A Hierarchically Organised Genetic Algorithm for Fuzzy Network Synthesis

Enrique Rafael Filloy-García

Ph.D.
University of Edinburgh
2002

Abstract

A hierarchical, two-level genetic algorithm to produce the rules for a fuzzy system is proposed. The underlying architecture of fuzzy networks corresponds with the structured, two-level representations used. At one level, a variable-length structure was designed to represent entire rule sets as individuals in a population; at a lower level, another population contains elements which represent single fuzzy rules. The two populations co-evolve simultaneously in an interdependent fashion. This method has been shown to be capable of producing effective fuzzy systems of an adequate size for particular classes of problems; examples of a control task and a classification problem are shown. Suitable replacement strategies for the elements population were devised, introducing the definition of a heredity factor. Additionally, means for the adaptation of system parameters like cut & splice probabilities were developed to further enhance system performance.

Acknowledgements

I want to express my deepest gratitude to Prof. Peter Ross, who has been incredibly patient with me. I hope I managed the exploitation of a vanishingly small portion of the vast knowledge space he provided me with to explore.

Many people I know wanted to see this thesis finished, but no one as badly as Jacqueline. Her comments were of invaluable help, although I am sure *my* English is still quite perceptible throughout. So, *muy, pero muy especialmente a JeeJee...*

This thesis is dedicated to Rosi & Leo.

My studies towards a PhD were partly funded by Mexico's *Consejo Nacional de Ciencia y Tecnología* (CoNaCyT).

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Enrique Rafael Filloy García
Edinburgh
February 20, 2002

Contents

List of Figures

Chapter 1

Introduction

This chapter presents the aims and underlying motivations, within the context of the Evolutionary Computation (EC) field, together with preliminary comments and an outline of the thesis.

1.1 Motivations

The main inspiration for this work stems from research in the fields of evolutionary computation, fuzzy logic and the possible interactions between the two. In the following sections, a brief introduction to the fundamental issues involved is presented in broad terms; detailed accounts of the most relevant topics will be given in subsequent introductory chapters.

1.1.1 Evolution and Adaptation

> "What limit can be put to this power, acting during long ages and rigidly scrutinising the whole constitution, structure, and habits of each creature, — favouring the good and rejecting the bad? I can see no limit to this power, in slowly and beautifully adapting each form to the most complex relations of life" [Darwin 59].

Scientific thinking has created many great ideas that, however complex, once the essential meaning has been understood, even if some of the details remain unclear to us, they are capable of changing the way we think about nearly everything that is around us.

Good examples are the different theories about fundamental issues in physics or chemistry, even abstract mathematics, which are learnt during the early stages of academic education. However, undoubtedly one of the greatest theories that scientific work has ever produced, is often dismissed, misinterpreted or simply taken for granted: the idea of evolution by cumulative selection.

Like in many other fields, the possibility to simulate natural processes in the computer has attracted many scientists to experiment and observe virtual evolutionary systems. Computer algorithms that implement simplified versions of natural phenomena often make it possible to identify new, distinct aspects of the observed systems. Many — but not all — computer simulations share a particular advantage over their natural counterparts, which in the case of EC turns out to be absolutely crucial: *speed*. "Darwinism is a theory of cumulative processes so slow that they take between thousands and millions of decades to complete. All our intuitive judgements of what is probable turn out to be wrong by many orders of magnitude. Our well-tuned apparatus of scepticism and subjective probability-theory misfires by huge margins, because it is tuned — ironically, by evolution itself — to work within a lifetime of a few decades" [Dawkins 91]. Computer implementations of evolutionary systems, however demanding in computational terms, certainly help "the imagination to escape from the prison of familiar timescale", without the need to wait for too long to observe and identify the interesting and potentially useful properties.

As the above quote from Darwin explicitly indicates, an intimately related concept to evolution is that of *adaptation*. In fact, John Holland's book [Holland 75] ([Holland 92]) is often considered the seminal work that initiated the field of study in Genetic Algorithms (GAs) and, more generally, in complex adaptive systems (cas) ([Holland 95]).

The fundamental concepts behind the theories of evolution and adaptation seem to extend far beyond the biological domain. It is now possible to design algorithms that exploit the main features of evolutionary processes, often in oversimplified versions, producing extraordinary results for specific kinds of problems. These issues are discussed in chapter 2. "Evolutionary algorithms are manifestly interesting algorithms — interesting to us, at least — not because what they are guaranteed to do is interesting to us, but because what they are guaranteed to *tend* to do is interesting to us" [Dennett 95]. Moreover, research in EC has started to produce significant insights into the abstract properties of this kind of algorithms; it is now possible to begin to explore aspects of evolutionary systems that were not directly observable in nature.

1.1.2 Function Estimators

Intelligent behaviour can be characterised in many different ways, so it is not surprising that machine intelligence has been approached accordingly, from several different angles. One such approach, that bears particular relevance to the present work, is based on "the single abstract property of adaptive model-free function estimation: *Intelligent systems adaptively estimate continuous functions from data without specifying mathematically how outputs depend on inputs*" [Kosko 92].

Even if the function, or behaviour, of a device is known, it is not possible to deduce from this a unique structural description. However, given the external description of the behaviour of a system, mathematical theory can provide the simplest internal structure that could yield the observed behaviour ([Arbib 87]). Although it may be too hard to explicitly program the behaviour seen in a black box, it may be possible to drive a system by the actual input-output behaviour of that box, or by some description of its trajectories, to cause it to adapt itself into a system with (approximately) that given behaviour.

As will be described in greater detail in chapter 3, it is possible to train fuzzy systems to "learn" input-output associations, later showing the capability to generalise and give an adequate response to unseen stimuli. The training process may be cast as one of search or optimisation which, in turn, may be carried out by an evolutionary process. This idea sets the basic framework for this thesis.

1.1.3 Soft Computing

The term *soft computing* (SC) was recently introduced in [Zadeh 94]. Closely related to the dichotomy between symbolic and sub-symbolic approaches to artificial intelligence (AI), SC is obviously regarded as the opposite alternative to *hard computing*.

In the former, methods that have an inherent capacity to deal with uncertainty and imprecision, often relying on stochastic algorithms, are used to find adequate solutions at a very low cost in terms of relative calculation effort. In the latter, on the other hand, only models using classical logic and based on formal methods are employed, providing precise but brittle solutions, normally involving a high computational cost. For several interesting classes of problems, the burden of deterministic accuracy becomes prohibitive, even relying on an increasing degree of parallelism. Frequently, a task is too difficult to acquire sufficient and adequate knowledge for its solution; traditional AI techniques are therefore not applicable and even the augmentation of a general problem solver by task-specific knowledge is not feasible.

Formalising knowledge in soft constraints rather than hard rules has important consequences. Hard constraints have consequences singly; they are context-independent rules that can be applied separately, sequentially, irrespective of whatever other rules may exist. But soft constraints have no implications singly; any one can be overridden by the others. It is only the entire set of soft constraints that has any implications. Inference must be a cooperative process, like the parallel relaxation processes typically found in subsymbolic systems. Furthermore, adding additional soft constraints can repeal conclusions that were formerly valid. Subsymbolic inference is therefore fundamentally non-monotonic ([Smolensky 86]).

SC can be seen as a non-traditional approach to increase machine intelligence, where methods from evolutionary computation, fuzzy logic and neural networks are used, often in combination. Each of the three areas of SC has its advantages and disadvantages, but the weaknesses of one seem to be *naturally* complemented by the strengths of at least one of the other two: the ability to deal with uncertainty and to express knowledge using linguistic representations of fuzzy rules, may be complemented with the robustness and learning capabilities of connectionist systems, or the effective, globally-oriented search performed by an evolutionary process. The most important issues and related relevant work are described in chapter 4.

1.2 Objectives

What is known about the principles of evolution can effectively be used to build adaptive computational systems. GAs are a subclass of methods performing evolutionary computation, commonly applied in search and optimisation problems. The principles that lie behind the operation of a generic GA are allegedly simple and yield robust, general, adaptive and efficient systems. For certain kinds of problems, however, the generic GA is either practically incapable of finding a suitable solution or, when it succeeds, its performance is poor in comparison with other techniques. Furthermore, designing a GA for a particular problem, or analysing its behaviour and properties, is usually not such an easy task. Normally there are a large number of possibilities to explore at nearly every step in the GAs' design/analysis process — e.g. representation (coding) schemes, selection procedures, combination (reproduction) methods, replacement strategies, fitness evaluation, etc. Most of the variations on each of these are the result of different attempts to overcome limitations in the basic model and its operators.

The main goal of this research work is to investigate the possibilities of using *higher order representations and transformations* through a multi-level organisation of the overall GA structure. One particular aim is to explore the implications of adopting hierarchical representation schemes, where the customary operations (e.g. selection, reproduction, mutation, etc.) perform the usual (simple) transformations but at different levels of the representational structure. The purpose is to provide the evolutionary process with the necessary means to perform an effective search, especially in a particular kind of complex problem space, without the need to design more elaborate, often problem-dependent, operators. Certain aspects of fundamental GA theory will be addressed, but maintaining an essentially pragmatic stance.

Initially, the synthesis of *fuzzy networks* was chosen as a problem domain for several reasons. One of them is that the underlying architecture of such networks lends itself very well to the application of structured (yet flexible) multi-level representations: each individual (i.e. rule set) is in turn composed of a variable number of smaller units (i.e. rule-like entities). Also, the interaction between evolutionary computation and fuzzy systems seems to be a promising source of new ideas with immediate application to real problems, particularly in the design of controllers and classification systems.

An important objective was not only to identify and describe relevant relations between a specific representation and the performance and operation of a hierarchically organised GA, but also to obtain suitable fuzzy systems efficiently. To this effect, several new variations on the basic model were introduced. This has been done in as general a way as possible, although a marked bias towards the aspects of integrating this kind of GA and systems based on fuzzy rules was unavoidable.

1.3 Main Contributions

Below is a summary listing the main contributions of this thesis; see chapters 7 and 8 for a complete discussion on these topics:

- hierarchically organised GA
- variable-length representation for fuzzy networks
- suitable replacement strategies for the elements populations
- definition of a *heredity* factor
- means for the automatic adaptation of heredity, cut & splice probabilities

1.4 Outline of the Thesis

Chapter 2: Genetic Algorithms presents first a revision of GAs in general, then concentrates on the unconventional aspects that were adopted in this work.

Chapter 3: Fuzzy Logic contains a broad presentation of fuzzy systems, introducing the basic concepts and practical considerations. Particular emphasis is placed on rule acquisition and design issues.

Chapter 4: Hybrid Systems reviews the current approaches to the combined use of fuzzy systems and genetic algorithms, pointing out the most relevant previous research that motivated and influenced this work.

Chapter 5: 2LGA Described presents in detail the proposed architecture and the operation of a hierarchical GA of a new kind. The basic building blocks this work is based upon are introduced and an overall picture of how all the pieces fit together is given. Special attention is paid to the more fundamental aspects and unique features of the system.

Chapter 6: 2LGA at Work shows the results of the application of 2LGA to control and classification problems. Several variations on the basic model are introduced, both to enhance the performance of the system and to investigate different design alternatives.

Chapter 7: 2LGA Analysed examines several considerations regarding the multi-level organisation of a GA. Observations are based on results obtained from experiments.

Chapter 8: Summary and Conclusions contains a summary of achievements, together with a discussion of the contribution of this thesis and the issues raised herein. Some possible directions for future work are also included.

1.5 Disclaimer

Terminology

It is easy to abuse the terminology when working with systems inspired by analogies to more familiar, natural phenomena — the word *neuron* is a good example. In this work, as is the case in practically all the evolutionary computation literature, several names for system entities and operations such as *gene, chromosome, parents, offspring, mating, reproduction,* etc. were adopted. It should be clear that the use of these terms is intended to facilitate the description of abstract properties of the system and the elements involved; those names were intuitively chosen only because of the relative role these elements play.

No Free Lunch

The so-called *No Free Lunch* (NFL) theorems have recently generated much controversy in the EC community ([Wolpert & Macready 95], [Macready & Wolpert 95]). These theorems indicate that there is no difference in performance, for all algorithms used to search for an optimum solution of a cost function, when averaged over all possible cost functions. "In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist as many other functions where B outperforms A" [Wolpert & Macready 95].

As stated in [Culberson 96], "The NFL theorems make it clear that any [claim about an algorithm outperforming all others] is inherently false unless it is coupled with a disclaimer such as 'under the prescribed assumptions' or 'on the functions tested'. In particular, no such claim can ever be made with respect to all possible functions, or even sufficiently large classes of functions". The main purpose of the work presented in this thesis is to study a particular system, analysing the most interesting features it seems to exhibit; certainly no claim about its superiority when compared against *all* other algorithms, with respect to *all* possible functions, will be made.

Philosophical Omissions

Several different themes that are often considered controversial, especially regarding fuzziness and Darwinism, were intentionally left out of this work, even when they bear implicit relevance to the material presented in the following chapters. "But there is no such thing as philosophy-free science; there is only science whose philosophical baggage is taken on board without examination" [Dennett 95]. Unfortunately, a proper discussion of the philosophical issues involved is certainly outside the scope of the present thesis. [Dennett 95] and [Kosko 94] give particular accounts of some of the fundamental questions and controversies concerned.

Chapter 2

Genetic Algorithms

This chapter presents a general review of genetic algorithms. The emphasis is placed on elements and ideas that are most relevant to this thesis, paying special attention to the origins and prior knowledge of the unconventional aspects that were developed in this research.

2.1 Overview

Genetic algorithms (GAs) have been used in science and engineering as adaptive processes for solving practical problems, normally formulated in search or optimisation terms, and as computational models of natural evolutionary systems. GAs are non-deterministic, adaptive and robust computational methods; they are built upon rough analogies to natural processes which are based on the principle of evolution by cumulative selection.

Several evolutionary approaches were proposed in the late fifties ([Box 57], [Fraser 57], [Friedberg 58], [Friedman 59]), during the sixties ([Bremermann 62], [Fogel *et al.* 66], [Rosenberg 67]) and in the early seventies ([Weinberg 70], [Cavicchio 70], [Rechenberg 73]); see [Fogel 98] for an excellent compilation of the major historical events that account for the evolution of the field. However, it is generally assumed that John H. Holland ([Holland 75]) set the stage for the development of research in GAs as such. Since then, interest from different research communities in the field has been growing rapidly, as reflected by the increasing number of conferences and articles on related subjects.

> "A major reason for this interest is that GAs really work. GAs offer robust procedures that can exploit massively parallel architectures and, applied to classifier systems, they provide a new route toward an understanding of intelligence and adaptation." John H. Holland, from the Foreword in [Goldberg 89c].

As pointed out by [Dennett 95]: "The 'genetic algorithms' devised by John Holland [...] have demonstrated their power in the no-nonsense world of software development and have mutated into a phylum of algorithmic variations." There are three distinct paradigms in which evolutionary computation approaches are normally classified ([Fogel 93], [Bäck & Schwefel 93], [DeJong 96]):

Genetic Algorithms The simple GA was introduced in [Holland 75] as an adaptation process; it was first analysed in [DeJong 75] using a set of optimisation tasks. It is an abstract model of evolution, characterised by a population of individuals represented by fixed-length binary strings. Population size and operator parameters remain constant. Of special importance is the total absence of any problem-specific features, both in the representation and in the operators used, which normally comprise some form of recombination as a main operator and bit-wise mutation as a background operator.

Evolution Strategies First suggested by [Rechenberg 73] and further developed by [Schwefel 75], Evolution Strategies (ES) use integer and real representations for numerical optimisation. Individuals are not only represented by the set of objective variables; they also include a set of strategic parameters which control certain aspects of the evolutionary process, i.e. variances and covariances.

Evolutionary Programming Initially proposed by [Fogel *et al.* 66] as an abstraction of behavioural processes, Evolutionary Programming (EP) emphasises the view of evolution as the adaptation and diversity of behaviours, rather than the cumulative selection of increasingly fitter building blocks. It was first used to evolve finite state machines to solve prediction tasks. [Fogel & Atmar 92] formulated EP's current form, relying upon real variables both for the object variables and the strategy parameters, which are adapted according to exogenous rules.

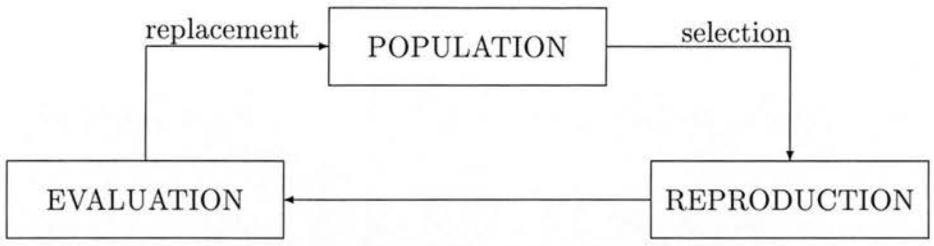
Surprisingly, very different and often contrasting design principles are traditionally emphasised by each research community. However, it would be difficult to draw a defining line between them without assuming a simplified characterisation of each one of the approaches. More recent systems normally incorporate techniques drawn from different algorithms. Very often, unless stated otherwise, the name GA is used in the broadest sense of the term, closely related to the *evolution program* concept, as defined in [Michalewicz 92]:

> "In [algorithms based on the principle of evolution (survival of the fittest)] a population of individuals (potential solutions) undergoes a sequence of unary (mutation type) and higher order (crossover type) transformations. These individuals strive for survival: a selection scheme, biased towards fitter individuals, selects the next generation. After some number of generations, the program converges — the best individual hopefully represents the optimum solution."

Several books and papers have been devoted to the presentation of the fundamental aspects of GAs. [Goldberg 89c] is perhaps the most cited general reference on the subject and is often regarded as the basic textbook; [Davis 91] and [Michalewicz 92] offer more pragmatic treatments, including examples of GA applications for specific classes of problems. [Mitchell 95] provides an accessible introduction, focusing particularly on machine learning, scientific modelling and the connections with artificial life. [Beasley *et al.* 93a] and [Beasley *et al.* 93b] present a succinct overview, describing the fundamental aspects of GAs; how they work, theoretical and practical issues, together with a short survey of research topics.

2.2 The Principles

The basic structure of a generic GA can be assumed to be as simple as the one shown in the following figure:



This elementary model depicts the essential components of a GA and the minimal interactions between them. First, it is necessary to represent a set or **population** of potential solutions. By means of a *selection* process, the relatively better individuals are chosen from the population and used to generate new individuals. Normally two kinds of operators are involved in this **reproduction** process: recombination and mutation. The fitness or merit of each new solution candidate is assessed using an **evaluation** function or method. Finally, the new individuals are introduced into the population, commonly adhering to a *replacement* strategy. This cyclic process, symbolising a *generation*, is iterated until a particular stopping criteria is met.

There are an increasing number of variations on each one of the components mentioned above and in the ways they interact; a comprehensive review of all of them is beyond the scope of this thesis. The first part of the following presentation focuses on the most significant features of GAs in general, while the later sections introduce the particular aspects that bear special relevance to this work.

2.3 Representation

As is the case with most problem solving techniques, an appropriate representation for *potential* solutions must be designed *a priori* for each particular task. Although the actual details of the final solution are obviously not known, it is necessary to define what it must be made of. The following definitions, inherited from the basic ideas of genetics, are generally used in the field:

Gene: the representation of a single parameter

Chromosome: the set of genes comprising a complete solution candidate

Genotype: the underlying set of parameters represented by a chromosome

Phenotype: the particular solution as determined by a genotype

For a standard GA, a potential solution normally consists of a predetermined set of parameters, described by the concatenation of their coded values. GAs work on a *population* of *individuals*, i.e. chromosomes, each one representing a candidate solution. These individuals traditionally consist of a fixed-length, binary coding of the parameters involved, joined together on a linear structure. For example, assuming the task consists in minimising a function of two variables $F(x, y)$, each variable represented by a 16-bit binary number, every chromosome would then contain two genes and consist of 32 binary digits.

The distinction between genotype and phenotype becomes apparent when the set of parameters, defined by the former, does not constitute a direct representation of the solution. Instead, these parameters are used to build the latter, following a specific development process, thus generating one possible solution originating from that particular set of parameters.

The choice of representation plays a critical role in GA design. There are many issues involved, both practical and theoretical. Once the other basic elements have been introduced, the most relevant implications concerning GA representations will be discussed in section 2.10 below.

2.4 Fitness Evaluation

Fitness is a crucial concept in evolutionary systems. It corresponds to a measure of the ability or utility of a particular individual to solve a specific problem. In GAs, this is normally evaluated using a fitness function which, depending on the problem, might range from a simple calculation, to multi-objective performance measures, to complex simulations. In any case, the obtained value should be proportional to the fitness of the individual to act as a potential solution to the problem, for this is the main indication the GA is going to use in order to follow an evolutionary process. In fact, once the coding scheme has been defined for a canonical GA, the fitness value is the *only* piece of information it needs to operate on a particular task, treating it like a black box. Unlike the commonly used hill-climbing search techniques, traditional GAs do not require explicit gradient information. Improved individuals are selected solely by their objective function, hence, the procedure is completely general. Unfortunately, the *real* value of a chromosome is not always easily or accurately assessed by a single measure. In some cases, it can be very difficult to provide a useful single quantity to guide genetic search.

Obviously, fitness evaluation is tightly related to the choice of representation, which specifies the space to be searched. If the fitness function is applied to every point in the search space, the hypersurface defined is known as the fitness *landscape*, which is actually explored by the algorithm. Together, the search space and fitness landscape provide an *environment* for the GA. As for any other search method, smooth and regular landscapes that do not have too many local maxima nor a very isolated global maximum, i.e. a needle in a haystack, are obviously preferred. Often, however, a "harsh" environment is unavoidable, perhaps inherent to the problem itself, but an adequate choice of representation/fitness evaluation combination can make a big difference and ease the GA task.

2.4.1 Performance Evaluation

Deciding when to stop or re-initiate the GA search can be based on a simple and deterministic strategy, e.g. after a fixed number of generations, or it might involve a more complex calculation, taking into account population diversity and average fitness statistics, for example. In certain situations, it makes sense to assess the performance of the algorithm based solely on the fitness values of the solutions obtained, but this is not always the case. [DeJong 75] defined two measures to quantify the effectiveness of a particular GA:

On-line performance $x_e(s)$ gives an indication of the ongoing performance, for a particular strategy s on environment e, comprising the average of all function evaluations up to and including the current trial T:

$$x_e(s) = \frac{1}{t} \sum_{t=1}^{t=T} f_e(t),$$

where $f_e(t)$ is the objective function value for environment e on trial t

Off-line performance $x_e^*(s)$ is aimed at measuring convergence (see below). It involves a running average of the best performance values to a particular time:

$$x_e^*(s) = \frac{1}{T} \sum_{t=1}^{t=T} f_e^*(t),$$

where $f_e^*(t) = \text{best } \{f_e(1), f_e(2), ..., f_e(t)\}$.

As explained in [Goldberg 89c], "The names off-line and on-line refer to the difference in emphasis between off-line and on-line applications. In an off-line application, many function evaluations can be simulated and the best alternative so far saved and used after the achievement of some stopping criteria. An on-line application does not afford this luxury and function evaluations are achieved through real experimentation on line; as a result, a premium is placed on getting to acceptable performance quickly."

2.5 Populations and Generations

Although the fundamental cyclic process prevails, several different global strategies have been developed for GAs. A distinguishing feature is the number of individuals that are directly affected on each iteration of the algorithm. The proportion of individuals in the population that are replaced in each generation is called the *generation gap*, as introduced by [DeJong 75]. There are obviously two extreme situations: 1) the maximum number of individuals in a population, i.e. all of them, are replaced in each generation, or 2) the small number of individuals created during the reproduction stage, typically two, take the place of a number of individuals from the previous generation. This leads to two different general strategies for dealing with the concept of a "generation" in GA populations, called *generational* and *steady-state*, respectively.

2.5.1 Generational GA

In the first case, when the whole population is replaced on each cycle, no breeding occurs between individuals of different generations: the generation gap is 1. This is called a *generational* GA, characterised by non-overlapping populations. Most of the early systems operated in this way, probably influenced by De Jong's suggestions; this approach is still very much in use in optimisation tasks, perhaps for the same reason.

2.5.2 Steady-state GA

In the second case, the new individuals, resulting after the reproduction process has taken place, are introduced into the population straight away. They normally replace an equal number of "old" individuals and are thus available for reproduction in the next generation. This is known as a *steady-state* GA, characterised by overlapping populations; an early example can be found in the G*ENITOR* system ([Whitley 89]). Occasionally the need for a particular replacement strategy becomes evident under this scheme. In fact, selection and replacement methods are often very closely related (see section 2.6 below).

The main difference between the steady-state and generational approaches is that, in the former, the new offspring are immediately available for reproduction. Purportedly, this characteristic would allow the GA to exploit a promising new combination of genes as soon as it is produced. Although this might intuitively seem like a desirable feature, no conclusive evidence has been presented to this effect so far ([Goldberg & Deb 91]). Obviously, it is also possible to use an intermediate generation gap. [Schwefel 81] introduced, in the context of Evolution Strategies (ES), the idea of generating λ offspring that compete for survival, either amongst themselves or together with the whole population, preserving at the end of each generation a total of μ individuals. Two examples of ES are the $(\mu + \lambda)$-ES and (μ, λ)-ES. In the first one, μ parents produce λ offspring; the population is then reduced again to μ parents by selecting the best solutions from among both the parents and the offspring. Thus, parents survive until they are replaced by better solutions. The (μ, λ)-ES is closer to the generational model used in canonical GAs; offspring replace parents and then undergo selection. Recombination operators for ES also tend to differ from from the typical crossover, allowing operations such as averaging parameters, for example, to create an offspring.

2.5.3 Islands and Migration

Not only can the basic components interact in a plurality of ways, but the various subparts that make them up can also be specifically arranged, displaying diverse forms of interrelation. It is common to have not just one population, but a small number of them. Individuals evolve in isolation within their respective population for several generations. When a predetermined condition is met, e.g. the system has evolved for a certain number of generations, selected individuals are allowed to *migrate* between populations. This is known as the *island* model, because partial geographical isolation in nature is simulated by using multiple subpopulations and intermediate migration rates ([Grosso 85]). This scheme is closely related to some of the object-based models of parallel GAs proposed by Goldberg ([Goldberg 89c]), and has special bearing on the concepts of niche and speciation discussed below (section 2.6.3).

2.6 Selection and Replacement

From biology, the idea of *survival of the fittest* is transferred to the GA analogy by means of selection and replacement methods. In broad terms, the aim is to provide the mechanisms through which highly fit individuals may acquire more opportunities to survive longer and reproduce more often, so that they have better chances of disseminating their valuable genetic material.

There are many different selection schemes and variations thereof. The actual implementation also depends on the choice of generation gap, but the principle is the same: some individuals must be chosen for reproduction. This is done in such a way that relatively fitter chromosomes are more likely to take part in reproductive trials. Basic theory (see section 2.8.1) says that mating opportunities should be allocated to individuals in direct proportion to their relative fitness. However, because actual GAs operate on finite populations, selection strategies must be modified in order to avoid *premature convergence*. One of the main problems faced by GAs is that it is possible to have a few individuals that, although being far from optimal, have a comparatively very high fitness. They may rapidly spread their genetic makeup, which will come to dominate the entire population. As soon as the GA converges to this local maximum, the effectiveness of recombination operators is severely hampered. The ability to continue and advance with the search relies almost exclusively on mutation operators, which at this stage will perform little else than slow random exploration.

It is possible to select parents according to fitness and replace old individuals by inverse fitness, although sometimes either one or the other is performed randomly. It is obvious that selection/replacement strategies that are too strongly biased towards selecting the best/removing the worst can lead to premature loss of diversity and hence sub-optimal solutions. However, "too little fitness bias results in unfocused and meandering search. Finding a proper balance is important but difficult to determine *a priori* with current theory" [DeJong 96].

It was in fact [DeJong 75] who introduced the first variations on biased sampling mechanisms and [Brindle 81] proposed several further modifications. Analysis of different categories of selection procedures can be found in [Baker 85], [Baker 87], [Bäck & Hoffmeister 91]. According to [Beasley *et al.* 93a], parent selection techniques can be divided into two types of methods, depending on whether fitness values are explicitly remapped in order to allocate reproductive trials or not.

Some methods involve sorting individuals according to raw fitness values, and then allocating reproductive opportunities based on this ranking. Scaling is another common technique, where fitness values are remapped in order to compress the range; it can be done linearly or exponentially, depending on the characteristics of the fitness landscape.

A very simple and commonly used sampling mechanism can be modelled as a "roulette wheel", where the slots are sized according to fitness:

- Calculate the fitness value $f(c_i)$ for each chromosome c_i $(i = 1, ..., N)$ where N is the population size.
- Obtain the fitness sum of the whole population
$$F = \sum_{i=1}^{i=N} f(c_i)$$
- Compute the probability p_i of selecting each chromosome c_i $(i = 1, ..., N)$
$$p_i = f(c_i)/F$$

- Calculate a cumulative probability q_i for each chromosome c_i $(i = 1, ..., N)$

$$q_i = \sum_{j=1}^{j=i} p_j$$

Selecting S chromosomes now consists on "spinning" the roulette wheel S times:

- Generate a random number r in $[0, 1]$.
- If $r < q_1$ then select c_1; otherwise select c_i such that $q_{i-1} < r \leq q_i$

An interesting improvement on this sampling mechanism, called *stochastic universal sampling*, was devised in [Baker 87]. Instead of spinning the wheel S times using just one marker, the wheel is spun only once but using S equally spaced markers. This modified version does not introduce the sampling errors incurred by the original.

2.6.1 Tournament Selection

An alternative approach does not involve an explicit remapping of the fitness values, but still performs reliable selection regardless of function scaling. Although there are several variations on the theme, a class of techniques is based on what is known as *tournament selection*: a set of M individuals is picked at random from the population and the fittest one is given the opportunity to reproduce. The choice of M is important, because it has a direct effect on selection pressure. To reduce the risk of premature convergence, the minimum value of $M = 2$ is commonly used, hence the name *binary* tournament selection.

2.6.2 Élitism

Unless explicitly precluded, in a steady-state GA it is possible to have the best individual replaced when new offspring are introduced into the population; in a generational GA, it might just be left behind. GAs are highly robust methods and, as such, good specimens just like the one that has been lost, are likely to be produced again in a relatively short time. However, it is common practice to prevent this temporary backward-step by always keeping the best individual found so far under protection from replacement. In a generational GA, at least one copy of the best chromosome is carried over to the next generation. Selection/replacement techniques comprising protectionism of this sort are called *élitist*.

2.6.3 Crowding, Niche and Speciation

In his pioneering work ([DeJong 75]), De Jong also introduced the concept of *crowding*. When a new individual has been created, it will be introduced into the population replacing an "old" one. Analogous to the tournament size, a crowding factor (CF) is used: CF candidates for replacement are chosen from the current population at random. The new individual will take the place of the one it resembles the most, using some measure of similarity — e.g. a simple bit-by-bit comparison to calculate the Hamming distance. Related concepts are *niche* exploitation and *sharing* ([Holland 75], [Goldberg & Richardson 87]), where several individuals which occupy neighbouring areas in the search space are made to share the fitness payoff among themselves.

The aim is not just to maintain diversity in the population: by enforcing this crowding pressure, kinds of niches are created. This is of special interest if the GA is used with a multi-objective evaluation function, when it is important to locate several peaks of high fitness rather than just one. This is intended to be analogous to the speciation phenomena in nature ([Perry 84]).

