be transformed for example into SVG graphs.

An XSLT stylesheet is used to apply the GE mapping. The mapping reads a sequence of genomes and using a given rule will select a sequence of grammar symbols in order to construct a program tree. In XMLGE the Genome population is merged with the grammar so the XSLT file can act on one document to generate the resulting XML based S-Expression population.

SOAP [W3C, 2005c] is the specification of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. Further description of SOAP is not required to understand its role in XMLGE. Suffice to say, it defines the Web Service packet structure that is used to exchange data in the distributed model of XMLGE. Even Dynamic Link Libraries (DLLS) can be passed over the wire in this way as binary data encoded in strings passed in XML nodes.

4.3 XML and Evolutionary Algorithms

In XMLGE, XML represents both genetic material and genetic mechanics. High level languages in turn describe the fitness environment in which this genetic material is evaluated. In XMLGE the fitness environment consists of the fitness function and the set of symbols defined in the grammar.

XML is used to represent all data in XMLGE including parameter files, serialised data types and result data. It is used to represent genomes and X-expressions generated from genomes. In this text we will generally refer to XML based X-Expression trees as X-Expressions to avoid ambiguity. Because data is represented in XML it can be persisted directly to file, transmitted easily in a SOAP message and shared by users who know the document format. XML files, which are verbose and large,

are generally compressed in XMLGE using GZip compression for both storage and transmission.

XML genomes consist of one genome element contains a variable number of codon elements which have integer values between zero and an upper limit. Genomes store fitness values and id's as attributes.

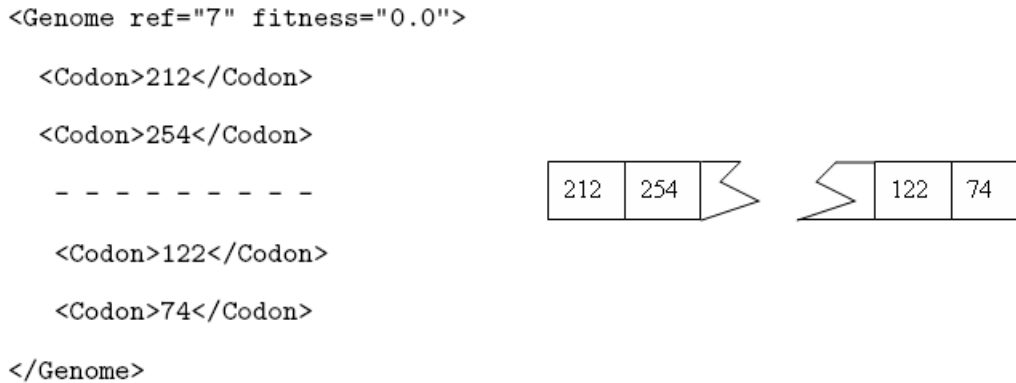


Figure 4.1: Sample Genome

S-Expressions in XML, called X-Expressions henceforth to differentiate from other meanings, have an arbitrary structure of Symbol elements as defined in the grammar. Terminal symbols contain primitive string or numeric values. The Start-Symbol contains a ref attribute that corresponds to the Genome ref. A production sequence in GE such as

1. $\langle \text{StartSymbol} \rangle \Rightarrow \langle \text{SymbolType1} \rangle$
2. $\langle \text{SymbolType1} \rangle \Rightarrow \langle \text{SymbolType2} \rangle \langle \text{SymbolType3} \rangle \langle \text{SymbolType2} \rangle$
3. $\langle \text{SymbolType2} \rangle \Rightarrow T1$
4. $\langle \text{SymbolType3} \rangle \Rightarrow T2$

5. `<SymbolType2> ⇒ T1`

is structured as follows.

```
<Start-Symbol ref="1" fitness="0.0">
  <SymbolType1>
    <SymbolType2>T1</SymbolType2>
    <SymbolType3>T2</SymbolType3>
    <SymbolType2>T1</SymbolType2>
  </SymbolType1>
</ Start-Symbol >
```

Each of the above structures representing genomes or X-Expressions exists within its respective XML document under a single root `<Population/>` element.

We now build an XML Schema that corresponds to a BNF grammar definition. The complete schema can be found on page 71 in figure 4.2. We start with the root element and its namespace declaration. Notice how the root is namespace qualified and the namespace is declared inside the node.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- Schema/Grammar Content Goes Here -->
</xs:schema>
```

Within this schema element, all the elements that will appear in the XML instance will be defined. The schema describes the structure and content of the document by specifying what elements and attributes can be contained within them. An element that can contain other elements is defined as a complex type while an element that

cannot be of simple type. A complex type would be used to define every non-terminal BNF symbol. An element can be defined with an embedded complex type defined inline, or the complex types can be defined and reused. The later approach is better for our purposes.

The first part of the schema specifies that XML instances can have expressions, or `<expr>` tags as the root node. This corresponds to the start symbol in the BNF grammar used in GE. This element will sit one level below the `xs:schema` root element and does not have an explicit BNF counterpart.

```
<xs:element name="expr" type="Expression"/>
```

The schema element definition specifies the element's name and type. 'type' corresponds to its schema complex type definition shown below. There is a sequence with three elements, and a loose element. Note that the single loose element might well have been wrapped in a sequence element. Note also the XML Schema definition elements prefixed with `xs`, which reflects the XML Schema namespace defined in the root.

There are various types of sequences, bags, enumerations etc. that can be used in XML Schema. XML Schema also provides data type constraints. The data type might specify that an element or attribute must contain a number, string or date for example. These data types can be extended. These details are not important to the current discussion.

The BNF specification

```
<Expression> ::= <Expression><Operator><Variable>
                | <Variable>
```


is represented in XML Schema using explicit choice and sequence keywords as shown below.

```
<xs:complexType name="Expression">
  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:sequence>
      <xs:element name="expr" type="Expression"/>
      <xs:element name="op" type="Operator"/>
      <xs:element name="expr" type="Expression"/>
    </xs:sequence>
    <xs:element name="var" type="Var"/>
  </xs:choice>
</xs:complexType>
```

Each of the elements have name and type attributes. Their 'type' attributes refer to their complex type definitions. Each element, including grouping elements e.g. `xs:choice` can have `minOccurs` and `maxOccurs` attributes. These are not necessary for XMLGE.

eXtensible Stylesheet Language Translations (XSLT) is a language for translating a source XML document into a result using an XSLT stylesheet*. Like XML Schema, XSLT stylesheets are written in XML. Each of the stylesheet elements is qualified with an `xsl` prefix and an XSLT processor knows to use these `xsl` prefixed elements. Below the root node of an XSLT stylesheet is shown.

*XSLT is a declarative language. Some of the details explained here are given to stress the stylesheet language's ability to *compute*. As such, it may seem strange that we discuss what would be trivial details in a high level programming language.

```
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<!-- Stylesheet Content Goes Here-->
</xsl:stylesheet>
```

Stylesheets are composed primarily of templates and match clauses. An XSLT processor will read a source document, parsing through each of its nodes and attributes. If a node or attribute matches a template, that template is applied. Within the template, the stylesheet author will specify how the output should look, most likely using elements and attributes (or their values) from the source. Here is a simple template that matches the root document and does nothing. The Forward slash refers to the root node.

```
<xsl:template match="/">
<!-- Template Content Goes Here-->
</xsl:template>
```

XSLT uses a separate language called XPath which is a language for identifying node sets in XML documents; this forward slash is a simple example of XPath. The root node of the source might contain a `<Population>` element. If so, within the template (empty in this case) one might choose to process the individual node of the current context. It is in many respects similar to navigating a directory structure. `./population/individual` would identify an individual child of 'population', which is itself a child from the current context.

In an XML file there may be multiple children of the same name, unlike in file systems. The above XPath expression would therefore refer to a 'nodeset' i.e. all 'individual' children. XPath can specify the second child; `./population/individual[2]...` the second and third child; `population/individual[2|3]...` the individual with an

'id' attribute set to 135;

`population/individual[id='135']@`, where @ refers to attributes.

XPath also defines *axis* which are written as 'AxisType::'. The child axis, `child::*` can be used to the same effect as `./*`. Similarly, the parent axis, `parent::`, can be used to the same effect as `../`. There other axis also like following-sibling::*, or ancestor-or-self::*.

XSLT has a number of functions such as `position()`, `current()`, `sum()`, `count()`, which can be applied to nodes or node sets. It is also possible to extend XSLT by writing functions in scripting or high level programming languages to be included in XSLT stylesheets. For example, XMLGE uses extension functions written in Java or C# for random number generators and tournament selection.

XSL has elements for processing XML files. For example `xsl:for-each`, `xsl:if` and `xsl:choose` allow iterative and conditional processing. `xsl:value-of`, `xsl:copy-of` and `xsl:copy` allow elements or their values to be processed. `xsl:element` and `xsl:attribute` allow for the creation of new elements and attributes in results. `xsl:variable` and `xsl:param` allow the use of variables and parameters as XML sources are processed. `xsl:apply-templates` and `xsl:call-templates` allow templates that are defined in the stylesheet to be called. Consider the following source file.

```
<RootElement>
  <AnotherElement attrOne= 'whatever' attrTwo= '4' />
  <ElementWithChildren>
    <Child fitness = '19' />
    <Child fitness = '5' />
  </ElementWithChildren>
</RootElement>
```

We will use two templates to arrive at the following result. This result is a simple XML file with some text content.

```
<SimpleRoot>
  <TransformResults>
    This is Text: Child Two had fitness of 5 and
    child One had fitness of 19. The average fitness is 12
  </TransformResults>
</SimpleRoot>
```

We get the result by matching `<ElementWithChildren>`, getting its children, getting their attributes and printing them in the results. We also calculate the average fitness in a variable and print that. Here are the two templates. This first part matches the root and applies templates to all children.

```
<xsl:template match="/RootElement">
  <xsl:apply-templates= "child::*"/>
</xsl:template>
```

The template for the second child is shown below. This is specified in the match clause.

```
<xsl:template match="ElementWithChildren">
  <xsl:variable name = "average" select="sum(Child/@fitness)
    div count(child:*)" />
  <xsl:element name='SimpleRoot'>
    <TransformResults>
      This is text: Child Two had fitness of
```

```
<xsl:value-of select="Child[2]/@fitness"/>
and child One had fitness of <xsl:value-of
select="Child[1]/@fitness"/>.
The average fitness is <xsl:value-of select="$average">
</TransformResults>
</xsl:element>
</xsl:template>
```

We have used a variable element to compute the average. This uses the sum, div, and count functions of XSL. Within the sum function, we use a node-set or attribute-set argument. These evaluate to numeric 'fitness' values in this case. The count function counts the child nodes in the current context. We have created the result by creating the elements and text output within the template. Notice we have used two methods for creating elements. One (SimpleRoot) uses the xsl:element and the other (TransformResults) writes the element directly. `<xsl:value-of select/>` gives the value of a node or variable in the select clause. In this case the 'average' variable is used. Variables are prefixed with the dollar symbol.

4.3.1 The GE mapping using XSLT

XSLT works by transforming a source document and creating a result output. In XMLGE, two source documents are used by one stylesheet in order to produce a result in the mapping process. The main source document is the genome population but the grammar is also used as the genome is parsed; this is the principle. What is in fact happening is the grammar is being transformed using values from each genome individual to make choices. *For each genome individual, record the codon values on a stack. Transform the XML schema (grammar) into an XML instance for*

that individual, sequentially using the codon values at each stage of the mapping. A complete schema and corresponding BNF can be found on the next page.

Note the `S = expr` is added for consistent comparison and would not appear in a BNF grammar.

Applying GE styles

A stylesheet processes the schema by matching a template or handler for each important element in the schema. The most important elements to GE are the `<xsd:choice>` and `<xsd:simpleContent>` as it is in these elements that the decisions are made about mapping. The schema defines

```
<xsd:element name="expr" type="Expression"/>
```

to bootstrap processing. This is the only element not contained within a complex or simple type. The process will begin to build an XML element with name 'expr' and of type 'Expression'. The use of this 'Expression' type attribute is twofold in XMLGE. For the stylesheet it decides which complexType definition to process. Also, this will be used later as a Java or C#'class in constructed by wrapping the XML individual. Note we differentiate between 'Expression' and 'expr' logically and for discussion. However it is more reasonable to use the same value and preferably the shorter in an actual schema. Once processing has been redirected to the appropriate complexType definition i.e. 'Expression', it will reach an `<xsd:choice>` element (storing the current source node in context).

```
<xsl:template match="xs:complexType">
```

```
  <xsl:apply-templates select="xs:simpleContent|xs:choice|xs:sequence"/>
```

```
</xsl:template>
```

<pre> <xs:schema xmlns:xsd="[NAMESPACE]" > <xs:element name="expr" type="Expression"/> <xs:complexType name="Expression" mixed="false"> <xs:choice minOccurs="1" maxOccurs="1"> <xs:sequence> <xs:element name="expr" type="Expression"/> <xs:element name="op" type="Operator"/> <xs:element name="expr" type="Expression"/> </xs:sequence> <xs:element name="var" type="Var"/> </xs:choice> </xs:complexType> <xs:complexType name="Operator"> <xs:simpleContent> <xs:restriction base="xsd:string"> <xs:enumeration value="+"/> <xs:enumeration value="-"/> <xs:enumeration value="*"/> <xs:enumeration value="/"/> </xs:restriction> </xs:simpleContent> </xs:complexType> <xs:complexType name="Var"> <xs:simpleContent> <xs:restriction base="xsd:string"> <xs:enumeration value="X"/> <xs:enumeration value="1"/> </xs:restriction> </xs:simpleContent> </xs:complexType> </xs:schema> </pre>	<pre> S = <expr> <expr> ::= <expr><op><var> <var> <op> ::= + * - / <var> ::= X 1.0 </pre>
--	--

Figure 4.2: XML Schema and corresponding BNF grammar.

This template will look to find one of the elements separated by '|'. Recall that the value of the select attribute is an XPath query[†]. If the template finds such an element, it will call a matching template and pass the tree context to that. In this way, each of the complex types in the schema, whether they have 'simpleContent' or 'choice' children, can be handled in this 'switch'. The choice element will be handled as follow;

```
<xsl:template match="xs:choice">
  <xsl:variable name="wraps" select="Xgenetic:wrapCount($XgenInstance)"/>
  <xsl:variable name="allele" select="number(Xgenetic:popA($XgenInstance))"/>
  <xsl:variable name="childCount" select="count(child:*)"/>
  <xsl:variable name="pointer" select="$allele mod number($childCount)"/>
  <xsl:variable name="choice">
    <xsl:choose>
      <xsl:when test="$wraps &lt;= 1">
        <xsl:value-of select="number($pointer) + 1"/>
      </xsl:when>
      <xsl:otherwise><xsl:value-of select="number($childCount)"/></xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <xsl:for-each select="child::*[position()=$choice]">
    <xsl:apply-templates select="."/>
  </xsl:for-each>
</xsl:template>
```

[†]The use of the separation character is actually part of an XPath expression and means 'or'. This is not related to the use of the same character in a BNF grammar.

As the XSLT processor operates on the schema, it pops codons (integer values) from a stack and uses the modulus of that integer to make a mapping decision. The popped value is stored in the variable 'allele'. This stack is implemented as an XSLT extension because XSLT has no memory. The namespace prefix Xgenetic before 'popA' refers to a class that implements stacks and random number generators. popA is a function within that class. The choice variable in the middle section uses the codon value in the mod rule (and adds 1 to index from 1 in XSLT). In the last section, the stylesheet calls a template to match what will evaluate to the xs:sequence or xs:expression child at the chosen position.

The xs:simpleContent element from the schema above makes decisions in this fashion when choosing terminal values.

The xs:element handler constructs the actual tags and attributes of the element. An XML element is created with the name specified in the 'name' attribute. Following this, the complex type definition for that element is processed as discussed above. In this way, elements are constructed recursively.

```
<xsl:template match="xs:element">
  <xsl:element name="{@name}">
    <xsl:attributename="class"><xsl:value-of select="@type"/></xsl:attribute>
    <xsl:apply-templates select="//xs:complexType[@name=current()/@type]"/>
  </xsl:element>
</xsl:template>
```

We do not discuss other operators such as crossover. However these also use XSLT to transform source documents into result documents.

4.3.2 Evaluating Programs

XMLGE programs are constructed from XML expression trees at runtime using reflection. Reflection in object oriented languages allows runtime scrutiny and instantiation of class libraries, classes and methods. The use of reflection is far slower than constructing programs in a standard way. However this has been done to support remote runtime evaluations. For example, imagine remote clients that evaluate arbitrary simulation-based problems for a central server. These clients do not know ahead of time what Dynamical Link Libraries (DLLs) are being used for a particular problem or if DLLs have been updated. The details of remote evaluation will be explained.

X-Expressions are evaluated as programs within the *fitness environment*, which as we have mentioned consists of symbol implementations and fitness function implementations in a high-level programming language. There must be a symbol class to correspond to each symbol defined in the grammar. For example in symbolic regression the above abstract X-Expression could be seen as the expression or function $X + 1$ i.e.