2.7 Reproduction

Once some individuals have been selected, they will be used to produce new offspring, which will become part of the population in the next generation. The reproduction process typically involves two classes of operators: recombination and alteration, applied in that order. The former is normally performed by *crossover* operators, while *mutation* is generally used for the latter. Depending on the overall strategy and the chosen generation gap, these operators are applied with a certain probability. In fact, crossover operators are often used with a relatively high frequency, whereas the probability of mutation is usually kept low (see section 2.9). In broad terms, the use of crossover is intended to attempt different arrangements of the genetic material already present in the population, while mutation is used to introduce new gene values that are possibly missing, in order to build better individuals.

2.7.1 Crossover

In traditional GAs, two parents are normally used to produce the same number of children, but it is obvious that simple modifications to the basic model provide different alternatives. Assuming chromosomes of fixed length l, one-point crossover works as follows:

1. Choose randomly a cutting point c $(0 < c < l)$.
2. Copy the first c genes from one parent and the last $l - c$ from the other and put them together to form the new chromosome. If required, create a second child in a complementary fashion.

For example, if the following are the parent chromosomes $(l = 7)$:

A	B	C	D	E	F	G
a	b	c	d	e	f	g

and the crossover point c turns out to be 3, these children would be produced:

A	B	C	d	e	f	g
a	b	c	D	E	F	G

Very frequently, a two-point crossover variation is used, where genes from each parent are transferred to the offspring alternately between cutting points. Using the same parents from the previous example, assuming crossover points at $c_1 = 2$ and $c_2 = 5$, the following individuals would result:

A	B	c	d	e	F	G
a	b	C	D	E	f	g

Apart from the obvious extension to N-point crossover, an interesting variation is the so-called *uniform* crossover: each gene is, in turn, randomly chosen from either parent ([Syswerda 89]). Optionally, a fitness-dependent bias can be introduced, so that children receive relatively more genes from the fitter parent ([Eshelman *et al.* 89]).

2.7.2 Mutation

Before a new individual is evaluated and inserted in the population, each one of its genes has a small probability of being altered. In the traditional binary string representation, a random number in the range $[0, 1]$ is generated for each bit in the chromosome. If the value is smaller than the current mutation probability p_m, the corresponding bit is flipped.

Many extensions of the basic ideas, derived from binary representations, have been proposed for higher cardinality alphabets, such as direct integer or real number representations. Meaningful interpretations of recombination and alteration operators have been thus defined. For example ([Beasley *et al.* 93b]):

- Recombination operators

  Average — take the arithmetic average of the two parent genes.

  Geometric mean — take the square-root of the product of the two values.

  Extension — take the difference between the two values, and add it to the higher, or subtract it from the lower.

- Alteration operators

  Random replacement — replace the value with a random one.

  Creep — add or subtract a small, randomly generated amount.

  Geometric creep — multiply by a random amount close to one.

2.8 Theoretical Framework

"Theory is crucial. Serendipity may occasionally yield insight, but is unlikely to be a frequent visitor. Without theory, we make endless forays into uncharted badlands. With theory, we can separate fundamental characteristics from fascinating idiosyncrasies and incidental features. Theory supplies landmarks and guideposts, and we begin to know what to observe and where to act" [Holland 95].

Despite Holland's concerns and efforts, it could be difficult to talk about the *theoretical foundations* of GAs, for this is still a rather controversial topic within the field. Evolutionary computation in general, and GAs in particular, often emphasise "traditional engineering concerns: using simulated evolution as a method to expand the practical design powers of programmers or software engineers" [Dennett 95]. The reason for this pragmatic emphasis is probably the following: despite the fact that GA practical applications, in general, lack a strong theoretical background, they very often perform better than traditional systems and, not unusually, outperform the best-known algorithm for a particular class of problems.

Some of the theoretical notions that provoked the interest in developing this research are introduced in the following sections. Intentionally, these concepts are presented first in rather general, abstract terms. More specific, concrete examples of some attempts to explore and exploit these ideas are provided below.

2.8.1 Schema Theorem and Building-Block Hypothesis

The traditional description of a *building-block* (BB) involves the notion of *schema* ([Holland 75], [Goldberg 89c]). A GA operates over a population of individuals representing potential solutions to a problem, a subset of which can be defined using a similarity template. Those individuals matching the values that were explicitly stated in the template are said to be instances of it; this particular template is called a schema. A schema which is tight (short), with few specified values (low-order), and with relatively high fitness, is called a building-block. Higher-order BBs are obtained over time by combining information from low-order ones; this gradual, cumulative process is said to involve *mixing events* ([Thierens & Goldberg 93]). In [Goldberg *et al.* 92], an attempt is made to create analytical models of building-block exchange in linear problems which are verifiable through computational experiments.

According to [Forrest & Mitchell 92], "the *building-block* hypothesis states that the GA works well when short, low-order, highly-fit schemas recombine to form even more highly fit higher-order schemas. The ability to produce fitter and fitter partial solutions by combining building-blocks is believed to be a primary source of the GA's power, but the GA research community currently lacks precise and quantitative descriptions of how schema processing actually takes place during the typical evolution of a GA search."

Although these statements regarding BBs are generally accepted, they are often considered to be weak as predictors for the working of GAs ([Beyer 95]). The main emphasis in schema-based theories is on the fitness proportionate selection. Mutation and recombination appear in these theorems as disturbances changing the selection equation into a weaker inequality. But even if selection is considered alone, such theories cannot predict the performance of the GA, as can be seen in the case of deceptive problems.

Several alternatives have been proposed, such as *Markov Chain Analysis* ([Davis & Principe 93]), which have been used to prove/disprove the convergence to stationary state properties. The problem is that, although they are exact models, a link is still missing that connects the microscopic level of description to macroscopic variables such as the expected fitness change over time or the number of generations needed to reach a certain state.

The *Mesoscopic Approach* ([Mühlenbein 92], [Thierens & Goldberg 94]), as the name implies, is an attempt to incorporate exact microscopic theory together with some phenomenologic ingredients obtained by empirical methods. Even though from the current state of the theory it is not plausible to derive formal basic principles, the mesoscopic approach is able to give estimates for optimal mutation rates and the expected run-time complexity of the algorithms ([Mühlenbein & Schlierkamp-Voosen 94]).

Nevertheless, it is very likely that the "building-block" notion will be a crucial one in the context of a more general theory of complex adaptive systems:

> "[The theory's] mathematics would emphasise the discovery and recombination of useful components — building-blocks — rather than focusing on fixed points and basins of attraction." [Holland 92]

While it seems that it is necessary for BBs of the appropriate size to come into existence, grow, and be well selected, it is most important in a recombinative scheme that good building blocks in one structure be combined with good building blocks on another structure to form a new structure with a larger number of effective BBs. It should be possible to create new mechanisms to facilitate the observation and description of BB manipulation in GAs. Once the underlying processes are understood in those terms, suggestions for predictability and performance improvement should be more easily drawn.

The way information is represented, i.e. the chromosome's coding scheme, plays a fundamental role in the operation of the GA, as will be discussed next. Furthermore, how BBs are formed, combined and preserved largely depends on the form of representation and operators chosen.

2.8.2 Epistasis

In most cases, the influence that a particular gene has on the fitness of an individual depends on the values of other genes in the chromosome. The term *epistasis* is used to refer to any kind of strong interaction between the different genes that make up a chromosome.

It is important to note that this idea extends beyond gene boundaries. BBs usually interact with each other in complex, non-linear ways. Two or more separate BBs that indeed contribute independently to the fitness of several individuals in a positive way, might interact negatively when they are assembled together in one single chromosome, resulting in a decrease in fitness for the individual that juxtaposed them. Therefore, the GA not only has to find all the required BBs; it must also discover a good way to combine them in order to yield an appropriate solution in the end.

2.9 Adapting Parameters

Unlike most deterministic methods, GAs are normally not highly sensitive to the choice of system parameter values, which contributes enormously to their reputation of being robust. Nevertheless, several attempts have been made to devise guidelines for choosing optimal parameter values. Techniques that employ dynamic system parameters seem very appealing, since the optimal use of each operator may vary during the course of a GA run ([Grefenstette 86]). An intuitive approach is to apply a linear variation to crossover and mutation probabilities ([Davis 85], [Syswerda 91]): GA operation starts with a relatively high value for the former and a low one for the latter; as the generations pass by, the probability of crossover decreases while the probability of mutation increases. The idea is to use the recombination powers of crossover to find useful combinations of building blocks at the beginning, and to give more opportunity for mutation to introduce the missing elements needed for refining individuals in the final stages. This does not need to be done using a fixed schedule. [Booker 87] proposed a dynamical crossover rate, which varies depending on the spread of fitness values; when the population diversity is reduced, the crossover probability decreases in order to augment the effects of mutation, thus preventing premature convergence.

Another adaptive technique is based on positive reinforcement ([Davis 89], [Davis 91]), where credit is given to each operator according to the success with which it creates fitter chromosomes. For each reproductive trial, an operator is selected with a certain probability, depending on an associated weighting figure which is updated according to its current effectiveness. These weights are adaptively modified during the course of a GA run.

In most cases, self-adapting mechanisms present many advantages over fixed strategies using predetermined parameter values. Moreover, the idea of self-adaptation can be extended to other GA elements, such as representation, introduced in the next section.

2.10 Further Issues on Representation

It is clear that representation is a central question concerning GAs. For any given environment or problem domain, the choice of which features to represent on the genotype and how to represent them is crucial to the performance of the GA. The trade-offs are clear in that binary encodings and traditional operators have a much broader range of applicability, but are normally outperformed by problem-specific representations which exploit additional knowledge, when available, about a particular class of problems.

Thus, many applications and analysis of GAs have departed from the basic binary string representation. Despite the fact that traditional GA theory suggests that low cardinality alphabets yield enhanced schema processing, other representation schemes such as real-coded genes ([Rechenberg 73], [Schwefel 81]) have been applied successfully in the solution of many practical problems. Theoretical analysis regarding the use of high-cardinality coding in GA chromosomes has made some progress accordingly ([Goldberg 91b]), but much work is still required in this area.

One generally adopted approach is to design a problem-specific type of representation and the appropriate new/modified genetic operators to work on it. Enhanced capabilities and improved performance are indeed commonly achieved, the price being a severe loss of generality and the lack of a solid theoretical account for the behaviour of that particular system.

2.10.1 Linkage Problem and Deception

For most chromosome representations, like the commonly applied binary strings with fixed positions for each gene, the actual coding scheme has to be very carefully designed. Operator-oriented views of fitness landscapes ([Jones 95]) emphasise the tightly coupled interaction between, on one hand, the choice of a certain representation and an evaluation function defining a fitness landscape and, on the other hand, the operators used to explore it. In particular, given the way in which crossover-like genetic operators work, if two inter-related genes are not located or easily placed by the GA itself close to each other in the chromosome, useful combinations will be both hard to identify and easily disrupted. The same applies for higher-order BBs; this is known as the *linkage problem*. If the choice of chromosome coding does not allow the required building block combinations to take place, the GA will converge to sub-optimal solutions, particularly in the case of so-called *deceptive* problems ([Goldberg 89a], [Goldberg 89b], [Goldberg 91a], [K. Deb 92]).

For example, [Bethke 80] used the Walsh-schema transform to construct functions that mislead the GA, by directly assigning the values of Walsh coefficients in such a way that the average values of low-order schemas give misleading information about the average values of higher-order refinements of those schemas — i.e. higher-order schemas contained by the lower-order schemas. Specifically, it is possible to choose such coefficients so that some short, low-order schemas have relatively low average fitness, and then choose other coefficients so as to make these low-fitness schemas actually contain the global optimum. Deception has been a central focus of theoretical work on GAs.

Walsh analysis can be used to construct problems with different degrees and types of deception, and the GA's performance on these problems can be studied empirically. The goal of such research is to learn how deception affects GA performance, shedding light on the reasons for the GA failure in certain cases, and to learn how to improve the GA or the problem's representation in order to enhance performance.

2.10.2 Variable Length

Most current GAs use fixed-length chromosomes comprising specific genes at fixed positions, normally incorporating one of the usual varieties of crossover operators. This bias towards simple, basic structures and operators is undoubtedly caused by early studies, mainly [Holland 75] and [DeJong 75], which adhered to a number of abstractions and simplifications in their quest for fundamental principles. As pointed out in [Goldberg *et al.* 89]: "It is interesting, if not ironic, that neither man intended for his work to be taken quite so literally. Although De Jong's implementation simplifications established [*sic*] usable technique in accordance with Holland's theoretical simplifications, subsequent researches have tended to treat both accomplishments as inviolate gospel".

However, the use of variable-length strings, although notably uncommon, is not new. [Cavicchio 70] is an early example, presenting a study of GAs as a search tool in the context of pattern recognition detectors. [Smith 80] presented the LS-1 system in a study of machine learning in a poker-playing task. Modified forms of crossover operators were introduced; crosses both at and within rule boundaries were allowed. Additionally, the inclusion of an inversion operator permitted the rearrangement of rules within a string, with the purpose of bringing closely related rules together, thereby reducing the disruptive effects of crossover. His work set the stage for machine learning applications using variable-length representations and modified versions of recombination operators ([Cramer 85], [Fujiko & Dickinson 87]).

Over- and Under-specification

In some cases, although a variable-size representation is used, a chromosome of a predetermined length is in fact required in order to perform fitness evaluation. On one hand, it is possible to have several instances of the same gene but with different values; this is a case of *over-specification*. On the other hand, a chromosome might also be lacking the presence of certain genes, in which case it is said to suffer from *under-specification*.

Handling over-specification is relatively easy. The requirement is to choose between conflicting instances of the same gene contained within the same chromosome. There are several ways of dealing with this problem, choosing by means of a probabilistic or deterministic voting procedure, adaptive precedence or, the one most commonly used, simple positional precedence.

Under-specification is more difficult to deal with, especially without making strong assumptions that might compromise the generality of the approach. One such assumption is the *partial string, partial evaluation* idea, which presupposes that the fitness function may be calculated as a sum of sub-functions ([Goldberg *et al.* 89]):

$$f(x_i) = \sum_{j=1}^{j=m} f_j(x_k, k \in K_j), i = 1, ...l,$$

where the f_j are themselves functions of non-overlapping subsets of the Boolean variables x_i and the sets K_j form a partition of the full index set $1, ..., l$.

Alternative approaches for dealing with under-specification are *independent* and *in-common averaging*, both of which were found to be unacceptable in [Goldberg *et al.* 89] due to a high schema-difference-to-noise ratio introduced by sampling errors. Instead, they proposed the *competitive templates* alternative (see below).

Parsimony

Some of the fundamental issues associated with representations of variable size, in contrast to linear strings of fixed length, have been described in different ways, particularly in the Genetic Programming literature (see section 2.11.1). One obvious difficulty is that these structures may grow too big, and too quickly, without any significant improvement in the quality of the solution they represent. In [Nordin & Banzhaf 95], for example, the effects of compression pressure in evolutionary algorithms using individuals of variable length are discussed; it is suggested that there are both positive and negative effects and that the key point is to balance the pressure towards low *effective complexity* (i.e. the length of the active parts of an individual), according to the equation for *effective fitness*, which takes into account this complexity factor.

Of particular interest to this thesis is the concept of parsimony, which is directly linked to the fundamental relationship between the performance and complexity of evolved individuals ([Zhang & Mühlenbein 95], [Zhang & Mühlenbein 93]). It has been modelled after the intuitive idea behind the principle of Occam's Razor ([Blumer *et al.* 87]) which simply indicates that a problem (or a solution, as it may be) should be stated in its basic and simplest terms. This is in line with the Aristotelian principle that entities must not be multiplied beyond what is necessary. In science, the simplest theory that fits the facts of a problem is the one that should be selected. The rule is interpreted to mean that the simplest of two or more competing theories is preferable and that an explanation for unknown phenomena should first be attempted in terms of what is already known.

2.11 Structured and Hierarchical Representations

As mentioned before, high-cardinality alphabets are often used to represent parameters in a GA's chromosome. Additionally, a large number of different approaches, even if they retain a binary-string representation, often impose a structure of some sort upon the chromosomes they use. The most relevant examples are briefly discussed below. It is important to note that few of these approaches had been developed at the time this project commenced.

GENES

An interesting example of a structured representation was presented in [Bickel & Bickel 87]. They used a variable-length chromosome (a rule set), treated as a linked list of genes, each gene being a tree-like structure (a single rule), used to develop expert systems. The smallest units to handle, the nodes forming the tree in each gene, were chosen from a table of possible options, so that the GA was used to find:

- good combinations of nodes within a gene — i.e. good rules
- good combinations of genes — i.e. good rule sets

where the latter was obviously the final goal.

The mutation, crossover and inversion operators were modified so that they could be used on that particular representation. A few variations of a mutation operator were proposed, which could affect either a whole gene or just a node within a gene, thus operating at two different levels of the representation. On the other hand, a crossover-like operator was designed to recombine complete genes, occurring at points between rules only, with no recombination of tree nodes. Likewise, an inversion operator would just exchange the order of complete genes within a chromosome. The actual number of rules in each linked list was randomly determined via an average length parameter and the number of rules allowed was optionally bounded. There was only one action node per rule; additional nodes each representing a possible condition precedent to carrying out the action, linked by boolean operators. In their system, named GENES, each individual consists of a linked list of rules. Each rule is in turn parsed syntactically as a tree structure. Thus, it could be said that their work bears resemblance, in several aspects, to the Genetic Programming approach (see section 2.11.1).

sGA

Another form of structuring comes about by adopting hierarchical representations. An interesting example is the so-called *Structured Genetic Algorithm* (sGA, [Dasgupta & McGregor 92]), developed for the design of application-specific neural networks. The main distinctive feature of the sGA is that a multi-level hierarchy is represented within the (linear) chromosome. In this scheme, high-level genes determine whether lower-level genes are active or not for that particular chromosome, making use of genetic redundancy represented as a set of binary strings in a haploid genetic model. Such redundancy is eliminated through regulatory genes that act as switching operators to turn other genes on or off. An important consequence is that a genetic operator may now produce, in a single high-level alteration, a result equivalent to multiple changes at lower-level parts of the chromosome. Furthermore, the operators may perform in a blind, uniform manner, regardless of the level of the gene(s) being affected. It could be said that the sGA quasi-hierarchical representation *implicitly* introduces an intra-chromosome operator.

GA-MINER

[Radcliffe & Surry 92], [Radcliffe 92], [Radcliffe & George 93] study the limitations of linear chromosome representations and the use of conventional recombination operators. They introduce the concept of *formae* as a generalisation of the schemata used in conventional GA theory. It is interesting to note that their work on non-linear representations, in the context of formulating data mining as a search problem, led them to the independent development of a system ([Radcliffe & Surry 94]) which is structurally very similar to the one presented in this thesis. They adopted the view that the most useful form in which the results of data mining can be presented is as explicit rules, commonly expressed as predicates of the form (*if x then y*). There too, the goal is not simply to find the single best rule describing a subset of the database in question, but to find a selection of rules representing different kinds and instances of patterns within the database. They acknowledged the need for a sort of niching technique (see section 2.6.3) and thus proposed the "use of the implicit niching encouraged by structured population models while strengthening this through the construction of a two-level *hierarchical* genetic algorithm". In their approach, rules are taken from each of the low-level populations to form a universal set of rules from which it will be the task of the high level genetic algorithm to find the best set of some given size. In this manner, competition at two different levels of the hierarchy results in the discovery of co-operatively useful sets of rules.

The issue of determining a suitable evaluation function for both levels in the hierarchy was addressed. For the low-level one, they postulated a number of desirable features, in order to formulate a single numerical quantity encapsulating them; an ideal rule would be [Radcliffe & Surry 94]:

- *interesting*—capturing some trend in the data which is of use to those using the data-miner;

- *relevant*—capturing meaningful trends, not simply "truisms" about the database;
- *approachable*—formulated and presented in a form easily digested by humans;
- *general*—rules which apply to larger portions of the database are preferred;
- *accurate*—so that truth of the conditional part of the rule implies truth of the rule's prediction with high probability;
- *covering*—so that truth of the prediction part of the rule implies truth of the rule's conditions with high probability;
- *statistically meaningful*—there must be a higher probability that the prediction of the rule is true when the conditions are satisfied than when they are not.

In the end they defined the fitness of individual rules to be the product of two terms, one measuring the quality of a rule, and the other measuring its generality.

With regards to the high-level GA, at first their work concentrated on coverage as the key criterion for rule set fitness evaluation, but the fitness of the individual rules was also taken into consideration. They defined a measure of the difference between two rules and then evaluated a set of rules by multiplying the fitness of each rule by the sum of its difference from each other rule, and summing over all rules, satisfying both the requirement for coverage and that for high individual rule fitness. Their algorithm starts with a large universal set of rules produced by genetic search in the low-level algorithm. The high-level GA then searches over fixed-size subsets of this universal set for good collections of rules.

They arrived at the conclusion that the hierarchical genetic algorithm was of relevance not only to data mining, but also to general covering problems. However, they seem to have later abandoned the idea of a hierarchical GA for data mining ([Radcliffe 95]).

GAANT

In the realm of engineering design, evolutionary and adaptive search algorithms have been integrated at different levels within the global design process, from the conceptual phase to the detailed design stages. However, [Parmee 96] introduced a dual-agent strategy to search the complex spaces involved in engineering design hierarchies defined by complete systems when taken as a whole. In general, such design hierarchies comprise, at a first level, many discrete configurations which are described, in turn, by dependent continuous variable sets. Often these sets of dependent variables differ between discrete design paths, making it difficult to search across such hierarchies without using a structured approach.

It is as an attempt to solve this problem that the GAANT system ([Parmee 96], [Parmee 97]) was proposed, incorporating both a GA and elements of an ant colony search paradigm ([Coloni *et al.* 92], [Bilchev & Parmee 95]). The distinction between discrete and continuous variable sets is made evident in the overall architecture of the system. In practice, however, the GAANT strategy uses a single chromosome to represent both kinds of elements (see figure 2.1), but establishes a distinction between individuals according to their discrete variables: crossover is only permitted amongst solutions with identical discrete configurations, thus preserving information exchange within members of the same kind. While the GA is guiding the continuous variable set through an evolutionary process, the values of the discrete parameter set remain constant for a predetermined number of generations, called an epoch. Then, an overall chromosome fitness is calculated for each individual, based on a ratio of average fitnesses and the mean fitness for the current generation. Using scaled values in accordance with the standard deviation, upper and lower bounds are defined in order to select which chromosomes are discarded, which ones are modified (mutated) and which others are preserved for the next generations.

Variations of this dual-agent strategy have also been applied in different engineering design domains ([Chen & Parmee 98]). In [Watson & Parmee 97], a genetic programming operator is used to deal with the discrete functional structure while the GA performs the search in the continuous coefficient space. The GAANT system and variations thereof have been used in practical settings to perform an efficient search across the discrete/continuous design hierarchy. See [Parmee 99] for a concise presentation of how the GAANT system fits into the whole domain of engineering design.

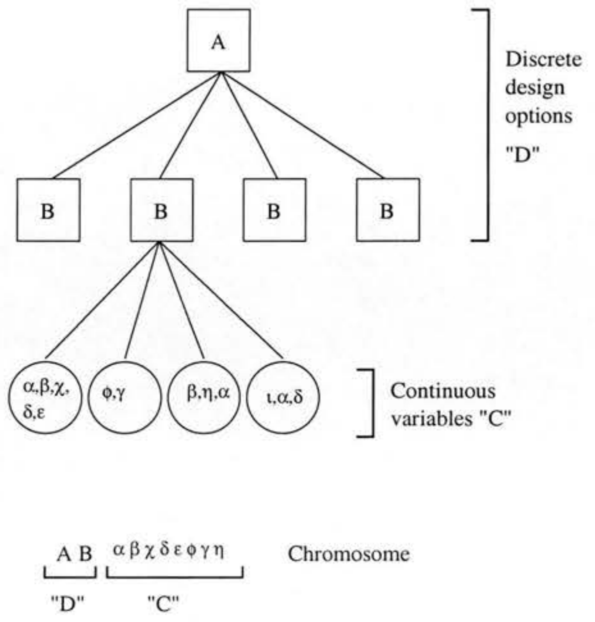


Figure 2.1: Design hierarchy representation in GAANT

Design Grammars

The concept of using grammar-like mechanisms providing syntactic generative capabilities to represent design plans is not a new one ([Stiny 80]). It is possible to integrate their representational power into an evolutionary system. Genes in a chromosome — i.e. the genotype — are defined by specific grammar constructs which can be used to generate a meaningful solution — i.e. a phenotype. Grammar rules can be regarded as fundamental operators, which are capable of defining a certain design space ([Rosenman & Gero 99]). The role of the evolutionary system is to explore this space, guided by the requirements as formulated in a fitness evaluation function. In the end, a particular genotype will be found that embodies a set of grammar rules which are capable of generating a satisfactory solution, given the interpretation of the actual phenotype. This concept is closely related to the use of rewrite grammars ([Gruau 93]) to represent connectionist systems as they are evolved by a GA of some sort; see section 4.6.

2.11.1 Genetic Programming

Often considered a variation of GAs, Genetic Programming (GP) has been gaining increasing popularity within the Evolutionary Computation community. First introduced by [Koza 92], problem solving in GP is formulated as a search in the space of computer programs, in an ad-hoc language, represented by structures of dynamically varying size and shape. Populations of such individuals are genetically bred, based on variations of the main genetic operations: reproduction, crossover and mutation. Naturally, each operation is based on a previous selection of fit individuals.

Genetic programs are stored as complex expressions with a specific recursive format, in the provided language, typically using prefix notation. Given that an expression is the application of an operator to a specific number of operands, prefix notation denotes that the ordering of the expression is the operator followed by the correct number of operands. If in turn each operand is itself an expression, a recursive tree structure develops with the operator as an internal node of the tree and the operands as the leaves. Borrowing the syntactic convention of LISP, the traditional Artificial Intelligence programming language, most GP notations denote complete subtrees in evolved programs enclosed within parentheses.

The fitness of each member of the population is determined by evaluating the individual program using domain dependent performance/cost functions. The usual selection methods can be applied to obtain candidates for reproduction, just as in the generic GA. Then, a special form of crossover is used to obtain (usually) two new programs from an exchange of genetic material between two old programs: a node is randomly chosen in the tree representing each parent and is used as a *pivot*, swapping subtrees rooted at the two pivots between parents. Mutation is applied in such a way that a newly generated subtree replaces the subtree rooted at a random pivot position in a selected individual. Because all individuals in the population are represented as recursive tree structures, simply removing a complete subtree and replacing it with another automatically preserves all syntactic constraints.

Because expression trees are recursive, they are of variable depth and hence of unlimited size. In practice, however, a specific system parameter is used to restrict the maximum size of the evolving expression trees, while a certain kind of mechanism (see section 2.10.2 above) is often used to discourage unjustified growth.

GP has been extended in several different ways ([Koza 94], [Kinnear 94a]). Such extensions have focused on the automatic discovery of functions that improve the ability to search for solutions by exploiting opportunities to parameterise and reuse previously generated code. Examples of these techniques are *automatic definition of functions* ([Koza 92], [Kinnear 94b]), which allows the evolution of reusable subroutines, and *adaptive representation* ([Rosca & Ballard 94]), which is focused on the discovery of useful building blocks of code; these blocks are identified by analysing their evolutionary trajectory and then generalised and transformed into new functions which extend the function set in an adaptive way.

2.12 Messy GAs

In order to circumvent some of the problems associated with traditional fixed-length and fixed-coded GAs, mainly the linkage problem discussed earlier, a different approach called the *messy genetic algorithm* (mGA) has been proposed and analysed ([Goldberg *et al.* 89], [Deb 91], [Goldberg *et al.* 90]), implemented ([Deb & Goldberg 91]) and enhanced ([Goldberg *et al.* 93]). Several new variations on the initial definition of an mGA have been put forward in the last few years ([Kargupta 97], [Knjazew & Goldberg 00]), but the main distinctive features remain, as described below.

There are four basic differences between simple GAs and mGAs (for a succinct presentation of what mGAs are, see [Goldberg *et al.* 91]):

1. mGAs use a variable-length coding scheme that may lead to over- or under-specification with respect to the problem being solved

2. mGAs use two complementary operators, *cut* and *splice*, instead of variations of the usual, fixed-length crossover operator

3. mGAs divide the evolutionary process into two phases: a primordial phase and a juxtapositional phase

4. mGAs sometimes use competitive templates to accentuate salient BBs

In broad terms, the operation of a mGA is not unlike that of a traditional GA, using selection and recombination procedures repeatedly on a population to make it follow an evolutionary process. The main distinctive property of the mGA is that it incorporates a very flexible coding scheme, which allows the reordering of genes within the chromosome. This feature is said to promote the discovery and, more importantly, the preservation of useful combinations of genes — i.e. building-blocks ([Watson & Pollack 99]).

Cut & Splice

The complementary *cut* and *splice* operators were introduced to recombine structures of variable length (see figure 2.2). The cut operator divides a chromosome of length λ with probability $p_c = (\lambda - 1)p_k$, where p_k is a predetermined bitwise cut probability. The splice operator joins together a pair of chromosomes, end to end, with specified probability p_s. Once the parents have been selected for reproduction, using the preferred selection method, the cut operator is applied according to the evaluated probability for each individual. The resulting chromosomes are then checked for splicing in successive pairs.

The purpose of applying these two operators in combination is to reduce potential disruption of BBs, while preserving the juxtapositional power of simple crossover operators. Cut and splice also have reordering capabilities, declared to be superior to those of unary operators such as inversion.

Competitive Templates

Since the use of random templates is too noisy to detect salient building blocks reliably, additional measures were taken to overcome the under-specification problem. A greedy procedure to generate a locally optimal point is employed, and the result is used as a *competitive template*, which is a string specifying a default value for each gene position. Whenever a chromosome suffers from under-specification, the unspecified genes are borrowed from the template.

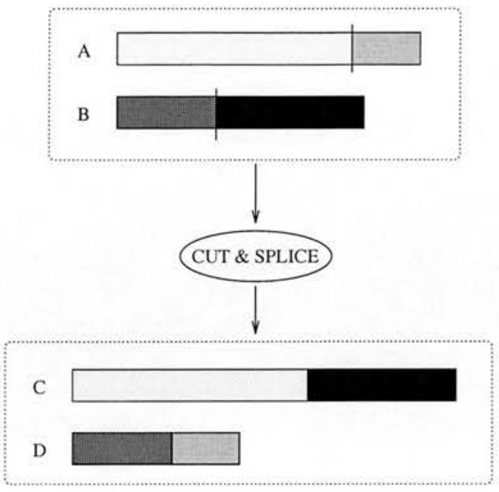


Figure 2.2: Cut & Splice operators

2.13 Classifier Systems

Learning Classifier Systems (CSs) have been used as models of stimulus-response behaviour and of more complex cognitive processes ([Holland *et al.* 86]). A succinct definition is given in [Goldberg 89c]: "A classifier system is a machine learning system that learns syntactically simple string rules (called *classifiers*) to guide its performance in an arbitrary environment." CSs comprise three basic components:

1. Rule and message system.
2. Apportionment of credit system.
3. Genetic algorithm.

These components reflect the underlying principles involved in CSs: hierarchies of internal models that represent the environment, intermittent feedback from the environment, and learning mechanisms. Traditional CSs are organised in three layers corresponding to the components mentioned above. There is an *internal performance system*, which operates using "messages" and controls its state with if-then rules, called classifiers, that specify patterns of messages. The GA is used to discover useful rules, based on intermittent feedback from the environment and an internal credit-assignment algorithm, the most common being called the *bucket brigade*; profit-sharing and other alternatives from the literature on reinforcement learning have also been proposed ([Wilson & Goldberg 89]).