```
<Expression ref="ID" fitness="val">
  < Expression >
    <Variable>X</ Variable >
    <Operator>+</ Operator >
    < Variable >1.0</ Variable >
  </ Expression >
</ Expression >
```

An implementation for each symbol Expression, Operator and Variable must be provided in code. Each symbol class must behave in a specific fashion. Each non-terminal must pass parameters down to its children. Each terminal must know how to evaluate itself and pass appropriate results to its parents. The Non-terminal Expression class in the XMLGE implementation first determines if it is a variable (containing only a Variable child) unary expression (Containing a Preoperator and a Variable or Expression), or a binary operator. It then computes values for child variables and expressions and applies child operators. Parameters are passed in hashtables. For example an input parameter X is mapped to a value and the parameter Y is mapped to a value for each test case. This allows the appropriate variable instances to pull the values required. For swarm evolution as we shall see, many types of parameters can be passed in at runtime representing arbitrary types of information.

By passing variables to children and managing child evaluations, X-Expressions in XMLGE are evaluated recursively at runtime and programs are effectively serialisable. This is a trivial point but an important part of XMLGE nonetheless. The term behavioural transfer is used to describe the remote transfer of code. In XMLGE as we shall discuss shortly, the fitness environment is sent to remote clients in a dll that contains all necessary symbol classes and the fitness class. As symbol classes are transferred to remote clients, which see them for the first time, the serialized form of the evolved programs can accompany them to be properly reconstructed.

4.4 Overview

This section describes the XMLGE framework architecture for stand-alone and distributed scenarios. The XMLGE abstraction means all evolutionary operators and

genetic material are represented in XML. High level programming languages provide the fitness environment. Code is also used in automation. Automation includes wrapping the specification file to retrieve values, *applying* operators and reading or saving results.

There are five stages in XMLGE; The start and end of the experiments, before and after each run, and during the run. XMLGE defines a 'transform pipe' for each of these stages where stylesheets at each stage are applied to genetic material and the results are passed to the next stylesheet in that stage. The specification file points to the appropriate stylesheet files. In this way the XMLGE core should not need to know about evolutionary logic.

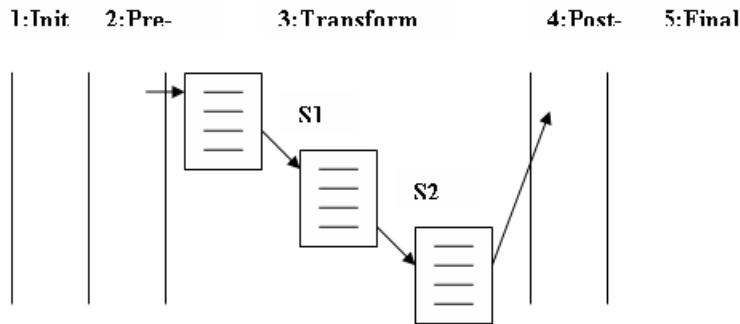


Figure 4.3: A stylesheet transformation pipe is used at each stage. For brevity, this stylesheet pipe is shown only for the Transform stage. The other five stages may also have transformation pipes. This notion is trivial and simply separates XSLT tasks (genetic operators) from other tasks such as program evaluation. The GE mapping is applied at the Transform stage while crossover will be applied in the Post stage.

In the run method, the main transform stage generates the X-Expressions from the genomes and constructs the programs at runtime based on the grammar-defined types. These programs are evaluated via the fitness function and results are tagged on the genomes and expressions in the XML document.

Distributed Mode

In distributed mode, this process is replaced by an event model to work in an asynchronous manner. A Task Manager is also added and a Task class that manages individual tasks by keeping track of genomes and their corresponding X-Expressions. X-Expressions are generated and tagged with results on remote clients and sent back to be collected by the Task class. The task logger is exposed to the world through a web service through which jobs are published and results returned.

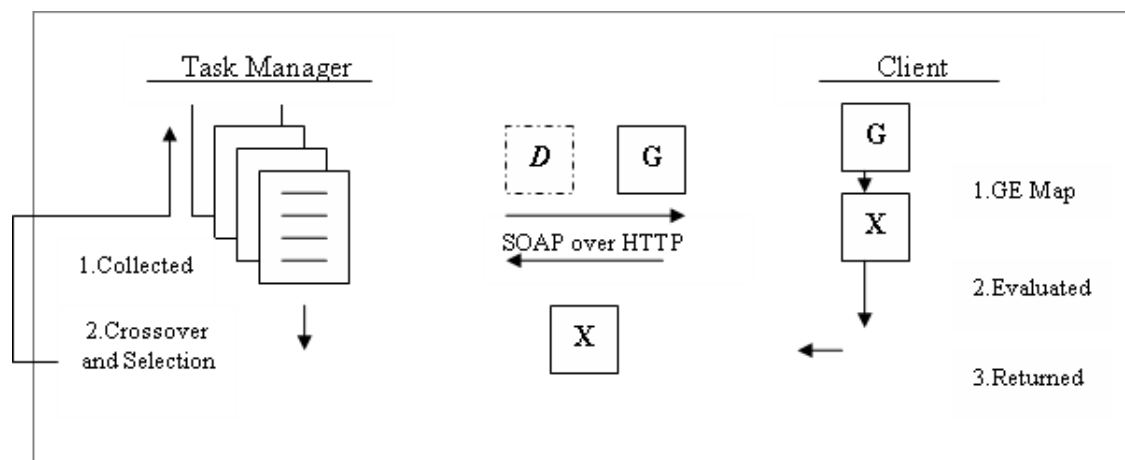


Figure 4.4: The Server (left) Generates a population of genomes i.e. tasks. These are sent over HTTP in SOAP packets to the client on request. The appropriate DLLs and XML files may also be sent to the client. The client applies the GE map to generate an X-Expression from a local GE stylesheet and subsequently evaluates the expression using the appropriate fitness environment. The expression with tagged ID and fitness is sent back to the server. When all tasks (entire population) are evaluated, crossover is applied and the next generation of tasks is created.

In order to publish an experiment in XMLGE, a web service is hosted on a web server. This web service must have access to project files such as dll(s) and XML files. A client servicing a particular server will request the active project specification and hence determine if it needs any additional files. The client requests a job (client can specify how many jobs to take at once) and the Task Manager returns one or

more genomes through the web service. These genomes are then used to generate X-Expressions using the mapping stylesheet and grammar which are contained in the project files. The client will construct the program using Symbol elements found in the project dlls supplied. This client will evaluate the program using the fitness environment supplied. When evaluated, the X-Expression is tagged with fitness. The tagged X-Expression(s) are then returned to the task manager via the web server. Representing evolutionary logic in XML means that arbitrary clients can perform operations on data on ways that are not pre-defined. This makes moving processes from the server to the client very easy as both data and logic can be exchanged.

The distribution of genome evaluations makes little sense except in the case where individuals evaluations are expensive such as evaluating simulations. In another case, where the population being evaluated is very large, applying evolutionary operators to produce a new generation is computationally expensive (particularly using XML). The Task class described above can be serialized as XML. This stores the partial or complete state of the evaluated population with genomes and X-Expressions tagged with fitness. This Task can be serialized into partial tasks and sent to remote clients (as compressed data). For example the population could be divided into ten equal segments. Clients could generate new sub-populations using for example tournament selection and send the new population segments back to the primary server for composition. We have not explored this later type of parallelization but concentrate on the 'expensive individual' case. In the next section, we consider an application of XMLGE where an individual evaluation is expensive.

4.5 Evolving a Dynamical System

This section describes an application of XMLGE to evolving cooperative swarm behavior. XML structures evolved through Grammatical Evolution correspond to 'ant genes', which are the serialized form of transducers that govern behaviours in a colony of homogeneous ants.

We use the term *synergenesis* to denote the emergence of synergy between particles that previously behaved chaotically from the macroscopic perspective. In the following sections we develop a particle system and a special-case function induction problem. Grammatical evolution is used to evolve particle behaviours in order to reduce a global entropy measure. We believe this type of evolutionary problem, while often difficult to formulate is of great importance. Much of the prominent challenges in science (ecology, immunology, neuroscience, economics) stems from the need to understand how *systems* function and how microscopic changes affect or are effected by order parameters.

These ideas are also important to evolutionary computing itself. For one, self-organizing dynamics can be used to build better algorithms. More generally, it is a reminder that all natural systems are complex and that nature's evolutionary search space is not the vast sea of variation it was once thought to be - much complexity is a natural consequence of self-organizing dynamics. The emergent order parameters that regulate 'social systems' are in effect an extended phenotype for the lower, non social scale of selection. As we consider what information is in the genome, we are reminded that much of the phenotype complexity may not be encoded. The following study is a good example of this.

Dynamical systems are interesting evolutionary problems. The paradoxical contradiction of the second law of thermodynamics seen in selforganizing systems has been explained in terms of a coupling between the macro level that hosts self-organization (where entropy is reduced) and the micro level (where entropy is increased through random processes) [Parunak and Brueckner, 2001]. The micro level is said to serve as an entropy sink permitting overall system entropy to increase while sequestering this increase from the interactions where self-organization is desired.

4.5.1 Dynamical Particle System

We have abstracted a Dynamical Particle System. Each system consists of an environment with objects or gases. We refer to animate objects as *particles*. A system of homogenous particles (animate) are given a number of behaviours. Each particle has a short memory which is a matrix of occurrences - each row corresponds to a particular piece of information. Particles can perceive gases and particles directly and probe their properties. However, memory stores occurrences only as real values e.g. 5 objects, 20,000 units of pheromone. Either memory or object properties at a given moment can be used in functions. These functions are defined by a grammar. Particles can move and act by physical manipulation of objects or by pheromone deposits.

It can be seen that our Dynamical Particle System corresponds closely to an *active cognitive transducer* [Wiedermann and Van Leeuwen, 1999] as described below. It is also consistent with the work of myrmecologists who observe that ants use functions of their recent, local experience in directing their actions [Wilson, 1971, Hölldobler and Wilson, 1990, Gordon, 2001]. These spatially extended systems which

communicate through their environment have significant computational abilities.

Wiedermann and Van Leeuwen's *cognitive transducer* differs from finite transducers or Turing machines. The former, while finite (and sometimes limited) in reactions, the order and intensity of reactions to input streams can vary in infinite numbers of ways. In this sense, while our artificial ants are genetically homogenous, heterogeneities do emerge within the colony. As cognitive transducers can influence their environment they are described as active transducers. Input streams can be causally linked to single or joint past reactions [Wiedermann and Van Leeuwen, 1999].

Cognitive transducers by definition are always on i.e. *infinity of operation* and are embodied in their environment within which interaction is describable i.e. *interactivity* [Wiedermann and Van Leeuwen, 1999]. Processing of environmental information is dynamic yet implicitly stateful. Multiple inputs may be processed in parallel.

A community of active cognitive transducers is a time varying set of devices which, at each moment, consists of finite number of homogenous units sharing the same environment. All members jointly play the role of an 'advice'. This advice keeps emerging online, incrementally and is blind, possessing as a whole no specific information processing, or computational intention [Wiedermann and Van Leeuwen, 1999]. *Stigmergy* meaning *incite to work* [Wilson, 1971] is used in the context of eusocial insects to describe communication through the environment. This allows for plebiscitary activity, which supports Wiedermann and Van Leeuwen's non-computable advice.

4.5.2 Swarm Evolution

We consider three primary concerns in evolving dynamical systems. The rich XML representation method of XMLGE and the ability to distribute processing may be

useful abstractions for addressing some of these issues.

Tractability The problem tends to be intractable in terms of evolutionary time and search. Emergent computation performed by Cellular Automata [Wolfram, 1983] is determined by its overall space-time behavior [Crutchfield et al., 2003]. This is also true of swarms. Individual experiments evaluated by the evolutionary system may run for several minutes. The problem contains non-linearities and therefore mirrors a rugged search space. On a desktop PC it could take weeks to evolve a solution, if a solution is to be found at all.

Representation As an automatic programming problem we must consider how to encode the problem, what terminals to use and how to map the genetic material onto the swarm program. This representation should contribute to a Language that allows us to relate microscopic and macroscopic phenomena [Kubrik, 2003].

Fitness Evaluation A Method to determine the fitness of the swarm can be difficult to produce for dynamical systems problems [Williams, 2002]. Methods to identify patterns in space or time are required. For example Spatial Entropy values [Bonabeau et al., 1999], Hough Transforms [Williams, 2002] and techniques from computational mechanics [Crutchfield et al., 2003] each describe patterns and hence fitness of dynamical pattern-forming systems. This is not to say there is a one-size-fits-all fitness evaluation method for complex systems but that dynamical systems might be evaluated based on the recognition of patterns.

Different functions of information effect the swarm's ability to model its environment although it is difficult to determine appropriate information and functions. Each

dynamical environment has its own properties and information - where information is the meaning of events. In complex environments there are consistent patterns in space and time. If an entity can recognise relevant patterns, it has the potential to be adaptive and anticipatory [Rosen, 1985]. The information-theoretic notion of relevant information and the potential of taking an agent-centered theoretic approach to the design of distributed adaptive systems is discussed in [Nehaniv et al., 2002].

Every creature in nature survives by modeling its environment as it senses it and through a transducer system makes a response that is fit for that environment. Using GE, we evolve templates for simple transducers that describe how environmental information should be modeled by an ant to produce a response that favours the swarm's fitness.

Grammar-based Genetic Programming approaches such as GE are a powerful means to describe legal interpretations of such terminal information yet still allow the open-ended evolution of novel solutions. GE's distinction between genotype and phenotype aids in representation of the problem and provides a substrate for processes that will regulate swarm construction.

4.5.3 Experimental Setup

We describe a multi-agent or swarm simulation model where information processing ants cooperate to solve an abstract clustering problem. Ants cluster identical objects which is a distinct problem from sorting (where a similarity measure is used to sort like objects). The ant's world is a square toroidal grid of 177*177 pixels (similar area to the circular world used in [Bonabeau et al., 1999]) and uses a Moore neighbourhood. We use the Repast simulation tool and the Colt math library [Repast, 2004].

Ants will move one pixel per time step. Ants can move in random directions based on a Gaussian probability distribution centered around the forward direction. A 'left antenna' at the north-west Moore pixel surrounding the ant and a 'right antenna' at the north-east are used to sense the concentration gradient. An ant will move deterministically in the direction of highest concentration or will continue moving straight if there is no difference in concentrations. These design choices are based on natural phenomena [Wilson, 1971, Hölldobler and Wilson, 1990]. An ant will deposit a single pheromone signal of a certain concentration as it moves. This pheromone signal diffuses and evaporates at a constant rate. Diffusion and evaporation is implemented by the repast simulation tool [Repast, 2004]. An ant may pickup or deposit objects in it's environment. Ants have a 'browsing' function whereby objects and other ants are observed and 'remembered'. The ant has a limited memory map where each object type encounter over a certain period is stored.

Ants use non-deterministic threshold functions [Bonabeau et al., 1999] to determine action. Equations 1 and 2 give the probability to pick up and drop an object respectively. $k1$ is the threshold value for picking up an object and $k2$ is the threshold value for dropping an object. f is the stimulus that the ant perceives - here it is the fraction of objects perceived over some period i.e. the ant's memory length.

$$\rho_p = \left(\frac{k1}{k1 + f}\right)^2 \quad (4.1)$$

$$\rho_d = \left(\frac{k2}{k2 + f}\right)^2 \quad (4.2)$$

The parameters used in the simulation model are shown in Table 4.1. We experimentally decided the simulation time. This time is a short duration that can be used

Parameter	Value
World Dimensions	177 X 177
Ant Count	400
Object Count	1000
Memory Length	50
Decay factor	80 timesteps
Evaporation Rate	0.87
Diffusion Rate	0.42
Simulation Duration	5000 timesteps

Table 4.1: Swarm simulation parameters.

to effectively evaluate the clustering performance. Evaporation and diffusion rates were chosen to represent 'recruitment' signals with medium spatial effect and short temporal effect. Other parameters were chosen over a number of trials.

In the clustering task, ants have a number of behaviours. There is a behaviour for picking up objects, dropping objects, sensing stimulus, dropping pheromone and depositing 'pheromone traces' on objects. Each of these has a corresponding *gene*, which is the evolved template mentioned above. These templates are equivalent to GP S-Expressions, arranged in sequences that make up a complex of S-Expressions or genes. These genes regulate ant responses using environmental information as inputs. Each ant individual is encoded by a complex of genes mapped to behaviours. These genes regulate the values used for k_1 , k_2 and f in equations 1 and 2. When objects are deposited, they emit pheromone traces of concentration specified by the appropriate gene. The concentration of pheromone used in ant trails is also regulated by a gene. Pheromone is emitted for a period of time specified by the decay factor in Table 4.1. GE uses the following grammar to map a genome to an S-Expression, or a complex of S-Expressions called GeneComplex. Start Symbol, S, Non-terminals, N, Production rules, P, and Terminals, T are shown below. The terminal values in `<var>`

correspond intuitively to information in the ants world. Encounters with ants and objects, or ants in a 'working state' are remembered in a matrix (for example over 50 timesteps). Parameters are retrieved from the memory depending on the Gene Complex function. `current_pherenome` is the value of pheromone surrounding the ants at the given moment in time while `pheromone` is the total pheromone occurrence in the matrix.

```
<GeneComplex> ::= <expr> <expr> <expr> <expr>
```

```
                <expr> (0)
```

```
<expr> ::= <expr> <op> <expr> (0)
```

```
        | <var> (1)
```

```
<op> ::= + (0)
```

```
        | - (1)
```

```
        | / (2)
```

```
        | * (3)
```

```
<var> ::= 10 (0)
```

```
        | 100 (1)
```

```
        | ants (2)
```

```
        | working_ants (3)
```

```
        | current_pheromone (4)
```

```
        | pheromone (5)
```

```
        | objects (6)
```

Notice the terminal symbols of the grammar correspond to environmental information. Production rule 1 describes a complex of 5 X-Expressions. These expressions are mapped onto ant behaviours. A complex of X-Expressions are mapped in an ar-

bitrary but consistent manner from a single genome using the grammar above. This complex is passed to a swarm simulation and used to construct genes for ants using a sequential mapping of S-Expressions onto the behaviour to be encoded. During the simulation at each time step, the ant senses its environment and passes in a table of all sensed variables corresponding to terminal symbols. Terminal symbols represent the number of ants, working ants, objects etc. (see grammar) encountered at each time step. Each behaviour class then computes a response value based on the environmental information supplied. These computed values are used for example in equations 1 and 2 to decide when to pick up and drop objects. In this way, each S-expression determines the value for a gene or parameter that determines the expression of a behaviour based on information inputs.

Initially objects are distributed randomly. As ants *work*, objects are picked up and deposited around the grid. After the specified experiment length, the fitness of the swarm is computed and returned to the evolutionary engine. As a measure for clustering performance, a spatial entropy value [Gutowiz, 1993, Bonabeau et al., 1999] is used, calculated over the lattice which is divided into a number of grids. The equation below gives the spatial entropy E , at a certain grid scale, s . P is the fraction of objects in one grid of total objects. In our experiments $s = 6$, there are 36 grids.

$$E_s = - \sum_{I \in s\text{-patches}} P_I \log P_I \quad (4.3)$$

Spatial Entropy [Gutowiz, 1993] is a macroscopic measure that corresponds to the individual ant's microscopic goals. As the ants work, spatial entropy values tend to decrease as the world becomes more ordered. In time-dependant problems it is advantageous to have fitness evaluation measures where the fitness value tends towards

Parameter	Value
Mutation rate	.1
Crossover Rate	.7
Population size	200
Generations	5
Fitness Measure	Spatial Entropy

Table 4.2: GE parameters.

the optimum. We run simulations for 5,000 time steps. However, this time reasonably approximates clustering behaviour over 50,000 time steps.

It is the nature of the swarm clustering task that it should be easy to find a good solution (approximate ratio between stimulus and threshold values) while it can be difficult to identify the features of the multi-parameter problem. Consequently we focus on showing evidence of progressive search rather than on finding an optimal solution. GE genome individuals are evaluated only once (assuming they are not mutated). We use a variable-length generational GA with tournament selection, one-point crossover and integer mutation (as opposed to bit mutation). See evolutionary parameters Table 4.2.

4.5.4 A Case for Remote Evaluation

In the previous section, we described how a web service hosts a GE experiment while clients request genomes, evaluate them and send back results. Clients can request XML files or dlls on demand. In this section we describe how the dynamical system problem could be used in the distributed system.

For evolving dynamical systems, a simulator and an ant gene class constitute the fitness environment. These are packaged in a dll. The XML files required at the client are the grammar (an XML schema file), the GE Mapping stylesheet (XSLT file) and

the problem specification file. Documents can be validated by XML Schema files also (for example a schema ensures the specification is valid).

When the client starts, it will ask for a Job providing the specification ID it is currently using (if any). The client can request one or more jobs which the server returns as a list of genomes. If the specification ID on the server is different to the one supplied by the client, new project files are sent to the client.

The client will process each genome by applying the GE mapping stylesheet to generate an X-Expression (phenotype derivation tree), constructing the fitness environment to evaluate the X-Expression and returning the result. The fitness environment is a simulator that constructs ants using the X-Expressions to describe transducers that process local information and determine behaviours. The simulation is run for a specified period of time and the X-Expression is used to regulate ant behavior in each time step. For the clustering-based problem addressed here, fitness is a spatial entropy value. At the end of the simulation, the fitness is tagged to the X-Expression and returned to the server.

The simulation is performed on the client. The optimal speed of the XMLGE distributed system is to have as many clients as there are individuals in the GE population. In this case each generation will take approximately as long as the simulation takes. Clients will send X-Expressions with tagged fitness image snapshots (e.g. Scalable Vector Graphics (SVG) or string encoded image files) back to the server. A fit X-Expression i.e. an Ant's GeneComplex may be used to re-construct a simulation at the server.

The server will wait until it has the correct amount of fitness assignments for the population and apply generational crossover. The clients will continue to request jobs at this time but the server will respond that there are no jobs at that time.

Runs	1	2	3	4	5
GE(mean)	1.79	1.79	1.70	1.43	1.70
GE(sd)	0.47	0.11	0.11	0.35	0.33
Rand(mean)	2.02	1.99	2.42	1.65	1.98
Rand(sd)	0.60	0.45	0.49	0.47	0.15

Table 4.3: Comparison between random run (Rand) and Grammatical Evolution (GE) showing mean Spatial Entropy results and standard deviation. (Initial entropy values approx. 2.7 on average). The standard deviation values for the GE runs are generally lower than the random case.

When crossover is finished, a new task set is generated for the new population and client processing will continue. XMLGE experiments are driven by one parameter file. Custom parameters as used by custom fitness environments such as the simulator are included in the parameter file along with evolutionary parameters. The parameter file is consistent and is constrained by an XML schema. In this way a simulation, as an evolutionary algorithm fitness function, can be controlled and monitored remotely.

4.5.5 Results

Over many clustering experiments, individual spatial entropy values ranged from average 2.7 (worst) to 0.9 (best) using given parameters. Results showing best solutions found in each generation for both GE search and random search are shown in Table 4.3. Random search generates and evaluates a random population on each generation. Due to computational time requirements, we have only taken 5 samples of each. Although not discernable from the tables above, GE did find the most favourable average spatial entropy value of 1.059 although only marginally better than the best value in the random search which was 1.127. However GE made improvements over successive generations in most samples and the standard deviation values were predominately lower.

All experiments used the same agent models and differed only in the information-processing templates used. Clustering models in the literature show consistent formation of several small clusters, gradually becoming three or four large clusters. We observed these similar patterns but also observed patterns where objects seemed to be 'swept' into regions of the ant's world. The regions first contained sparse clusters that were gradually swept into dense compact clusters. We observed clusters that formed stripe-like patterns in addition to 'spots'.

4.5.6 Discussion

The purpose of this section was to demonstrate evolutionary pattern-forming swarms using Grammatical Evolution, with the result that the ant colony successfully evolved templates that exhibited clustering behaviour based on a spatial entropy measure. GE provides independence between the evolutionary aspects and the program representation. Many complex systems can be considered in terms of swarms of information processing particles. The use of grammars provides a means to describe transducers for information processing in complex system nodes. Grammars provide a powerful means to describe legal interpretations of information yet still allow the open-ended evolution of novel solutions.

Fitness evaluation methods that evaluate patterns are an interesting way to evolve dynamical systems. The choice of fitness function is important as depending on how well it approximates or anticipates performance of the dynamical system in the early stages of a simulation, one can use shorter simulation times and less runs in the evolutionary stages.

The use of templates in a homogenous colony leads to behaviorally heterogenous ants based on their environmental information context. In a sense it also realises a type of ontogeny in that over time, the ant has features that may have variable fitness. This is an important aspect to exploit given the temporal development of simulations and makes the colony more adaptive. We have observed this through comparisons between models using static parameters and those using template-based parameters.

In all areas of complex system research, a bridge of understanding between microscopic and macroscopic phenomena is required. Some research perspectives focus more on one or the other of these suffering the critique of others. Templates represent for us loci at which to study this connection. The evolutionary search implicitly identifies these templates as features of complex systems where for some global task, we can see how individuals process information.

4.6 Conclusions

We have demonstrated the use of genetic operators such as crossover and an implementation of the GE mapping in XMLGE. These use XSLT, which is a declarative stylesheet language. We have shown how genetic material can be represented in XML.

The ability to first evolve structures in data form which can be used to reliably drive software processes is good software practice. XML is used extensively for a similar purpose with manually programmed software. It is used to describe user interface layouts, application configurations and application messaging.

XMLGE is a candidate *program DNA*. We suggest that such ideas should be developed to consider artificial DNA as *active data* rather than the most abstracted and simple binary string. We have used reflection to react at runtime to data and

fire events in the presence of certain information in a deterministic fashion. Future work could create more intelligent, possibly non-deterministic environments using rich representations.

This chapter has also demonstrated the evolution of dynamical systems as a special case of evolution. Studies in gene networks [Kauffman, 1993] and cell systems [Furusawa and Kaneko, 2002] suggest that the complexity in life emerges spontaneously and naturally as a consequence of system dynamics. Whether the use of complex interactions and state are used in the phenotype as we have done, in the genotype-phenotype mapping as done in certain embryogenies or whether it is in the development machine itself, the genome can be used in more rich ways than is the case with deterministic, ballistic mappings.

A lesson from self-organizing systems is that we must enrich environments so as to create regularities that an evolutionary system can exploit. This is a special type of compression or *implication* and a possible route to the evolution of artificial complexity. Evolutionary systems are inherently suited to exploit regularities in an environment while traditionally the only feedback from the environment is fitness based.

In the remainder of this thesis we exploit the information that is tagged on genomes or X-Expressions during the GE mapping. We evaluate the usefulness of this approach.

Chapter 5

Coevolving Antibodies for Selfish Genes

Everything else is without importance from the Creator's point of view, and is only a play on permutations and combinations within a general program, which is not a prophetic anticipation of the future but merely sets the limits of possibilities within which all power of decision has been left to chance.

- Milan Kundera 'Immortality'

We consider an evolutionary algorithm to be a cognitive or perceptual system. These terms do not differ greatly from how Holland [Holland, 1992] used the term *adaptive* meaning *to fit to*. In each case a system models or constructs an image of its environment and reacts differentially in the context of environmental stimuli or information. An important property of cognitive systems is their anticipatory nature, where past experience can be used degenerately to model new experience.

These systems often make 'leaps of imagination' using *analogy* in order to innovate. Understanding the *physics of innovation* is an important objective in evolutionary algorithm research [Goldberg, 2002].

It can be seen that a cognitive system operates dual processes of feature detection and feature integration [Hershberg and Efroni, 2001]. An anticipatory system must have the *physical* means to detect regularities in its environment and these means must be *biased* towards useful examples [Hershberg and Ninio, 2004]. In particular we use the term *perceptual* to emphasize a cognitive system's *selective* ability to perceive relevant information. Like all perceptual systems so too must an evolutionary system detect useful features and integrate them into global images. This is termed the linkage learning problem within the EC community and requires that linkage be learned quicker than alleles are crystallized. We note that most variants of the *competent GAs* use explicit dual processes including messy and fast messy GAs, gene expression messy GAs and BOA [Goldberg, 2002] and suggest also that if not explicit, all use implicit dual processes. Thus, we stress that one of the simple GA's main shortcomings is its unfortunate coupling of feature detection and feature integration processes. This is an important theme in this chapter.

In this chapter, queries are evolved which mirror primary structures in evolutionary algorithms and extract useful features. These features are then combined into higher-order solutions in a second process. We refer to this type of algorithm as a *dual process* evolutionary algorithm and point out that similar dynamics and information processing could be achieved in different ways. We believe an important contribution of this approach which uses XMLGE, is the ability to evolve complementary structures for any primary structure. These 'complements' form the basis of a distributed perceptual system. In the chapter title we have used the term *selfish*

gene taken from [Dawkins, 1989]. This simply emphasizes the *selfish* or *independent* definition of tightly linked features outside of the individual. Specifically we develop a model where complements are coevolved to detect features or genes. Our convolutionary model is based on the immune system and referred to as *antibody coevolution* [Amarteifio and O’Neill, 2005].

5.1 Principles

5.1.1 An Immune System Overview

The model-building evolutionary algorithm literature refers to the problem of identifying features as the *linkage learning* problem at both inter-gene and intra-gene levels. First suggested by Holland [Holland, 1992], later work addressed this problem through alternate encodings [D. E. Goldberg and Korb, 1989] and stochastic model building [Pelikan and Mühlenbein, 1998, Pelikan et al., 1999]. Learning inter-gene linkage refers to the adaptive *definition* and *propagation* of strongly linked alleles, which are essentially features of a cognitive system’s world.

The question of how a system can learn fundamental features that are later recombined into high-order features is not confined to genetics. This problem is common to any cognitive system. In any such system one can take the perspective that there are two simultaneous, non-discrete high-level processes. One that identifies features as elements and another that forms concepts or learns through integration and diversification of features. The immune system can be seen as such a cognitive system [Hershberg and Efroni, 2001]. If we are to engineer these processes in evolutionary computation we might see this as a co-evolutionary problem as both systems are tem-

porally mingled. The immune system provides an example of a co-evolving feature detector system. An antibody library is continuously evolving in the immune system. It evolves continuously and simultaneously with respect to the recombination process in which antigens are suppressed by dynamic networks of library samples.

The immune system is an autonomous, distributed, adaptive system composed of trillions of immune cells. The functions of the immune system are accomplished by combined action of many of these entities. The single function of the immune system is to maintain homeostasis in the body. This is achieved by detecting and eliminating harmful non-self pathogens that would lead to the eventual demise of the host [Hofmeyr, 1999]. This must be done while not harming self. The immune system consists of layered protection from the skin, to physiological conditions such as temperature or pH in and around the skin, to the innate immune system and the adaptive immune system.

The innate immune system is a built-in defence mechanism from birth giving the adaptive immune system the chance to build up. The adaptive response consists of the primary response which learns and memorises new attacks. This response is both slowly initiated and slow to complete. A secondary response responds to known attacks that the primary response has remembered. This memory lasts up to the lifetime of the organism. It is also associative in the sense that it can recognise similar attacks. This is a property of many distributed representations. The adaptive immune response consists primarily of lymphocytes. These white blood cells move around the body seeking out pathogens. There are trillions of these lymphocytes each moving about the body, interacting with each other and detecting pathogens through simple localized rules [Cohen, 2000]. The stage when the immune system chooses the appropriate response is called the effector choice stage. The correct cells must be

activated to respond appropriately.

Pathogen recognition in the immune system involves establishing chemical bonds between lymphocyte receptors (antibodies) and the surfaces of pathogen peptides (protein fragment). These surfaces are called epitopes. The strength of these protein bonds is termed affinity. Receptors are specific in the sense that they bind tightly onto a few similar epitope structures or patterns. Each lymphocyte has in the order of 10^5 identical receptors, making lymphocytes specific to certain pathogens. Lymphocytes will only be activated if the number of receptors bound exceeds some threshold. Therefore the affinity of receptors to epitopes must be high and there must be a sufficient amount of local pathogens. Pathogens are complex structures and they consist of a variety of epitope compositions. There is a many to one relationship between lymphocytes and pathogens. There are significantly more pathogen proteins than the proteins the immune system has available (at any one time) for recognition, even after resorting to DNA recombination. It has been suggested that there is a lymphocyte turnover rate of about 10^7 a day. The significance of this is that over a period of about 10 days, a completely new repertoire of lymphocytes has been produced giving the immune system greater defences over time [Hofmeyr, 2001]. Antibodies have variable and constant regions. The constant region is used to communicate with other immune cells and thus regulates responses. The variable region is randomly generated and binds to specific pathogens.

The adaptive process of antibody production involves a Darwinian cycle of activation, proliferation and differentiation of B cells [Hofmeyr, 1999]. B cells are a class of lymphocytes that learn and remember pathogens, 'B' indicating bone marrow where they originate. Another class of adaptive lymphocytes are T cells which originate in the thymus. When B cells are activated they migrate to lymph nodes throughout

the body where they undergo somatic hypermutation. This is a very high rate of mutation and there is a high probability of different or new B cells. At this point B cells are exposed to pathogens. If they do not bind they die after a short time. If they do bind they differentiate into plasma B cells or memory B cells. This continuous cycle ensures only the fittest cells, those with the highest affinity, survive over time. B cells must compete for available pathogens which imposes selection pressure.

As T cells originate and mature in the thymus, they are exposed to most self cells. If they are activated by self cells they are censored. This negative selection process is called central tolerance and is a first step towards self tolerance, which is vital in preventing autoimmunity. However central tolerance is only a half-step towards self tolerance as there may be a very large amount of autoreactive B cells (that have recently undergone hypermutation). Distributed tolerance is achieved by only allowing B cells to be activated when they have been stimulated by both pathogens and T cells (T helper cells/Th cells). In the event that B cells are stimulated by only pathogens, they die.

Noest describes the designer lymphocyte, designed without regard for biological constraints but based on the theory of statistical optimal detection [Noest, 2001]. He bases his description on detection theory (Neyman-Pearson); in this approach it is desired to maximize the probability of choosing a hypothesis (H1) when it is true (detection), while not exceeding a fixed probability of choosing H1 when it is false (false alarm). This binary decision maker can be generalized for multiple hypothesis testing for multiple, distinct epitope encounters. In consequence Noest points out the need for specificity to reduce interference and following this, diversity to cover the problem space. Noest observes that natural lymphocytes behaviour is consistent with a mathematically derived designer lymphocyte [Noest, 2001]. However, it may

be the case that no individual lymphocyte is so discerning but rather a distributed response alone ensure robustness.

5.1.2 The Emergence of Meaning

The primary task of any adaptive systems is to model the environment by creating functional representations of it. The brain can be classified as an adaptive system. It responds to situations in an adaptive manner and involves understating and responding to new situations in relation to what is already known. Similarly, in the immune system, the response must mirror the stimulus. The immune system does not recognize pathogens as they exist but aspects of the pathogen whose context is deconstructed according to immune system rules. The response to the deconstructed antigen is not the sum of the responses of each individual agent; the immune response is the cooperative outcome of the mutual interactions of the different agents and their diverse perceptions [Cohen, 2001].

This research uses rich representations to allow for the evolution of meaning [Hershberg, 2003]. An adaptive system must be seen as part of an environment. This environment is expected to contain regularities and the system has certain sensitivity to such regularities. Meaning evolves as a relationship between an adaptive entity and its environment.

The *exemplar learning* model [Hershberg and Ninio, 2004] describes how cognitive systems, the immune system in particular, determine the rules or general properties or *useful examples* of their environment*. They suggest a class of examples that are ubiquitously encountered (ensuring reinforcement) that are almost meaningless in their generality. For example in infants learning to speak, parents use a high

*[Wu and Stringer, 2002] developed a chunking genetic algorithm based on similar observations.

frequency of verbs that are almost empty semantically - which are the first verbs learned by infants. It is widely accepted that the statistical shape of language is such that a relatively small subset of words are highly frequent while the rest are used at lower frequency and these words might have an important role in syntactic development [Zipf, 1965].

A central component of contemporary immune systems understanding is the clonal selection theory. This states that the immune system relies on the existence of receptors that can bind preferentially to pathogens rather than self cells. In contradiction the cognitive immune system viewpoint [Cohen, 2000] observes that 1) due to common ancestry there exists an essential similarity between cellular biology and ourselves and 2) A necessary benign self-affinity is known to exist in healthy immune receptor repertoires.

Based on the cognitive perspective, 'useful examples' in the immune system are expected to be; 1) Centrally important and therefore should not have changed much though time, 2) to be common to both self and pathogens and 3) should be expressed in times of stress. A possible candidate for such structures are *housekeeping proteins*, which fulfill these criteria [Hershberg and Efroni, 2001]. These constitute an 'achieved set' which describes generalities - a repertoire built to react degenerately at a median level of affinity to a few self antigens. This repertoire will not change much. Then throughout lifetime, a refitting mechanism is used for refitting to specific encounters allowing for the deconstruction and reconstruction of the immune 'image'. The second non-self-reactive receptors evolve and adapt over time as the immune system encounters antigens. As such this observes that the immune system like any perceptual system uses the self as a background signal, which it ignores, while reacting to changes that the background emphasizes [Hershberg et al., 2003].

5.1.3 Artificial Immune Systems

The immune system at a teleological scale protects the host by distinguishing harmful entities from non-harmful entities. This has inspired application to security models such as network monitoring. Work on this line follows from Hofmeyr's important thesis on the subject [Hofmeyr, 1999]. Other metaphors inspire work in other areas such as scheduling and data clustering [Hart, 2002]. In the last few years, many artificial immune systems (AIS) are emerging as the immune system's information processing capabilities are becoming better understood within the EC community. This is perhaps best seen in the last three ICARIS gatherings. See for example [Nicosia et al., 2004]

The current work follows in the tradition of Artificial Immune Systems in recognising properties such as feature detection, diversity maintenance and memory and perception. Both Kim [Kim and Bentley, 2001] and as mentioned, [Hofmeyr, 1999], exploit the security analogy in intrusion detection methods. Similar to the current work, Hightower et al. explore the use of gene libraries and complementary matching in a genetic algorithm while contrasting *potential* and *expressed* genes [Hightower et al., 1995]. Questions about efficient pathogen space coverage which is equivalent to strategic *sensor evolution* in perceptual systems, have also been addressed [Hightower et al., 1995, Oprea, 1999] while [Oprea, 1999] puts some emphasis on germline diversity. In a recent application to GP, [Hasegawa and Iba, 2004] exploit an antibody library to tackle multimodal problems. Artificial immune systems seems well positioned to exploit this *duality*, which may well be one of the important properties sought for competent GAs.

Of particular interest is work in [Hightower et al., 1995], which best supports our motivations. The antibody analogy is used to model a lock and key mechanism. Bit

strings are used to represent both the genotype libraries and the antibody molecules of the phenotype [Hightower et al., 1995]. The number of bitwise complementary matches are used as a measure of affinity between complementary structures. Fitness of an individual (collection of gene segments) is determined by how well it recognizes a set of antigens overall.

5.1.4 Dual Processes

Our work considers the perspective of the cognitive immune system [Cohen, 2000, Hershberg and Efroni, 2001, Hershberg and Ninio, 2004]. The evolutionary model is developed based on the immune system's process of antibody production. Antibody co-evolves with antigen to be retained subject to co-affinity. Affinity depends on complementary shape and electrical charge matching. In the natural immune system, an immune regulatory network subsequently emerges from a library of fit antibodies as selected based on affinity.

In order to adaptively detect and integrate features or concepts, natural intelligent systems seem to exploit distinct, simultaneous phases [Hershberg and Efroni, 2001]. These are referred to in this article simply as *dual processes*.

While the selection of antibodies and the emergence of an immune network can be seen as two independent processes, this is an idealization; biological processes can be broken down into many sub processes. It is well understood that biological processes are spatially and structurally distributed without central control. However, it is important to consider, and distinguish, temporal distribution of processing. [Butcher et al., 2001] gives a relevant example in the context of the immune system where multistep navigation leads to combinatorial targeting. In this view, immune

system cells need to make simple yes-no decisions at different states, gradually leading to the appropriate action. This can clearly be appreciated from an information theoretic perspective.

[De Jong and Oates, 2002, De Jong, 2003], discuss the use of Evolutionary Multi-Objective Optimization to make informed decisions about the suitability of features; While this may be an effective method, considering biological principles we suggest that a dual process evolutionary algorithm can operate distinct selection phases. The first phase; a crude selection of general features the second, a simple or informed recombination event that chooses from a library of general features.

In our abstract model, complementary structures can be evolved to match phenotype fragments and hence suggest features based on the binding. A library of these features are maintained based on a number of criteria. A second process must integrate these features. Emphasis is put on isolating the phases of feature detection and feature recombination. These processes can be seen as a set of independent selection phases.

5.2 Methods

The methods described below exploit rich representations in XMLGE. The XML implementation allows a feature detection mechanism at a high level of representation. Features as used in XMLGE are described in the following subsection. A method for evolving antibodies which is more like somatic hypermutation than crossover is used as well as a *chaining operator* for recombination of multiple features.

5.2.1 Using Grammatical Evolution for Feature Detection

Representations and Anti-Representations

In XMLGE, XML-based genomes are mapped to XML-based phenotype tree structures or expression trees. XML Expression trees (X-Expressions) can encode information captured during the mapping process. The result is a rich XML tree that describes the phenotype including information from the grammar context and codon that generated it. A reverse phenotype-genotype feedback may inform the genome of the context it is used in. With a hint of the Lamarckian, this simply represents both 1) regularities that would be implicitly expected as natural genomes and interpreters co-evolve and 2) the more dynamic and information-rich process of mapping the natural phenotype.

Using XML-based rich representations a novel feature detection method can be implemented. The XPath language is a powerful XML query language that can select nodesets on XML trees (in the current EA application termed X-Expressions). Queries can be seen as phenotype structure complements. Using a translation grammar, a *mirror* grammar can be generated from any primary grammar. This secondary grammar is used to generate XPath queries that bind to the primary expressions. Thus we explicitly recognize the importance of *complementary structures* which are pervasive in biological systems and can be used to construct distributed perceptual systems. While previous work [Hightower et al., 1995] used similar immune system inspired ideas at a binary string level, grammars and antigrammars suggest an approach at a higher level of representation.

An algorithm that coevolves complementary structures may be important for bootstrapping 'perceptual' systems because the recognition process is inherently im-

plicit. Features may be associative making recognition pleiotropic and redundant at higher scales. We refer to this general principle as *complement coevolution* in emphasis of the use of the associative 'lock and key' mechanism. However to avoid ambiguity, we refer to this in the immune system context as as antibody coevolution. Closer to ideas in EC, this type of distributed perception sees through the simple and discrete fitness feedback and can preserve features of variable interest. This property has already been noted by [Hightower et al., 1995] as discussed above yet here we can deal with the issue at higher level of representation.

Generating XPath

The XPath queries are generated to mirror the primary phenotype structure. The technique is expected to be domain-independent. A mapping grammar maps from the primary grammar to an XPath grammar. While many mappings are possible and may exploit variable features from the standard W3C XPath grammar [W3C, 2005d], we use a rudimentary mapping given below. Note VALID-CONSTRAINT, VALID-CHILD-FILTER and X are expanded according to the production choices in the primary grammar. <NTF>, <TCF> and <TVC> denote non-terminal filter, terminal constraint/filter and terminal value constraint respectively.