A parameter called strength is associated with each classifier. This measure reflects the utility of that rule, based on the system's past experience. The apportionment of credit mechanism, normally based on the bucket brigade algorithm, is responsible for altering the strength of each rule. The algorithm is based on the metaphor of an economy, with the environment acting both as the producer of raw materials and the ultimate consumer of finished goods, and each classifier acting as an intermediary in an economic chain of production. Using the bucket brigade, a classifier system is able to identify and use the subset of its rule base that has proven useful in the past. The GA interprets a classifier's strength as a measure of its fitness and periodically, after enough time has passed for strengths to have stabilised under the bucket brigade, the GA deletes rules with low strength, which have not been useful or relevant in the past, and generates new rules by modifying existing high-strength rules through mutation, crossover, and other special-purpose operators. See section 4.3.4 below for a brief description of a particular kind of CS involving fuzzy logic concepts.

The main conceptual difference between CSs and GAs is that, in the latter, population members are functionally independent, normally interacting only in a competitive fashion through the selective process ([Smith & Valenzuela-Rendón 94]). In contrast to this, population members in a CS are interdependent. A balance between cooperation and competition must be sustained: on one hand, classifiers must cooperate in order to improve overall system performance; on the other hand, they compete for valuable credit in order to survive the selective process.

Chapter 3

Fuzzy Systems

This chapter contains a general presentation of fuzzy systems (FSs), introducing the basic concepts and practical considerations. Special attention is paid to rule acquisition and design issues, since these topics are of particular relevance to the work presented in this thesis.

3.1 Overview

The beginning of fuzzy logic is most widely associated with Lotfi Zadeh, whose original paper [Zadeh 65] formally defined fuzzy set theory, from which fuzzy logic emerged. He extended traditional set theory to resolve problems sometimes generated by the hard and rigid bivalent classifications of Aristotelian logic. Traditionally, a logic condition or expression can only be either absolutely true or absolutely false. However, in fuzzy logic, values range from 0 to 100% true or false, so that statements have some degree of truth between 0 and 1, inclusive. In this way, sets can be defined qualitatively using linguistic terms and the elements of the sets assigned degrees of membership. Additionally, any action or response resulting from a statement being at least partially true executes to a strength reflecting the degree to which the statement is true. Thus, FSs produce smooth and continuous outputs, regardless of inputs crossing set boundaries.

The principles of fuzzy logic have been successfully incorporated in many different areas, such as signal processing, approximate reasoning, decision making, classification and uncertainty handling; however, it is for control applications that the majority of

FSs have been developed. There is no question that these controllers present several interesting features, often comparing favourably with conventional control theory methods.

Notwithstanding the fact that fuzzy logic is a well-developed, broadly applied computational method, engineers and scientists in general still react with scepticism towards fuzzy theory and concepts ([Lindley 87], [Elkan 93]). Nevertheless, the practical benefits of FSs are very compelling and include such claims over conventional methods as shorter development time, increased maintainability, better performance, less-expensive hardware, and more robustness. It is possible to design them in a rather simple and consistent way, since knowledge can be represented in readable, explicit rule-like structures, without requiring a mathematical model describing how the output functionally depends on the input — FSs are *function estimators*. Furthermore, Fuzzy Logic Controllers (FLCs) are capable of modelling non-linear relations and their performance has proven to be very competitive in commercial and industrial applications. Thus, the first use of the particular kind of FS developed under this thesis' framework will be in a control application. In order to demonstrate how the basic FS model can be modified and extended, a second one will be applied to solve a classification problem.

There are many variations on the basic FS, e.g. the choice of membership functions definition, the operators used to fuzzify/defuzzify input/output values, the rule-base structure, etc. Traditionally, these systems are constructed starting with a knowledge engineer-domain expert interaction, not unlike the rule acquisition process for an expert system, to be further improved and fine-tuned at a later stage in the developing process, usually in an ad hoc way. An expert may articulate linguistic associations based on his/her domain knowledge, or another system may adaptively infer and modify these fuzzy associations, usually from representative numerical samples. Good examples of such auxiliary techniques include statistical and neural systems.

If a standard FS is adopted, there are two major components to design for a particular problem, namely, 1) the input/output fuzzy sets and their membership functions and 2) the rules that establish a functional relationship between them — *fuzzy associative memory* (FAM) rules [Kosko 92]. For the former, there are generally applicable ways to characterise the domain of a fuzzy variable, assuming conventional fuzzy-set values and membership functions. For the latter, knowledge engineering techniques are commonly applied. However, both of these components can be automatically tuned, modified, or even constructed from scratch by learning or searching mechanisms, such as GAs. In this thesis, a particular framework based on evolutionary techniques is proposed for the same purpose: to synthesise fuzzy rules directly from problem-domain sample data.

3.2 Basic Concepts

In fuzzy logic, relationships between imprecise concepts are evaluated instead of mathematical equations. FSs store and process fuzzy rules in parallel, associating output fuzzy sets with input fuzzy sets. Structured knowledge is directly encoded but, unlike traditional knowledge-based systems, this is done in a numerical framework. FSs process information using parallel associative inference, with fuzzy or multivalued sets instead of bivalent propositions. The basic process is described in the following sections.

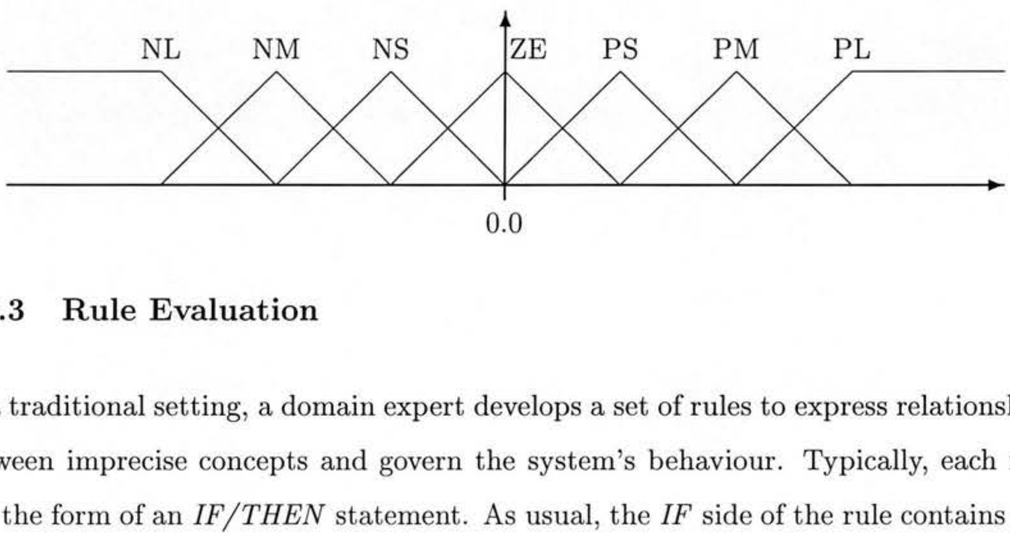
3.2.1 Fuzzification of Inputs

Each system input is associated with a group of qualitative classifications, called fuzzy sets. An input has some degree of membership, possibly zero, in each of its fuzzy sets. Such degree of membership in a fuzzy set is defined by a function, appropriately called a membership function. *Fuzzification* is the process of determining a value to represent an input's degree of membership in each of its fuzzy sets. The variable containing the resulting degree of membership is called a fuzzy input.

3.2.2 Fuzzy Sets and Membership Functions

Central to the fuzzification process is the collection of membership functions. Traditionally, it is the system designer or domain expert who must define these functions based on intuition or experience, often involving a trial-and-error tuning phase. Generally, once the system is in operation, the membership functions remain fixed. However, it is possible to provide the system with adaptive means, so that membership functions may change in order to reflect alterations in the system's state, environment, operating conditions, etc. Simple shapes such as triangles and trapezoids are commonly used to define membership within fuzzy sets, but any suitable function can be used. Obviously, the simpler the shape of the membership function, the easier the construction, representation and execution of that function.

In conventional systems, the domain expert also decides on the number of fuzzy sets per input/output variable. The fuzzy sets span the entire range or universe of discourse for the system variables. Mapping to the y-axis typically ranges from 0 to 1 and represents the degree to which an input value is a member of that particular fuzzy set. Overlapping between set boundaries permits membership in multiple sets, even if seemingly contradictory. Binding imprecise, linguistic terms to membership functions gives them computational meaning. Consequently, it is possible to use "natural" language to define the behaviour of a system, which enhances the ability to describe complex tasks clearly and concisely. A standard way of defining fuzzy sets, usually over a scaled or normalised input/output domain, is depicted in the figure below. Leaving zero (ZE) aside, negative (N) and positive (P) values are further qualified as being large (L), medium (M) or small (S), hence the abbreviated names:



3.2.3 Rule Evaluation

In a traditional setting, a domain expert develops a set of rules to express relationships between imprecise concepts and govern the system's behaviour. Typically, each rule has the form of an *IF/THEN* statement. As usual, the *IF* side of the rule contains one or more conditions, or antecedents, and the *THEN* side contains one or more actions, or consequents. The antecedents of rules correspond directly to degrees of membership (fuzzy inputs) calculated during the fuzzification process.

A frequent design and implementation choice is to model fuzzy rules using decision (look-up) tables, or linguistic matrices ([Kosko 92]). For example, if the system has two input and one output variables, using a reduced version of the standard membership function definitions, a particular fuzzy rule set could be defined as follows ([Wasserman 93]):

	NM	NS	ZE	PS	PM
NM			PM		
NS			PS	**ZE**	
ZE	PM	PS	ZE	NS	NM
PS		ZE	NS		
PM			NM		

Columns are indexed by the fuzzy sets that quantise the universe of discourse of $input_1$, whereas rows are indexed by those for $input_2$. Each entry in the table represents a fuzzy rule. For instance, the highlighted one is interpreted as:

$$\begin{array}{l} \textit{IF } input_1 = \text{PS } \textit{AND } input_2 = \text{NS} \\ \textit{THEN } output = \text{ZE} \end{array}$$

During rule evaluation, rule strengths are computed based on antecedent values and then assigned to the rule's fuzzy outputs. Generally, a minimum operation is applied, making the rule strength equal to the weakest antecedent value; however, other alternatives can also be used, like multiplying the antecedents together. Often, more than one rule applies to the same fuzzy output, in which case the common practice is to use the strongest rule.

3.2.4 Defuzzification of Outputs

After the rule evaluation process has assigned strengths to actions, further processing is required for two different purposes. First, it is necessary to decipher the meaning of vague or fuzzy actions, in order to produce an exact, crisp output value. Second, more than one action may have been triggered by certain conditions during the rule evaluation process, so the system needs to resolve conflicts between competing actions. A process that employs compromising techniques to resolve the vagueness and conflict issues is called *defuzzification*.

One common defuzzification technique, called the centre-of-gravity method, consists of several steps. Initially, a centroid point on the x-axis is determined for each output membership function. Then, the output membership functions are shortened in height by the applied rule strength, and new output membership areas are computed. Finally, the defuzzified output is derived by a weighted average of the x-axis centroid points and the newly computed areas, with the areas serving as the weights.

3.3 The Implementation of a Fuzzy System

The principles and requirements of fuzzy logic systems are relatively simple and there are a few public-domain systems available. For the work presented in this thesis, however, a proprietary implementation of a FS was developed. In broad terms, the design adhered to the generic FLC architecture depicted in figure 3.1, using constructs derived from [Viot 93].

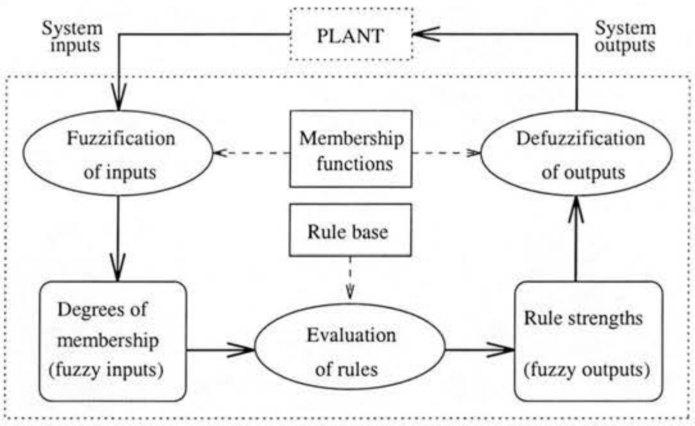


Figure 3.1: Generic Fuzzy Logic Controller

The operation of the basic system can be described as a (simplified) five-step process:

1. Read system inputs — the current value of state variables from the external system to be controlled.

2. Translate these (real) values into qualitative classifications — calculate the degree of membership for each of the fuzzy sets defined for each input. This is done using the membership functions, generating the *fuzzy inputs*.

3. Evaluation of rules — the values of the rules' antecedents, corresponding to the degrees of membership of the inputs' fuzzy sets, are propagated to the conse-

quents using standard (min-max) conventions, generating *fuzzy outputs*. This process is carried out in a (simulated) parallel fashion.

4. Translate the fuzzy outputs into an exact (crisp) value — calculate, for each output variable, the centre of gravity of the degrees of membership for that output's fuzzy sets.

5. Write system outputs — the updated values for the control variables.

Each particular system is characterised by 1) the set of rules and 2) the membership functions. The program was designed in such a way that these two components can be independently supplied either by the user or by other programs, using a standard format. The system to control will be simulated using a computational model of it.

In most cases, it is not too difficult to define evaluation criteria for a particular system in terms of performance, robustness, stability, compactness, etc. This information will be crucial to define an adequate fitness evaluation function for a GA that is trying to evolve a suitable controller or classifier. Section 5.2.3 presents a brief description of the information used for fitness evaluation in this research.

Chapter 4

Hybrid Systems

A review of the different approaches to the combined use of fuzzy systems (FSs) and genetic algorithms (GAs) is presented, pointing out examples of the most relevant research that has motivated and influenced this work. A brief overview of the relevant combinations of GAs and neural networks (NNs) is also included.

4.1 Overview

It is possible to combine fuzzy logic and evolutionary systems in several different ways. Although most of these have been developed particularly for control tasks, a generic architecture for a fuzzy system can be assumed to share the same basic properties (see figure 3.1). The two obvious targets for a GA-based automated search/optimisation process are

- the fuzzy set membership functions
- the fuzzy input/output associations — i.e. fuzzy rules

Early examples of how GAs could be effectively applied to synthesise FLCs can be found in [Karr 91], [Thrift 91], [Valenzuela-Rendón 91] and, as described in more detail in section 4.4 below, [Feldman 93]. These papers summarise the state of the art at the time the research for this thesis began. They show how membership functions and/or rule bases can be not only tuned, but even generated by a GA, producing FSs that may outperform those developed in more traditional ways. Their work has shown that

GAs are a potentially effective, robust tool for complex FSs design. They can be used to both improve FSs performance and to gain insight into the problem at hand, letting the GA generate a set of readable, explicit rules and the appropriate membership functions.

Since then, many more alternatives of soft computing approaches have been proposed, combining genetic algorithms, fuzzy logic and neural networks of some sort ([Ishigami *et al.* 94], [Shaffer 94], [Zimmerman 94], [Fukuda & Shibata 94], [Shimojima *et al.* 95], [Linkens & Nyongesa 96], [Russo 98], [Tang *et al.* 98]).

As is the case in any other discipline, it is not easy to uniquely classify research in the field into different, concrete categories. The following classification scheme is based on that proposed by [Cordón *et al.* 95] and [Cordón *et al.* 96]. It is particularly relevant to note that different representational commitments might play a very important role in establishing further categories for research within the field.

4.2 Fuzzy Genetic Algorithms

Concepts from fuzzy logic can be used to enhance GAs in many ways. A common approach is to regard the parameter set of the GA as the control variables of a fuzzy logic controller, according to some measure of performance. It is also possible to define fuzzy versions of the usual genetic operators, such as crossover or mutation. On the other hand, when a GA is used to solve problems that involve uncertain or imprecise environments, it becomes necessary to provide it with the appropriate mechanisms to deal with this kind of information; a frequent choice is to use fuzzy sets as a modelling tool.

4.2.1 Improving GAs Using FL

In this case, techniques drawn from FSs are used to improve GA behaviour or model GA components. Several different approaches have been reported; they can be grouped as follows:

Representation Tasks

The aim is often to establish fuzzy relationships between genotypes and phenotypes; a FS can be used to define the fuzzy mapping between a chromosome and the potential solution it represents. It is normally required to design more flexible forms of representation in order to work directly with the fuzzy sets that are used to define the genotype/phenotype fuzzy relationship.

Dynamic Control of GAs Using FLCs

This category includes attempts to incorporate "expert knowledge" in the form of fuzzy rules. These rules summarise what are generally considered as contributing factors towards desirable behaviours of GAs. This rule base is then used by a FLC to guide the evolutionary process, modifying the parameters of the GA in order to encourage positive trends, such as an adequate balance between exploration and exploitation, and prevent unwanted tendencies, like premature convergence. The obvious problem with this approach is the requirement to define standard measures describing what the positive features are and, perhaps more difficult still, the different relations between GA parameters that can lead to them. [Lee & Takagi 93] have considered a GA as a dynamical system and used an FLC to control the alteration of the parameters; the FLC itself may in turn be evolved by a meta-GA.

Operator Definition

Fuzzy crossover and mutation operators can induce predefined diversity levels in the population. These operators are especially designed for GAs using real-coded genes.

Assessing Solution Quality

It is often difficult to establish adequate criteria to decide when to stop a particular GA run, especially in a non-interactive setting. It is possible to use fuzzy logic to deal with predefined levels of accuracy or quality in order to assess overall system performance.

4.2.2 Managing Fuzzy Information Using GAs

There seems to be two basic different forms of dealing with fuzzy information using a GA. First, the problem to be solved needs to be defined in fuzzy optimisation/search terms, where imprecision, uncertainty or ambiguity measures are involved, and these have been represented with fuzzy variables and their associated membership functions. Then it is possible to either represent the values of fuzzy variables directly in the chromosome, or to represent non-fuzzy values but use fuzzy fitness evaluation methods. In other words, there is a choice of either representing explicitly fuzzy sets and values, or using a non-fuzzy representation with a fitness evaluation comprising a fuzzification step.

4.3 GAs in Search or Optimisation of FSs

It has been increasingly common to apply GAs in various optimisation and search problems concerning FSs design. As mentioned in the previous chapter, there is a wide range of applications for this kind of system. In many of these cases, expert knowledge is either not available or is difficult to express in an adequate fuzzy rule-form, and it is often complicated to characterise the domain adequately in terms of fuzzy sets. However, adaptive FSs can use an evolutionary process to abstract fuzzy principles from simulations or sampled data. It is feasible to obtain a complete fuzzy knowledge base from scratch. However, if some components are previously available, either initially proposed by an expert or obtained using other means, like neural or statistical techniques, the GA can gradually refine them too.

As described in detail in section 3.2, a fuzzy *knowledge base* is made up of two major components: the membership function definitions of the fuzzy sets involved, and the collection of linguistic associations that define transformations between input and output fuzzy variables. Therefore, it has been common to classify reported studies on GAs used to design FSs in three different categories as follows.

4.3.1 GAs Defining Fuzzy Membership Functions

When a suitable knowledge base is available, the main concern centres on the fine-grained characterisation of the domain in terms of fuzzy sets and variables. Although standard ways of performing this characterisation are commonly assumed, fine-tuning membership function definitions can often lead to significant improvements in performance for the resulting FS.

Fuzzy membership functions can be defined using several parameters, designating their shape, size or position, even the actual number of linguistic terms or values of the fuzzy variables — see section 3.2.2.

Each chromosome encodes a combination of values for the parameters chosen to represent distinct membership functions. Normally, these functions are triangular, so a centre and a base-width for each of them are used. A series of simulations or tests are then used to evaluate the fitness of a particular individual. The GA is thus employed as a search/optimisation tool to obtain suitable fuzzy membership functions.

As mentioned above, [Karr 91] shows a straightforward method to apply a GA to FLC design by adaptation of membership functions of a fixed rule base. Defining these functions consists of specifying a number of numerical parameters, which are tuned by the GA. A direct chromosome representation that is simply a concatenation of the pre-specified number of parameters encoding the membership functions is commonly used.

[Sakurai *et al.* 94] and [Topchy *et al.* 96] also use a GA to optimise the membership function parameters of a FS in a classification task. [Abbattista *et al.* 98] present an "integrated approach to rule structure and parameter identification for fuzzy systems". Although they work on both the rule structure and the optimisation of membership function parameters, their approach uses a GA for the latter purpose only.

4.3.2 GAs Defining Fuzzy Rule Bases

This can be regarded as the complement of the previous case. Once suitable fuzzy membership functions have been defined, characterising the input/output domains, the aim is to devise an appropriate set of fuzzy rules to operate on them. Nevertheless, it is important to note that the adopted definition of the fuzzy membership functions plays a crucial role, since it determines the number of input/output variables and the fuzzy sets that characterise them. In most cases, the definition of rules depends on this information.

Many of the systems belonging to this category employ what is known as a *decision table*, or *look-up table* to represent fuzzy rule bases — see section 3.2.3. Fuzzy associations consisting of n input variables and one output variable can be represented using a n-dimensional decision table. Each "row" on each dimension represents a fuzzy set value for that particular variable. There is a corresponding fuzzy set value on the table cells for each intersection of rows, defining the value of the output variable for the corresponding combination of input values. This structure is encoded in chromosome form and used by the GA to search for a suitable set of rules. Examples of this approach can be found in [Thrift 91], [Hwang & Thompson 94], [Ng & Li 94] and [Li & NG 95].

Several alternatives have been proposed to overcome some of the limitations of the decision table representation. For example, [Hoffmann & Pfister 94] present an alternative hierarchical rule base structure, where "hidden" fuzzy variables are introduced in an attempt to reduce the total number of rules; these variables are used to group several premises together, dividing the rule base in smaller parts. It is important to note that a reorder operator was necessarily introduced, in order to assure a unique and consistent interpretation of the hidden fuzzy terms, before a crossover operator could be applied.

[Ishibuchi *et al.* 94] propose a GA-based approach to the design of fuzzy classification systems. First, a large number of rules are generated from numerical data. Second, a rule selection task for constructing a compact system is formulated as a combinatorial optimisation problem, which is then solved using a GA.

[Chowdhury & Li 96] present a learning method based on the messy genetic algorithm for the optimisation of neurofuzzy controllers. They adopt the conventional cut & splice operators for mGAs, in order to recombine variable-length chromosomes representing neurofuzzy structures, very similar indeed to the Fuzzy Networks adopted for this thesis. However, they retain a flat, non-hierarchical structure: rules are simply strung together in a linear chromosome.

In fact, this thesis could be classified along with the previous examples in this category, since the actual product obtained by the two-level genetic algorithm (2LGA) is a particular kind of fuzzy rule base representation. See section 4.4 below for a detailed description of the work by Feldman that led to the interest in these systems as the subject for this research. Since then, some attempts to overcome the problems associated with fixed-length representations have been proposed. [Buhusi 94] presents a GA for FSs synthesis featuring structures that represent a variable number of rules, introducing an *unequal crossover operator* to work on these structures. [Cooper & Vidal 93] and [Cooper & Vidal 94] propose an encoding scheme that maintains only those rules necessary to control the target system, representing each FS as an unordered list of an arbitrary number of rules. [Hoffmann & Pfister 96] show another approach, similar to the one that had been developed for this thesis, representing a fuzzy rule base by means of a variable-length encoding scheme; it is also based on the principles of messy-GAs.

4.3.3 GAs Defining the Complete FSs Knowledge Base

It is obvious that the membership functions and the rule base of a particular FS are tightly related. Establishing one of them and then searching for the other does facilitate the task somewhat, but it is also possible to use a GA to synthesise both, either at the same time or alternately in stages ([Wang *et al.* 98]).

[Cordón & Herrera 94], [Herrera *et al.* 95] and [González & Herrera 96] describe the use of a GA for learning fuzzy control rules from examples. They propose a 3-stage methodology:

1. A genetic generating process for obtaining desirable fuzzy rules capable of including the complete knowledge from a set of examples, based on an iterative rule-learning approach.

2. A post-processing method for combining and simplifying rules, avoiding the possibility of "overlearning" and removing redundant fuzzy rules.

3. A genetic tuning process for adjusting the membership functions.

An interesting aspect of this approach concerns stage 1 above. First, a *generating method* is used to find one "good" rule over the set of examples. Then a *covering method* determines the covering value this rule has over the set of samples and removes those samples being "covered enough" by this rule, so that the next iteration will generate another rule covering different samples. An earlier, similar approach can be found in [Venturini 93].

[Liska & Melsheimer 94] use a GA for discovering fuzzy rules and membership functions simultaneously. Once the GA approaches convergence, conjugate gradient descent is used to further improve the best solutions by fine-tuning membership function parameters.

4.3.4 Fuzzy Classifier Systems

Another method used is to cast the evolutionary search of fuzzy knowledge bases in a classifier system framework (see section 2.13). [Valenzuela-Rendón 91] and [Carse & Fogarty 94] show examples of the Michigan and the Pittsburgh approaches, respectively. In the former, the model used is very similar to the conventional classifier system; the operation of the system involves an 8-step basic cycle (see figure 4.1, taken from [Valenzuela-Rendón 91]):

1. There is an *input unit*, which receives input values. These values are encoded into fuzzy messages and added to a *message list*.

2. A *classifier list* is scanned to find all classifiers whose conditions are satisfied by the messages in the message list.

3. The message list is erased.

4. All matched classifiers are fired and the produced messages are stored in the message list.

5. An *output unit* detects the output messages and erases these messages from the message list.

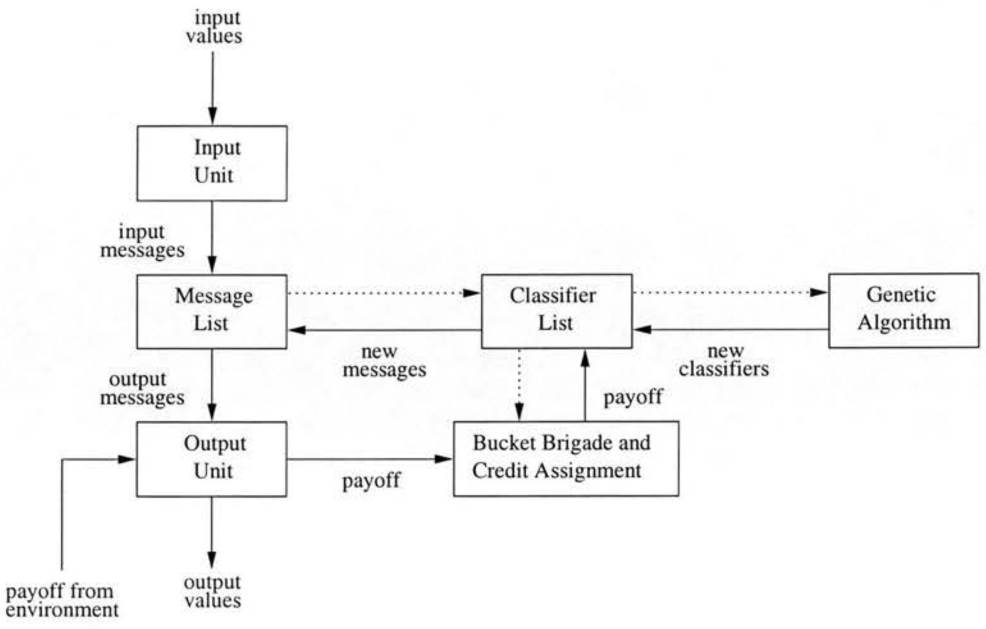


Figure 4.1: Fuzzy Classifier System

6. Output messages are further decomposed into minimal messages in the output unit.
7. Minimal messages are defuzzified and transformed into output values.
8. Payoff from the environment and classifiers is transmitted through the messages to the classifiers.

4.4 Fuzzy Networks

The work presented in [Feldman 93] was fundamentally the starting point for this research — a rational reconstruction of that system was in fact the first step. Thus, the most important features of his approach are presented next.

According to Feldman, a fuzzy network (FN) "is a connectionist extension of a fuzzy logic system allowing partially connected associations, or rules, that incorporate fuzzy linguistic terms". Formally, it is defined as "a transformation from the fuzzy variable domain space $X = \{x_1, ..., x_n\}$ to the fuzzy variable range space $Y = \{y_1, ..., y_p\}$. The rules are evaluated in parallel and the outputs are calculated based on a weighted average of the activated rules." As is the case with other fuzzy systems, a domain of discourse U is defined for each fuzzy variable $x_i, 1 \leq i \leq n$ and $y_j, 1 \leq j \leq p$ with its corresponding fuzzy sets S, which are characterised using membership functions $\mu_S(x)$ for each of the chosen fuzzy set values. Using the same short notation, fuzzy rules can be represented by associations of the form $(A_1, ..., A_n; B_1, ..., B_p; w)$, where the weight term w indicates the relative strength or credibility of the connection between this particular association and the output variables; it is considered to be analogous to the rule:

IF $x_1 = A_1$ AND ... AND $x_n = A_n$
THEN $y_1 = B_1$ constrained by w AND ... AND $y_p = B_p$ constrained by w

A fuzzy network is, therefore, a connectionist representation of a set of fuzzy rules, where each rule indicates a set of connections between fuzzy inputs and outputs. The definition of a FN given in this thesis (see figure 4.2) comprises of:

- an *input node* for each input variable
- an *output node* for each output variable
- an *input subset node* for each fuzzy set value for each input variable
- every input subset node is connected with its associated input node
- an *output subset node* for each fuzzy set value for each output variable
- every output subset node is connected with its associated output node
- a *rule node* for each fuzzy rule
- a connection between a rule node and each input subset node designated by the antecedents in its associated rule
- a *weighted* connection between a rule node and each output subset node designated by the consequents in its associated rule
- the activation of an input node depends on the value of its associated input variable

- the activation of an input subset node depends on the activation of the input node; it is given by the membership function associated with the fuzzy set value it represents — *fuzzification*
- the activation of a rule node depends on the activation of the input subset nodes to which it is connected; it is given by the minimum of these values
- the activation of an output subset node depends on the activation of the rule nodes and the weight of the connections between them — rule node activations are multiplied by their respective connection weights and the output subset node activation is set to the maximum of such products
- the activation of an output node, which indicates the value of the associated output variable, depends on the centroid of the output waveform given by the membership functions of the activated output subset nodes — *defuzzification*

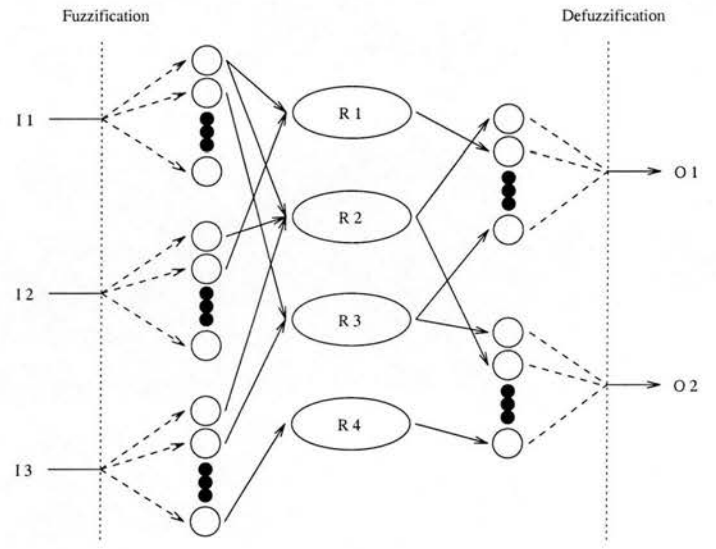


Figure 4.2: Example of a fuzzy network of four rules

By allowing the presence of a special value, represented by the symbol ϕ, it is possible to indicate that a certain rule is not dependent on — or does not affect — the variable at that position. In the connectionist model, this is considered a *cut* connection, meaning there is no link between that particular fuzzy subset node and the rule node.

4.5 Fuzzy Network Synthesis Using Structured and Hierarchical Representations

Even for problems of moderate complexity, the optimum solutions will often have to be found within a search space which is usually vast. Asuming the less stringent task of finding a solution that is simply *good enough*, according to some acceptance criteria, does not alleviate the problem, unless a proper exploration strategy is used that, at the same time, exploits the information which is readily available to speed up the search. It is clear that in the case of FNs, epistasis issues (see section 2.8.1) become particularly relevant, as representation structures modeling rule sets are likely to inherently impose a tighter link between possibly distant genes.

Sections 2.10 and 2.11 described some of the most important issues associated with complications regarding GA representations, along with several relevant attempts to address these in the literature. Most of the systems reviewed introduced a structured approach to the problem, reflected in the representation itself, the architecture of the system, or both.

It would be difficult to assess in an accurate way how well those schemes would fare, if given the task to produce FNs suitable to solve a particular problem, without actually implementing them and carrying out a comprehensive set of tests. However, it is still possible to deduce a few important facts from both the known features of those systems, on one hand, and the intrinsic properties and contrivancies of FNs and the problems associated with their construction, on the other.

Presumably, the GENES system ([Bickel & Bickel 87]) could be easily extended to accomodate the intricacies of FNs, as it already handled (expert system) rules, both individually and at the set level. Although it had no predefined method to adjust nor to optimise the chromosome's length, the system was capable of dealing with representations of a variable size, within a certain range. Given that many fuzzy system implementations assume standard fuzzy set coverages of the input and output domains, the GENES system could incorporate these into the tables it uses to link individual nodes to form each rule. The rest of the system could remain practically intact, although the introduction of a crossover-type operator at the rule level might prove to be a worthwhile addition. The use of a tree-like representation for FNs would certainly be an interesting path to explore (see section 8.3.5).

The basic system presented in [Feldman 93] could also be augmented with the salient features of the sGA, as described in [Dasgupta & McGregor 92] (see also [Ichimura *et al.* 95]), thus providing the original linear chromosome with quasi-hierarchical properties. It is easy to see that, at least in principle, this scheme might serve as an efficient mechanism to achieve a variable-size representation in practice, as the control genes would effectively switch on and off other genes that would, presumably, represent individual rules. The tasks of interpreting the active genes, adapting parameters and introducing relevant recombination and mutation operators at the different levels, would still present themselves as nontrivial problems to be solved.

It was interesting to learn, at the later development stages of this research, that [Radcliffe & Surry 94] had independently devised an analogous strategy to the one presented in this thesis, in order to tackle a different kind of problem, namely a data mining task, using conventional explicit rules in the form of causal predicates. Obviously, their system could be readily adapted to the task of FN synthesis with minor high-level modifications. However, their assumption that the fitness of individual rules can be easily evaluated would have to be re-examined outside the data mining domain, especially in the light of a generalised fuzzy rule characterisation of the problem at hand. It is not always the case, in practice, that the value of a single rule can be objectively assessed in isolation; more commonly, individual fuzzy rules only make sense in the presence of other rules, reinforcing and/or counterbalancing each other. In any case, the fact that their system and the one presented here share a considerable number of structural features, reinforces the notion that the common basis of both strategies exhibits desirable properties that are worth studying and pursuing further.

Although they were developed specifically to alleviate the problems associated with the engineering design hierarchies defined by complete systems, an adaptation from the dual-agent representations incorporated in the systems introduced in section 2.11 could certainly be relevant in the present context also. In fact, it seems intuitively appealing to formulate them in terms of the basic FN features. In particular, an approach similar to the one presented in [Watson & Parmee 97], where a GP operator would take care of the discrete rule sets, while a specifically adapted GA would deal with individual rules, could be an adequate solution to the FN synthesis problem, in close proximity to the proposition made above regarding the use of tree-like structures. Alternatively, an ant colony search strategy could be used instead of the GP scheme, more in line with the original GAANT system definition.

4.6 Combinations of Genetic Algorithms and Connectionist Systems

Since the representation used in this thesis has been modelled as a connectionist structure, this section presents a very brief overview of the main issues regarding the combination of GAs and NNs. However, due to the uniform nature of the structure for the chosen representation, only some general aspects are in fact relevant to the work developed for this thesis.

Apart from proposals to use GAs to replace a learning method, usually backpropagation, a more promising domain of application is that of using a GA to evolve the topology of a NN to solve a given problem. There are two general paradigms to design a representation for a NN structure: direct, or low-level encoding, and indirect, or high-level encoding. The former specifies each connection and unit individually, while the latter uses growth rules of some form or another — e.g. rewrite grammars; see [Gruau 93] for a notable example. Of course, some approaches fall somewhere in the middle, using a sort of parametric encoding, where a list of parameters describes the number of layers, their size and their interconnections.

As described in the previous section, the work presented in this thesis comprises FNs. In the present context, these networks are modelled using a direct representation, since the links and weights that define individual fuzzy rules are explicitly encoded, using a particular structure for a given set of "nodes", as shown in figure 4.2.

A thorough survey that covered the research in combinations of GAs and NNs up to 1992 can be found in [Schaffer *et al.* 92], according to whom, the following tasks need to be addressed by these approaches:

- A representation must be devised for the class of topologies of interest. As these schemes become more elaborate, the step of mapping the genotypes to the phenotypes takes on a more prominent role.
- There must be a protocol for exposing the phenotypes to the task environment.
- There is usually, but not always, a local learning scheme where the phenotypes tune themselves. Some researches have placed the dual burdens of learning the topology and tuning the weights onto the GA.
- There must be an evaluation of fitness.
- There is the evolutionary step of producing new genotypes from the old ones.

The following chapter describes in detail how particular instances of these problems were dealt with during the design stages of the 2LGA system.

Chapter 5

2LGA Described

This chapter describes in detail the proposed novel architecture and operation of a hierarchical GA. The basic building blocks this work is based upon are introduced and an overall picture of how all the pieces fit together is given. The chapter ends with a description of the interactions that take place between the different parts that make up the distinct two-level genetic algorithm (2LGA).

Special attention has been paid to the more fundamental aspects and unique features. Because it is meant to be a simple introduction to the peculiarities of the new system, it should be possible to perform a rational reconstruction of an elementary 2LGA based on the description given here. Several variations on the basic model, enhancements and implementation details were devised during the experimental stages, but these will be discussed subsequently as they are introduced.

For practical purposes, it is possible to give a global view of the system in two parts: 1) the kind of fuzzy system that will be obtained and 2) the evolutionary system that will be responsible for producing it. The inherent features of the former are closely related to the representation and operational scheme devised for the latter. However, it is important to note that, although the structure of the 2LGA owes much to the intrinsic properties of the sort of fuzzy systems it works upon, its use is not in principle restricted to the automated synthesis of such systems only.

5.1 The Final Product — Fuzzy Networks

As described in section 4.4, a simple GA is used in [Feldman 93] to synthesise the rule base of a specific kind of fuzzy system. The input/output fuzzy set membership functions are fixed beforehand and the task of the GA is to find a suitable set of rules to solve a particular problem. In the example shown in that paper, each single chromosome in the population represents a sequence of rules, whose fitness is determined by testing its performance as a controller strategy against a model of the system to be controlled.

Apart from the problems normally associated with traditional GAs, notably the linkage problem discussed in section 2.10.1, an obvious limitation of the approach presented there is that the coding scheme features a very low degree of flexibility: the size of the rule base, i.e. the chromosome's length, must be predefined and fixed for each run; a trial-and-error initial stage seems unavoidable under that scheme, as is suggested in the paper itself. However, the formalism that was proposed then to model the rule base of a FS, the so-called *fuzzy network* (FN), seemed to be a powerful yet simple one (see figure 4.2). Thus, the use of FNs was adopted in this work.

The need to design a more flexible GA representation for FNs gave rise to one of the main contributions of this research work: the development of a hierarchically structured GA. This form of representation strongly influenced both the architecture and the operation of the entire system, as is described in detail below.

FNs are made of rules, which in turn are made of links. Instead of concatenating groups of links into a long string to form a rule set, it was decided to give the representation a two-level structure. At one (high) level, there is a population of *sets*, i.e. rule bases. At the other (low) level there is a population of *elements*, i.e. single rules. Each set may contain a variable number of elements and each element may, in turn, be contained in several sets (see figure 5.1). Sets are recombined with other sets while elements are recombined with other elements, so that the two co-evolve in separate yet interdependent populations.

At this level of description, it is convenient to think in terms of the entity/relationship model ([Chen 76]) used in classical relational database systems ([Codd 70]): *sets* and

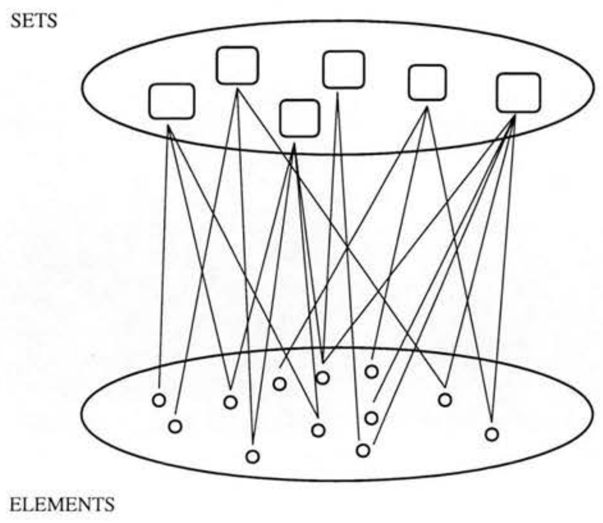


Figure 5.1: Two-level GA

elements are different entities, associated by the (many-to-many) *membership* relation. However, the actual design and implementation were based on closely related Object Oriented concepts instead ([Cox 86], [Booch 91]).

Since representation plays such an important role, it is imperative to attempt to provide GAs with the capability to somehow self-adapt or evolve their inner structure. This would allow them to effectively alter their form of representation and/or the functionality of their operators, according to the characteristics of the problem being solved. Although the choice of coding scheme is a design decision which cannot be fully automated, if the representation is flexible enough to self-adapt, thus reconfiguring itself to better suit the problem's environment, a good balance between a problem-specific representation and the universal binary encoding is achieved. That was the main objective of the two-level architecture, together with the variable-length representation used for the sets populations, described bellow.

5.2 The Sets Population — Rule Bases

The top level population is used to represent potential solutions as single units. Although the information is in fact indirectly represented at this level, it is here that the entities, which the entire system is actually trying to evolve, reside.

5.2.1 Representation

Each individual in the sets population will designate a particular FN. The basic idea is to represent each one of these — a complete fuzzy rule base — as a chromosome of variable length. Genes in these chromosomes will be indices into the elements population, each one designating a particular rule:

R_0	R_1	R_2	$\cdots$	R_n

Under this scheme, the efforts of the GA working at this level will concentrate on finding *useful combinations* of rules. The aim is to isolate the problem of searching for the best way of putting together independent fuzzy rules, so that they interact as positively and efficiently as possible. Furthermore, the adequate number of elements that are actually needed to solve the problem at hand will be discovered, and optimised, by the evolutionary process itself.

In contrast with other systems using a variable-length representation, it is important to note that it is possible to sidestep the problems of over- and under-specification altogether (see section 2.10.2), since any chromosome at least one gene long represents a valid FN. This is one of the major distinctions between 2LGA and other approaches utilising variable-length representations: each gene is in itself a complete solution; the purpose of the GA working at this level concentrates on searching for useful combinations of individual genes that, when put together in a single chromosome, produce a good quality solution. The actual information required per individual is thus very simple: a list with all the "names" of the elements which it contains — but not the elements themselves.

5.2.2 Operators

Selection

The operation of the system is not restricted to a particular selection mechanism. The choice is entirely arbitrary and would not have any fundamental effect on the system's components or their interactions. For practical reasons, and this is merely a case of personal preference (see section 2.6), a modified form of tournament selection, modelled on the 'marriage problem' from dynamic programming, has been used for most of the experiments:

```
1. choose one chromosome at random; call it best-so-far
2. repeat up to $N$ times:
        choose one chromosome at random; call it one-to-try
        if fitness of one-to-try $>$ fitness of best-so-far return one-to-try
3. return best-so-far
```

This selection strategy was devised by Peter Ross and it has been implemented in the Parallel Genetic Algorithms testbed (PGA) system ([Ross & Ballinger 93], [Ross 96]). It appears to be considerably less sensitive to the choice of N than conventional tournament selection is.

Recombination

Because they were specifically designed for variable-length representations, analogues to the *cut and splice* operators from mGAs were adopted (see figure 2.2). Two parents are selected for reproduction. Each of them can be *cut* into two parts with a certain probability, which is proportional to the length of the chromosome. Each of the resulting parts — from 2 (none were cut) to 4 (both were cut) — can be *spliced* to the others in a meaningful way, avoiding splicing back pieces that were just previously cut. It is clear then that 1, 2, 3 or even 4 children can be produced in this way each time these operators are applied (see figure 5.2).

Mutation

Before being introduced into the population, each gene of the newly created chromosomes can be subjected to mutation with a given probability. When a gene is altered in this way, another valid index to some element (rule) is picked at random to replace the current one.

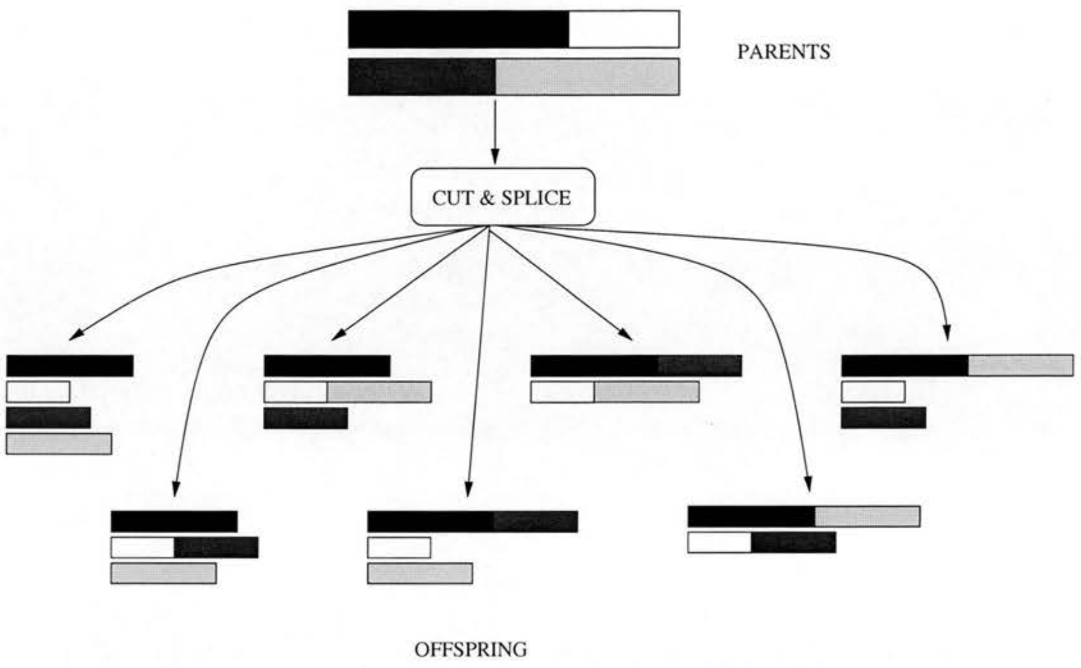


Figure 5.2: Cut & Splice possibilities

Replacement

The new chromosomes replace individuals selected randomly from all but the best in the population, so a form of élitism is enforced. Since both populations are tightly interconnected, it is necessary to update those elements that are affected by the introduction of the new sets: each element contained in the new sets will have to add the name of this new set to its list of sets to which it belongs. Likewise, the names of the old sets being replaced will have to be deleted from the list of all those elements that were members of them.

5.2.3 Fitness Evaluation

This is entirely problem-dependent and specific examples will be given in sections 6.1.1 and 6.2.3 in the next chapter. For explanatory purposes, a brief mention of the approach that can be taken is included here. In a control task, for instance, the fitness of each set is calculated using a function averaging the performance and stability of the particular FN that it represents. This is done over a series of simulations as a FLC, for the specific control problem. The initial conditions for each simulation are uniformly distributed over the range of possible values. In a classification problem, calculation of fitness is based on the number of cases that the FN can classify correctly.

As will be presented in detail in the next chapter, the concept of parsimony (see section 2.10.2) was incorporated to further refine fitness evaluation. In the end, it played a very important role in controlling the behaviour and performance of the 2LGA.

5.3 The Elements Population — Single Rules

The bottom level population is used to represent rules as independent individuals. The aim is to isolate the problem of finding the appropriate elements that useful fuzzy rules are made of, conforming to the requirements of the problem at hand. Thus, the building blocks for the sets population are evolved at this level as single units.

5.3.1 Representation

The elements population uses an ordinary (bit-string) representation for each fuzzy rule. Thus, fixed-length chromosomes with binary-coded parameters can be used to represent each one of them, mainly based on what was suggested in [Feldman 93]:

Antecedents				Consequents				Weight		
#	#	#	···	#	#	#	···	#	#	#

###	FUZZY SUBSET
000	Negative large (NL)
001	Negative medium (NM)
010	Negative small (NS)
011	Zero (ZE)
100	Positive small (PS)
101	Positive medium (PM)
110	Positive large (PL)
111	No link (CUT)

It was certainly possible to use an alternative representation for the individuals in the elements populations. The choice of the traditional fixed-length, binary representation was a practical one. It seemed reasonable to keep the system simple whenever possible, in order to prevent obscuring the main features under investigation. Nevertheless, a more flexible and integrated form of representation will be a desirable feature to include in future versions of the system.

In practice, in order to keep computational costs to a minimum, each element also contains a list of all those sets to which it belongs, establishing bi-directional links between populations. This introduces a certain degree of redundancy, which must be dealt with accordingly, but permits a rapid identification of membership relationships that would be prohibitively expensive otherwise in computational terms.

5.3.2 Operators

Selection

No specific selection mechanism is required and the actual method used is simply a matter of choice. In this case, the same modified version of tournament selection that was used for the sets population (section 5.2.2) is used here.

Recombination and mutation

Since these are ordinary bit-string chromosomes, the usual two-point crossover (section 2.7.1) and bit-wise mutation (section 2.7.2) were chosen.

Replacement

In a large number of cases, the replacement mechanism plays a secondary role in the functioning of an ordinary GA. A sort of "inverse selection" is commonly applied to decide which individual will disappear in order to make space for the newcomer. However, in this case it was also necessary to devise a suitable way to introduce new elements taking the place of existing ones. The problem arises because something must be done with the links belonging to the individual that is going to be replaced and, more importantly, a way to assign the links for the new element must be carefully designed.

Two different strategies have been tested. In both of them, the new chromosome replaces another individual selected at random, with an élitist mechanism enforced (for full details, see section 6.1.4):

Adopted The new chromosome maintains all the links from the sets population that the element being replaced had. As a result, every set containing the element that was chosen for replacement will now contain the new element instead. However, these sets (their indices) remain unaltered; it is only the contents of one of the elements being pointed at, that has changed.

Inherited The new chromosome will 'inherit' some of the links from the sets population from its parents: some of the sets that originally contained the parents (randomly chosen with a given probability) will now contain the new child, either 'in addition to' or 'instead of'. In the first case, a set that contained one of the child's parents will now contain both the parent and the child. In the second case, it will no longer contain the parent and the child will take its place. The links originally pertaining to the replaced individual are removed in both cases.

5.3.3 Fitness Evaluation

For most problems, it would be difficult to assess the worth of an individual rule independently. One alternative is to use a scheme for apportionment of credit similar to that taken by the Michigan or the Pittsburgh approaches to classifier systems (see section 2.13). However, after obtaining encouraging results, it was decided that the fitness of each element is simply a function of the fitness of all those sets in which it is contained, i.e. the average. This scheme seems to provide an efficient way to estimate its fitness and, at the same time, reinforce the interdependent nature of the relation between the sets and the elements populations.

5.4 Interaction between Sets and Elements

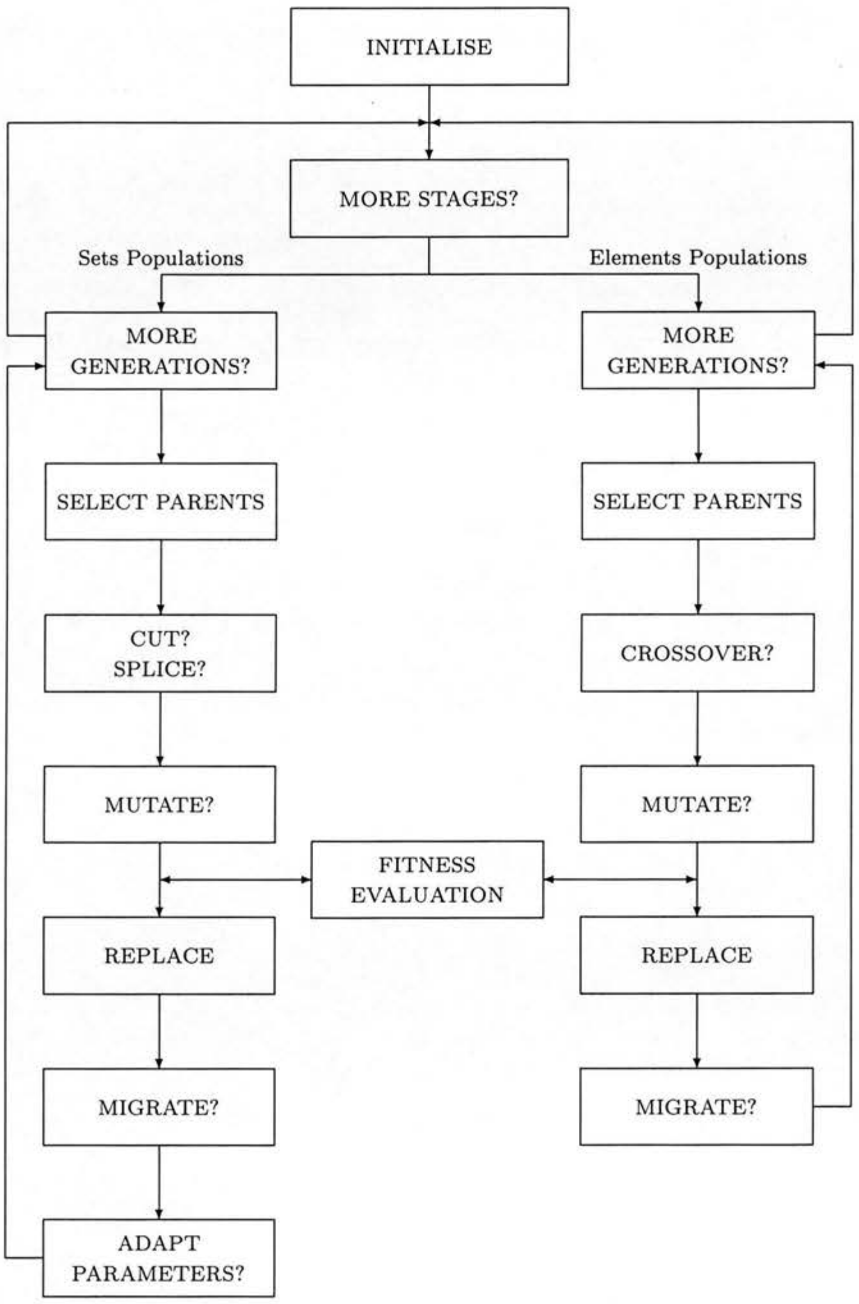
Within each level, it was decided to adopt the island model, thus having multiple subpopulations with intermediate migration rates. After the initial experimentation phase, where different values for these system parameters were systematically tested, it was decided to perform the majority of experiments using five populations at each level, containing one hundred individuals each. One migration takes place every one hundred generations, at both levels, from alternating populations. A steady-state approach was adopted, so that newly produced individuals are inserted immediately in the population (see section 2.5 for the general issues, and section 6.2 for a comparison of different parameter choices).

Obviously, the final solution(s) will be defined by one (or more) of the individuals in the sets population, which is (are) in turn made of a number of specific individuals from the elements population. Ideally, both the elements and the sets populations would co-evolve in parallel, the former attempting to improve individual rules while the latter is trying out different combinations of these. The current implementation simulates this process in a serial fashion, the only disadvantage being the deterioration in time-related performance. It is important to note that, while the fitness of a set depends entirely on how well the elements it contains combine together, the fitness of an element depends, in turn, on the fitness of those sets in which it is contained.

As a result of this dual, hierarchical representation, it is possible to define a separate developmental pace for each population, i.e. different reproduction rates for each level (see section 6.1.3). Thus, a complete *stage* will consist of M generations for the elements population and N generations for the sets population. As will be shown later, the values of M and N have direct implications on the operation of the system. This can be exploited to enhance the behaviour and overall performance of the 2LGA.

5.5 Diagrammatic Overview

A schematic diagram of the elementary execution steps performed by the system is shown below. It is evident that the fundamental evolutionary process prevails: the basic GA cycle is essentially maintained within each population. The distinguishing features of 2LGA reside in the underlying hierarchical representation, and the interactions between system components and parameters. At the algorithmic level, apart from the obvious high-level stage-related tasks, these implicit interrelations become apparent when fitness is evaluated for either a set or an element, as shown in the diagram.


INITIALISE
MORE STAGES?
Sets Populations
Elements Populations
MORE GENERATIONS?
SELECT PARENTS
CUT? SPLICE?
MUTATE?
FITNESS EVALUATION
REPLACE
MIGRATE?
ADAPT PARAMETERS?
MORE GENERATIONS?
SELECT PARENTS
CROSSOVER?
MUTATE?
REPLACE
MIGRATE?

Chapter 6

2LGA at Work

"But to achieve anything you must tackle the abstract property in a concrete situation, that is you must build a program to do some task that requires search to be controlled, knowledge to be represented, knowledge to be learnt." [Bundy *et al.* 89]

The results of the application of 2LGA to control and classification problems are presented in this chapter. The previously described basic model was used, but several variations were introduced, both to enhance the performance of the system and to investigate different design alternatives. Some of them led to interesting results and were of great help in finding feasible explanations for the observed phenomena. The first experiment was mostly used as a proof-of-concept model, derived in its most elementary form from the research that inspired the development of 2LGA. The software modules corresponding to the fuzzy system were implemented in C++, while the GA modules were initially programmed in C, within a Unix operating system. The subsequent experiments served as testing grounds for more elaborate and complex ideas, and the whole system was re-implemented in C++, observing object-oriented design and programming methodologies. It has been ported to the Linux and Windows environments.

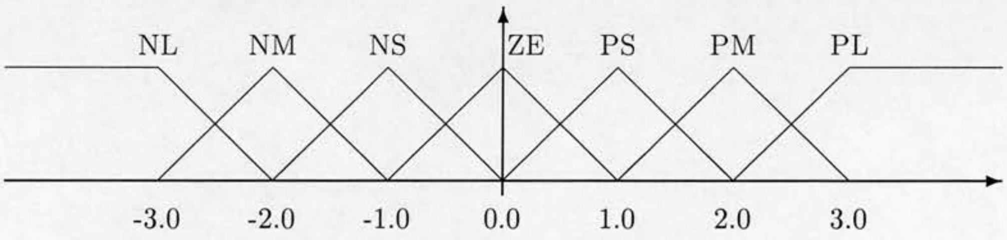
6.1 First Experiment — a Control Problem

For comparison purposes, 2LGA was initially used to work on the same problem that was presented in [Feldman 93] (see section 4.4) which, in turn, was taken from [Thrift 91], namely the *cart centring* problem. Succinctly put, a good control strategy should be capable of taking an imaginary cart, moving along a straight, frictionless rail to a specific central point as quickly as possible. It should also keep the cart as close to that point as it can and for as long as possible. Moreover, the controller should be able to perform well regardless of the initial conditions for the position x and speed v of the cart.

The cart is assumed to have a mass m and the variable to control is the external force F exerted upon it to achieve the centring task. The simulation is based on Euler's method to approximate the dynamical system, with a time step $\tau = 0.02$ sec, given by:

$$\begin{aligned} x(t+\tau) &= x(t) + \tau v(t), \\ v(t+\tau) &= v(t) + \tau \frac{F(t)}{m}. \end{aligned}$$

The same fuzzy set values {NL,NM,NS,ZE,PS,PM,PL} were used for the two input variables x and v, and for the output variable F, characterised by triangular membership functions, defined over each domain in the interval [-5, 5] as follows:



With the initial, intuitively chosen values for mutation (0.002), cut and splice probabilities (0.5), ordinary and simple selection/replacement strategies, the system produced results comparable to those obtained using the conventional GA reconstructed from Feldman's work. However, since a few new parameters had been introduced, an initial exploration of the parameter space became an important task. Consequently, several distinct aspects of the new system were tested, such as different cut and splice probabilities, reproduction rates between populations and various replacement strategies for the elements population. Before these issues are discussed, the next section gives details about the fitness evaluation function used and the parsimony concept involved.

6.1.1 Fitness Evaluation

The fitness measure for each control strategy was initially calculated by the following function:

$$f = \frac{1}{N} \sum_{i=1}^{i=N} (T_{max} - T_i)(1 - e_i)$$

where

- N : number of runs (varying initial conditions)
- T_{max} : number of time steps to complete a simulation
- T_i : number of time steps to reach the target position in simulation i
- e_i : normalised error term in simulation i

$$e_i = \frac{1}{x_{max}(T_{max} - T_i - 1)} \sum_{j=T_i+1}^{j=T_{max}} |x_{ij}|$$

- x_{max} : maximum possible distance from target position
- x_{ij} : distance from target position at time j in simulation i

Parsimony

In order to exert controlled pressure against unnecessarily big individuals, it is possible to add explicitly to the fitness evaluation function a penalty factor, usually called *parsimony,* (see section 2.10.2) which grows proportionally with the size of the solution. The new fitness value can then be defined as

$$f_{new} = max(0, f_{old} \cdot (1 - P \cdot S))$$

where P is the parsimony factor and S is the size of the set, i.e. the number of rules it contains. Suitable values of P yielded moderate and justified set growth patterns, so that larger sets are fitter because of the quality of their genetic material, not because of its quantity. As will be discussed in section 6.2 below, although other parameters also had a strong influence on the length of individuals, parsimony provided a direct way to keep their size under control.