```
<NTF> ::=
    <TYPE-NAME>
    |<TYPE-NAME> [<VALID-CONSTRAINT>]*
    |<TYPE-NAME> / <NTF>
    |<NTF> / <NTF>
```

```
<TCF> ::=
```

```

(<TCF> <BOOL> <TCF>)
| (not (<TCF>))
| <VALID-CHILD-FILTER>*

<TVC> ::=
<TYPE-NAME> [(<TVC> <BOOL><TVC>)]
| <TYPE-NAME> [not (<TVC>)]
| <TYPE-NAME> [text()= <X>* ]

```

XPath queries are generated from the grammar by mapping the genomes using the standard Grammatical Evolution mapping. The fitness of an XPath query is the average fitness of all phenotype structures it binds to. In theory, the fitter the XPath library, the more accurately it detects useful features to be used for composition. A population of 100 genomes is used in these experiments for the XPath population.

Using XML with Grammatical Evolution, a population of Expression tree phenotypes are co-evolved with XPath expressions. Expression trees are represented as detailed XML trees with mapping details encoded on each node. Information is purely *local* to the mapping event such as the grammar context used in the mapping or the codon and its position that was used to produce a given part of the tree.

Evolved XPath queries will match non-terminal and terminal regions with variable specificity using boolean filters. The example below shows an XPath query with *isotype regions*. Isotype regions are chosen downstream from a random read point. The term isotype comes from the natural antibody structure although the analogy is not stressed. Here, the tail and leaf of an XPath lymphocyte regulates the context and specificity of a feature. In the hypermutation based operators an emphasis can be put on modularity and hierarchy where genome regions that encode loosely defined

'isotype regions' might be identified.

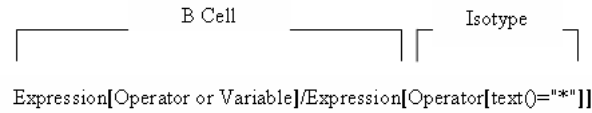


Figure 5.1: Example Lymphocyte

Features in Grammatical Evolution

Features are groups of alleles with tight linkage, which together contribute to some phenotype function. Feature detection is the first of the processes in the dual process model. Every cognitive system must have the *physical means* to selectively attend to features. Antibodies for example match reciprocal antigen ligands through physical and chemical affinity.

In XMLGE, features can be identified at the phenotype using XPath queries i.e. regular expressions. XPath queries are used to select sub trees on X-Expressions. The features themselves are genome sub sequences that were recorded on the phenotype during the GE mapping event. Genome sub sequences are stripped from the phenotype when the XPath queries match phenotype X-Expressions. In the example below, the XPath expression marked (1) will match the expression marked (2) because the expression has an operator child with a '+' value and the expression does **not** have a variable with a 'Y' value; The codon sequences or context sequences can then be stripped from the tree. In this study, sequences such as (1/2),(3/3),(2/4),(1/3) are considered degenerate features or modules. This values which can be seen in the expression below simply state that 1st of 2 (1/2) production rules were chosen. This information is important because it captures the essence of the mod rule. An arbitrary codon integer value can be used to satisfy that choice such that the codon

value $x \bmod 2$ equals one. Thus, when ultimately used, these context sequences are converted into appropriate (arbitrary) codon sequences.

(1)

```
/Expression[Operator[text()='+' ] and not(Variable[text()='Y'])]
```

(2)

```
<Expression ref="123" fitness="12.89">
  <Expression codon="240" context="1/2">
    <Variable codon="12" context="3/3">X</Variable>
    <Operator codon="126" context="2/4"/>+</Operator>
    <Variable codon="79" context="1/3">1.0</Variable>
  </Expression>
</Expression>
```

In this study, the GE mapping is exploited in the definition of features. The first stage of selection is to identify stable sequences. Some context sequences will simply not exist for a given grammar. This is analogous to the basic physical and chemical laws of nature that make some compositions impossible. This hints at an important theme; the use of available information in rich representations.[†] Stable sequences are identified on stable phenotypes but are retrieved as context sequences and then stored as degenerate codon sequences. Thus there are a number selection and filtration events. Ultimately it is desirable to have a set of encapsulated features yet allow exploration. As such adaptation should be achieved with the use of degenerate sequences that have 'proved' themselves in historic contexts but are sensitive to variation. From the discussion of grammars and antigrammars and the discussion of

[†]It is engineering practice to abstract as much as possible and allow the same abstraction to work in all suitable contexts. This is limiting for adaptive system design. Here we suggest exploiting information that is inherent to a given algorithm, in this case GE, in order to detect regularities.

degenerate codon sequences, it becomes clear how XMLGE constructs a perceptual system as a *distributed library of degenerate complements*. The memory of this system i.e. the model of degenerate codon sequences is sampled to construct new individuals.

5.2.2 Immune System Model

The Binding Event

XPath queries are somewhat analogous to lymphocytes and will be referred to as such.

In each generation, given a population of lymphocytes and a population of primary expressions, each lymphocytes will *bind* to each expression with a probability factor used simply to provide a degree of binding uncertainty (in these experiments this is 0.7). Lymphocytes traverse the tree depth first with the same probability of binding at each *site*. If another lymphocyte has already bound to a site, lymphocytes will compete for the site. In the current implementation the winner is randomly chosen. However there are other possible strategies such as *shortest query wins* or *most bindings wins*. The product of the binding event is what is referred to as an antibody secretion and is stored in a library of sequences.

This library records sequences, grouped by common grammar context sequences or production sequences. A production sequence is different to a codon sequence. As explained in chapter 2, when the genome is mapped sequentially, the grammar context evolves. Two codon sequences such as 21,19,13 and 21,11,38 may have the same phenotype impact. Given production rules choices from the grammar, the mod rule in both cases will, for example, chose the first of five choices, followed by the first of two, followed by the second of four. There can be any number of *instances*

for a given production sequence grouped together. These instances are grouped and stored with details from the mapping event. Note that in GE, the ability to map many codon sequences to the same phenotype provides a degree of neutrality at the syntactic level. The sequences retain information about positional context, the host they come from, the fitness of the host they came from, the grammar context they were used in and the XPath query that 'found' them.

During the binding, in addition to preserving sequences, the lymphocyte's details will be updated with the fitness of the Expression tree it bound to and its binding count will be incremented. The average fitness of hosts a lymphocyte binds to determines its own fitness.

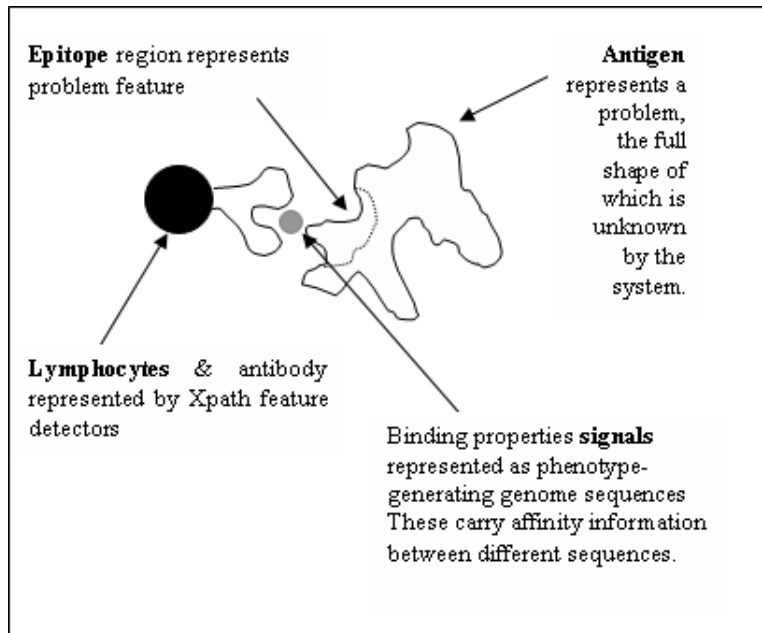


Figure 5.2: Binding Analogy

Evolving Antibodies

In the natural immune system Lymphocytes or B cells contain Y-shaped receptor i.e. antibodies. The filtering effect of an XPath query is achieved using receptors, which are regular expression constraints. Yet the combined effect of the entire query in addition to the context or the 'side effects' from the binding event are more generally attributed to these artificial lymphocytes. The analogy is carried further in the evolution of antibodies. In a somatic hypermutation based algorithm described below, antibody regions along the XPath query (artificial lymphocyte) can evolve to best match the pathogen samples.

All XPath queries are mapped from the genome using Grammatical Evolution. Two search strategies have been used. The first is the the simple genetic algorithm and needs no explanation. The second method is based loosely on the immune system.

Only a small number (approx 10-15%) of XPath queries bind successfully in each generation. These are given exclusive propagation rights (survival and ability to reproduce) although there is a small probability called a *survival factor* that an unsuccessful XPath query will propagate.

Using feedback from the genotype-phenotype mapping event, genomes are encoded with the grammar production they are used for. This information is used to identify regions of the XPath. The more specific regions are referred to as isotypes. Isotypes are defined and selected as follows. Each grammar symbol in the XPath grammar has a code that associates it with a particular function on the primary genome. For example, a set of XPath symbols that filter a symbolic regression expression will have one code, whereas any XPath symbol that filters an operator will have another. Isotypes are selected by picking a random position on the genome and

looking downstream for the next region as specified by these codes.

Using this type of information, mutation rates can vary at the isotype region on the genome or isotype regions on the genomes can be swapped (isotype switching). When isotypes are swapped, they are selected randomly from the pool of all matching isotypes. Each time an XPath is used in reproduction, one of the following operators is used; 1) low mutation (0.1) across the genome, 2) high mutation (0.3) at isotype regions 3) Isotype switching, taking the isotype from the pool of isotypes on all antibodies 4) Isotype switching with (0.3) mutation at that region. The purpose of these operators are to promote diversity and improve hierarchical composition through isotype regions. Using phenotype-genotype feedback to bias operators in GE is a topic for further research made particularly interesting in light of degenerate coding in Grammatical Evolution.

Recombination

The previous section described evolution of XPath. XPath queries are used to produce a library of sequences through the binding event. These sequences are available to potential operators when regenerating the primary population. The simplest method is to use a one-point crossover. Two sequences are chosen from the library through tournament selection and joined. Intuitively, this will only work with particular problems unless the library has managed to incrementally detect higher-order features.

The second method, called chaining, allows any number of features to be added to the solution thus overcoming the 'two feature limitation'. This operator is based on the integration of multiple antibodies or similarly the formation and folding of a polypeptide chain. A number of sequences are chosen from the library randomly. These are layered according to their fitness onto a stack of variable length sequences

with higher sequences overwriting lower sequences. As sequences are of variable length, sequences may be totally overwritten or partially overwritten. Positional information is used to choose where to add the new sequences to the stack. There is a probability (0.1) of shifting the position once to the left or right and a smaller probability (0.01) of picking a random position. A number of features (in these experiments, 8) are stacked in this way.

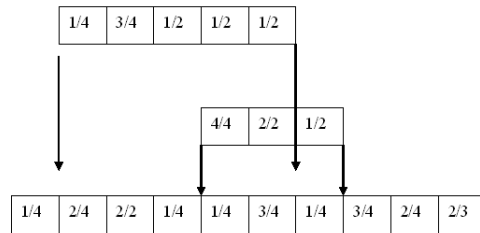


Figure 5.3: The fittest fragments are in the higher layers and will mask lower layers in that region when the genome is expressed.

Two observations should be made. Firstly, sequences are chosen randomly from the library for chaining. It might seem more reasonable to use, for example, tournament selection. However, we aim to promote diversity in choosing sequences for recombination. The layering method can still express fitter sequences by adding them at higher layers. The second point is about measuring sequence fitness. A number of direct options exist. Recall that sequence instances are grouped by having a common 'production sequence'. These sequences will have a range of fitness values. A sequence could be measured by its best fitness, average fitness or perhaps the entropy of its fitness distribution. Fitness might further be affected by *affinity distances* between other sequences involved in the chaining event. The many possibilities are beyond the scope of the current study. Best fitness is used with hesitancy. The reason for this hesitancy comes from [De Jong and Oates, 2002, De Jong, 2003] observation

that evaluating modules on single criteria can be inaccurate. However motivations link back to the end of section three as we operate a two phase selection process. Thus we return to the first observation just mentioned; by randomly picking modules from the sequence library, which is a small subset of all possible sequences in the search space, it is possible to build many variations during the second chaining event.

5.3 Experiments and Results

The purpose of these experiments is to determine if *antibody coevolution* is a feasible module detection mechanism. These experiments require the modular algorithms to build solutions from features detected by the feature detectors. A number of variants of the model have been tested on standard GP problems, specifically, two Symbolic Regression functions and a Multiplexer problem.

Parameters are shown in Tables 5.1 and 5.2. The immune system models uses a number of specific parameters (Table 5.2) explained in the previous section.

The first dimension of variation is with respect to the evolutionary algorithm. Note we refer to genotypes or phenotypes for the problem itself as the primary population and genotypes or phenotypes for XPath as the secondary population. We investigate a standard GA approach (S) to evolving the secondary population and a method modeled on the immune system (I). Two means of recombination within the first population are tested. One is a one-point crossover-like model(X) for exchanging identified features and the other is the chaining method (C) which layers a number of identified features using contextual information. There are therefore, four variations in coevolution, namely SX, SC, IX and IC.

Property	Primary	XPath
Generations	31	-
Tournament Size	3	3
Wrap Threshold	3	3
Population	300	100
Elitism	0.1	0.2
Mutation	0.1	0.1
Crossover	0.7	0.7
Binding Rate	0.7	0.7

Table 5.1: Parameters for both primary and XPath population evolution.

Property	Value	Property	Value
Binding Rate	0.7	Chain Position Shift	0.1
General Mut.	0.1	Chain Position Mut	0.01
Isotype Mut.	0.3	Survival Factor	0.02

Table 5.2: Parameters particular to Immune System

Symbolic Regression

Two symbolic regression functions are tested, each over 30 runs with test cases drawn from the range [0,1]. The first function is quartic symbolic regression and the other is taken from [Keijzer, 2003]. Linear scaling is used following [Keijzer, 2003].

$$f(x) = x^4 + x^3 + x^2 + x^1 \quad (5.1)$$

$$f(x,y,z) = \frac{30xz}{(x-10)y^2} \quad (5.2)$$

The grammar in section is used in both cases.

`<expr> ::= <expr><op><expr>`

`| <var>`

`<op> ::= + | - | * | /`

`<var> ::= x | y | z | 1.0 | 2.0`

Model	Avg Pop.(S.Dev)	Avg Best(S.Dev)	Success(Gen)
Crossover	28.6(1.1)	0.07(0.2)	28(579)
SX	28.4(0.4)	0.5(0.8)	19(399)
IX	34.0(1.6)	5.2(5.4)	6(79)
SC	35.9(0.8)	1.6(3.5)	19(353)
IC	33.1(0.9)	0.04(0.2)	28(603)

Table 5.3: Results for quartic symbolic regression i.e. function 1. From 30 runs, average population fitness and average best fitness are shown with standard deviation in brackets. Successful runs are shown with successful generations in brackets. This indicates how quickly solutions are found.

Model	Av.Pop.(S.Dev)	Av.Best(S.Dev)	Hits(Gen) at 15,12,10
Crossover	79(2.0)	8.3(1.7)	8(183), 5(111), 1(23)
SX	81(1.8)	10.8(3.0)	2(62), 1(31), 0
IX	67(4.2)	10(2.7))	3(29), 1(3), 0
SC	76.8(1.9)	10.7(3.2)	3(49), 2(40), 1(14)
IC	79.1(0.5)	8.6(2.8)	6(186), 6(186), 6(186)

Table 5.4: Results for symbolic regression function 2. Probably due to the use of constants, no algorithm found the perfect solution using the given grammar. Hits for less than 15,12 and 10 are shown.

Results in Tables 5.3 and 5.4 show negligible differences between the crossover and IC model. However the intermediate models SX, IX and SC suggest contributions from aspects of the IC algorithm. We used hits less than scores 15,12,10 as a means to exaggerate the difference between these algorithms on function 2.

While it is difficult to reduce the algorithms completely, the results suggest that there is little difference between crossover recombination (X) and chaining (C) when the standard method S is used to evolve lymphocytes. However when the hypermutation based method (I) is used to evolve lymphocytes, the chaining method can exploit it in IC while the crossover based recombination methods cannot (IX). Graphs (Figures 5.4 and 5.5) of average best and average population fitness for both functions show the IC algorithm to converge slowly.

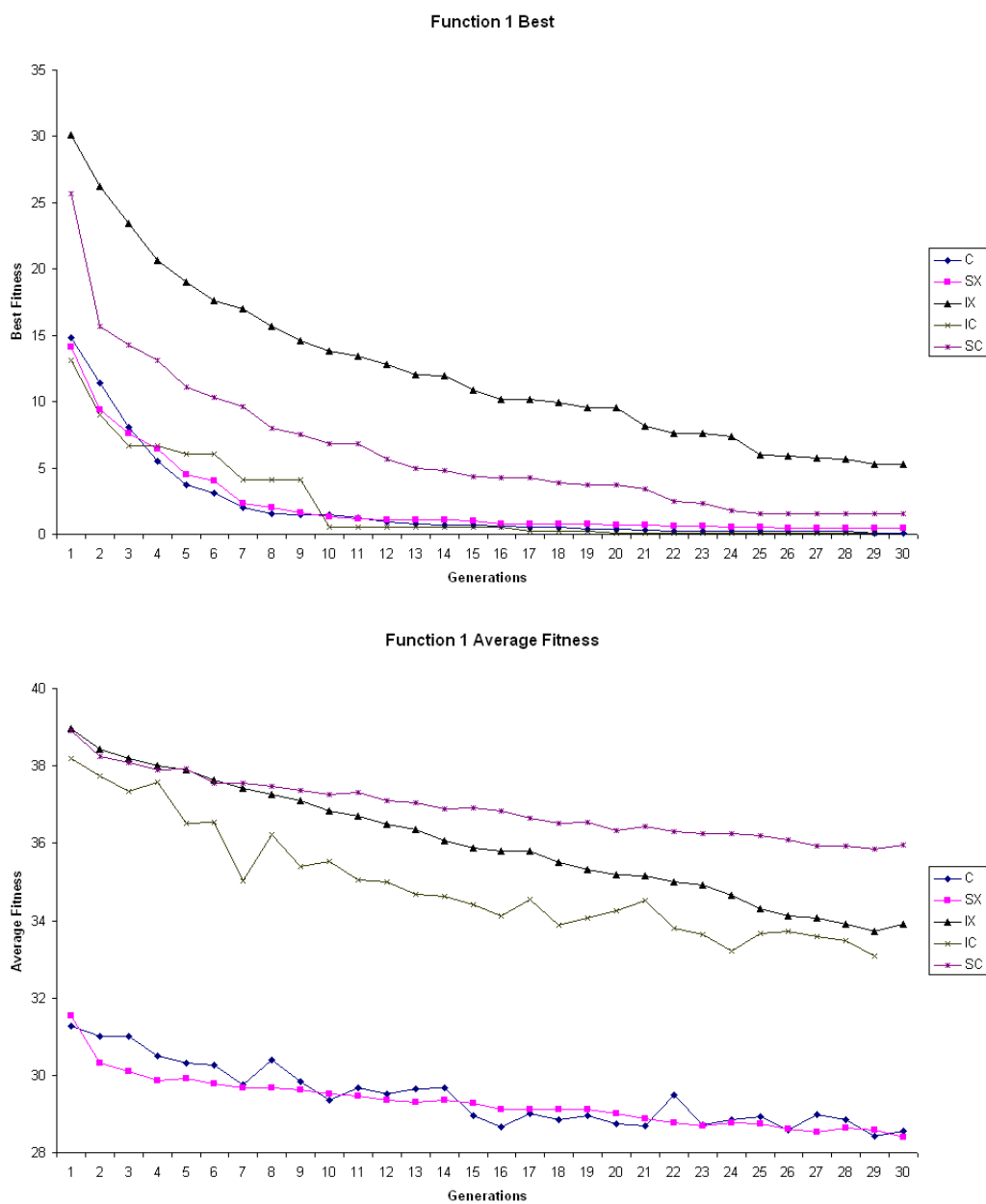


Figure 5.4: Symbolic Regression Best Fitness and Average Fitness over 30 samples

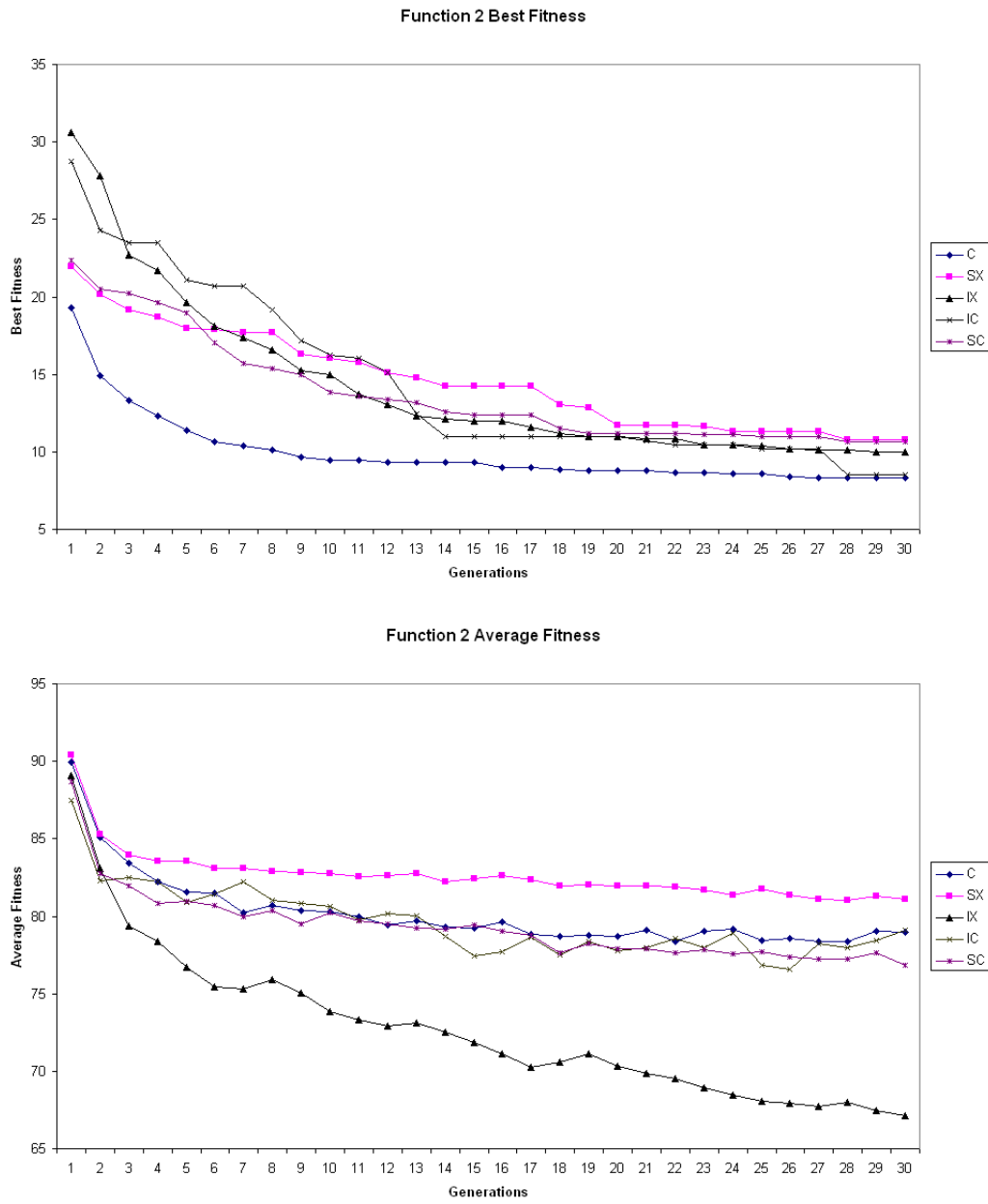


Figure 5.5: Function 2 Average Fitness over 30 samples

Model	Avg Pop.(S.Dev)	Avg Best(S.Dev)	Success(Gen)
Crossover	28.5(1.4)	9.7(1.8)	1(28)
SX	31.1(1.2)	9.7(3.1)	2(51)
IX	28.7(1.3)	9.7(1.8)	4(70)
SC	31.2(1.2)	9.7(3.1)	0
IC	27.9(1.6)	8.1(4.0)	6(116)

Table 5.5: Results for 3-Multiplexer

3-Multiplexer

The multiplier problem requires discovery of a boolean expression that behaves as a three multiplexer. 8 fitness cases are used to represent all possible input-output pairs, where fitness is the number of the correct output cases. The grammar is shown below.