6.1.2 Cut and Splice Probabilities

The cut and splice operators work in a complementary fashion: while the former is responsible for breaking down rather large chromosomes into smaller chunks, the latter provides the means to build up fitter chromosomes out of separate blocks. Consequently, the probability assigned to the occurrence of one operator is likely to affect the usability of the other, i.e. they are interdependent.

In order to explore the relation between the probabilities of the two operators, several tests were carried out. As a first approach, every parameter was set to a default value for all the experiments, while the cut and splice probabilities varied systematically over the [0, 1] range, averaging each test over a series of 20 runs.

As is made evident later on, the relation between the cut and splice probabilities has a strong influence in the quality of the final results, shaping the average size of sets throughout the evolutionary process. In fact, it became possible to predict, for a given replacement strategy (see section 6.1.4 below), whether a higher or a smaller probability for either the cut or splice operator, or both, would lead to improved performance. In the second experiment, discussed below (section 6.2.6), this idea was exploited to allow the system to adjust these parameters automatically in an informed manner.

6.1.3 Reproduction Rates between Populations

A complete stage can be arbitrarily defined as M generations for the elements population and N generations for the sets population, making it possible to regulate the relative speed at which both populations evolve with respect to one another. Different fixed rate combinations were systematically tested in an attempt to see how these affect the overall performance of the system and to disclose how they relate to each other. The results were as expected: with $M/N > 1$, performance is severely degraded in the initial stages but seems to have a rather positive effect towards the end of the experi-

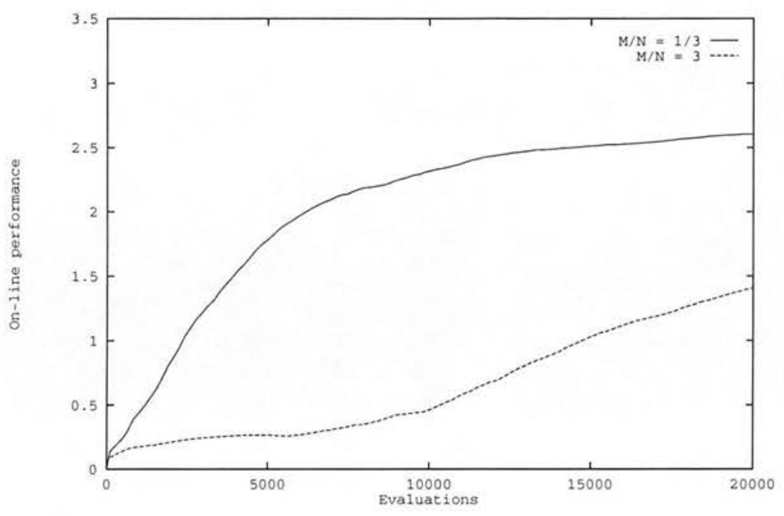


Figure 6.1: $M/N = 3$ vs. $M/N = 1/3$ — On-line performance

ment, when the evolutionary process starts to converge; conversely, with $M/N < 1$, a very fast and effective initial progress is observed, but it is followed by a rather slow improvement in the later stages. This is made evident by comparing the on-line performance — i.e. the accumulated sum of fitness values divided by the total number of evaluations — of both strategies (figure 6.1).

As the graph comparing the average set size for both approaches suggests (see figure 6.2), a small M/N allows the system to find very quickly important combinations of those rules that are readily available, but fails to help fine-tuning individual rules. On the other hand, a large M/N prevents the system from building useful sets at first, but allows it to improve the quality of the more elaborate rules found in the later stages. These observations opened up new possibilities for self-adaptive mechanisms, which will be described in section 6.2 and discussed further in chapter 8.

6.1.4 Replacement Strategies for the Elements Population

Once the new individuals have been produced in the elements population, a suitable way to introduce them while replacing some of the existing ones poses a deceitfully simple problem. Apart from the usual considerations concerning replacement methods, two additional issues have to be dealt with:

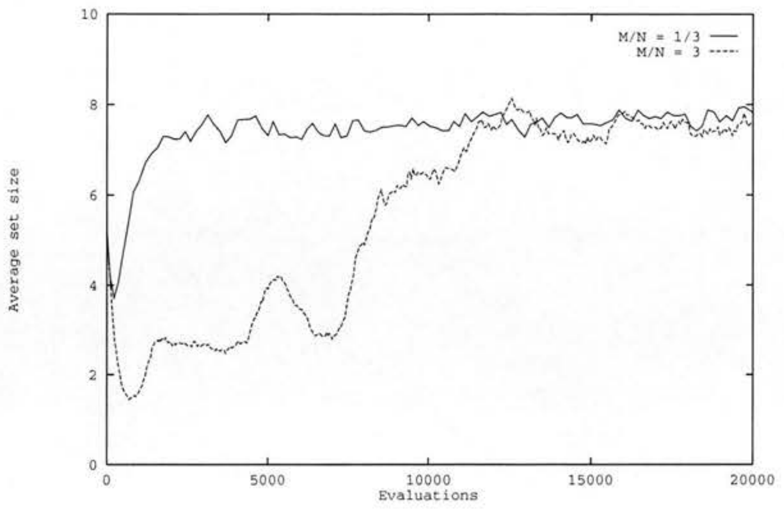


Figure 6.2: $M/N = 3$ vs. $M/N = 1/3$ — Average set size

- What to do about the sets that contained the elements to be removed
- How to appropriately assign the new elements to some sets

To account for both of these problems, two basic strategies were devised: *adopted* and *inherited* replacement, with two variations for the latter.

Adopted Replacement

This is a rather naive but surprisingly effective approach: the new element simply replaces the old one, arbitrarily *adopting* all the links to/from the sets population, i.e. it will be contained by the same sets that contained the one it is replacing. In practice, this is computationally very efficient, since no other elements are affected and there is no need to identify relationships with the other population and make alterations to the sets involved. Figures 6.3 and 6.4 show the typical observed behaviour.

Every time an element is replaced in this fashion, only the sets that previously contained it will be directly affected. Obviously, the new rule can be quite different from the one it is replacing. For all intents and purposes, specially from the affected sets point of view, this is a sort of *external* mutation. It can be seen in fact as an "indirect-but-directed" form of set alteration: the material contained in new elements is always

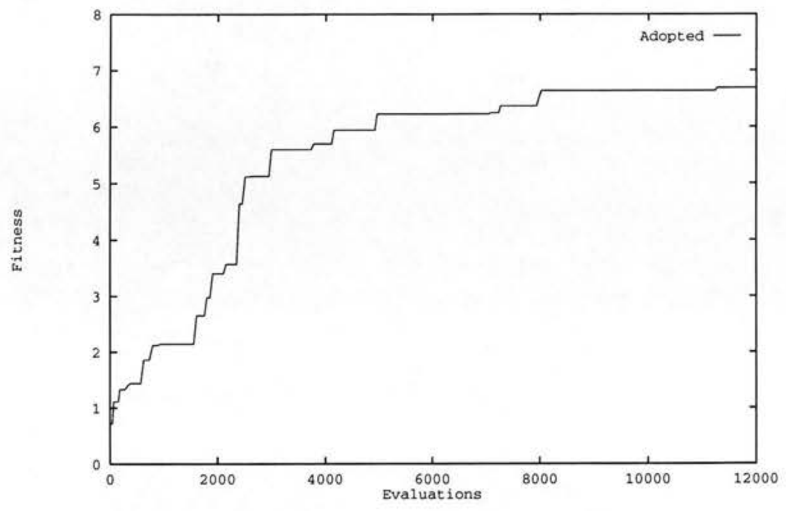


Figure 6.3: *Adopted* strategy — best set's fitness

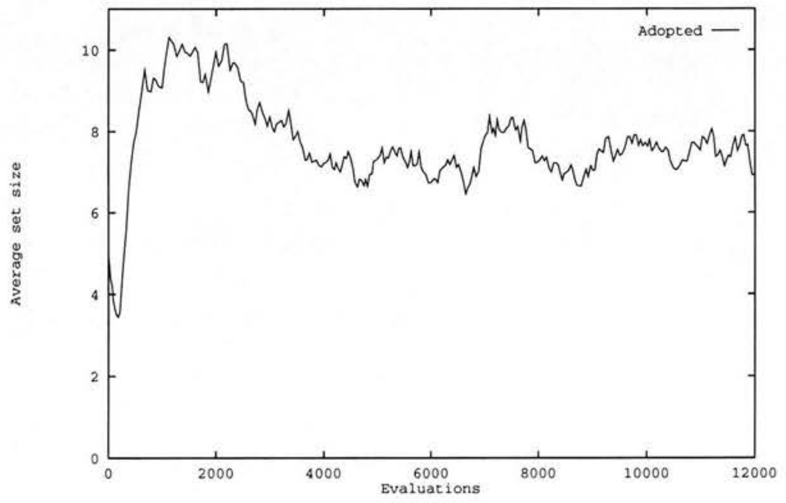


Figure 6.4: *Adopted* strategy — average set size

obtained from the relatively fitter ones.

It is important to note that unfit elements are probably not contained by many sets. In fact, many of them are not contained by any set at all. That is why the elements to be replaced were first chosen at random. Exerting pressure to select only the relatively bad ones proved to produce too small (and slow) an impact to be of any help. This is because new elements would not be used, at least not for a long time, unless they replaced an element that was actually contained by one or more sets. A different approach will be described for the second experiment, where a crowding bias was incorporated into the replacement mechanism.

Under this replacement strategy, the best performance was obtained when the cut probability and the splice probability were equal. No significant difference was perceived within the [0.5, 0.75] range.

Inherited Replacement

This form of replacement was devised as an attempt to improve the way in which new individuals from the elements population are assigned to individuals from the sets population. Instead of using the information that the element to be replaced can provide to this effect, which obviously in most cases bears no relation at all to the new element, the idea is to obtain it from the parents themselves. New individuals, although not necessarily better than their parents, often preserve their most important features. Adding the new element to a set already containing one of the parents is therefore not likely to produce a negative effect too often; in fact, if the offspring is identical or very similar to the parent, it will produce no discernible effect and parsimony will certainly help to discard either of them later. Two alternative ways for the offspring to inherit information from its parents were designed and can be described as follows:

Substitution. The new element will inherit, with a given probability (see section 6.1.5 below), some of the links from its own parents, thus taking their place in some sets, i.e. a number of links to the parents will be possibly *substituted* by links to the offspring. The following steps are required:

1. Clear the element to be replaced:

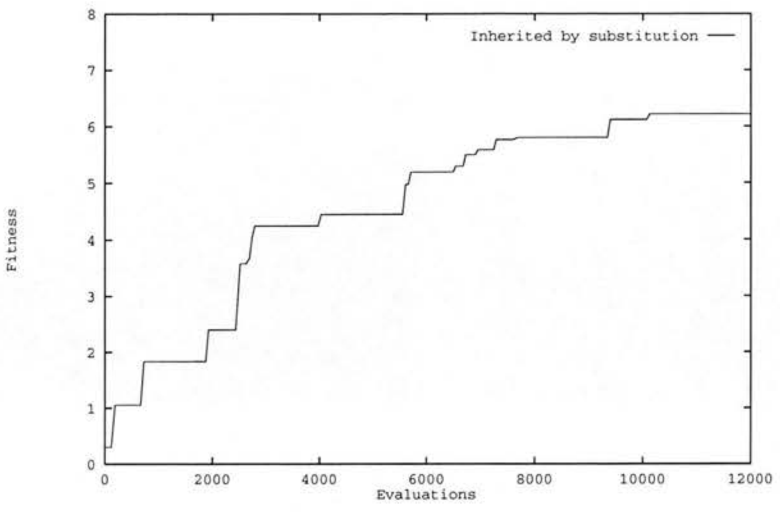


Figure 6.5: *Inherited by substitution* strategy — best set's fitness

Ask the sets containing it to remove it from their list.

2. Substitute the new element in some of its parents' sets:

   With a given probability, take one of the sets containing one of the parents and ask it point to the newly created child *instead of* pointing to the parent.

Examples of the effects of this strategy are shown in figures 6.5 and 6.6 below.

Addition. The newly created element will be *added*, with a given probability, as a member to some of the sets that already contain one of its parents. The aim is to prevent fit parents from losing their links as generations pass by (see figures 6.7 and 6.8):

1. Clear the element to be replaced:

   Ask the sets containing it to remove it from their list.

2. Add the new element to some of its parents' sets:

   With a given probability, take one of the sets containing one of the parents and ask it point to the newly created child *in addition to* pointing to the parent.

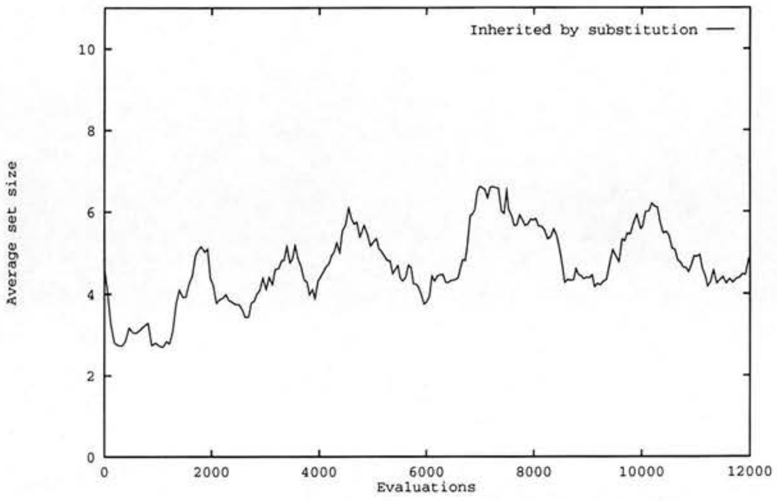


Figure 6.6: *Inherited by substitution* strategy — average set size

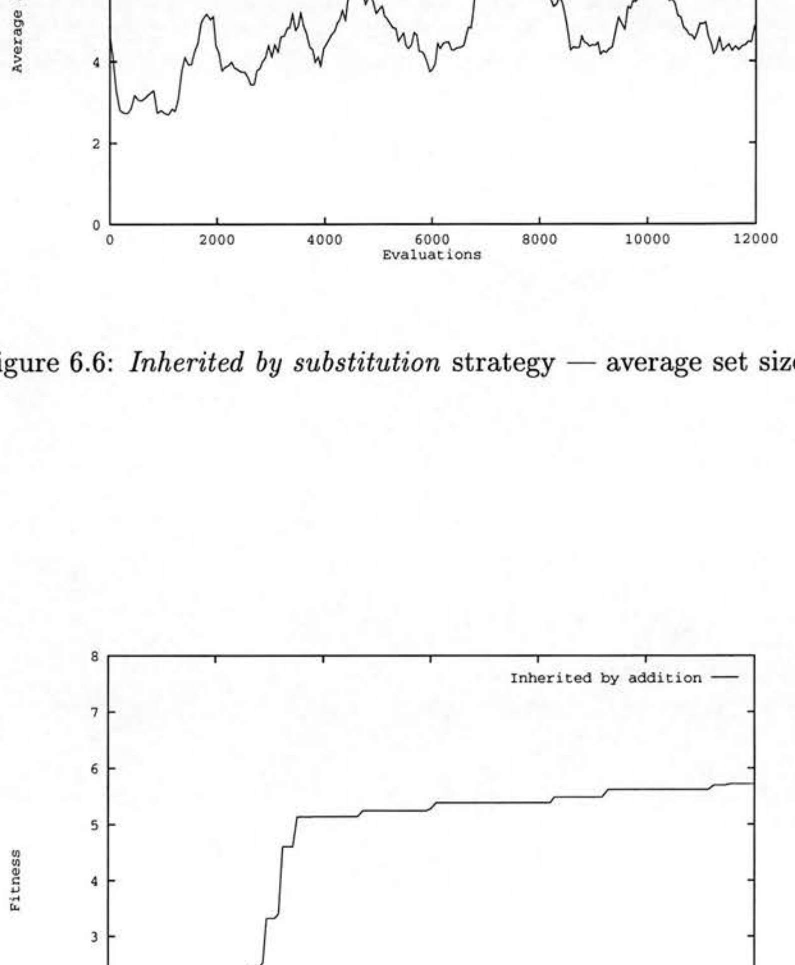


Figure 6.7: *Inherited by addition* strategy — best set's fitness

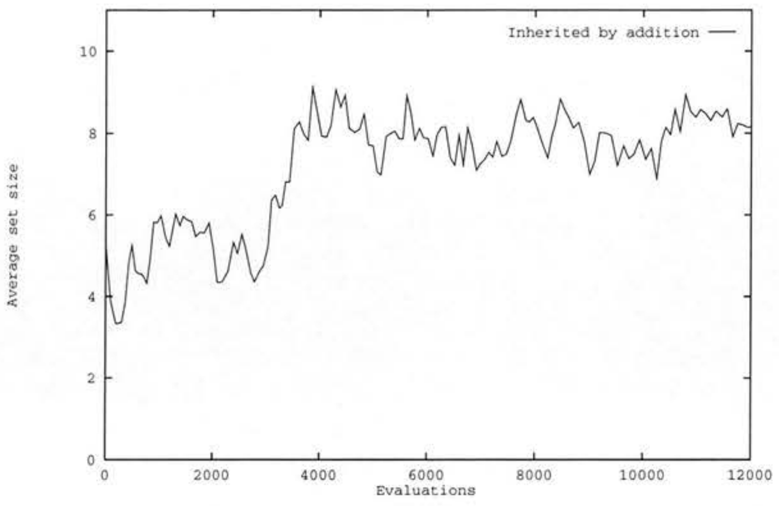


Figure 6.8: *Inherited by addition* strategy — average set size

In both cases, it clearly pays off to favour the selection of relatively unfit individuals for replacement. Under these inheritance strategies, not only the sets that contain one of the parents of the new element are affected, the replaced element is removed from every set that previously contained it as well. Since the disappearing rule is likely to be a bad one, this is in fact a desired consequence. Moreover, because it is possible to occasionally render a set totally empty, it is necessary to reinitialise the rather unfit ones when this happens. Thus, as another positive side effect, membership links between the two populations are more uniformly distributed among individuals in the elements population.

6.1.5 Heredity

In order to control the number of links that a new individual from the elements population will inherit from its parents, a new parameter was introduced: the *heredity* factor. It is a real number between 0 and 1 (typically 0.5) which simply designates the probability of inheriting, either by substitution or by addition, each link from each parent.

After an initial exploration of the parameter space, it became clear that, together with both the cut & splice probabilities and the parsimony parameters, heredity played an important role in directly shaping the curve corresponding to the average set size. This, in turn, is a determinant factor in the overall performance of the system.

Experimentally, keeping the parsimony factor constant and low ($0.005 \leq p \leq 0.01$), optimal values for the cut & splice probabilities were found to be around 0.7 and 0.3, respectively. This is in accordance with the intuitive notion that more cutting and less splicing would tend to yield shorter individuals, keeping in balance with the increase in set size encouraged by the inherited replacement strategies, particularly the addition variety.

6.1.6 Suppressing Spurious Evaluations

Every time a new set is created, its fitness is evaluated before it is inserted in the population, as is customary with most GAs. However, every time a new element is created and inserted in the population, several sets (their links) might be altered too as a result of this operation. While some sets have received a new element, some others might have lost another one.

It is possible to keep a totally faithful fitness value for each set, but this implies a series of *update evaluations*, performed necessarily every time a new element is introduced into the population. Because elements may indeed be contained by a large number of sets, and because fitness evaluation is often computationally a rather expensive process, it was decided not to re-evaluate the fitness for the affected sets, allowing unfaithful fitness values to occur. The rationale for this decision is as follows:

- The vast majority of alterations due to an element insertion have minimal immediate effect on most sets, particularly on the relatively fit ones. Evidence supporting this assumption is presented in figure 6.9, which shows the histogram for the difference in fitness values before and after the update evaluation was performed for a typical run.

- New sets, obtained from old sets with or without faithful fitness values, will always be evaluated before they are introduced into the population. The alterations that might have affected the parents will now be reflected in the faithful fitness value just calculated for the offspring, even if the parents' fitness values remain

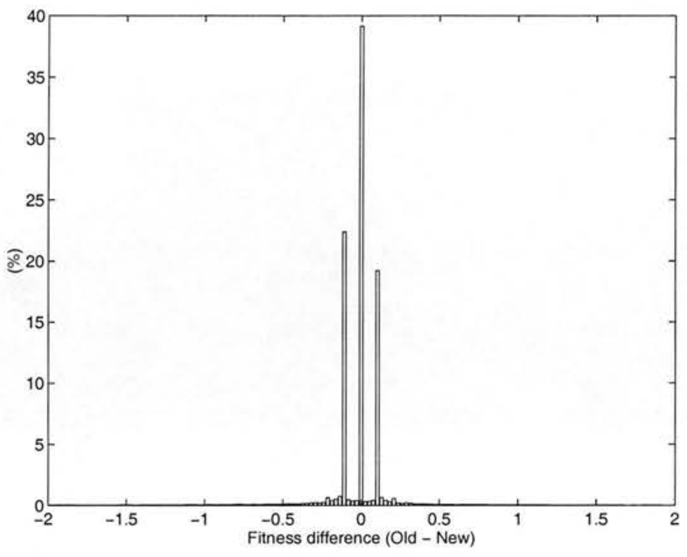


Figure 6.9: Histogram of fitness difference for update evaluations

outdated. It is clear that the influence of a single individual whose fitness has not been updated could only be significant for a rather limited number of generations.

- Due to the élitist mechanism, the best set is never affected by the insertion of a new element. Its fitness value is guaranteed to remain accurate until a new, better individual comes along and proves to be the fittest set, up to that generation.

6.1.7 Summary of Results

2LGA did not fail to produce a suitable FN for the control task, literally, in thousands of runs. It would sporadically take a long time to converge and, mostly during the first stages of experimentation, it would occasionally come up with a rather large set of rules to solve the problem. However, after the initial exploration of the new parameter space had been performed, the relations between these parameters and the system started to become clear. It was then possible to predict, to some extent, the behaviour of the main aspects of the system. As discussed in the next chapter, it became feasible to make informed decisions to tune up some of the different parameters in order to obtain improved performance. This is often a difficult task when working with ordinary GAs.

2LGA was compared against the simple GA, as reconstructed from [Feldman 93]. Every system parameter, such as heredity and parsimony factors, mutation, cut and splice probabilities, were explored in turn (see section 6.2). However, at this stage, the focus was placed on:

- the newly developed replacement strategies for the elements populations,
- the different reproduction rates between populations.

The results are summarised in the table below, where [i-s] means *inherited-substitution* replacement, [i-a] means *inherited-addition* replacement, M is the number of generations for the elements population and N is the number of generations for the sets population. The average over 10 different runs was used in every case.

GA	Average max. fitness	Evaluations to reach 80% of average max. fitness
Generic	5.92	5600
2LGA adopted, $M/N = 1$	6.24	5566
2LGA [i-s], $M/N = 1$	6.25	4158
2LGA [i-a], $M/N = 1$	5.94	4467
2LGA [i-s], $M/N = 1/3$	6.60	3686
2LGA [i-s], $M/N = 3$	6.34	13518

It is clear that, for this particular problem, the inherited-by-substitution replacement strategy proved to be superior in terms of both convergence speed and quality of results. Additionally, a faster reproduction rate for the sets populations resulted in a considerable reduction in the number of evaluations needed to obtain a good solution.

6.2 Second Experiment — a Classification Task

In an attempt to investigate further the capabilities of the new approach, it was considered important to apply the 2LGA to the solution of a different kind of problem. The actual purpose of the system is still that of synthesising a FN capable to solve a particular problem. However, the characteristics of the classification task provided further insight into some of the system's important features.

On one hand, the size of both elements and sets was increased: 6 input variables instead of 2, 3 outputs instead of 1, and solutions consisting of 20 rules or so instead of about 5. On the other hand, fitness evaluation turned out to be significantly faster, making it practical to carry out several different runs and tests; this is a delicate and important issue in evolutionary systems, specially at the early stages of development and testing.

Most of the observations made for the control problem above recurred here in a very similar fashion. What follows is a description of the new problem in detail and some of the considerations that were applied differently, along with new findings, extended testing and results.

6.2.1 The Problem Data

The data was obtained from the *Machine Learning Database Repository*, maintained by the Department of Information and Computer Science at the University of California, Irvine. In this case, the data consist of the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. All attributes are continuous.

Since the main purpose of the experiment was not to test the capabilities of fuzzy networks as classifiers, but to investigate the operation of the multi-level GA in a different problem domain, the original data was pre-processed in two ways, as is common practice, in an attempt to simplify the task. First, a principal component analysis was performed to reduce the dimensionality of the data; only the six most significant components obtained in this way, whose eigenvalues showed a difference of over an order of magnitude, were further utilised. Second, the class distribution was equalised, randomly eliminating instances of overrepresented classes from the data set.

The exercise can then be summarised as follows: the 2LGA will be used to synthesise fuzzy networks that are capable of classifying instances into three distinct categories, based on the values of six continuous attributes. Commonly employed techniques such as *leave-one-out* will be used to cross-validate performance measures.

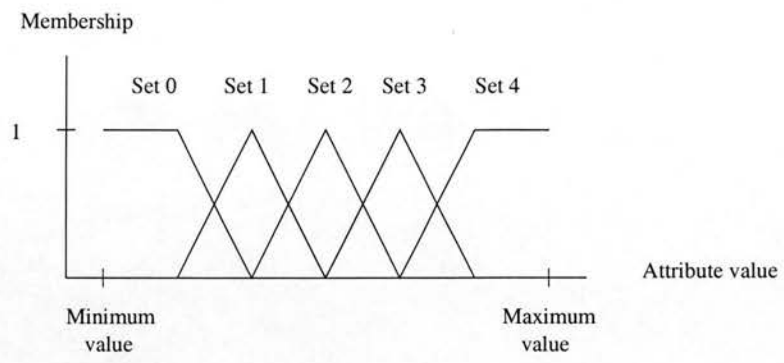


Figure 6.10: Standardised fuzzy sets and membership functions

6.2.2 Fuzzy Sets and Membership Functions

As was the case in the control problem, standard fuzzy sets and membership functions were used. An auxiliary program was developed that analyses each variable's data and defines a pre-specified number (7) of fuzzy sets, which have uniformly shaped and spaced, overlapping membership function definitions (see figure 6.10). Under this arrangement, it was not considered necessary to rescale data values. In the general case, this arrangement will ensure that the linguistic semantics of the original fuzzy terms are preserved.

6.2.3 Fitness Evaluation

The simplest way to evaluate a FN's fitness in this domain is perhaps to use the percentage of instances it can correctly classify. Thus, an initial fitness function may be simply defined as

$$f = \frac{1}{N} \sum_{i=1}^{i=N} C(i)$$

where

- N : total number of training instances
- and $C(i)$ is defined, for instance i, as

$$C(i) = \begin{cases} 1, & \text{when } |\text{output - class of } i| < \text{tolerance} \\ 0, & \text{when } |\text{output - class of } i| \geq \text{tolerance} \end{cases}$$

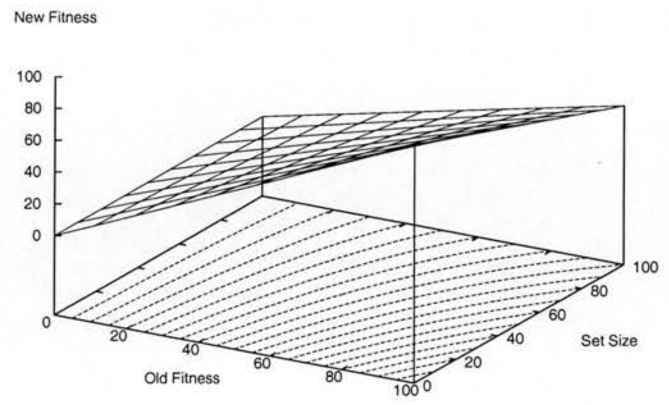


Figure 6.11: Effects of parsimony on fitness values: P = 0.005

Parsimony

As in the previous experiment, long individuals ought to be penalised, in order to induce a tendency to produce compact solutions. Therefore, a parsimony factor was also incorporated to refine the fitness function that was actually used:

$$f_{new} = max(0, f_{old} \cdot (1 - P \cdot S))$$

where, again, P is the *parsimony* factor and S is the size of the set.

Since f_{old} is a value between 0 and 100, and acceptable values of S would be in a similar range, allowed parsimony values were $0.005 \leq P \leq 0.01$. Graphs 6.11 and 6.12 show the effect on f_{new} at these two extremes. Figures 6.13, 6.14 and 6.15 show the actual effects on a typical run with low, medium and high values for the parsimony factor, with regards to the average set size and the training and testing fitnesses of the best idividual.

6.2.4 Crowding Elements

Since the final solution will consist of several different rules, it was decided to facilitate the production (and preservation) of dissimilar individuals in the elements populations.

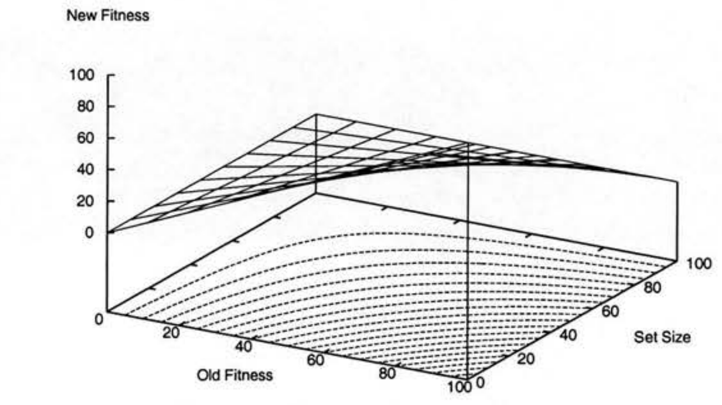


Figure 6.12: Effects of parsimony on fitness values: P = 0.01

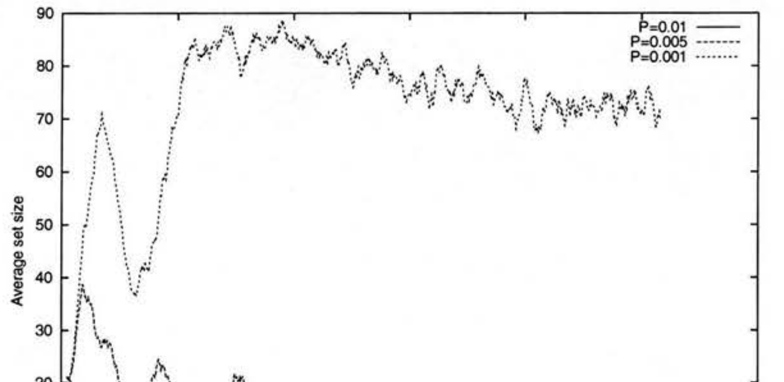


Figure 6.13: Effects of the parsimony factor on average set size

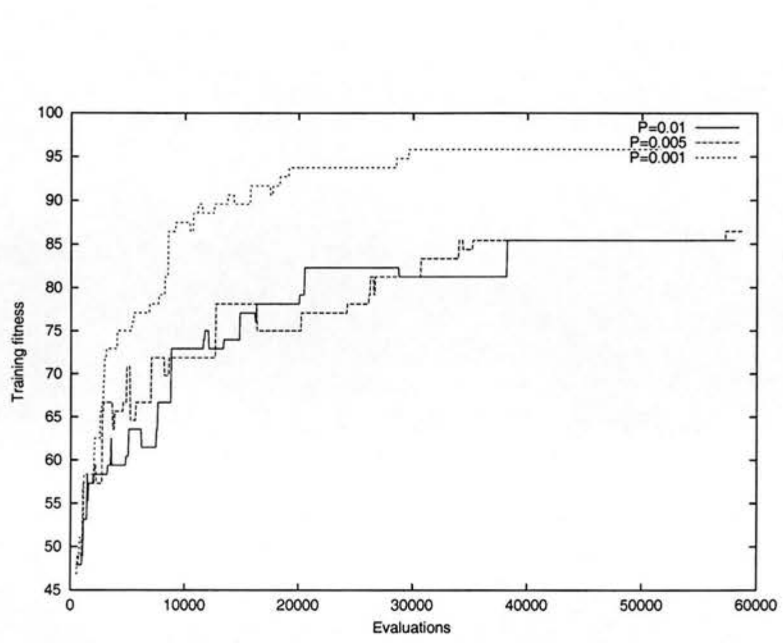


Figure 6.14: Effects of the parsimony factor on best fitness

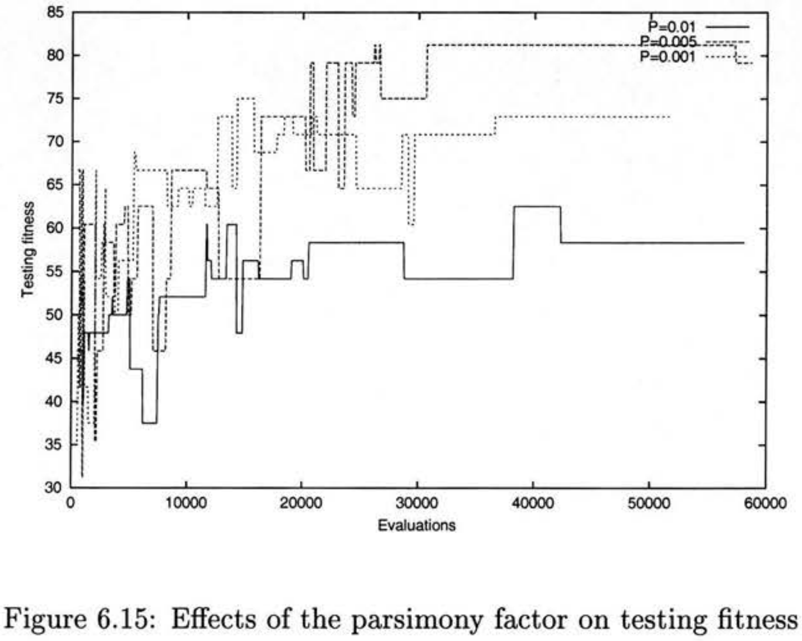


Figure 6.15: Effects of the parsimony factor on testing fitness

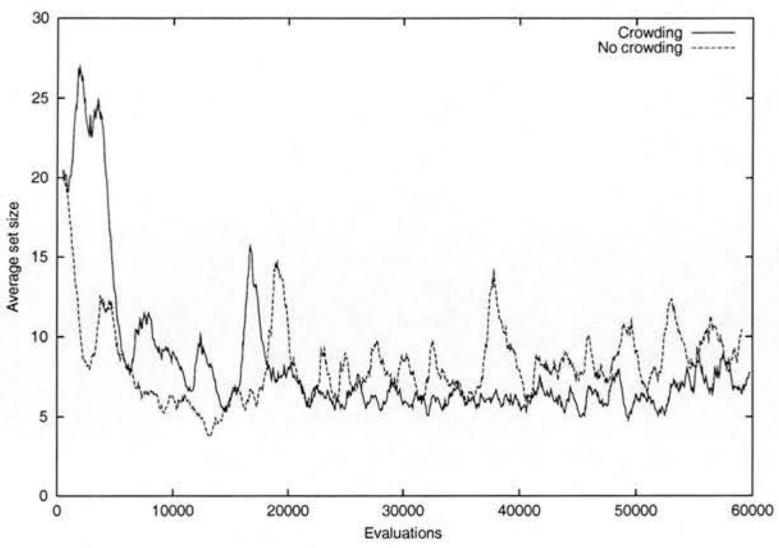


Figure 6.16: Effects of crowding on average set size

Instead of replacing a previously existing element at random, a kind of crowding mechanism based on tournament selection was incorporated. When a new element is to be introduced in the population, n candidates are randomly selected and the new one will replace the candidate it resembles the most. In this case, Hamming distance was used as a measure of similarity. The effects of crowding can be seen on graphs 6.16, 6.17 and 6.18.

6.2.5 Other parameters

"**The Exception Principle:** It rarely pays to tamper with a rule that nearly always works. It's better just to complement it with an accumulation of specific exceptions." [Minsky 85]

Élitism

It is generally assumed within the evolutionary computation community that holding on to the best solution found so far is a good idea. GAs tend to be robust enough to re-produce in the short term a relatively highly fit individual that has been replaced by mistake: the genetic material necessary to build it is very likely to remain in the pool for several generations. Nevertheless, empirical data shows that performance is

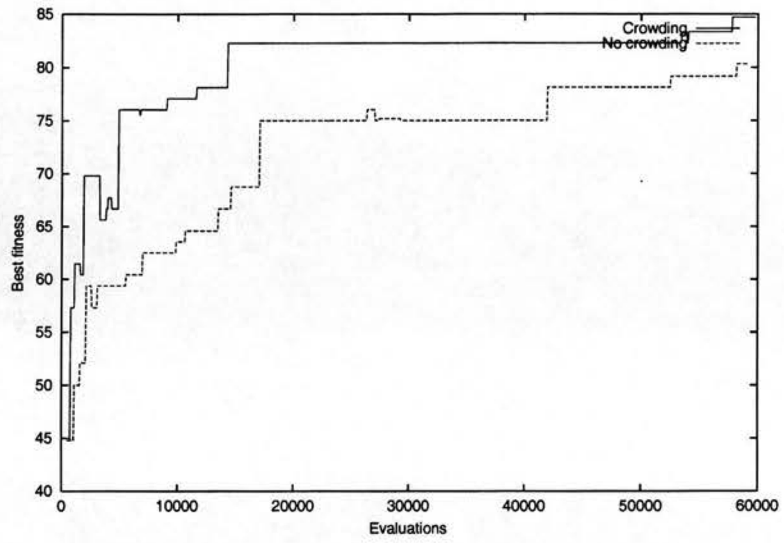


Figure 6.17: Effects of crowding on best fitness

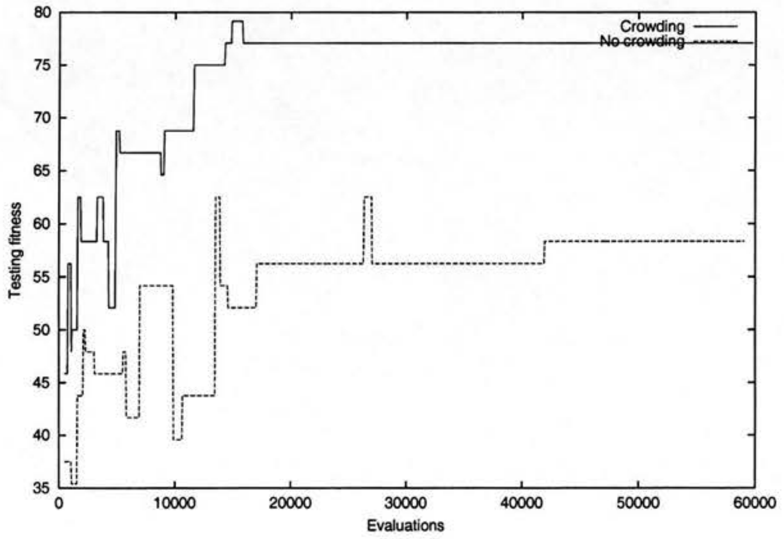


Figure 6.18: Effects of crowding on testing fitness

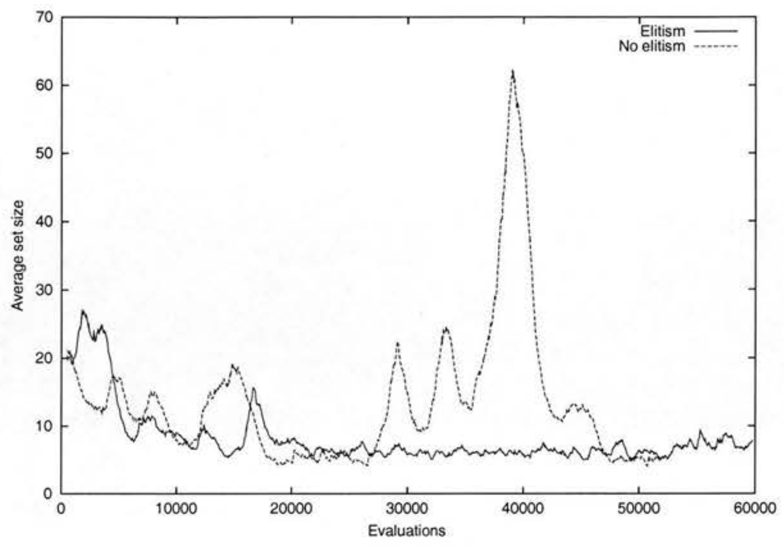


Figure 6.19: Effects of élitism on average set size

usually enhanced if a form of élitism is enforced. Several tests were performed using 2LGA with and without the élitist regime in force. The results are shown in graphs 6.19, 6.20, 6.21 and 6.22.

Island Model - Subpopulations and Migrations

In the following comparisons, because the number of subpopulations affects proportionally the number of evaluations per stage, the graphs indicate the number of stages in the abscissa and not the total number of evaluations. As in the previous section, it is often the case that the number and sizing of populations to use is not investigated properly, even though it can sometimes have a strong impact on the overall performance of the system.

The next graphs show the outcome of different migration rates between sub-populations (islands).

6.2.6 Adaptive Cut and Splice Probabilities

Assuming that the size of the best individual found so far gives a good indication of the size of the solutions that the system should be considering, at that particular stage

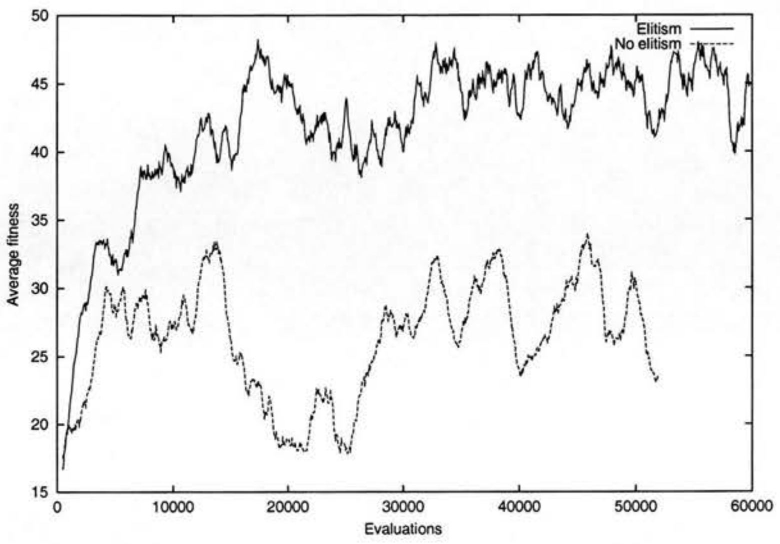


Figure 6.20: Effects of élitism on average fitness

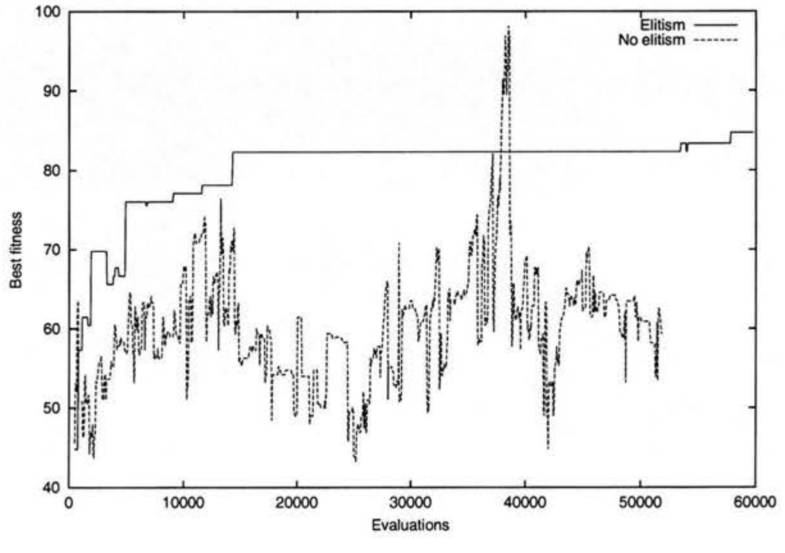


Figure 6.21: Effects of élitism on best fitness

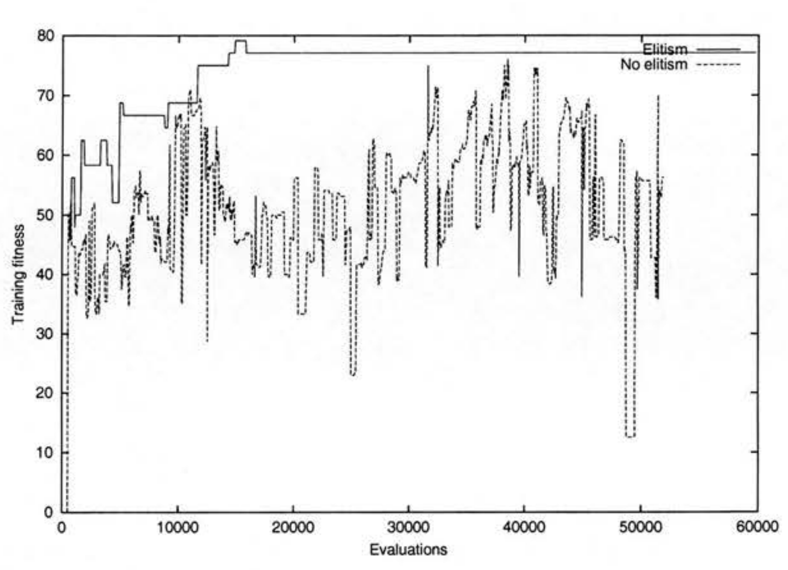


Figure 6.22: Effects of élitism on testing fitness

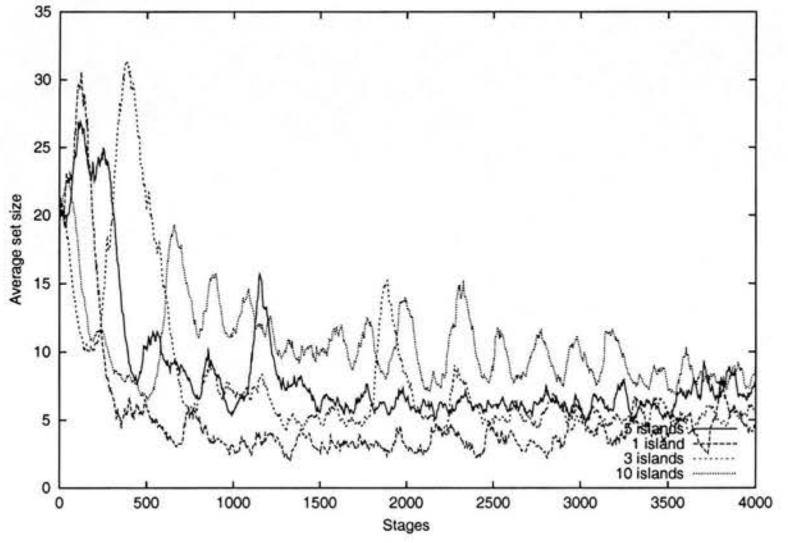


Figure 6.23: Effects of the number of islands on the average set size

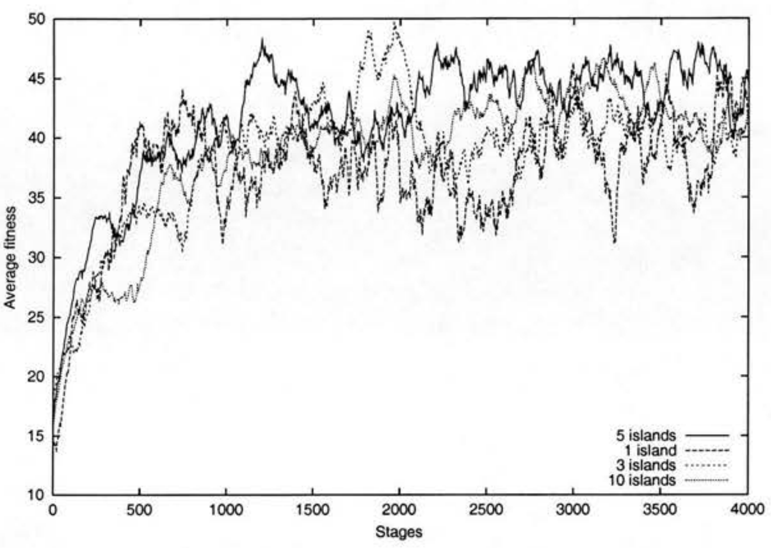


Figure 6.24: Effects of the number of islands on the average fitness

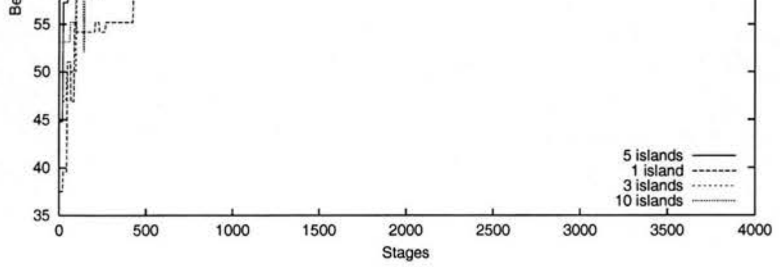


Figure 6.25: Effects of the number of islands on the best fitness

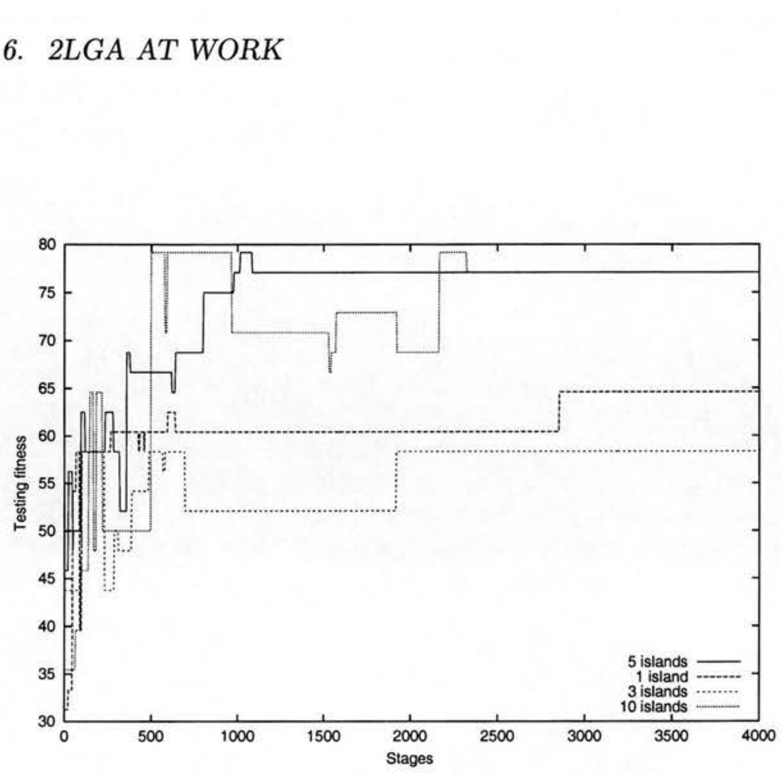


Figure 6.26: Effects of the number of islands on the testing fitness

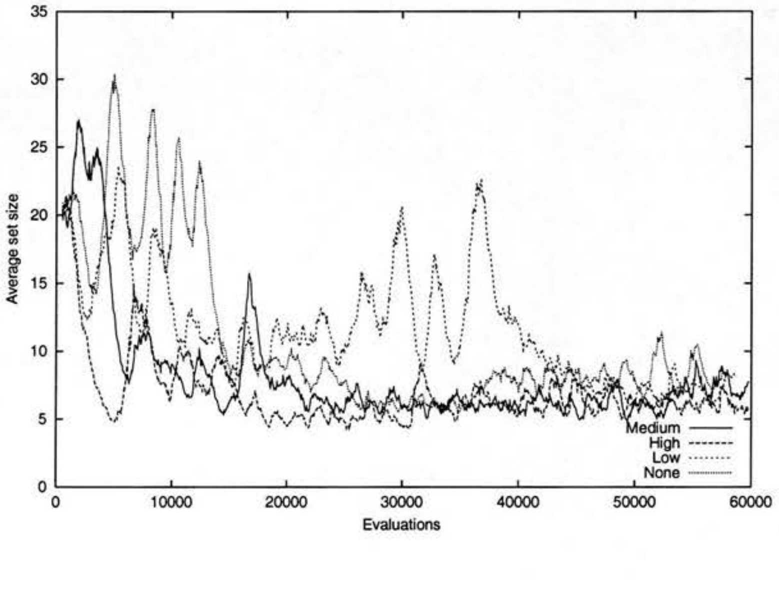


Figure 6.27: Effects of the migration rates on the average set size

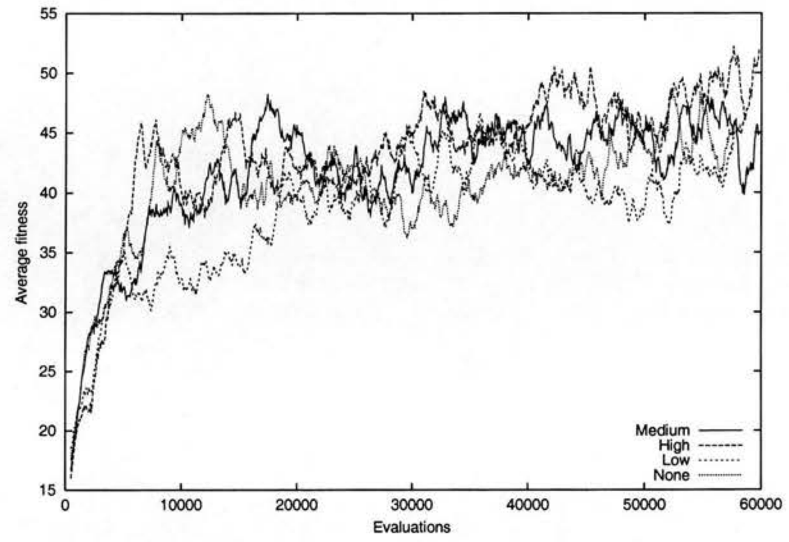


Figure 6.28: Effects of the migration rates on the average fitness

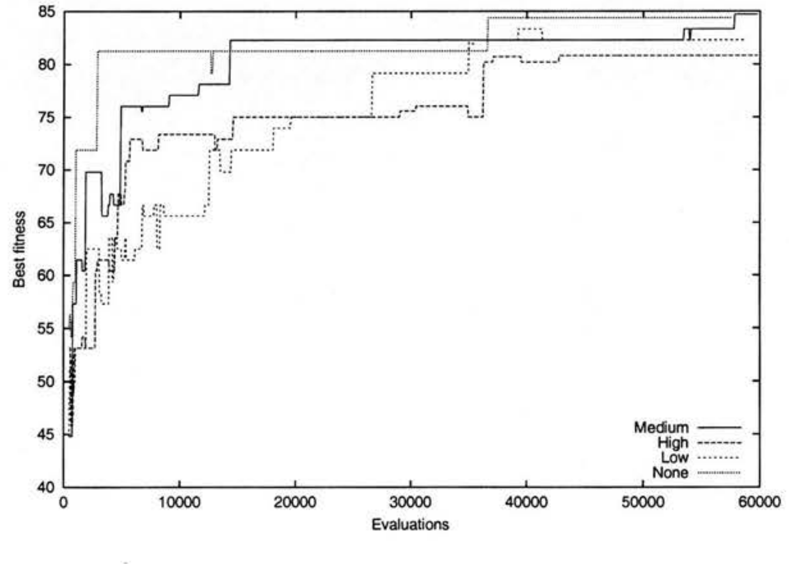


Figure 6.29: Effects of the migration rates on the best fitness

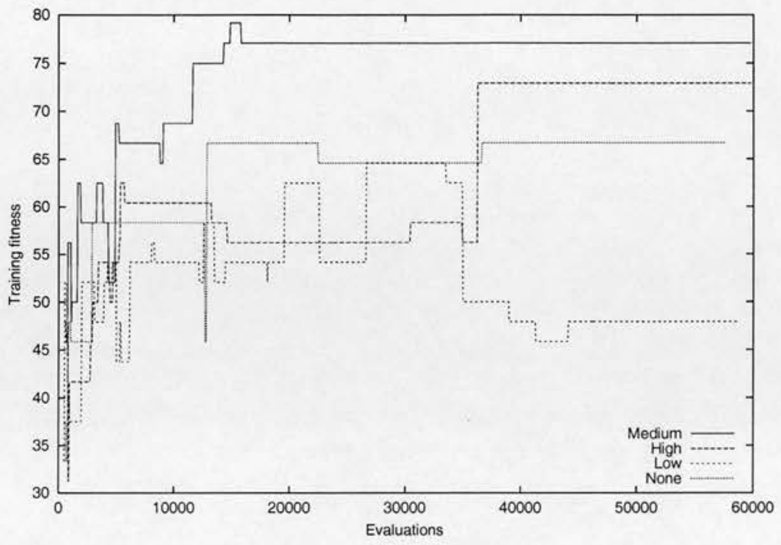


Figure 6.30: Effects of the migration rates on the testing fitness

in the evolutionary process, it was decided to compare that value with the average size of all the sets populations. The difference was used to guide the adjustment of the cut and splice probabilities.

It is clear that increasing the probability of the cut operator will induce the production of smaller individuals, while decreasing it will result in larger ones. Likewise, the inverse relation holds true for the splice operator — they perform complementary tasks. With this in mind, a simple form of adaptation, guided by population statistics, was implemented at the beginning of each generation:

- if the average length is smaller than the length of the best then decrease the cut probability and increase the splice probability
- otherwise, increase the cut probability and decrease the splice probability

Maximum (0.8) and minimum (0.2) values for both parameters were always preserved to guarantee the presence and influence of each operator at all times. It was found that relatively small values for the increase/decrease adjustments (0.001) produced smoother, more consistent behaviour, and better results too. Figures 6.31 and 6.32 illustrate the activity of these parameters subjected to automatic modification and the behaviour of the size of individuals — compare with figure 6.33.

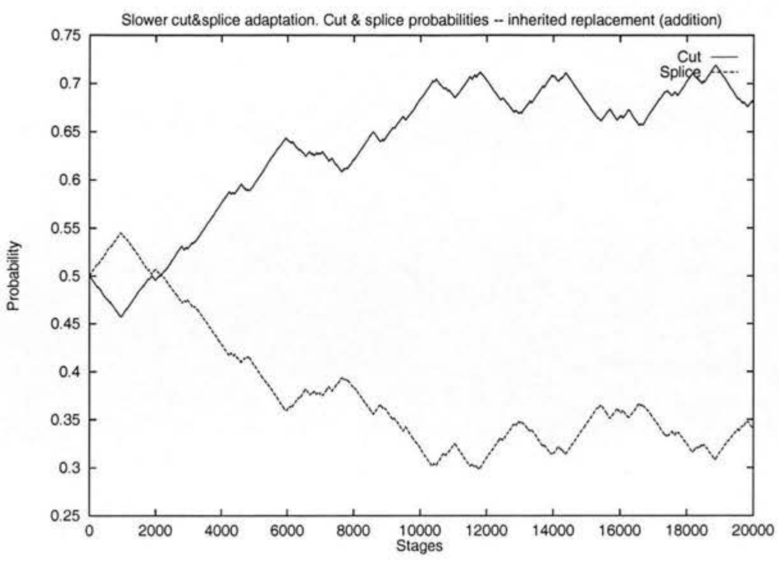


Figure 6.31: Automatic adjustment of the cut and splice probabilities

It is important to note that the optimum values obtained earlier, experimentally, are very close to the ones obtained by self-adaptation. Very similar results were produced using the other replacement strategies.

Adaptive Heredity and Parsimony

In the same vein, the heredity and parsimony factors were also subjected to similar self-adaptation mechanisms. Like the cut & splice operators, heredity and parsimony play antagonistic roles: a relatively large heredity factor induces an increase in individual size while the opposite is true for the parsimony factor. Nevertheless, the nature of the link between these parameters is not so obvious, as opposed to the natural interrelationship between the cut & splice operators.

6.2.7 Adjusting Reproduction Rates

Observing the results obtained in the previous experiment, it became clear that having different rates at which the two populations evolve can have a direct implication in the performance of the system. Trying different, fixed rates for both populations provided an indication of the role played by the M and N parameters and the relation between them. As was the case in the first experiment, if fixed reproduction rates — i.e. number

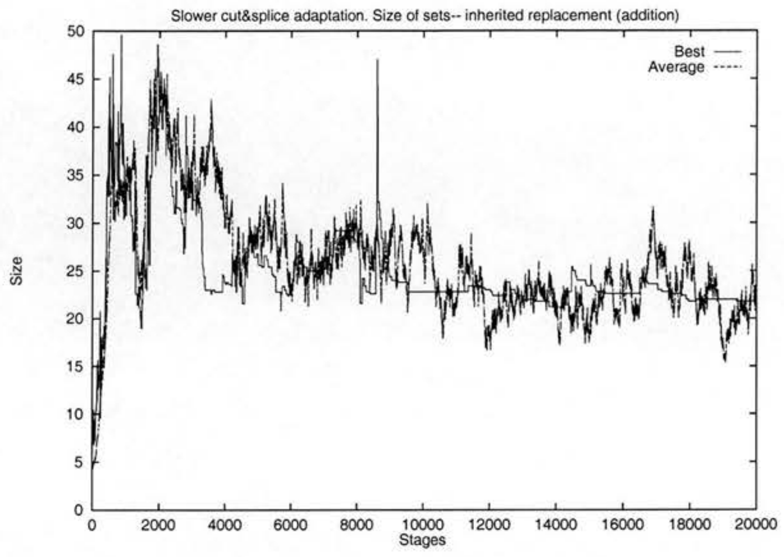


Figure 6.32: Average size and size of best — adaptive probabilities

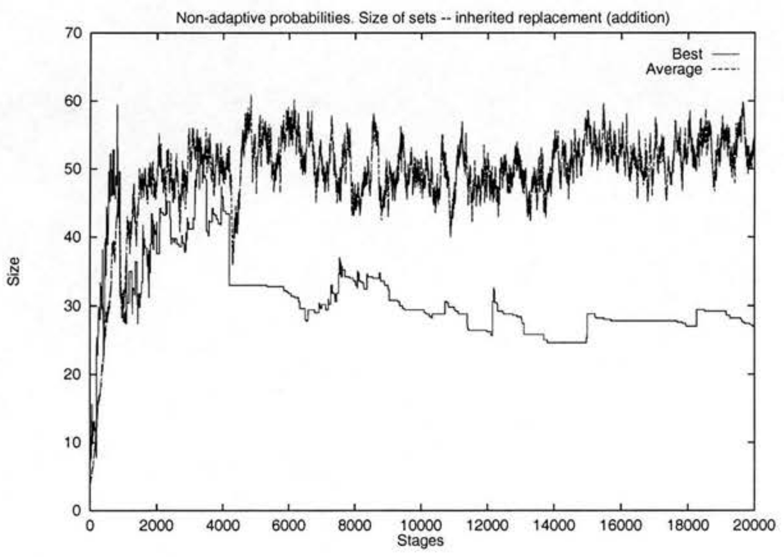


Figure 6.33: Average size and size of best — fixed probabilities

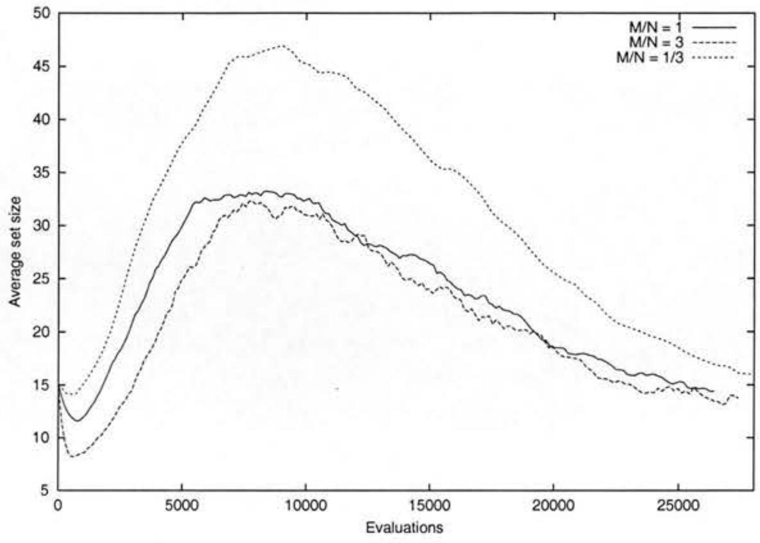


Figure 6.34: Average size — fixed reproduction rates

of generations per stage — were used, a 1/3 ratio yielded marginally better results; see figures 6.34 and 6.35.