```

<bexp> ::= <bexp><bop><bexpr>
        | <uop> (<bexp>)
        | input
<bop>  ::= and | or
<uop>  ::= not
<input> ::= input1|input2|input3

```

Results in Table 5.5 show IC to be effective on this problem, while the SC model which also used chaining performed badly. The IX model was the next best model to IC. The suggestion is that the hypermutation algorithm maintains diversity better than the standard crossover model and that these diverse features can be exploited by the chaining operator. However the differences between IX and IC are not so significant that the chaining operator can be considered superior. Graphs (Figure 5.6) show the IC algorithm to be better on average in terms of best fitness.

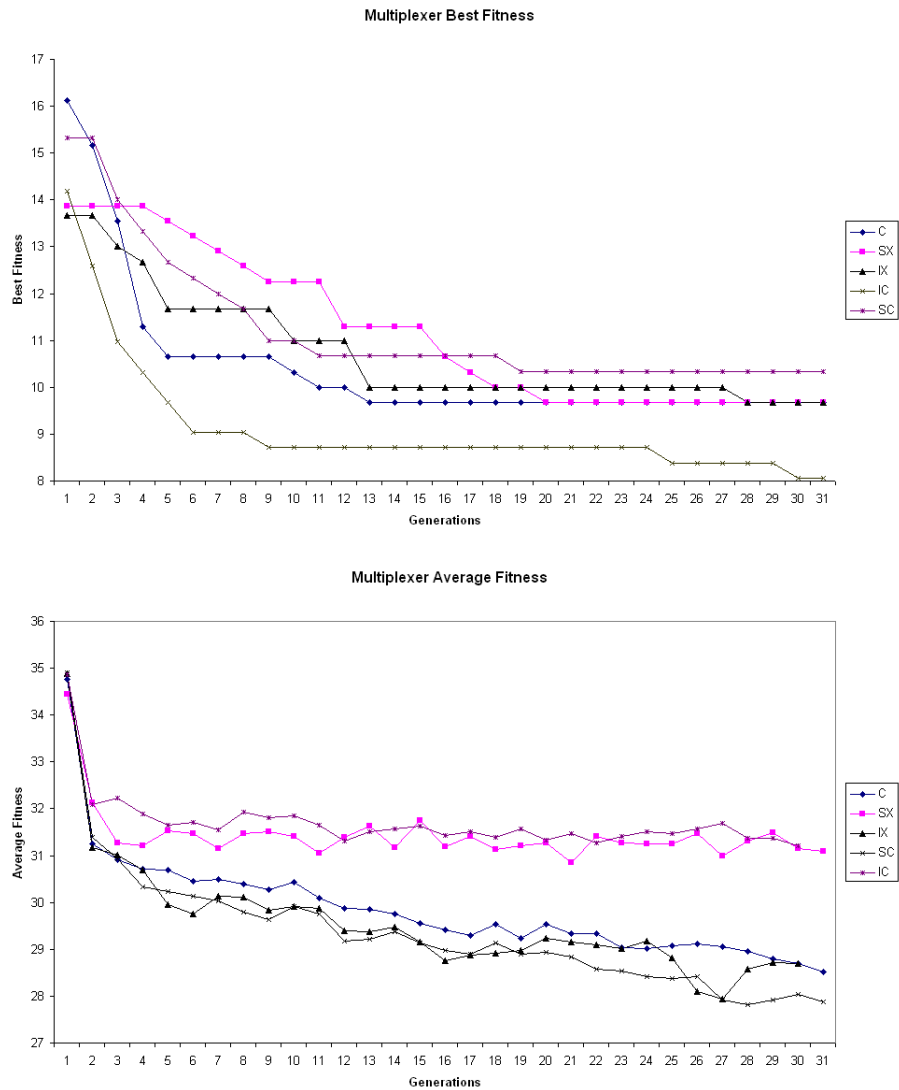


Figure 5.6: Multiplexer Best Fitness and Average Fitness over 30 samples

A note about the Multiplexer problem is required. Multiplexer is an easy GP problem. However the crossover operator only solved the problem once. This was alarming. We ran the algorithm for the standard length with standard populations size and found it to be considerably more effective, finding the solution in almost all cases. Fortuitously, this has led to an interesting observation. Multiplexer fitness is based on the number of successful outputs over 8 fitness cases. Fitness cases can range from 1 to 8. At the start of the run, many solutions get between 2 and 4 successful output. By the middle of the run, many solutions find all but one. However, there is nothing to differentiate between different solutions. While it is easy to find a 3 multiplexer, as a GP problem it is still 'deceptive'. This might be seen as the problem of having different, equally fit lineages each solving parts of the problem [Gustafson, 2004]. By using a library of features in recombination, antibody coevolution seems to be more effective on this problem under the given constraints.

5.3.1 Quartic symbolic regression with small populations

Having shown the performance of the different algorithms on a range of problems, we considered the issue of population sizes using the quartic symbolic regression problem.

Quartic symbolic regression was repeated, this time with a population of size 50. Results are shown in Table 5.6. Success rates scaled down uniformly giving little indication as to the differences in the models.

Following this, the XPath population was increased from 100 to 300, which is equal to the size of the initial primary population. IC scored 11(191) hits which was almost twice as good as any previous run[‡]. When the XPath population was increased

[‡]This is interesting when considering the evolution of simulations or other resource intensive evaluations. Is it possible to attach a second, evolving process to adaptively model and improve an expensive primary evolutionary process?

Model	Avg Pop.(S.Dev)	Avg Best(S.Dev)	Success(Gen)
Crossover	34.5(2.3)	15.7(7.7)	5(102)
SX	34.8(2.3)	17.5(8.4)	2(32)
IX	34.2(2.0)	14.2(7.3)	1(4)
SC	33.1(2.2)	7.9(8.7)	4(78)
IC	36.2(1.9)	15.7(10.7)	6(79)

Table 5.6: Function 1 with primary population size at 50.

Model	SX (100)	SX (300)	IC(100)	IC(300)
Multiplexer	<i>2(51)</i>	1(18)	<i>6(116)</i>	13(238)
Quartic SR	<i>19(399)</i>	20(428)	<i>28(603)</i>	20(402)

Table 5.7: Results with both populations set to 300 compared to results with xpath populations set to 100 (italics)

to 500, IC only managed to achieve 4(90) hits. This is less than the result achieved with an XPath population of 100. While the differences between these are small, the difference between IC with XPath population of 500 and IC with XPath population of 300 is quite big i.e 4(90) against 11(191).

Following this result, quartic symbolic regression and multiplexer were evaluated with both populations at 300 for models SX and IC as summarized in table 5.7.

The SX model changes slightly in both cases. For the quartic symbolic regression problem, the improvement is from 19(399) to 20(428) while on the multiplexer problem there is slight degradation from 2(51) to 1(18), which in both cases is trivial. The IC model improves on the multiplexer from 6(116) to 13(238) yet degrades on the quartic symbolic regression case from 28(603) to 20(402). Later when XPath was set to 500, IC scored 4(90).

These results suggest that when there is deception or over-representation of different, equally fit lineages, complement coevolution can exploit large XPath populations to improve search space modeling. However, there is an upper limit in increasing

XPath population size probably due to competition at binding sites leading to inappropriate distribution of XPath survival. We suspect that this upper limit will be higher in deceptive problems (or in general problems that do not quickly converge). This may be because a larger number of diverse XPath queries can be sustained through the binding events that are crucial for their survival. Future work will analyze the behavior of the XPath population under different conditions.

5.4 Discussion and Conclusions

We have introduced the concept of *antibody co-evolution* at a high level of representation using grammars. Associative templates (complements) are evolved to mirror primary structures and act as adaptive feature detectors. This method exploits a genotype-phenotype mapping to filter the possible genome sub sequences through phenotypic significance. It achieves further adaptability as features, identified at the phenotype via antibodies, are ultimately represented as genome subsequences. These sequences are degenerate in GE and not tied to reproducing the same phenotype as can be the case with ADFs for example. There is naturally a tradeoff between the information loss and the diversity gain. However work such as [Wu and De Jong, 1999] suggests that this type of degeneracy is more beneficial than not.

The results are interesting. The IC algorithm performed as well or better than standard crossover on the tested problems. The effect of increasing XPath population sizes seems to have a positive effect on performance to a limit in certain cases and may be useful for evolution of resource-intensive problems. The differences between the various models suggest contributions from isolated processes in the general model. Further work will determine how the individual processes can be improved

through either parameter choices or alternative representations (e.g. XPath mapping grammars). Future research will also explore *dual process evolutionary algorithms* as evolving networks of adaptive low-level features. Particular attention will be paid to the correlations between problem characteristics and performance fluctuations when comparing standard genetic algorithms to these models.

At one level, we demonstrate an analogy with cognitive or perceptual systems. Such systems have selective interest, selecting environment micro features of interest. These are used to construct a reality or global perception. Through this analogy we suggest the importance of dual processes. Unlike other evolutionary algorithms which consider a single process of optimization, we suggest that two processes should be distinguished. This mechanism overlaps existing ideas in the literature. On the one hand, the importance of modularization for certain problems and on the other hand the importance of memory for maintaining mutually exclusive features.

The perceptual system is distributed in the sense that a library of feature detectors or antibodies are maintained. Collectively, these antibodies mirror aspects of the pathogen i.e. the problem. These antibodies can be evolved using grammars that are themselves reciprocal structures of primary grammars, which makes the general approach domain neutral. This method is believed important in bootstrapping perceptual systems in a bottom-up fashion. The use of antibodies or more generally, *complements*, in evolutionary algorithms can be used as adaptive module detectors while a library of antibodies can sustain a library of features or genes which are their complements.

The evolution of antibodies which mirror phenotype structures has been made possible through the use of rich representations. A number of information-based filtration phases result in meaningful degenerate codon sequences. Grammar contexts

and other details are stored on phenotypes and constitute and implicit relationships with the genotype. Such methods suggest approaches to performing 'local search' within genetic material without making problem-specific assumptions.

The next chapter explores another method for exploiting rich representations in the GE mapping.

Chapter 6

Codon Compression and Delayed Expression

[Agnes] wondered what kind of existence the computer had programmed for life after death. Two possibilities came to mind. If the computer's field of activity is limited to our planet, and if our fate depends on it alone, then we cannot count on anything after death except some permutation of what we have already experienced in life... At best, existence after death would resemble the interlude she was now experiencing while reclining in a deck chair: from all sides she would hear the continuous babble of female voices.

- Milan Kundera 'Immortality'

6.1 Introduction

It is becoming clear that artificial genomes used in genetic algorithms may not be best appreciated as literal strings, read in a linear fashion. It becomes extremely difficult for an evolutionary algorithm to present a perfect string as problems become large and complex. Work in evolutionary algorithms such as artificial embryogenies [Bentley and Kumar, 1999] and gene expression based methods [Kuo et al., 2004] have begun to address this issue by using developmental processes and combinatorial gene expression.

In a more generic perspective we consider the genome interpreter as an anticipatory system which discovers regularities in a noisy genome environment. An anticipatory system can be considered as *a system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's prediction pertaining to a later instant* [Rosen, 1985]. This is a useful way to think about innovation in evolutionary algorithms yet requires the use of *rich environments* to enable information-centric rather than purely algorithmic-centric investigations.

The work presented in this section makes two contributions. 1) It presents a new and effective genetic mechanism while 2) asking questions about the information content in the artificial genome. We consider a method that compresses information in a single genome which may lead to more adaptive algorithms. A second feature of the algorithm is referred to as delayed expression and will be explained.

Algorithmic Information Content

We consider the Algorithmic Information Content of a string, which is defined as the length of the shortest program needed to generate that string [Kolmogorov, 1965, Chaitin, 1990]. We ask; what is the shortest string that is needed to map to promising regions of phenotypic space with high probability given many potential genome encodings? This could be evaluated as the probability that a genome will yield a good phenotype following a random perturbation. This could in turn suggest that the regularities that define the search space are encoded and possibly compressed in the genome in that it facilitates a mapping that is adaptive to perturbations. Note as we discuss an encoding, the mapping process that interprets it is implied.

In a simple direct mapping in Grammatical Evolution, how much information does a single genome contain? To add information to the encoding one could add complexity to either the mapping or to the environment. More complex mappings in evolutionary algorithms such as those evaluated in [Ebner et al., 2001] use alleles in complex ways and appear to contribute towards evolvability. Other approaches enrich the environment and the mapping for example in artificial embryogenies [Bentley, 2004b, Federici and Roggen, 2004]. This study considers exploiting trivial information inherent to the genome and the mapping.

Gell-Mann's perspective on the matter of complexity attempts to account for how complex adaptive systems learn to anticipate in a world that exhibits regularities as well as random deviations from those regularities. Gell-Mann defines effective complexity as complexity inherent in patterned regularities. Effective complexity is characterized mathematically in terms of an algorithmic information measure that measures the extent to which regularities can be compressed into a mini-

mal representation or schema. Gell-Mann also characterizes the effects of randomness mathematically in terms of a Shannon information measure [Shannon, 1948] that measures the extent to which random deviations depart from the patterned regularities in question. He then defines total information as the effective complexity in addition to the complexity inherent in the effects of such randomness [Gell-Mann and Lloyd, 1996, Gell-Mann and Lloyd, 2004].

Christoph Adami and co-workers consider biological complexity in the context of the genome [Adami, 2003, Adami et al., 2000] and ask what information a genome stores about the environment. However Adami acknowledges that lack of agreement on what is 'complexity' while defining *physical complexity* which he suggests corresponds exactly to what biologists feel is increasing when biological systems self-organize. The physical complexity of a sequence (such as a genome) is the amount of information which is stored about an environment. Entropy can be seen as the potential states a system could have and Adami equates sequence entropy to *length*. Adami suggests that adaptation (evolution) is filling empty slots along the length of a sequence with information (which reduces uncertainty at that position) thus increasing complexity. Information is revealed as symbols that are conserved (fixed) under mutational pressure - if a mutation which is beneficial (fits the environment) occurs then the amount of information (and hence complexity) has increased. A beneficial mutation that is lost before fixation does not decrease information. In short, Adami views natural selection as a filter or semipermeable membrane that allows information to flow into the genome but not out. He compares this to Maxwell's demon *. These questions about complexity have influenced developmental evolutionary algorithms

*Maxwell's demon is the elusive creature that disobeys the second law of thermodynamics by selectively allowing particles to move in certain directions.

[Lehre and Haddow, 2003].

6.2 Algorithm Description

Two parents are selected for crossover. Rather than splice and merge as in traditional crossover, a function is applied along the genome. Where possible, a number that will satisfy both parents' production rules choices is chosen as the new codon, allowing the child to satisfy both parent genomes at that allele. In the case where both parents can not be satisfied, the information from both parents can be stored. When it is possible to satisfy both parents we will call this *codon compression*. When it is not possible, we will call this *codon divergence*.

In a rich-representation-based approach, feedback is applied from the genotype-phenotype mapping. After evaluation, in addition to having a fitness value recorded on the genome, the grammar context at each codon is recorded. This allows a genetic operator to exploit information about the grammar in order to choose codon values. Specifically, rather than use codons in recombination, the grammar contexts may be used directly in a manner as described below.

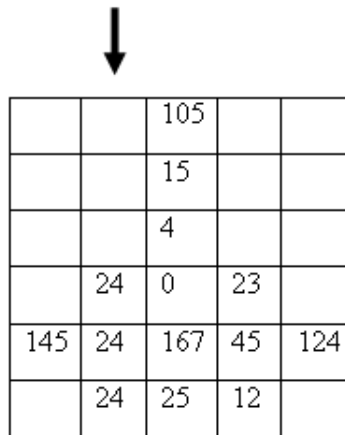
A simple function is used to choose the new codon value. Recall that using the mod rule, 'codon value' modulus 'grammar choices' gives the grammar choice to used in production. If the codon value was 10 and there were 6 choices, $10 \bmod 6$ would give 4, and so the grammar choice at index 4 would be used in phenotype production. Hence, consider the case where one parent chooses rule '3 of 6' at the first allele. Assume on the corresponding allele in the second parent the genome value is 46 and the number of choices is 4 causing the second codon value to be chosen in the given context. Thus, given $X \bmod 6 = 3$ and $X \bmod 4 = 2$, X could be compressed

to become 15. Clearly, there may not be such a real number in all cases but where there is, the codon is said to be compressed as both parents are satisfied. This can be computed for any *order*. Order is the number of generations of alleles remembered and compressed. Order-1 satisfies only the parents as just shown. Order-2 satisfies the parents and the grandparents. And so on.

When a genome encoded in this way is to be expressed, there may be more than one possible expression path. For example consider the following sequence '34|210', '12|51', '47'. The numbers separated by '|' denote a divergent codon. This could be called diploid in that both parents' alleles are represented. An isolated number such as '47' has either been compressed or mutated. Note a mutation event will have chosen a random value for that position while a compressed codon will have found one codon value that satisfies both parent's intentions. The genome can be visualized as columns of lineages and rows of codons. The cursor sits above a given column and may be incremented at random moments to explore different columns. The maximum number of columns in these experiments is 5.

A cursor can be used and incremented (and wrapped) by some value with a small probability. This probability of increment is given by the Cursor rate parameter and the jump value is given as the Cursor Jump parameter in Table 6.1.[†] Using a small cursor increment probability means that as gene expression takes place row by row, linked codons are likely to be expressed together. Because each genome stores a number of genome columns (e.g. the genetic material for both parents and a couple of the grandparents), more information is retained. The 'compound' genome can be read a number of times reusing variable genetic material over a number of gene

[†]A number of trial runs were used to determine these parameters. Jump values of 1,2 and 3 were used. The suitability of these values may depend on the number of columns used which in this case is 5.



		105		
		15		
		4		
	24	0	23	
145	24	167	45	124
	24	25	12	

Figure 6.1: The Genome and read cursor. The cursor moves along the columns and is normalized to the number of columns in the particular row. Columns correspond to entire chromosomes from parents while rows are allele positions. In this illustration the later rows are divergent with row five retaining all genetic material. Earlier rows have been compressed or mutated. This is a realistic outcome.

expression events. we refer to this as *delayed expression*. This preserves ancestral features despite not always being able to express them.

Encoding Mapping Details on the Genome

We evaluate two methods for encoding grammar contexts on the genome which are used in the algorithm. These values are encoded on the n th generation while compression-divergence occurs at generation $n + 1$.

The first method is an accurate and intuitive feedback mechanism. Recall that the genome may have divergent alleles and depending on the read cursor, there are a number of possible expression paths. The feedback mechanism records the actual grammar context used on the derivation tree (i.e. X-Expression). Feedback allows the genome to be informed as to how it was actually used in a given mapping event.

A *read ahead* method ignores the expression path and ignores divergent codons. It records grammar contexts for all alleles that are compressed by reading along the genome before the mapping event. This involves reading a genome's values as though a regular GE mapping (as opposed to Codon Compression Delayed Expression) is actually taking place. If there are divergent codons, GE can not use the allele position. Otherwise, GE will use the codon value in choosing a grammar production. If a grammar context is not recorded, a grammar context may be already recorded at the given allele from a previous read ahead event. The method is called 'read ahead' because prior to the actual mapping event, this approximation of the genome's interpretation is recorded. In Figure 6.1, the alleles at positions 3, 4 and 6 will not have a grammar context stored because codons are divergent at that position.

$\frac{1}{4}$	$\frac{3}{4}$	X	X	$\frac{1}{4}$	X	$\frac{3}{4}$	$\frac{1}{2}$
12		121	43		156	12	
	32	14	210	21	97		14
1	2	3	4	5	6	7	8

Figure 6.2: Read ahead method for storing grammar contexts. Grammar context values such as 1 of 4 ($\frac{1}{4}$) are chosen based on an evolving grammar context.

The grammar context values shown in Figure 6.2 will be used in favor of codon values in a subsequent recombination event to generate new codon values for the alleles.

Using parameters given in Table 6.1, when we evaluated each method on symbolic regression, read ahead performed significantly better. This may not be the case for other problems. However this paper concentrates on *read ahead* grammar encoding.

Property	Value	Property	Value
Samples	30	Cursor Rate	0.1
Genome Length	30	Cursor Jump	3
Generations	50	Crossover	0.7
Tournament Size	3	Mutation	0.1
Wrap Threshold	3	Elitism	0.1
Population	50	Max Columns	5

Table 6.1: Parameters.

Model	Sample Hits	Generation Hits
Read Ahead	20	61
Feedback	9	85
Crossover	5	102

Table 6.2: Hits comparison of read ahead and feedback grammar context encoding against standard crossover algorithm on symbolic regression. The generation hits are lower for the compression algorithm in both Read Ahead and Feedback cases than for crossover. This is due to the non-deterministic mapping.

6.3 Experiment Results

For these experiments we use small population sizes. We consider this a more suitable measure when comparing algorithms. Evolutionary algorithms can rely quite heavily on large populations depending on the problem. For example, on the 3-Multiplexer problem we observed a significant performance difference between runs using 300 individual over 31 generations as opposed to 500 individuals over 51 generations. However on symbolic regression, performances scale down uniformly with population sizes. We are interested in an algorithms ability to evolve a solution and as such try to restrict the influence of many random samples. Parameters for all problems are shown in Table 6.1.

For all fixed length experiments, genomes are fixed at 30 integer codons. The cursor in the new model is incremented at a fixed rate, and by a fixed rate. These rates are shown in Table 6.1.

Fixed Model	Crossover	CA	Comp.	Diverg.
Quartic	4	20	300	750
Multiplexer	1	4	160	900

Table 6.3: Total Hit results for fixed crossover and fixed compression based algorithm (CA) on both problems. Approximate compression and divergence rates over all generations are shown.

Variable Model	Crossover	CA
Quartic	4	6
Multiplexer	2	7

Table 6.4: Results (Hits) for variable length comparisons show a slight improvement over variable length crossover using variable length compression. Variable length compression is similar to variable length crossover except compression is used when adding a segment to the recipient genome. This does not perform as well as the fixed case (shown in Table 6.3) on symbolic regression but performs slightly better on multiplexer.

Quartic Symbolic Regression

Results for quartic symbolic regression are shown in Figure 6.3 The following grammar is used.

```

<expr> ::= <expr><op><expr>
        | <var>
<op>   ::= + | - | * | /
<var>  ::= x | y | z | 1.0 | 2.0

```

The fixed length compression algorithm and to a lesser extent the variable length compression algorithm are seen to oscillate on average despite the use of elitism. These algorithms use a number of genome columns which can be read in random orders. This means that genomes can have variable fitness over a number of 'expression events'. In these algorithms the best fitness can for example be found in generation 12, then lost and recovered in generation 24. This explains the average fitness readings. However it

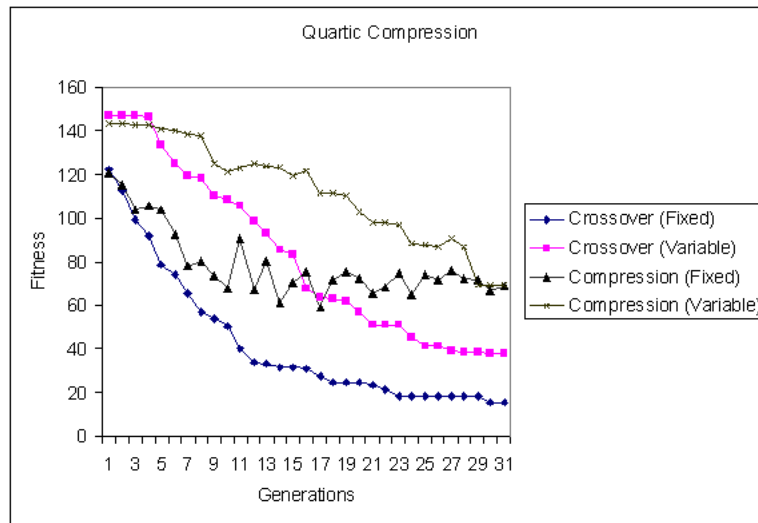


Figure 6.3: Population best fitness plots for Compression Experiments on Quartic Symbolic Regression

is interesting that on average the fitness does improve until about generation 15 and then levels out. A similar pattern is seen for the multiplexer problem. This suggests that the complexity of these genomes is increasing in that they become less likely to depart from fitter regions of the search space.

3-Multiplexer

The fixed length compression algorithm performs better on average in the multiplexer problem. This can be seen in Figure 6.4. The grammar is given below.