Before attempting a self-adaptive way to adjust reproduction rates for either population, different scheduled strategies were tried first. However, none of them showed significant differences in performance or behaviour. As an example, figures 6.36 and 6.37 illustrate the results of applying a different reproduction rate in "burst mode": each stage would consist of one generation of each population ($M/N = 1$) for a predetermined number of stages, then change to a faster reproduction rate for one of them ($M/N = 1/3$).

6.2.8 Alternative Recombination Operators

In an attempt to test the alleged recombinative power of the mGA operators, namely cut and splice, a simple form of crossover was devised for 2LGA. Designed by analogy to the usual uniform crossover method, commonly employed in binary, fixed-length representations, a recombination mechanism called *mixing* was defined as follows:

Mixing: Take each gene of each of the two parents in turn, and assign it randomly to one of the two children.

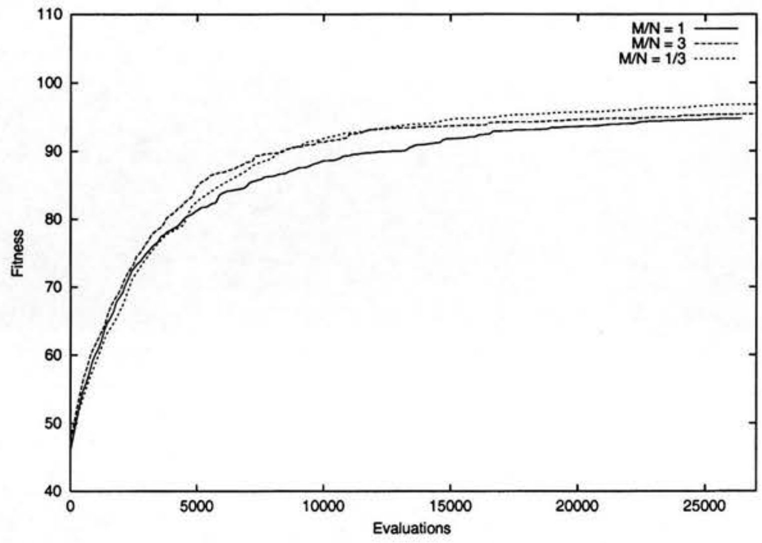


Figure 6.35: Average fitness — fixed reproduction rates

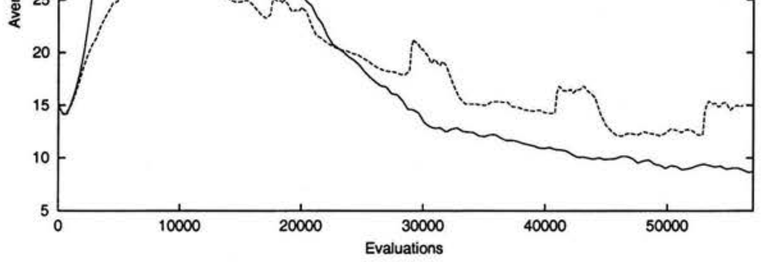


Figure 6.36: Average size — reproduction rates in burst mode

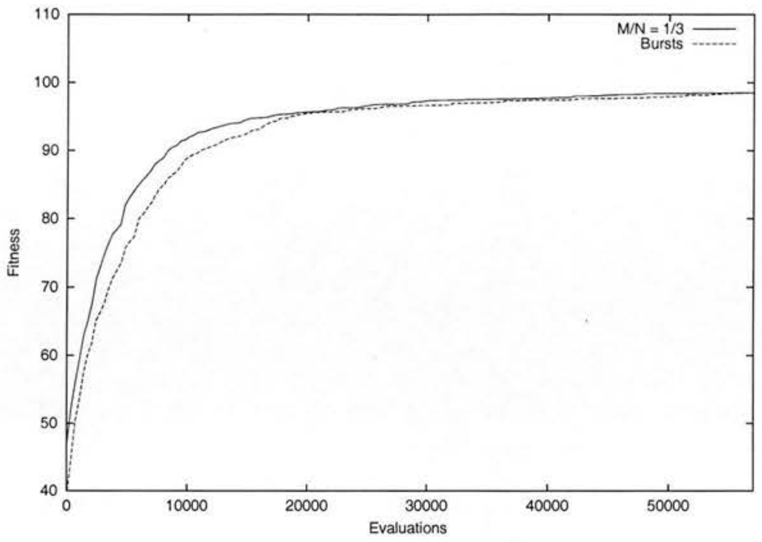


Figure 6.37: Average fitness — reproduction rates in burst mode

Obviously, this mechanism will take two parents and produce two offspring, unlike cut & splice, which may produce a different number of new individuals each time they are employed. Even though mixing does produce individuals with a variable number of genes, it is clear that it induces a tendency to make homogeneous populations with regards to size: the children tend to be the average size of the parents. Of course it would be possible to set a certain bias, based on the parents' size differences or even their fitness. However, the main purpose is to have a viable recombination operator to compare against. The disruptive features of mixing are discussed in the next chapter.

6.2.9 Comparative Results

A brief summary of the results obtained trying out the different alternatives is presented below. In every case, 20 runs were carried out, each one comprising of 4000 stages. A standard *t-test* ([Bailey 81]) was then performed (with the aid of a small program developed by Peter Ross) in order to assess the statistical significance of the results obtained so far. T-tests are a special type of inferential statistic used to compare two different means to each other. The t-test produces a statistic that is a ratio of the difference between the means being compared to the variability of the observations within each set of data on which the means are based. A 95% confidence interval is

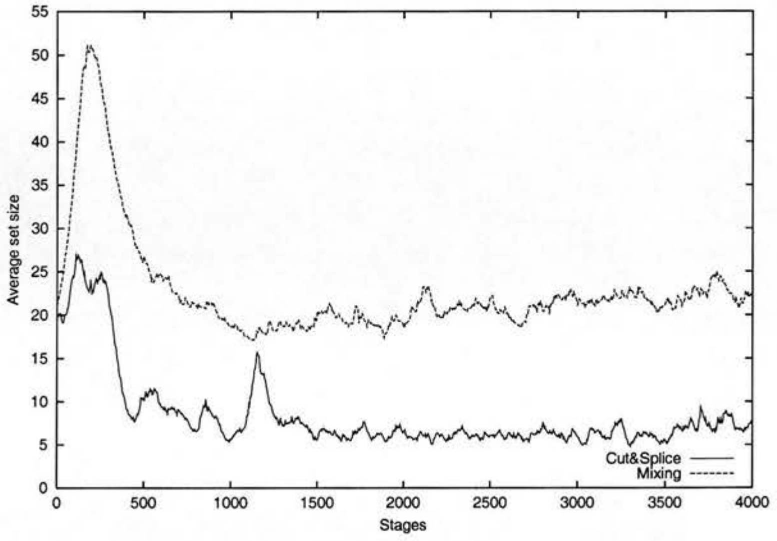


Figure 6.38: Effects of mixing on the average set size

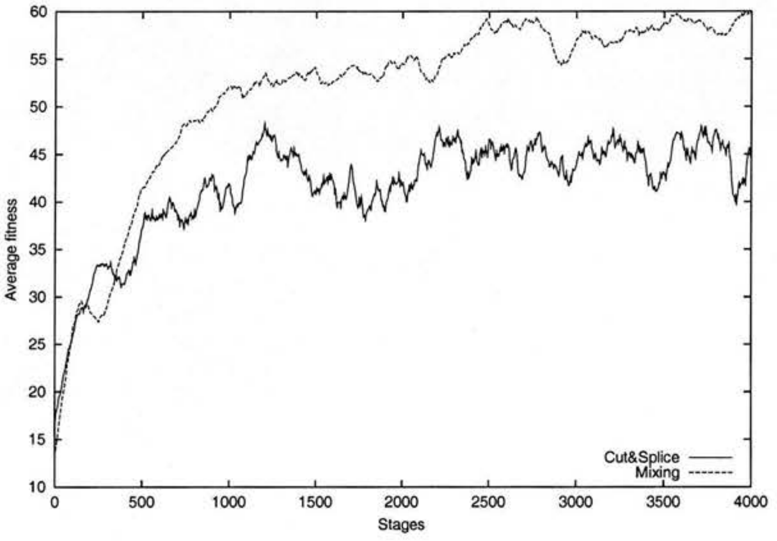


Figure 6.39: Effects of mixing on the average fitness

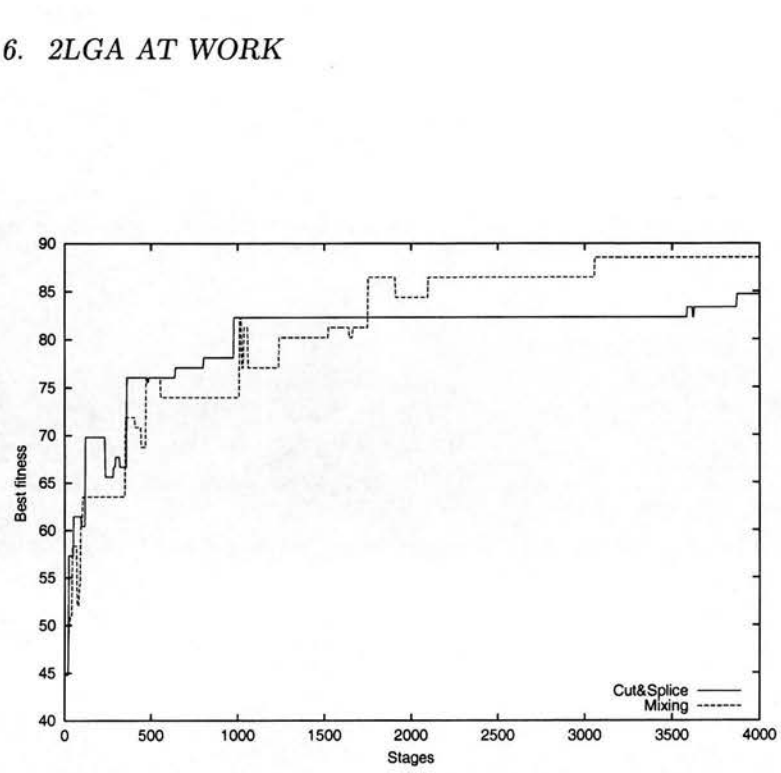


Figure 6.40: Effects of mixing on the best fitness

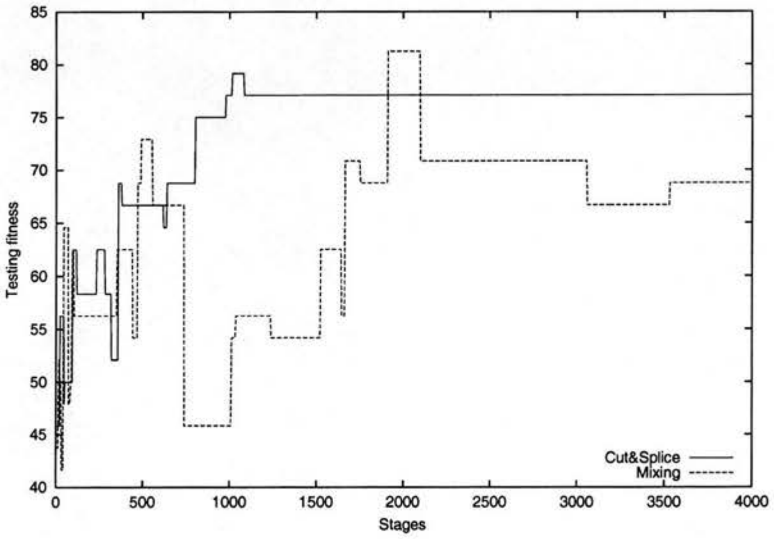


Figure 6.41: Effects of mixing on the testing fitness

traditionally required to be able to draw statistically valid conclusions.

Parsimony

In section 6.2.3, it was described how a parsimony factor was included in the fitness function, so that excessively large individuals were duly penalised, in an attempt to promote a preference for succint solutions. Three different values for this parameter were tried: 0.01 (high), 0.005 (medium) and 0.001 (low). The results of the t-test are presented below; in this case it is particularly relevant to distinguish between the results obtained for the training set and those obtained for the testing set.

For the training set:

Parsimony	high	medium	low
high	–	0.9999	1.0000
medium	1.0000	–	0.9999
low	1.0000	1.0000	–
Average	81.3140	85.5100	93.9079
Standard deviation	2.1296	1.5066	1.4054

For the testing set:

Parsimony	high	medium	low
high	–	0.8512	0.9990
medium	0.8512	–	0.7865
low	0.9990	0.7865	–
Average	68.1293	74.1393	78.3487
Standard deviation	7.7729	9.9151	2.8658

It can be observed that, as expected, the standard deviation in the testing set is considerably larger than in the training set. It can be said that, although for the training set a relatively small parsimony factor yields better results, for the testing case the difference against a moderate parsimony factor is not conclusive. In both cases, a high parsimony pressure appears to be detrimental to the performance of the system.

Additionally, the effects of "over-training" are more visible in the case of low parsimony pressure, where a considerably higher success rate for the training set was achieved, at the cost of requiring a much larger number of rules, without carrying over the advantage difference to the testing set.

It appears that keeping a value between the low and medium level for the parsimony factor could result in a balanced performance without increasing the risk of overtraining.

Crowding

The rationale for the introduction of a crowding mechanism and its effects were presented in section 6.2.4. A simple comparison against the same system without such a mechanism yielded the following results for the training set:

Crowding	yes	no
yes	–	0.7790
no	0.7790	–
Average	84.0882	82.8323
Standard deviation	1.9987	2.4115

The difference does not appear to be overly large; furthermore, the claim that a crowding scheme enhances the performance of the system is not statistically valid under the present testing conditions. The same is true for the testing set, where the difference is considerably larger. It is likely that for more complex problems requiring a larger number of rules, crowding could have a stronger impact than in this particular case.

Élitism

The generally acknowledged notion that a form of élitism should be enforced in order to accelerate the search process has been confirmed for the 2LGA system. Again, a comparison against an otherwise identical system reflected through the t-test the validity of the original assumption:

Élitism	yes	no
yes	–	1.0000
no	1.0000	–
Average	84.0882	58.9348
Standard deviation	1.9987	4.2868

It is worth noticing that the variation of the results was considerably higher without an élitist system; and it does show that such a mechanism directly influences the quality of the final results. Presumably, for longer runs the absolute difference might not have been so abrupt, as the inherent robustness of GAs would allow the system to recover from the temporary losses of the best individuals.

Subpopulations

Another factor that seemed to play an important role was the adoption of the Island Model, together with a migration scheme, whereby individuals are allowed to evolve only within their respective subpopulations (islands) for a given interval and then, according to a predefined migration rate, certain individuals are exchanged between different subpopulations. 1 (none), 3 (low), 5 (medium) and 10 (high) subpopulations were tried, producing the following results:

Migration	none	low	medium	high
none	–	0.9999	0.9984	0.5488
low	0.9999	–	0.9460	1.0000
medium	0.9984	0.9460	–	1.0000
high	0.5488	1.0000	1.0000	–
Average	80.8133	85.9883	84.0882	80.3130
Standard deviation	1.9560	2.1205	1.9987	0.6278

These tests confirm the statistical validity of the choice of implementation using an low/medium number of subpopulations with a moderate migration rate between them.

Top-down Induction of Decision Trees

2LGA was also compared against Quinlan's C4.5 algorithm ([Quinlan 92]). C4.5 builds a decision tree using the standard TDIDT (top-down induction of decision trees) approach, recursively partitioning the data into smaller subsets, based on the value of an attribute. At each step in the construction of the decision tree, C4.5 selects the attribute that maximises the information gain ratio. The induced decision tree is pruned using pessimistic error estimation. There are several parameters that can be adjusted to alter the behaviour of C4.5. In the experiments with C4.5, the default settings for all parameters were used (see figure 6.42). The results obtained by C4.5 are very good, reaching an estimated accuracy of roughly 90%. Keeping in mind that it was specifically for this kind of task, is encouraging to see that 2LGA was capable of producing comparative solutions, albeit in a rather longer period of time. It would be necessary to compare the performance of both systems over a larger set of tests in order to produce a meaningful assessment.

Several experiments were carried out using different training and testing sets for cross-validation. Figure 6.43 presents the results obtained using half the data set for training and the other half for testing, while figure 6.44 shows the results obtained using the "leave-one-out" technique.

```
C4.5 [release 8] decision tree generator        Thu Jul 10 16:13:40 1997
----------------------------------------

    Options:
        File stem <wine>
        Trees evaluated on unseen cases

Read 144 cases (6 attributes) from wine.data

Decision Tree:

A1 <= 752.264 :
|   A4 <= -3.6955 :
|   |   A5 <= -1.5909 : 2 (37.0/1.0)
|   |   A5 > -1.5909 :
|   |   |   A3 <= 20.5716 : 2 (3.0)
|   |   |   A3 > 20.5716 : 3 (5.0)
|   A4 > -3.6955 :
|   |   A3 <= 16.9409 : 2 (5.0)
|   |   A3 > 16.9409 : 3 (38.0/1.0)
A1 > 752.264 :
|   A4 <= -6.0623 :
|   |   A3 <= 22.9852 : 2 (3.0)
|   |   A3 > 22.9852 : 1 (3.0/1.0)
|   A4 > -6.0623 :
|   |   A1 > 881.493 : 1 (38.0)
|   |   A1 <= 881.493 :
|   |   |   A3 <= 20.6894 : 1 (6.0)
|   |   |   A3 > 20.6894 : 3 (6.0)

Evaluation on training data (144 items):

         Before Pruning           After Pruning
        ----------------   ---------------------------
        Size      Errors   Size      Errors   Estimate

          19    3( 2.1%)     19    3( 2.1%)    (10.9%)   <<

Evaluation on test data (132 items):

         Before Pruning           After Pruning
        ----------------   ---------------------------
        Size      Errors   Size      Errors   Estimate

          19    3( 2.3%)     19    3( 2.3%)    (10.9%)   <<


         (a)  (b)  (c) <-classified as
        ---- ---- ----
          42    1    1 (a): class 1
           1   43      (b): class 2
                    44 (c): class 3
```

Figure 6.42: C4.5 Solving the Wine Classification Problem

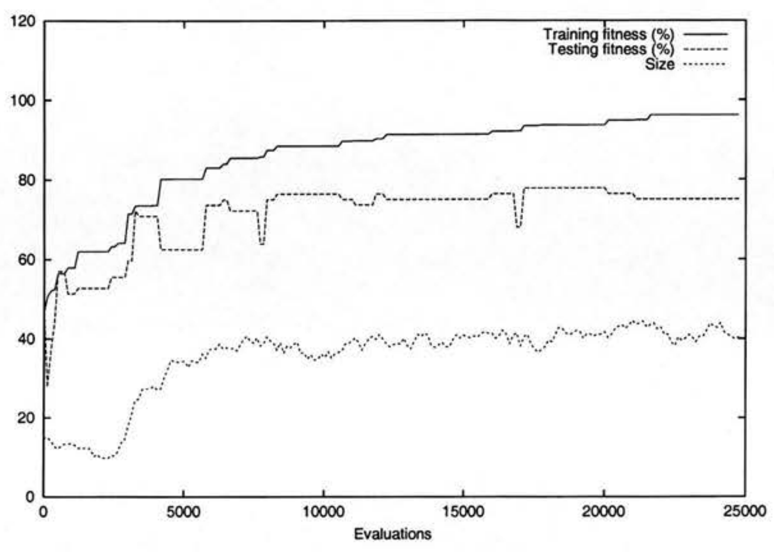


Figure 6.43: Cross-validating (50%–50%)

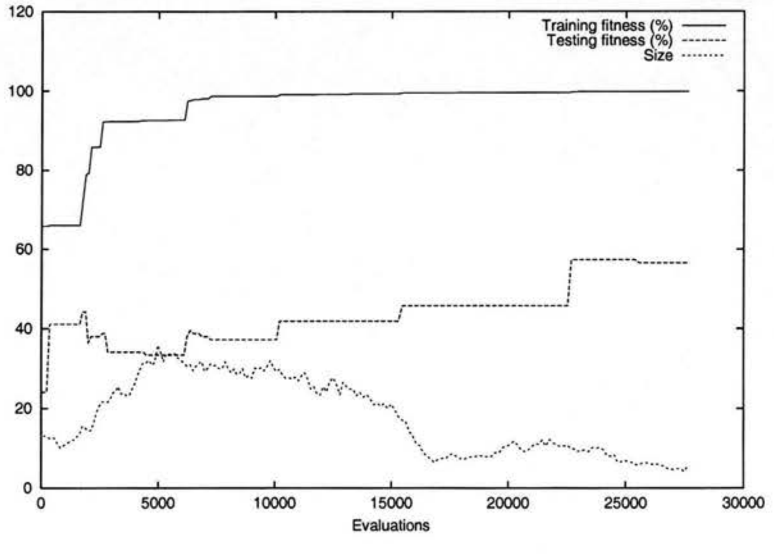


Figure 6.44: Cross-validating (leave-one-out)

Chapter 7

2LGA Analysed

Practical and theoretical considerations regarding the multi-level organisation of 2LGA are discussed. Although it is difficult to draw a precise line of demarcation because they are intricately related, important issues have been grouped into sections describing search, representation and learning aspects of the system.

7.1 Controlling Search

> "It can save a lot of mental work if one makes each arbitrary choice the way one did before. The more difficult the decision, the more this policy can save. The following observation by my associate, Edward Fredkin, seems important enough to deserve a name:
>
> **Fredkin's Paradox:** The more equally attractive two alternatives seem, the harder it can be to choose between them—no mater that, to the same degree, the choice can only matter less."
>
> [Minsky 85]

Most practitioners in the field agree that, if particular knowledge about the class of problems being solved is available, it should be incorporated into the GA to enhance its search capabilities. As described in chapter 2, different GA approaches can be extended in many different ways, and 2LGA is no exception. The particular mechanisms that were proposed and adopted in this work to enhance the system's search power are discussed below.

7.1.1 Adapting Parameters

Theoretical results might be difficult to obtain, but the idea of having adaptive reproductive operators is supported by many empirical studies. The effectiveness of adaptive operator probabilities has been demonstrated in several papers, both for mutation ([Fogarty 89], [Bäck & Hoffmeister 91], [Fogel & Atmar 92]) and recombination ([Schaffer & Morishima 87], [Davis 89]).

Self-adapting mechanisms present many advantages and obviously play an increasingly important role in GA design. This trend is due in part to the absence of strong predictive theories capable of specifying optimal parameter values. However, not only can operator probabilities be modified dynamically; the idea of self-adaptation can be extended to other GA elements, notably representation. This is a main feature of the work presented in this thesis and the subject of section 7.2 below.

An adaptive mechanism requires some information on which to base the adaptations. GAs are adaptive systems that use a population to maintain information about the space being searched. This population is used by the GA to adaptively guide the trajectories through the search space. It seems natural to consider using the GA as an adaptive mechanism for adjusting itself as it solves a problem.

In this work, the recombination operators used are based on the cut & splice ones that were defined for messy GAs and, therefore, are very similar to them. It was clear from the beginning that, despite 2LGA's robustness and low sensitivity to small parameter variations, the probabilities assigned to the application of each of these operators would have a noticeable impact in overall system performance. That is the reason why it was decided to devise a mechanism to allow the self-adaptation of these parameters (see section 6.2.6).

The assumption that the size of the best individual found so far could give a good indication as to the size of the solutions that the system should be considering, at each particular stage in the evolutionary process, proved to be an effective one. Furthermore, this idea was extended to incorporate the heredity and parsimony factors in the set of self-adapting parameters. The results were also as expected, achieving balanced levels of activity, although not as significant in enhancing system performance as the adaptive probabilities of the recombination operators. Further study of the properties of parameter adaptation may certainly help to identify the complementary nature and relationship between distinct parameters.

7.1.2 Crowding

In rule learning systems in general, fitness is a function of how well an individual complements other individuals in the population. Rather than searching for an optimal group of genes assembled together in a single chromosome, the goal is to evolve groups of individuals which collectively solve a particular problem. When the typical optimisation-oriented GA is applied in this situation, the strong pressures towards homogeneity in the population make it difficult to maintain different but cooperative individuals. Either an additional mechanism for encouraging groupings of individuals, as introduced in section 2.6.3, or a suitable structural imposition are required.

In 2LGA, pressure towards homogeneity is especially harmful in the elements population. Due to the inherent properties of fuzzy rules, their definition and evaluation, it is certainly the case that a good solution, i.e. a set, must contain several rules that are substantially different from one another. To circumvent this problem, a simple crowding mechanism was enforced, as described in section 6.2.4. This method resulted in a better fitness distribution in the elements populations, and more individuals of relatively high fitness were allowed to coexist. That was exactly the desired behaviour. Noticeable improvement was observed not only in the overall performance of the system, but also in the quality of the final solutions that were produced.

A similar problem, introduced in [Goldberg *et al.* 90] in the context of messy GAs, relates to the differences between arbitrary substrings, which might pose a problem for the sets populations at the time of tournament competition. The proposed solution will be considered as a possible future extension to the 2LGA system (see section 8.3.3).

7.1.3 Parsimony

Genes that get "trapped" inside a building block but serve no useful purpose are called *parasitic*. Even if they do not negatively affect the fitness value of the containing building block directly, their presence might thereafter prevent the expression of a larger building block. In [Goldberg *et al.* 90], they proposed the use of *null bits*, acting as explicit placeholders, to guide a tie-breaking mechanism: strings with the least effective length are preferred when fitness ties occur between building blocks. This is similar to the treatment given to the hitchhiking effect, i.e. genetic material with low contribution towards an individual fitness is carried along between segments that do contribute positively to enhance it, as defined in the Artificial Life literature ([Forrest & Mitchell 92], [Mitchell & Forrest 93]), where, paradoxically, the addition of introns seems to alleviate the problem.

As a different approach, the effect of the introduction of a parsimony factor was very positive. It provided a direct and explicit way of maintaing growth rates for individuals in the sets populations under control. This is in agreement with the notion of effective fitness and complexity compression described in [Nordin & Banzhaf 95]. Clearly, the principle of Occam's Razor is evident in this case: a shorter solution is in essence a more generic solution. Likewise, as pointed out above, hitchhiking genes become less of a burden because they are more easily discarded. This is evidenced in section 6.2.3, where the sample graphs show that parsimony alone can make the difference between over-fitting and generalisation ability. They also demonstrate that too strong a parsimony factor might impede an efficient exploratory search, necessitating at times the use of relatively large individuals, particularly during the early stages.

7.1.4 Exploration and Exploitation

Intuitively, without adhering to the analysis formalism provided by schemata, a BB can be said to be a part or component that contributes significantly towards an increase in the fitness of a complete individual, i.e. a partial solution. The task of the GA is to identify those BBs and try to put them together in a single individual. This is performed incrementally in such a way that promising combinations are tried first, until a complete, satisfactory solution to the problem at hand is produced. But the GA search is characterised by two coexisting factors with seemingly antagonistic purposes: exploration and exploitation. The tension between exploitation and exploration is a recurring theme in genetic algorithms, as initially suggested in [Holland 75]. A proper balance between these two aspects is crucial when adaptively searching an unknown space for optimal solutions.

In 2LGA, premature convergence was hardly ever a problem. The reason for this stems from the co-evolution of the two populations at different levels. Long phases of exploration or exploitation did not take place in a synchronised way at both levels: the occurrence of one of them at one level would be inevitably interrupted by the complementary activity in the other level. A clear example is the effect produced by the replacement strategies of the elements populations, which would be seen as an externally induced mutation at the level of the sets populations.

It is interesting to note that, particularly when the 'adopted' strategy was being used, it was possible to observe a sort of *adaptation anomaly*, as defined in [Davidor 90], regarding the average set size. That paper postulates that adaptation in evolutionary systems normally starts from a state of relatively high redundancy and complexity, so that they can adapt to very diverse conditions by simplifying and rearranging the basic components of the system first, followed by increased specialising and sophistication. In this work, the evidence suggests that the high-level population must first turn the disordered, randomly initialised sets into relatively small, non-redundant ones; these will then start to grow and turn into useful, more complex combinations of increasingly specialised and fine-tuned rules. This effect was emphasised when a faster reproduction rate for the sets population was used, leading to an improved performance and overall quality of the results.

7.2 Representation

Ideally, the generic GA requires only two problem-dependent pieces of information: a fitness evaluation function and the length of the binary string used to represent each chromosome — there may be many other parameters, but they relate to the inner workings of the GA and are only tuned to prevent premature convergence and improve performance considerations. The GA will operate on the specified chromosomes with no additional information about the problem domain other than the value returned by the evaluation function. Indeed, this can be regarded as an advantageous feature of

GAs, making them a very general-purpose search technique. Alas, addressing the issue of what kinds of problems are adequately solved by GAs, it is known that if a suitable special-purpose, knowledge-based search technique does exist for a particular class of problems, GAs are often outperformed. However, "if the space to be searched is not so well understood and relatively unstructured, and if an effective GA representation of that space can be developed, then GAs provide a surprisingly powerful search heuristic for large, complex spaces" [DeJong 90].

One of the main objections to the traditional linear, fixed-length, binary representation arises because the linkage between the necessary gene combinations is too weak. If the choice of chromosome coding does not facilitate the required building block combinations, the GA will converge to suboptimal solutions. The simplest way to give some structure to a basic representation is to define special treatment for particular genes or explicit intra-chromosome boundaries — e.g. places where crossover points are permitted. One obvious drawback of these approaches is that domain-specific information that used to reside in the evaluation function, such as the encoding details, must now be at least partially incorporated within the functionality of the GA operators.