```

<bexp> ::= <bexp><bop><bexpr>
        | <uop> (<bexp>)
        | input
<bop>  ::= and | or
<uop> ::= not
<input> ::= input1|input2|input3

```

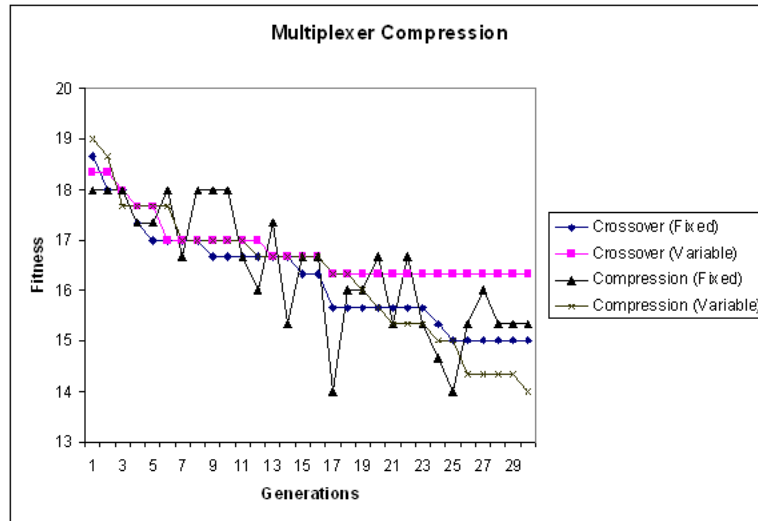


Figure 6.4: Population best fitness plots for Multiplexer Compression Experiments.

Figure 6.5 shows compression-divergence ratios for both problems. The compression and divergence plots show consistency in the population-wide rates.

6.4 Discussion

One aspect of complexity is the amount of information that is compressed in an entity. This is to say, that the *proportion* between how many variants there might have been and the state(s) the system finds itself in is significant. We have not addressed complexity measurement explicitly in this study. Measures such as Kolmogorov complexity have been used in consideration of genotype/phenotype complexity [Lehre and Haddow, 2003] while a measure of *physical complexity* has also been developed and used for this purpose [Adami, 2003]. Yet we relate our use of codon compression to this issue and hypothesize that a useful relationship between such measures and codon compression might be found in future work. Intuitively, evolution proceeds by reducing uncertainty thus adding information.

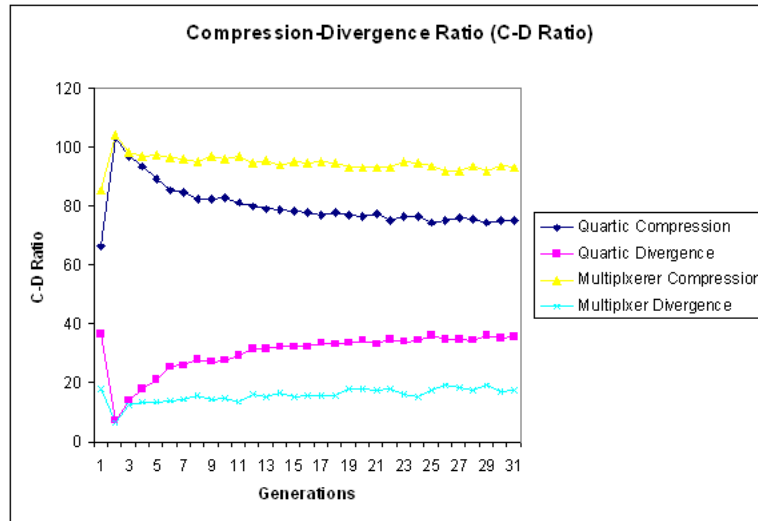


Figure 6.5: Compression Ratios for Symbolic Regression and Multiplexer. Average population compression rates peak and then gradually level out while divergence rates mirror them. This is the same for both problems. The intensity of the variation seems to increase for more successful applications.

The algorithm uses codon compression and delayed expression. Codon compression gradually crystallized alleles sequentially while divergent codons can explore. Codon compression is a type of 'incremental commitment' mechanism. We suggest that the compression of codons which both parents 'agree on' is, in effect, gradually forming building blocks. Pollack and Watson suggest that one of the earliest competent GAs, the messyGA has many unique features and it is difficult to suggest the contribution of these. However they argue that the feature that most distinguished it is *partial commitment* where individuals commit to specifying only a subset of the entire gene set [Watson and Pollack, 1999].

Delayed expression can preserve genetic material that may not have had the opportunity to prove itself and thus guarantee its survival. While we mentioned in the last chapter that many competent GAs work via dual processes, the LLGA

[Harik and Goldberg, 1997] is said to overcome the mixing problem through probabilistic expression. This delays allele convergence while linkages can be learned [Goldberg, 2002].

It seems as though the problem of preserving good genetic material is more of a concern than the problem of building block disruption as is often discussed. Studies on degenerate floating representations in genetic algorithms showed interesting results for abstract problems and on symbolic regression [Wu and De Jong, 1999]. The average population fitness increased faster and leveled off at higher values using floating representations while diversity was also higher. Interestingly while 60% of building blocks survived from one generation to the next, 97% of those that did not survive, did not survive due to not being selected - while only 3% did not survive due to disruption. Indirect or non-deterministic mappings such as delayed expression aim to preserve genetic material while it is not always expressed.

6.5 Chapter Conclusions

Codon compression rates are correlated with search progress. The trend in the compression algorithms success seems to correlate with the trend in GE's success and problems that are difficult for the compression algorithm are also difficult for GE. This suggests that the compression algorithm may be simply exaggerating what GE is already doing. The algorithm in its fixed-length and variable-length modes is compared to fixed and variable length standard crossover. We report performance improvement on a multiplexer problem and significant improvement on symbolic regression problems. Interestingly, high compression rates appear to correlate with

success on these problems. Results seem to indicate that search proceeds through gradually compressing genome segments. For the moment we do not say which of the features *codon compression* or *delayed expression* is most important. Future work will perturb various parameters to determine feature contribution.

Chapter 7

Conclusions

... But there is a second possibility: beyond our planet's computer there may be others that are its superiors. Then, indeed, existence will not need to resemble our past life and a person can die with a vague and justified hope.

- Milan Kundera 'Immortality'

7.1 Lessons and Future Work

The work is based on Grammatical Evolution [O'Neill and Ryan, 2003], which was described in chapter 2. Grammatical evolution exploits a grammar-based genotype-phenotype mapping to evolve programs in arbitrary languages. Inherent properties such as degeneracy and intrinsic polymorphism add robustness to the evolutionary procedure while recent applications of grammars using GE consider modularity and coevolution of genetic codes.

In chapter 3, a wider scope encompassed ideas in evolutionary algorithms that are relevant to the theme of genome interpretation. Methods for representing genome information such as floating representations [D. E. Goldberg and Korb, 1989, Wu and Lindsay, 1996] were reviewed. Some of these methods led to representations that composed fragments or templates of genetic material from a dynamic model [D. E. Goldberg and Korb, 1989, Wu and Stringer, 2002]. Other model building methods use probabilistic model building. These models are sampled to choose allele values in generating new genomes. In addition to emphasis on evolving a genome which is then mapped to a phenotype, the mapping process itself has been another point of research [Banzhaf, 1994, O'Neill and Ryan, 2003, Lones, 2004]. Gene expression methods have been developed using both ballistic mappings and indirect mappings. It is expected that combinatorial gene expression methods will have an important impact in the future [Kuo et al., 2004]. Other gene expression-like methods use explicit developmental processes. These are known as artificial embryogenies or artificial ontogenies [Bentley and Kumar, 1999].

In chapter 4 we presented the XMLGE framework. This is a novel XML application of Grammatical Evolution with a number of unique properties. XML based tools such as XSLT and XPath are used in XMLGE to generate, manipulate and analyze genetic material. This framework builds on GE to allow evolutionary algorithms to be studied at a high level of representation with implied semantics.

The use of a rich genotype-phenotype map was explored in chapters 5 and 6. We discussed the immune system. Artificial immune systems [Hofmeyr, 1999] are a class of evolutionary algorithms that among a number of emerging properties of interest, may use an antibody library. We took the perspective of the cognitive immune system emphasizing the ability to perceive and remember features while

generating new information. An implementation of an artificial immune system which can be seen more generally as a dual process evolutionary algorithms exploited a high level representation and used information from the mapping event. The compression algorithm in chapter 6 also used this information in a novel recombination mechanism. In both cases, the adoption for rich representations led to improvements over GE in many cases and was shown to perform as well in all cases.

General Interpretation

The genome is like a word puzzle, with many features waiting to be found. As evolution proceeds even in the simple GA, above average sequences are propagated. However, these are lost in a number of ways due to disruption or repositioning. This may be just as well since these blocks are not 'good features' *per se* but merely a route to solutions, where that route must be reasonably canalized. Following this we suggest that certain blocks should be selected preferentially but used degenerately. In this way they are removed from an *explicit context*, promoting future exploration and reuse.

We believe the artificial genome is misinterpreted and should not hold information in isolation. Blocks of genes can be constantly rediscovered as meaning itself evolves. If meaning evolves, an adaptive system can learn 'contexts'. For example in an early evolutionary phase, if a sequence of genes matches a fit context, the context should be preserved not the gene. If the gene reoccurs in that context, the context will be expressed. However other genes can fulfil that context also, just as similar genes can activate different contexts. Many biological systems define contexts that are manifested as system attractors. These attractors are stable despite the loss of control parameter information such as gene mutations. We suggest that macroscopically

(having no relation to dynamical systems), this is the most important feature of Grammatical Evolution i.e. the degenerate and intrinsically polymorphic code that packs meaning into redundant sequences.

7.1.1 Extending the Dual Process Algorithm

We have suggested a method to preserve features as degenerate sequences. This method could be applied to dynamic and multimodal problems which are difficult for evolutionary algorithms. Extensions to the XPath grammar may allow more specific and powerful matches. It will be interesting to see if modification of this grammar has any effect on the algorithms performance. This, and a number of parameter variations may improve the first process in the dual process algorithm.

The dual process algorithm was based on ideas about perceptual systems and useful examples [Hershberg and Ninio, 2004]. Such work discusses the statistical distribution of *useful examples* suggesting that many features that are the building blocks of perceptual systems are semantically neutral and are used ubiquitously, while specificities are emphasized on this background signal. Consolidation of our dual process algorithm and such observations will be pursued in the future. Firstly within the current algorithm, the occurrence rate of features could be measured and a means to detect feature anomalies could be investigated. Later extensions to the algorithm may be directed towards a closer *exemplar learning based* modeling.

The integration process might also be improved by borrowing again from biology and constructing networks. Just as pathogens are suppressed by dynamic antibody networks, or genes are activated in dynamic gene networks, so too might linkages evolve between degenerate features.

7.1.2 Exploiting Compression

We have based the compression algorithm on questions about complexity. However we have not explicitly considered complexity measures. The relationships between grammar complexity, phenotype complexity and compression will be investigated more closely in the future with respect to complexity measures.

Another development will be to add compression to the chaining operator in the dual process algorithm. This would mean that codon values that satisfy different *layers* could be exploited to a certain degree.

7.1.3 The Next Generation of XMLGE

The XMLGE framework has been developed as a means to explore XML in evolutionary algorithms and the benefits of rich representations. The use of XSLT for genetic operators has been demonstrated as a novel but resource hungry mechanism. Going forward, we aim to redevelop the ideas of rich representation without XML so as to improve performance. In certain cases it is beneficial to 'expose' structures as XML for either serialization or analysis. As such, XML will be used as an extension of regular classes. For example, a phenotype object structure can be exposed as XML for XPath queries and used in complement coevolution. The alternative to this would be to develop a custom query tool.

7.1.4 Towards Efficient Dynamical Systems Evolution

Future work is still required to improve these evolutionary algorithms. However work to date such as the use of XPath feature detection and compression suggest that smaller populations can be used with symbolic regression type problems. We

have shown how evolving complex systems is a special function induction problem with one or more functions to evolve. One of the suggestions of the dual process algorithm is its applicability to multimodal problems for example the case where a number of functions are simultaneously evolved. We observed this in the multiplexer problem which is a very simple case where mutually exclusive building blocks must be preserved. However work such as [Hasegawa and Iba, 2004] suggests that artificial immune systems are efficient at these types of problems in cases where standard GP are unable to find solutions. Given these two findings (i) that smaller populations can be used with symbolic regression problems (ii) dual process algorithms are useful for multimodal problems, we can attempt to evolve function-based dynamical systems more efficiently. This has been our goal and we are pleased to say that it seems now to be within reach.

Conference Papers

Parts of this thesis have appeared in the following publications.

- Saoirse Amarteifio and Michael O'Neill (2004), An evolutionary approach to complex system regulation using grammatical evolution in: *Artificial Life IX (Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems)* J. Pollock et. al (eds). MIT Press.
- Saoirse Amarteifio and Michael O'Neill (2005), Coevolving Antibodies with a Rich Representation of Grammatical Evolution, To Appear in CEC 2005.

Bibliography

- [Adami, 2003] Adami, C. (2003). Sequence complexity in darwinian evolution. *Complexity*.
- [Adami et al., 2000] Adami, C., Ofria, C., and Collier, T. (2000). Evolution of biological complexity. In *Proceedings of National Academic Science*, volume 97, page 44634468, USA.
- [Amarteifio and O’Neill, 2004] Amarteifio, S. and O’Neill, M. (2004). An evolutionary approach to complex system regulation using grammatical evolution. In Pollack, J., Bedau, M., Husbands, P., Ikegami, T., and Watson, R. A., editors, *Artificial Life IX (Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems)*, pages 551–556. MIT Press.
- [Amarteifio and O’Neill, 2005] Amarteifio, S. and O’Neill, M. (2005). Coevolving antibodies with a rich representation of grammatical evolution. In *CEC 2005 (To Appear)*.
- [Azad and Ryan, 2003] Azad, R. M. A. and Ryan, C. (2003). Structural emergence with order independent representations. In et al, C.-P., editor, *Proceedings of the 2003 Genetic and Evolutionary Computation Conference, GECCO 2003*, pages 1626–1638.
- [Baluja, 1994] Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning,. Technical Report CMU-CS-94-163, Pittsburgh, PA.
- [Baluja and Caruana, 1995] Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In Frieditis, A. and Russel, S., editors, *The Int. Conf. on Machine Learning 1995*, pages 38–46, San Mateo, CA. Morgan Kaufmann Publishers.
- [Banzhaf, 1994] Banzhaf, W. (1994). Genotype-phenotype-mapping and neutral variation. In Davidor, Schwefel, H.-P., and Manner, R., editors, *Parallel Problem Solving From Nature III*, pages 322–332, Berlin. Springer.

- [Banzhaf, 2003] Banzhaf, W. (2003). Artificial regulatory networks and genetic programming. In Riolo, R. and Worzel, B., editors, *Genetic Programming - Theory and Applications*, pages 43–61. Kluwer Academic, Boston, MA.
- [Banzhaf et al., 2003] Banzhaf, W., Christaller, T., Dittrich, P., Kim, J. T., and Ziegler, J., editors (2003). *Evolving Developmental Programs for Adaptation, Morphogenesis, and Self-Repair*, volume 2801 of *Lecture Notes in Artificial Intelligence*, Dortmund, Germany. Springer.
- [Banzhaf and Lasarczyk, 2004] Banzhaf, W. and Lasarczyk, C. (2004). Genetic programming of an algorithmic chemistry. In U.M. O’Reilly, T. Yu, R. R. and Worzel, B., editors, *Genetic Programming - Theory and Applications*, pages 175 – 190. Kluwer Academic, Birmingham, UK.
- [Banzhaf and Miller, 2004] Banzhaf, W. and Miller, J. (2004). *The Challenge of Complexity*, pages 43–61. Kluwer Academic, Boston, MA.
- [Bedian, 2001] Bedian, V. (2001). Self-description and the origin of the genetic code. *Biosystems*, 60(39-47).
- [Bentley and Kumar, 1999] Bentley, P. and Kumar, S. (1999). Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, Orlando, Florida, USA. Morgan Kaufmann.
- [Bentley, 2004a] Bentley, P. J. (2004a). Adaptive fractal gene regulatory networks for robot control. In *Genetic and Evolutionary Computation Conference (GECCO 2004)*, Workshop on Regeneration and Learning in Developmental Systems, Catania, Sicily. SpringerVerlag.
- [Bentley, 2004b] Bentley, P. J. (2004b). Fractal proteins. *Genetic Programming and Evolvable Machines*, 5(1):71–101.
- [Bentley and Timmis, 2004] Bentley, P. J. and Timmis, J. (2004). A fractal immune network. In *The Thirds international Conference on Artificial Immune Systems (ICARIS 2004)*, Catania, Sicily. SpringerVerlag.
- [Bonabeau, 1997] Bonabeau, E. (1997). From classical models of morphogenesis to agent-based models of pattern formation. *Artificial Life 3*, pages 191–209.
- [Bonabeau et al., 1999] Bonabeau, E., Theraulaz, G., and Dorigo, M. (1999). *Swarm Intelligence*. Oxford Press.

- [Bongard and Pfeifer, 2001] Bongard, J. C. and Pfeifer, R. (2001). Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In Spector, L., editor, *Proceedings of The Genetic and Evolutionary Computation Conference, GECCO-2001*, volume 2801 of *Lecture Notes in Artificial Intelligence*, pages 256–265. Morgan Kaufmann publishers.
- [Butcher et al., 2001] Butcher, E. C., Foxman, E. F., Pan, J., and Kunkel, E. J. (2001). Multistep navigation and the combinatorial control of cell positioning: A general model of living structure based on studies of the immune cell trafficking. In Segal, L. and Cohen, I., editors, *Design Principles for the Immune System and other Distributed Autonomous Systems.*, chapter 10, pages 227–240. Oxford University Press.
- [Caruana and Schaffer, 1998] Caruana, R. and Schaffer, J. (1998). Representation and hidden bias: Gray versus binary coding in genetic algorithms. In Laird, J., editor, *5th International Conference on Machine Learning*, pages 153–161, San Mateo. Morgan Kaufmann Publishers CA.
- [Chaitin, 1990] Chaitin, G. (1990). *Algorithmic Information Theory*. Cambridge University Press.
- [Cohen, 2000] Cohen, I. R. (2000). *Tending Adam's Garden: Evolving The Cognitive Immune Self*. Academic Press.
- [Cohen, 2001] Cohen, I. R. (2001). The creation of immune specificity. In Segal, L. A. and Cohen, I. R., editors, *Design Principles for The Immune System and Other Autonomous Systems*, chapter 6, pages 151–159. Oxford University Press, New York.
- [Conrad, 1990] Conrad, M. (1990). The geometry of evolution. *Biosystems*, 24(61–81).
- [Crutchfield et al., 2003] Crutchfield, J., Mitchell, M., and Das, R. (2003). Evolutionary design of collective computation in cellular automata. In Crutchfield, J. and Schuster, P., editors, *Evolutionary Dynamics*. Oxford University Press.
- [D. E. Goldberg and Korb, 1989] D. E. Goldberg, K. D. and Korb, B. (1989). Messy genetic algorithms: motivation, analysis, and first results. *Complexity*, 3(5):493–530.
- [Dawkins, 1989] Dawkins, R. (1989). *The Selfish Gene*. Oxford University Press.
- [de Garis, 1992] de Garis, H. (1992). *Artificial embryology : The Genetic Programming of an artificial embryo*, pages 373–393. John Wiley, New York.

- [De Jong et al., 2004] De Jong, E., Thierens, D., and Watson, R. A. (2004). Hierarchical genetic algorithms. In *8th International Conference on Parallel Problem Solving from Nature*, pages 232–241.
- [De Jong et al., 2005] De Jong, E., Watson, R. A., and Thierens, D. (2005). On the complexity of hierarchical problem solving. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-05*.
- [De Jong, 2003] De Jong, E. D. (2003). Representation development from pareto-coevolution. In *GECCO*, pages 262–273.
- [De Jong and Oates, 2002] De Jong, E. D. and Oates, T. (2002). A coevolutionary approach to representation development. In *ICML, Workshop on Development of Representations*.
- [De Jong, 1975] De Jong, K. A. (1975). *An Analysis of the Behaviour of a class of genetic adaptive systems*. PhD thesis, Univesity Michigan.
- [Ebner et al., 2001] Ebner, M., Shackleton, M., and Shipman, R. (2001). How neutral networks influence evolvability. *Complexity*, 7(2):19–33.
- [Eggenberger, 1997] Eggenberger, P. (1997). Evolving morphologies of simulated 3d organisms based on differential gene expression. In Husband, P. and Harvey, I., editors, *4th European conference on artificial life*, cambridge, Ma. MIT press.
- [Federici, 2004] Federici, D. (2004). Using embryonic stages to increase the evolvability of development. In *GECCO*, Seattle, Washington, USA.
- [Federici and Roggen, 2004] Federici, D. and Roggen, D. (2004). Multi-cellular development: is there scalability and robustness to gain? In *Proceedings of Parallel Problem Solving from Nature*, number 8, pages 391–400, San Francisco.
- [Furusawa and Kaneko, 2002] Furusawa, C. and Kaneko, K. (2002). Origin of multi-cellular organisms as an inevitable consequence of dynamical systems. *The Anatomical Record*, 268:27–342.
- [Garibay et al., 2004] Garibay, I. I., Garibay, O. O., and Wu, A. S. (2004). Effects of module encapsulation in repetitively modular genotypes on the search space. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO, Vol. 1, Seattle, WA*, pages 1125–1137.
- [Garibay et al., 2003] Garibay, O. O., Garibay, I. I., and Wu, A. S. (2003). The modular genetic algorithm: Exploiting regularities in the problem space. In *ISCIS*, pages 584–591.

- [Gell-Mann and Lloyd, 1996] Gell-Mann, M. and Lloyd, S. (1996). Information measures, effective complexity, and total information. *Complexity*, (2):44–52.
- [Gell-Mann and Lloyd, 2004] Gell-Mann, M. and Lloyd, S. (2004). Effective complexity. In Tsallis, C., editor, *Nonextensive Entropy: Interdisciplinary Applications*. Orford University Press, New York.
- [Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co.
- [Goldberg, 2002] Goldberg, D. E. (2002). *The Design Of Innovation. Lessons from and for Competant Genetic Algorithms*. Kluwer Academic.
- [Goldberg et al., 1993a] Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993a). Rapid accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S., editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 56–64, San Mateo, CA. Morgan Kaufmann.
- [Goldberg et al., 1993b] Goldberg, D. E., Dep, K., and Thierens, D. (1993b). Towards a better understanding of mixing in genetic algorithms. *Journal of the Society for Instrumentation and Control Engineers*, 32:10–16.
- [Gordon, 2001] Gordon, D. M. (2001). Task allocation in ant colonies. In Segal, L. A. and Cohen, I. R., editors, *Design Principles for The Immune System and Other Autonomous Systems*, chapter 14, pages 293–301. Oxford University Press, New York.
- [Gruau, 1994] Gruau, F. (1994). *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Supérieure de Lyon.
- [Gruau et al., 1996] Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In *First Genetic Programming Conference*, pages 81–89. MIT Press.
- [Gustafson, 2004] Gustafson, S. M. (2004). *An Analysis of Diversity in Genetic Programming*. PhD thesis, University of Nottingham.
- [Gutowiz, 1993] Gutowiz, H. (1993). Complexity seeking ants: (Unpublished).
- [Harik, 1997] Harik, G. (1997). *Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms*. PhD thesis, University of Michigan.

- [Harik and Goldberg, 1997] Harik, G. R. and Goldberg, D. E. (1997). Learning linkage. In Belew, R. K. and Vose, M. D., editors, *Foundations of Genetic Algorithms 4*, pages 247–262. Morgan Kaufmann, San Francisco, CA.
- [Hart, 2002] Hart, E. (2002). *Immunology as a Metaphor For Computational Information Processing: Fact or Fiction*. PhD thesis, University of Edinburgh.
- [Hasegawa and Iba, 2004] Hasegawa, Y. and Iba, H. (2004). Multimodal search with immune based genetic programming. In Nicosia, G., Cutello, V., Bentley, P. J., and Timmis, J., editors, *Third International Conference on Artificial Immune Systems ICARIS2004*, pages 330 – 341. Springer.
- [Hemberg and O’Reilly, 2002] Hemberg, M. and O’Reilly, U.-M. (2002). Using grammatical evolution in a surface design tool. In *Workshop on Grammatical Evolution at GECCO 2002*.
- [Hershberg, 2003] Hershberg, U. (2003). *The Emergence of Meaning in Biological Systems*. PhD thesis, Hebrew University.
- [Hershberg and Efroni, 2001] Hershberg, U. and Efroni, S. (2001). The immune system and other cognitive systems. *Complexity*, 6:14–21.
- [Hershberg and Ninio, 2004] Hershberg, U. and Ninio, A. (2004). Optimal exemplar learning in cognitive systems. *Cognitive Systems (ESSCS)*, 6(2-3).
- [Hershberg et al., 2003] Hershberg, U., Solomon, S., and Cohen, I. (2003). What is the basis of the immune systems specificity. *Mathematical Modelling and Computing in Biology and Medicine*.
- [Hightower et al., 1995] Hightower, R. R., Forrest, S., and Perelson, A. S. (1995). The evolution of emergent organization in immune system gene libraries. In Eshelman, L., editor, *Proc. of the Sixth International Conference on Genetic Algorithms*, pages 344–350, San Fran, CA. Morgan Kaufmann.
- [Hofmeyr, 1999] Hofmeyr, S. A. (1999). *An Immunological Model of Distributed Detection and its Application to Computer Security*. PhD thesis, University of New Mexico.
- [Hofmeyr, 2001] Hofmeyr, S. A. (2001). An interpretative introduction to the immune system. In Segal, L. A. and Cohen, I. R., editors, *Design Principles for The Immune System and Other Autonomous Systems*, chapter 1, pages 3–26. Oxford University Press, New York.

- [Hogeweg, 2000] Hogeweg, P. (2000). Evolving mechanisms of morphogenesis: On the interplay between differential adhesion and cell differentiation. *Journal of Theor. Biology*, 203:317–33.
- [Holland, 1992] Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press Edition, 1992.
- [Hölldobler and Wilson, 1990] Hölldobler, B. and Wilson, E. O. (1990). *The Ants*. Springer-Verlag.
- [Hornby and Pollack, 2001] Hornby, G. S. and Pollack, J. B. (2001). The advantages of generative grammatical encodings for physical design. In *Congress on Evolutionary Computation (CEC)*, pages 600–607. IEEE Press.
- [Huynen, 1996] Huynen, M. (1996). Exploring phenotype space through neutral evolution. *Journal of Molecular Evolution*, 43.
- [Kauffman, 1993] Kauffman, S. A. (1993). *The Origins of Order: Self-Organisation and Selection in Evolution*. Oxford University Press.
- [Keijzer, 2003] Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 71–83, Essex. Springer-Verlag.
- [Keijzer et al., 2001] Keijzer, M., Ryan, C., O'Neill, M., Cattolico, M., and Babovic, V. (2001). Ripple crossover in genetic programming. In Miller, J., Tomassini, M., Lanzi, P. L., Ryan, C., Tetamanzi, A. G. B., and Langdon, W. B., editors, *Proceedings of the Fourth European Conference on Genetic Programming (EuroGP-2001)*, volume 2038 of *LNCS*, pages 74–86, Lake Como, Italy. Springer Verlag.
- [Keller and Banzhaf, 1999] Keller, R. E. and Banzhaf, W. (1999). The evolution of genetic code in genetic programming. In W. Banzhaf, J. Daida, A. E. M. G. V. H. M. J. and Smith, R., editors, *first GECCO conference*, pages 1077–1082, San Francisco. Morgan Kaufmann.
- [Kennedy and Eberhart, 2001] Kennedy, J. and Eberhart, R. (2001). *Swarm Intelligence*. Morgan Kaufmann.
- [Kim and Bentley, 2001] Kim, J. and Bentley, P. J. (2001). Investigating the roles of negative selection and clonal selection in an artificial immune system for network intrusion detection. *Special Issue on Artificial Immune Systems in IEEE Transactions of Evolutionary Computation*.

- [Kimura, 1983] Kimura, M. (1983). *The Neutral Theory of Molecular Evolution*. Cambridge University Press.
- [Kitano, 1990] Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4.
- [Kolmogorov, 1965] Kolmogorov, A. N. (1965). Problems of information transmission.
- [Koza, 1994] Koza, J. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.
- [Kubrik, 2003] Kubrik, A. (2003). Towards a formalisation of emergence. *Artificial Life*, 9:41–65.
- [Kuo et al., 2004] Kuo, P. D., Leier, A., and Banzhaf, W. (2004). Evolving dynamics in an artificial regulatory network model. In Yao X., Burke E., L. J. S. J. M.-G. J. B. J. R. J. T. P. K. A. S. H.-P., editor, *Parallel Problem Solving from Nature Conference (PPSN-04)*, pages 571–580, Birmingham, UK. Springer, Berlin.
- [Lehre and Haddow, 2003] Lehre, P. and Haddow, P. (2003). Developmental mappings and phenotypic complexity. In *Proceedings of CEC*, pages 62–68.
- [Lewin, 1999] Lewin, B. (1999). *Genes VII*. Oxford University Press.
- [Lindsay and Wu, 1996] Lindsay, R. K. and Wu, A. S. (1996). Testing the robustness of the genetic algorithm on the floating building block representation. In *13th National Conference on Artificial Intelligence*, Orlando.
- [Lones, 2004] Lones, M. A. (2004). *Enzyme Genetic Programming: Modelling Biological Evolvability in Genetic Programming*. PhD thesis, University York, York, UK.
- [Lones and Tyrrell, 2004a] Lones, M. A. and Tyrrell, A. M. (2004a). Crossover and bloat in the functionality model of enzyme genetic programming. In *2002 World Congress on Computational Intelligence*. IEEE Press.
- [Lones and Tyrrell, 2004b] Lones, M. A. and Tyrrell, A. M. (2004b). Modelling biological evolvability. implicit context and variation filtering in enzyme genetic programming. *Biosystems*, 76(1-3).
- [Nehaniv et al., 2002] Nehaniv, C. L., Polani, D., and Dautenhahn, K. (2002). Meaningful information, sensor evolution, and the temporal horizon of embodied organisms. pages 345–349. MIT Press.

- [Nicosia et al., 2004] Nicosia, G., Cutello, V., Bentley, P. J., and Timmis, J., editors (2004). *Third Interational Conference on Artificial Immune Systems ICARIS2004*. Springer.
- [Noest, 2001] Noest, A. J. (2001). T cells obey the tenets of signal detection theory. In Segal, L. A. and Cohen, I. R., editors, *Design Principles for The Immune System and Other Autonomous Systems*, chapter 8, pages 185–202. Oxford University Press, New York.
- [O’Neill and Brabazon, 2004] O’Neill, M. and Brabazon, A. (2004). Grammatical swarm. In *GECCO (1)*, pages 163–174.
- [O’Neill and Brabazon, 2005] O’Neill, M. and Brabazon, A. (2005). mGGA: The meta-grammar genetic algorithm. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J. I., and Tomassini, M., editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 311–320, Lausanne, Switzerland. Springer.
- [O’Neill et al., 2004] O’Neill, M., Brabazon, A., Nicolau, M., McGarraghy, S., and Keenan, P. (2004). pi-grammatical evolution. In *GECCO (2)*, pages 617–629.
- [O’Neill and Ryan, 2003] O’Neill, M. and Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. KluwerAcademic Publishers.
- [O’Neill and Ryan, 2004] O’Neill, M. and Ryan, C. (2004). Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In Keijzer, M., O’Reilly, U.-M., Lucas, S. M., Costa, E., and Soule, T., editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 138–149, Coimbra, Portugal. Springer-Verlag.
- [Oprea, 1999] Oprea, M. (1999). *Antibody Reportoires and Pathogen Recognition: The Role of Germline Diversity and Somatic Hypermutation*. PhD thesis, University of New Mexico.
- [Parunak and Brueckner, 2001] Parunak, H. V. D. and Brueckner, S. (2001). Entropy and self-organisation in multi-agent systems. In *Proceedings of Fifth International Conference on Autonmous Agents*, pages 124–130. ACM Press.
- [Peitgen et al., 2004] Peitgen, H.-O., Jürgens, H., and Saupe, D. (2004). *Chaos and Fractals: new frontiers of Science 2nd. edition*. Springer New York.
- [Pelikan, 2005] Pelikan, M. (2005). *Hierarchical Bayesian Optimization Algorithm Toward a New Generation of Evolutionary Algorithms*, volume 170. Springer.

- [Pelikan et al., 1999] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL. Morgan Kaufmann Publishers, San Fransisco, CA.
- [Pelikan and Mühlenbein, 1998] Pelikan, M. and Mühlenbein, H. (1998). Marginal distribution in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel '98*, pages 90–95, Brno, Czech Republic.
- [Pelikan and Mühlenbein, 1999] Pelikan, M. and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., and Chawdhry, P. K., editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London. Springer-Verlag.
- [Piaseczny et al., 2004] Piaseczny, W., Suzuki, H., and Sawai, H. (2004). Chemical genetic programming - coevolution between genotypic strings and phenotypic trees. In *GECCO (2)*, pages 715–716.
- [Repast, 2004] Repast (2004). <http://repast.sourceforge.net>, Social Science Research Computing University of Chicago.
- [Rieffel and Pollack, 2004] Rieffel, J. and Pollack, J. (2004). The emergence of ontogenic scaffolding in a stochastic development environment. In *GECCO*, Seattle, Washington, USA.
- [Rosen, 1985] Rosen, R. (1985). *Anticipatory Systems: Philosophical, Mathematical and Methodological Foundations*. Pergamon Press.
- [Ryan et al., 2002] Ryan, C., Azad, A., Sheahan, A., and O’Neill, M. (2002). No coercion and no prohibition, a position independent encoding scheme for evolutionary algorithms: The chorus system. pages 131–142.
- [Sastry and Goldberg, 2003] Sastry, K. and Goldberg, D. E. (2003). Probabilistic model building and competent genetic programming. Technical Report 2003013, IlliGAL.
- [Schaffer, 1985] Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In Grefenstette, J. J., editor, *First International Conference on Genetic Algorithms and their Applications*, pages 93–100.
- [Shan et al., 2004] Shan, Y., McKay, R. I., Baxter, R., Abbass, H., Essam, D., and Nguyen, H. (2004). Grammar model-based program evolution. In *Congress on Evolutionary Computation*, Portland, USA.

- [Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423.
- [Streichert et al., 2003] Streichert, F., Spieth, C., Ulmer, H., and Zell, A. (2003). Evolving the ability of limited growth and self-repair for artificial embryos. In Banzhaf, W., Christaller, T., Dittrich, P., Kim, J. T., and Ziegler, J., editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life*, volume 2801 of *LNAI*, pages 289–298, Dortmund, Germany. Springer Verlag.
- [Suzuki and Sawai, 2003] Suzuki, H. and Sawai, H. (2003). Chemical genetic algorithms: coevolution between codes and code translation. In *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, pages 164–172, Cambridge, MA, USA. MIT Press.
- [Volkert, 2003] Volkert, L. G. (2003). Enhancing evolvability with mutation buffering mediated through multiple weak interactions. *Biosystems. Special Issue on Evolvability*, 69(127-142).
- [Volkert and Conrad, 1997] Volkert, L. G. and Conrad, M. (1997). The effect of weak interactions on the structure of adaptive surfaces. *Soft Computing*, 1(148–154).
- [Volkert and Conrad, 1998] Volkert, L. G. and Conrad, M. (1998). The role of weak interactions in biological systems: the dual dynamics model. *Journal of Theoretical Biology*, 193(287–306).
- [W3C, 2005a] W3C (Accessed 2005a). *eXtensible Markup Language (XML)*. <http://www.w3.org/XML/>.
- [W3C, 2005b] W3C (Accessed 2005b). *Extensible Stylesheet Language Transformations(XSLT)*. <http://www.w3.org/TR/xslt>.
- [W3C, 2005c] W3C (Accessed 2005c). *Simple Object Access Protocol (SOAP)*. <http://www.w3.org/TR/soap/>.
- [W3C, 2005d] W3C (Accessed 2005d). *XML Path Language (XPath)*. <http://www.w3.org/TR/xpath>.
- [W3C, 2005e] W3C (Accessed 2005e). *XML Schema*. <http://www.w3.org/TR/xmlschema-1/>.
- [Wagner and Altenberg, 1996] Wagner, G. P. and Altenberg, L. (1996). Complex adaptations and the evolution of evolvability. *Evolution*, 50(3).

- [Watson and Pollack, 1999] Watson, R. A. and Pollack, J. B. (1999). Incremental commitment in genetic algorithms. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 710–717, Orlando, Florida, USA. Morgan Kaufmann.
- [Watson and Pollock, 2003] Watson, R. A. and Pollock, J. B. (2003). A computational model of symbiotic composition in evolutionary transitions. *Biosystems*, 69(2-3):187–209.
- [Weberndorfer et al., 2003] Weberndorfer, G., Hofacker, I., and Stadler, P. (2003). On the evolution of primitive genetic codes. *Origins Life Evol. Biosph*, 33(491-514).
- [Wiedermann and Van Leeuwen, 1999] Wiedermann, J. and Van Leeuwen, J. (1999). Emergent computational potential of evolving artificial living systems. *AI Communications* 15, pages 205–215.
- [Williams, 2002] Williams, H. (2002). Spatial organisation of a homogenous agent population using diffusive signalling and role differentiation. Master’s thesis, University Sussex.
- [Wilson, 1971] Wilson, E. O. (1971). *The Social Insects*. Harvard University Press.
- [Wolfram, 1983] Wolfram, S. (1983). Statistical mechanics of cellular automata. *Review of Modern Physics*, pages 601–644.
- [Wu and De Jong, 1999] Wu, A. S. and De Jong, K. A. (1999). An examination of building block dynamics in different representations. In *Congress on Evolutionary Computation*, volume 87, pages 715–721.
- [Wu and Garibay, 2002] Wu, A. S. and Garibay, I. I. (2002). The proportional genetic algorithm: Gene expression in a genetic algorithm. 3(2).
- [Wu and Lindsay, 1996] Wu, A. S. and Lindsay, R. K. (1996). A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation*, 4(2):169–193.
- [Wu and Stringer, 2002] Wu, A. S. and Stringer, H. (2002). Learning using chunking in genetic algorithms. In *Proceedings of the 11th Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL, pages 243–254.

- [Wu and Stringer, 2004] Wu, A. S. and Stringer, H. (2004). Winnowing wheat from chaff: The chunking ga. In *Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, Seattle, WA*, pages 198–209. Springer-Verlag LNCS Series.
- [Zipf, 1965] Zipf, G. K. (1935/1965). *Psycho-Biology of Languages*. Cambridge, MA: MIT Press (first published by Houghton Mifflin).