The idea of subjecting the representation itself to adaptation is not new ([Holland 75]), but there are very few proposals that implement this approach. A remarkable example appears in [Shaefer 87], where the ARGOT system "not only employs [...] Darwinian evolution of the chromosome population, but also utilizes information from the current trial solutions to modify the translation mapping between the chromosomes and the parameter space".

The capacity to evolve the best representation for a particular problem depends on the ability of the learning algorithm to modify its own structure and the way in which it codes for potential solutions. Typical fixed-length GAs have little capacity, or none at all, for adapting their genotype (their structure) because the length and meaning of each component has been determined a priori. By evolving its structure, a variable-length genotype may be able to discover not only the values of the parameters of a solution, but also how many parameters there should be, what they would mean and how they needed to interrelate. This variability introduces many degrees of freedom into the evolutionary search that are missing in fixed length structures.

With this two-level representation, it is easy to draw analogies between learning and evolution as two adaptive processes, one taking place during the lifetime of an organism, and the other over much larger periods of time, spanning several generations. However, the different element and set generations interacting at different rates relate more to the coexistence of different organisms, where the lifetime of one of them might spread over several generations of the other; not unlike a host-parasite relationship, where the health of the host depends to a large extent on how "benign" the current parasites are, and viceversa, since a healthy host guarantees a suitable environment for the parasite.

Obviously, the possibility to simultaneously use another GA for tuning the membership function definitions used by individual rules is an interesting option. This does not appear to involve complicated modifications to the current system at different levels, since the elements population encode the linguistic terms, not their interpretation. The inclusion of a third GA could be done in an analogous way, expanding the hierarchical structure of the system one more level, as discussed in the next chapter.

7.3 Learning Fuzzy Knowledge

Initially, the rules in the knowledge bases of FSs were obtained directly by interviewing experts, using the same knowledge engineering techniques developed by mainstream AI. Indeed, the original motivation for developing fuzzy logic was that it could be used to neatly capture human statements involving vague quantifiers, providing a formal framework to reason effectively with such statements. Obviously, the well-known disadvantages of traditional knowledge engineering techniques were carried over to FSs development.

Since the 2LGA system was mainly used to synthesise fuzzy networks, the discussion of certain considerations regarding the engineering of fuzzy systems is in order. In accordance with the introduction given in chapter 3, it is assumed that FSs in general, and FNs in particular, present five important features which are relevant to this thesis:

1. Compact rule bases — knowledge bases of FSs are normally much smaller than systems built using traditional artificial intelligence formalisms.

2. Statically and dynamically shallow structure — rules do not produce conclusions that are then used as premises in other rules. Statically, rules can be organised in a flat list and, dynamically, there is no run-time chaining of inferences.

3. Knowledge recorded in a FS typically reflects immediate correlations between the system's inputs and outputs, without the need of either a mathematical or a deep, causal model of the system — i.e. they are function estimators; the antecedents refer to system inputs, e.g. qualitative sensor observations, and consequents refer to system outputs, e.g. qualitative actuator settings. This work demonstrates the possibility of using an evolutionary algorithm to synthesise the rule base of a FS.

4. Numerical parameters, such as membership function definitions, can also be tuned by learning or searching algorithms, such as NNs or GAs.

5. Fuzzy logic operators are used — typically *min* and *max*, together with explicit possibility distributions, usually of triangular or trapezoidal shape. Fuzzification/defuzzification processes map between fuzzy values and their corresponding crisp, real values.

These features, taken together, make FN synthesis an ideal problem domain for 2LGA. In particular, the credit assignment problem appears to be solvable: it is possible to discover how to modify part of a complex system in order to improve it, given only an evaluation of its overall performance.

The fact that FSs normally do not need a large number of rules to solve real-life problems, in practice represents an obvious advantage, because the knowledge base becomes a small system to modify. 2LGA decidedly encourages the generation of compact rule bases. The introduction of the parsimony factor has a direct implication to this effect. Additionally, the coordinated, adaptive use of the cut & splice operators yielded supplementary assistance in achieving an adequate balance between system size and performance.

The short paths between the inputs and outputs of a FN imply that the effect of a particular change in the system tends to be localised, so it is easier to discover a change that has a positive effect without having other undesired consequences. 2LGA permits the alteration of independent rules in isolation. Introducing a new individual in the elements population will have a localised effect in those sets where it is contained, and is immediately available for exploitation by the evolutionary process in subsequent generations.

Two test cases were presented in chapter 6: one was a control problem, using a software simulation to provide the means for system evaluation; the other was a classification task, where a set of cases was used to construct "training" and "test" subsets. The iterative way in which FNs are refined allows a large number of observations of input/output performance to be used for system improvement, without the need for a formal or mathematical definition of it — FNs produced by 2LGA are evolved function estimators.

FSs present several unique features. What makes them useful in practice, among other things, is the combination of a rule-based formalism with numerical factors qualifying rules, and the simplicity and efficiency of fuzzy logic "reasoning". The principal advantage of rule-based formalisms in general is that knowledge can be acquired incrementally: individual rules and premises can be refined independently, or at least more independently than items of knowledge in other formalisms. This aspect is clearly reinforced by 2LGA's structure, with one level containing a GA completely dedicated to the evolution of individual rules, as opposed to comlete indivisible rule sets alone.

Chapter 8

Summary and Conclusions

A summary of achievements and a discussion of the contributions, issues raised and future directions is the concluding part of the thesis.

8.1 Summary

In chapter 2, the basics of genetic algorithms were introduced. The underlying principles and fundamental concepts were first presented in relation to traditional GAs. Important issues regarding selection/replacement mechanisms, fitness and performance evaluation were discussed, such as convergence and selection pressure. Special emphasis was placed on concepts relating to representation and organisation of the several components that GAs comprise, including the notions of building block, recombination and epistasis. Relevant issues such as adaptive parameters, variable-size and structured representations, parsimony, crowding and messy genetic algorithms were also introduced.

Chapter 3 presented an introduction to systems based on fuzzy logic. The basic elements involved in FSs were discussed, including the notions of fuzzy sets and rules. A simplified step-by-step description of the general process was given, covering the fuzzification/defuzzification and basic rule evaluation methods.

An overview of hybrid systems and soft computing was the subject of chapter 4. The most relevant approaches to the combined use of fuzzy systems and genetic algorithms were reviewed. Particular attention was paid to the role GAs have played as search or optimisation methods to synthesise FSs. A first description of fuzzy networks was presented, including important issues regarding the combination of GAs and connectionist systems.

In chapter 5, the basic two-level genetic algorithm was described. The presentation included the details of 2LGA's novel architecture and its operation. The two essential components — i.e. sets and elements, as used in this thesis — were fully described, including their forms of representation, operators and fitness evaluation. The important aspects of the interacting, co-evolving populations were introduced, together with the concepts of parsimony, adaptive parameters, heredity and the different replacement mechanisms devised for 2LGA.

Chapter 6 presented applications of 2LGA for two different problems. Firstly, the system was used to generate the rule base of a FLC to solve a simple, standard control problem. The notion of different reproduction rates for sets and elements populations was introduced, including a comparison of the different replacement methods that were devised for the elements population. Secondly, 2LGA was used to solve a classification task. The possibility to use the information available to the system in order to guide the automatic modification of some system parameters was explored.

A discussion of the most important aspects of this work was presented in chapter 7. The way adaptive parameters, as developed for 2LGA, may affect the search performed by the system is analysed. Issues on representation, especially the hierarchical organisation and variable-length chromosome structures were discussed. The ability of 2LGA to generate fuzzy rules was reviewed in the context of more general knowledge engineering concerns.

8.2 Conclusions

As system complexity increases, it becomes difficult to distinguish between peculiarities that seem deceptively interesting and features that actually teach us something important and general, either about how to design more efficient mechanisms or about the intrinsic properties of the problem. In this hierarchically organised GA, the cut & splice probabilities for the sets population, the replacement strategies for the elements population, the heredity and parsimony factors are all important, interrelated features that directly affect the performance of the system in different ways.

The tests demonstrate that the system is capable of finding very good solutions, in an effective and robust manner, for the kind of problems on which it was tested. Perhaps more importantly, the two-level architecture allows us to focus on more isolated features of the evolutionary search carried out by the system. Future work involves a thorough analysis of the interaction between the different levels and the application of the system in different problem domains.

It is true that binary, linearly-ordered representations allow the use of standard mutation and crossover operators in a problem-independent way. Nevertheless, the flexible, variable-length representation devised for the individuals in the sets population presents several advantages. The GA is able to manipulate short individuals not only during the early stages, but all along the duration of a complete run. This allows the system to combine short, well-tested building blocks into longer, more complex assemblies. Furthermore, it becomes easy for the GA to "backtrack" whenever two building blocks that are put together interact in a negative way, because the building blocks are able to exist in isolation.

The cut & splice operators help reduce the inherent, positional bias in standard crossover towards breaking up correlated genes that are widely separated on the chromosome. This is particularly important, since the strong positional dependence of most typical representations is an artifact introduced by GAs.

Obviously, an additional advantage of using a variable-size representation is that the adequate or even required size of the solution does not have to be known nor estimated beforehand; the answer to that question is provided by the system itself as a by-product.

Suitable replacement strategies for the elements populations were required, so a few different alternative schemes were devised. These produced not only somewhat different results, but also distinct trends in the behaviour of the system. On one hand, the 'adopted' strategy seemed to induce an undesired increase in the number of rules per set. However, a higher probability for the application of the cut operator, or a lower probability for the splice operator, or a raised parsimony factor would all lead to less sizeable sets. Thus, the values of these parameters can be *predictably* set accordingly, in order to account for that potential problem. Conversely, the 'inherited by substitution' replacement strategy had an obvious tendency to decrease the average size of the sets; this bias could be cancelled out by increasing the splice probability, or by decreasing the cut probability or even the parsimony factor. Furthermore, a suitable combination of all of these actions was easily obtained, without the need for extensive, sensitive fine tuning; it is possible in practise to maintain a balance between these interacting parameters in order to preserve the size of the solution within the desired range.

However, it became apparent that the difference in size between the best solution found so far and the average size could provide the information needed to guide the automatic adaptation of these parameters. As expected, once the cut and splice probabilities were subjected to automatic adjustment, they consistently moved towards the values that had been predicted and corroborated experimentally in the previous experiments. These results confirmed that the sources being used to direct the adaptation of the representation were providing the relevant information at the right time.

The introduction of replacement strategies brought with it the definition of a *heredity* factor. This system parameter joined the parsimony factor and the cut & splice probabilities in the set of self-adapting features of the system. Together, these parameters created a balanced, self-regulated mechanism that allowed 2LGA to effectively influence the size of the genotypes available to it, in order to conduct the evolutionary search under the most promising conditions, as indicated by the status of the system itself.

> "Intelligence may be [...] an emergent property of an enormously complicated self-organising system that uses a large range of self-regulatory mechanisms, rather than being directly the result of a small number of fairly simple mechanisms." [Ross 94]

8.3 Future Directions

The number of open possibilities presented by the hierarchical organisation of 2LGA is enormous. It would be impossible to enumerate every conceivable potential enhancement. What follows is a short list of promising directions, some of which are currently under development.

8.3.1 MLGA

The first obvious addition is the extension of the hierarchical structure to more than two levels. It was mentioned in section 4.3.1 that GAs have been successfully used to evolve the membership function definitions for fuzzy sets. The numerical framework offered by FSs presents a number of features that facilitate the automatic design of several system components, as observed in section 7.3. Tuning the membership function definitions by further introducing another level to the system is a logical extension to the current implementation. The development of a *3LGA* seems a plausible future enhancement to the current system. Just as sets are made of elements, elements would be, in turn, made of membership function definitions (see figure 8.1). The system would be capable of further refining the quality of the solutions being produced. This is in complete agreement with several of the hierarchical approaches that were mentioned in chapter 2, and the trend in hybrid systems introduced in chapter 4.

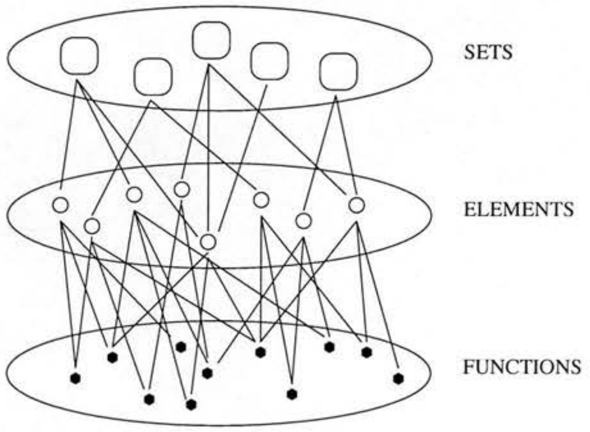


Figure 8.1: Three-level GA (3LGA)

For this new "bottom" level GA, a first attempt could simply take a similar approach to that presented in [Karr 91]: a predefined number of parameters are represented in a linear genotype so that a GA, using standard recombination and mutation operators, sets out to optimise the shape and position of these membership functions. Obviously, a variable-size representation could also be used, allowing the GA to attempt to work out the correct number of membership functions required to adequately characterise each fuzzy set. Fitness evalutation could be guided in a similar fashion to the one adopted for the middle level, i.e. taking the average of the fitness of those elements (rules) to which each membership function belongs. This would imply a top-down propagation of fitness, from whole rule sets, to indivitual rules, down to membership function definitions; it would become important then to certify that the final fitness value is genuinely representative of the aptitude of individual membership functions in the appropriate context.

8.3.2 Adaptive Reproduction Rates between Populations

As mentioned in section 6.1.3, a faster reproduction of sets (N) allows the system to promptly find useful combinations of available building blocks, but does not facilitate fine-tuning individual rules. Conversely, faster reproduction of rules (M) induces slow progress, but allows the system to improve the quality and sophistication of rules in the later stages.

In an initial attempt to explore this particular feature of the system further, it would be important to vary the M/N ratio in an dynamic way: start with a relatively small (< 1) value and let it increase (>1) as the system progresses towards the final stages. This can be done in several different ways, but a simple linear trend can be defined, maintaining each parameter inversely proportional to the other, as depicted in figure 8.2 below.

Taking this idea further, instead of having a predefined rate change schedule, it seems plausible to devise a mechanism to modify reproduction rates in an *adaptive* fashion. In this case, diversity at the different representation levels, together with on-line performance measurements, could provide the information required to influence the decision on which M/N ratio to have at the different stages of the evolutionary search. At any rate, a similar trend to that which has been determined experimentally, outlined above for the dynamic linear ratio adjustment, would be expected, as ilustrated in figure 8.3.

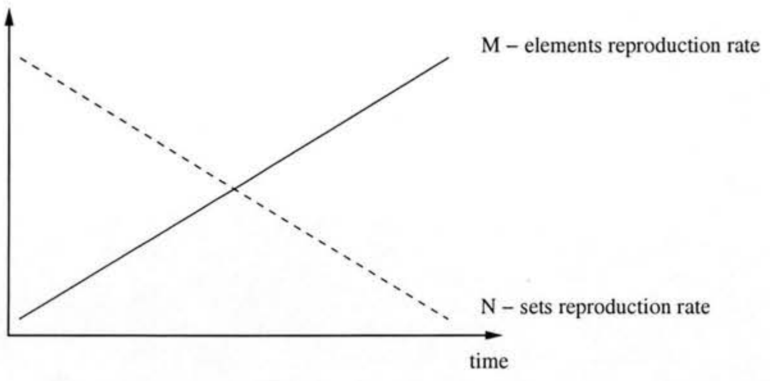


Figure 8.2: Dynamic linear M/N ratio adjustment

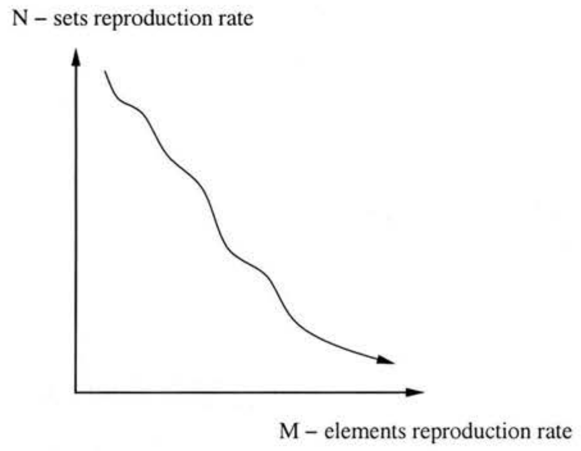


Figure 8.3: Adaptive M/N ratio adjustment

8.3.3 Genic Selective Crowding

In [Goldberg *et al.* 90], the concept of *genic selective crowding* was introduced. The idea is to compare individuals for selection purposes only when they are similar enough. Tournament selection is carried out as usual, except that individuals can only take part if they have at least some threshold number of genes in common. The aim is to counter the problem of nonuniform subfunction scaling and variable building block size: comparison of partial evaluations are meaningful to the extent that they refer to the same partial solution. Although rule modularity alleviates this problem for 2LGA, the same line of reasoning could be extended, so that individuals in the sets population can only compete with those other sets which share a minimum number of rules.

It would be relatively easy to modify the selection scheme proposed in section 5.2.2, in order to accomodate genic selective crowding. A new parameter would have to be defined, and its value tuned, establishing the minimum number of rules that two sets must share before they can be compared. Keeping in mind that there might be significant differences in set sizes, this threshold could probably be better defined in terms of the candidates' length — e.g. a fraction of their average size — instead of being an absolute number. Furthermore, because under certain circumstances it is desirable to allow distinct sets to recombine, a stochastic factor could be included, allowing the selection of dissimilar candidates with a given, presumably small, probability. A measure of diversity could be used to guide the adaptability of this new parameter, in an analogous way to the temperature parameter used in simulated annealing, for example.

8.3.4 Combination/Adaptation of Recombination Operators

As it was demonstrated in section 6.2.6, the automatic adaptation of the cut & splice operators, based on the information provided by the difference in size between the best solution found so far and the population's average, is an important contribution to enhance the system's robustness and performance. It also means that the potential user will not necessarily have to dedicate time to tune some of the system's parameters.

The introduction of the mixing operator (see section 6.2.8) was circumstantial, but it demonstrated that it is possible to improve on-line performance of the system under certain conditions. As is always the case, it is difficult to strike a balance between exploration and exploitation, but many of the original claims made in that context in favour of uniform crossover ([Syswerda 89], [Eshelman *et al.* 89]) could be in principle extrapolated to the mixing operator. It would be important to carry out research focusing on population analysis, monitoring the juxtapositional properties of mixing, in particular with regards to the preservation of diversity, but keeping building block disruption to acceptable levels at the same time. Careful analysis of the application of alternative recombination operators would undoubtedly increase the usability of the system under different settings.

8.3.5 2L*GP*

A more radical modification would be to use tree-like representations at either one or both levels of the 2LGA, based on formulations derived from the Genetic Programming literature. As introduced in section 4.5 (see also section 2.11.1), various analogies can be drawn from GP and applied within the hierarchical organisation of the system presented in this thesis.

A first attempt would be to substitute the variable-size linear genotypes of the sets populations in favour of a tree-like representation. A rule set would thus be defined by a structure of dynamically varying size and shape, using a recursive definition with simple "concatenation" operator nodes linking the leaves in the tree containing links to the elements populations, i.e. individual rules. As an illustration, the tree depicted in figure 8.4 would represent the set containing rules $R1$ to $R7$.

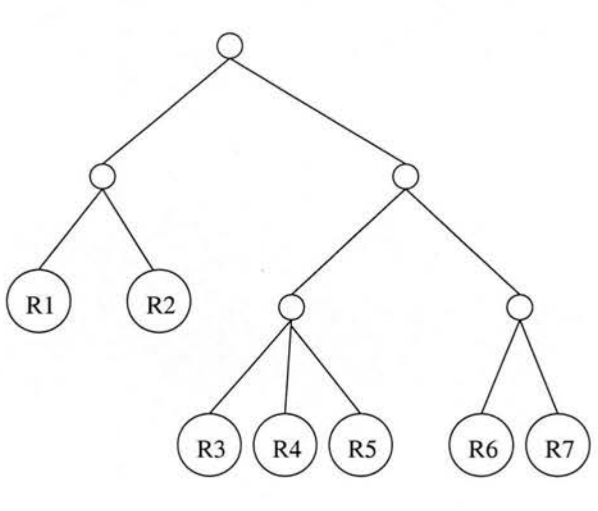


Figure 8.4: Rule set representation in 2LGP

Once two individuals have been chosen for reproduction, the crossover operator would be used to recombine individuals in the usual way: pivot nodes would be picked in both parents and the corresponding subtrees swapped between them. Likewise, mutation could be carried out by replacing a particular subtree with a newly generated one. Given that all genotypes share the same properties of recursive tree structures, syntactic constraints are always preserved.

Similarly, automatic function definition and adaptive representation techniques such as those presented in [Rosca & Ballard 94], which are focused on the discovery of useful building blocks, could be adapted to the 2LGP. Since the concatenation operator accepts a variable number of arguments, two or more levels of a subtree can be collapsed into one, effectively creating a larger building block. Proceeding in the opposite direction, one node with several branches could be reorganised by creating an intermediate new level, consisting of two or more new nodes, and distributing the original subtrees below them. These mutually complementary operators would facilitate the creation and destruction of more elaborate subsets (subtrees) or rules.

Obviously, the notions developed for the 2LGA regarding set size would still apply under the new scheme. The parsimony factor, the expansion and contraction of subtrees, branch insertion and branch deletion could all be used by the system to aid in the self-regulation of the average size of individuals.

Bibliography

[Abbattista *et al.* 98] F. Abbattista, G. Castellano, and A. M. Fanelli. An iterative two-phase approach to fuzzy system design. In P.K. Chawdhry, R. Roy, and R.K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, chapter 3. Springer-Verlag, 1998.

[Arbib 87] M.A. Arbib. *Brains, Machines, and Mathematics.* Springer-Verlag, second Springer-Verlag edition, 1987.

[Bäck & Hoffmeister 91] T. Bäck and F. Hoffmeister. Extended Selection Mechanisms in Genetic Algorithms. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 92–99. Morgan Kaufmann Publishers, 1991.

[Bäck & Schwefel 93] T. Bäck and H.P. Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[Bailey 81] N.T.J. Bailey. *Statistical Methods in Biology.* Edward Arnold, second edition, 1981.

[Baker 85] J.E. Baker. Adaptive Selection Methods for Genetic Algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 101–111. Lawrence Erlbaum Associates, 1985.

[Baker 87] J.E. Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21. Lawrence Erlbaum Associates, 1987.

[Beasley *et al.* 93a] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[Beasley *et al.* 93b] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[Bethke 80] A.D. Bethke. *Genetic Algorithms as Function Optimizers*. Unpublished PhD thesis, University of Michigan, 1980.

[Beyer 95] H.G. Beyer. How GAs do *NOT* Work: Understanding GAs without Schemata and Building Blocks. Technical Report D-44221, Systems Analysis Research Group, University of Dortmund, 1995.

[Bickel & Bickel 87] A.S. Bickel and R.W. Bickel. Tree Structured Rules in Genetic Algorithms. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 77–81. Lawrence Erlbaum Associates, 1987.

[Bilchev & Parmee 95] G. Bilchev and I.C. Parmee. The ant colony metaphor for searching continuous design spaces. In T.C. Fogarty, editor, *Proceedings fo AISB Workshop on Evolutionary Computing, Lecture Notes in Computer Science 993*, pages 25–39. Springer-Verlag, 1995.

[Blumer *et al.* 87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[Booch 91] G. Booch. *Object Oriented Design, with Applications*. Benjamin/Cummings, 1991.

[Booker 87] L. Booker. Improving Search in Genetic Algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 61–73. Pitman, 1987.

[Box 57] G.E.P. Box. Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society*, 6(2):81–101, 1957.

[Bremermann 62] H.J. Bremermann. Optimization Through Evolution and Recombination. In M.C. Tovits, G.T. Jacobi, and G.D. Goldstein, editors, *Self-Organizing Systems*, pages 93–106. Spartan Books, 1962.

[Brindle 81] A. Brindle. *Genetic Algorithms for Function Optimization.* Unpublished PhD thesis, University of Alberta, 1981.

[Buhusi 94] C.V. Buhusi. Learning by Simulating Evolution in Automatic Fuzzy Systems Synthesis. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1308–1313, 1994.

[Bundy *et al.* 89] A. Bundy, B. du Boulay, J. Howe, and G. Plotkin. The Researcher's Bible. Teaching Paper No. 4, 1989. Department of Artificial Intelligence, The University of Edinburgh.

[Carse & Fogarty 94] B. Carse and T.C. Fogarty. A Fuzzy Classifier System Using the Pittsburgh Approach. In Y. Davidor, H.P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature - PPSN III, Proceedings of the Third International Conference on Evolutionary Computation*, pages 260–269. Springer-Verlag, 1994.

[Cavicchio 70] D.J. Cavicchio. *Adaptive Search Using Simulated Evolution.* Unpublished PhD thesis, University of Michigan, 1970.

[Chen & Parmee 98] K. Chen and I.C. Parmee. A comparison of evolutionary-based strategies for mixed-discrete multi-level design problems. In I.C. Parmee, editor, *Adaptive Computing in Design and Manufacture.* Springer-Verlag, 1998.

[Chen 76] P.P-S. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM TODS 1*, 1976.

[Chowdhury & Li 96] M.M.M. Chowdhury and Y. Li. Messy Genetic Algorithm Based New Learning Method for Structurally Optimised Neurofuzzy Controllers. In *Proceedings of the IEEE International Conference on Industrial Technology*, pages 274–278, 1996.

[Codd 70]	E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *CACM*, 13(6), 1970.
[Coloni *et al.* 92]	A. Coloni, M. Dorigo, and V. Maniezzo. An investigation of some properties of the ant algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature - PPSN II*, pages 509–520. Elsevier Science Publishers, 1992.
[Cooper & Vidal 93]	M.G. Cooper and J.J. Vidal. Genetic Design of Fuzzy Controllers. In *Second International Conference on Fuzzy Theory and Technology*, 1993.
[Cooper & Vidal 94]	M.G. Cooper and J.J. Vidal. Genetic Design of Fuzzy Controllers: The Cart and Jointed-Pole Problem. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1332–1337, 1994.
[Cordón & Herrera 94]	O. Cordón and F. Herrera. A three-stage evolutionary process for learning descriptive and approximative fuzzy logic controller knowledge bases from examples. *International Journal of Approximate Reasoning*, 11:1–39, 1994.
[Cordón *et al.* 95]	O. Cordón, F. Herrera, and M. Lozano. A Classified Review on the Combination Fuzzy Logic-Genetic Algorithms Bibliography. Technical Report DECSAI-95129, Dept. of Computer Science and A.I., University of Granada, 1995.
[Cordón *et al.* 96]	O. Cordón, F. Herrera, and M. Lozano. On the Bidirectional Integration of Genetic Algorithms and Fuzzy Logic. In J. Periaux, G. Winter, M. Galán, and P. Cuesta, editors, *Proceedings of the Second Online Workshop on Evolutionary Computation (WEC2)*, pages 13–16, 1996.
[Cox 86]	B.J. Cox. *Object Oriented Programming: An Evolutionary Approach.* Addison-Wesley, 1986.
[Cramer 85]	N.L. Cramer. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of an International*

Conference on Genetic Algorithms and Their Applications, pages 183–187, 1985.

[Culberson 96] J.C. Culberson. On the Futility of Blind Search. Technical Report 9618, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, 1996.

[Darwin 59] C. Darwin. *On the Origin of Species by Means of Natural Selection.* Murray, 1859.

[Dasgupta & McGregor 92] D. Dasgupta and D.R. McGregor. Designing Application-Specific Neural Networks using the Structured Genetic Algorithm. In L.D. Whitley and J.D. Schaffer, editors, *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96. IEEE Computer Society Press, 1992.

[Davidor 90] Y. Davidor. An Adaptation Anomaly of a Genetic Algorithm. Technical Report CS90–05, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, 1990.

[Davis & Principe 93] T.E. Davis and J.C. Principe. A markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, 1(3):269–288, 1993.

[Davis 85] L. Davis. Job Shop Scheduling with Genetic Algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, pages 136–140. Lawrence Erlbaum Associates, 1985.

[Davis 89] L. Davis. Adapting Operator Probabilities in Genetic Algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.

[Davis 91] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[Dawkins 91] R. Dawkins. *The Blind Watchmaker.* The Penguin Group, 1991.

[Deb & Goldberg 91] K. Deb and D.E. Goldberg. mGA in C: A Messy Genetic Algorithm in C. Technical Report 91008, Illinois Genetic Algorithms Laboratory (IlliGAL), 1991.

[Deb 91] K. Deb. *Binary and Floating-Point Function Optimization Using Messy Genetic Algorithms.* Unpublished PhD thesis, University of Illinois, 1991. (IlliGAL Report No. 91004).

[DeJong 75] K. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Unpublished PhD thesis, University of Michigan, 1975.

[DeJong 90] K. DeJong. Introduction. *Machine Learning*, 5(4):351–353, 1990.

[DeJong 96] K. DeJong. Evolutionary Computation: Recent Developments and Open Issues. In *EvCA-96*, pages 7–17, 1996.

[Dennett 95] D.C. Dennett. *Darwin's Dangerous Idea.* Allen Lane The Penguin Press, 1995.

[Elkan 93] C. Elkan. The Paradoxical Success of Fuzzy Logic. Technical Report 92093-0114, Department of Computer Science and Engineering, University of California, San Diego, 1993.

[Eshelman *et al.* 89] L.J. Eshelman, R. Caruna, and J.D. Schaffer. Biases in the Crossover Landscape. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19. Morgan Kaufmann, 1989.

[Feldman 93] D.S. Feldman. Fuzzy Network Synthesis with Genetic Algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 312–317. Morgan Kaufmann Publishers, 1993.

[Fogarty 89] T.C. Fogarty. Varying the Probability of Mutation in the Genetic Algorithm. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 104–109. Morgan Kaufmann, 1989.

[Fogel & Atmar 92] D.B. Fogel and J.W. Atmar, editors. *Proceedings of the First Annual Conference on Evolutionary Programming.* Evolutionary Programming Society, 1992.

[Fogel 93] D.B. Fogel. On the Philosophical Differences between Evolutionary Algorithms and

Genetic Algorithms. In D.B. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 23–29. Evolutionary Programming Society, 1993.

[Fogel 98] D.B. Fogel, editor. *Evolutionary Computation, The Fossil Record.* IEEE Press, 1998.

[Fogel *et al.* 66] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution.* Wiley, 1966.

[Forrest & Mitchell 92] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, 1992.

[Fraser 57] A.S. Fraser. Simulation of Genetic Systems by Automatic Digital Computers. *Australian Journal of Biological Science*, 10:484–499, 1957.

[Friedberg 58] R.M. Friedberg. A Learning Machine: Part I. *IBM Journal*, pages 2–13, 1958.

[Friedman 59] G.J. Friedman. Digital simulation of an evolutionary process. *General Systems Yearbook*, 4:171–184, 1959.

[Fujiko & Dickinson 87] C. Fujiko and J. Dickinson. Using the Genetic Algorithm to Generate LISP Source Code to Solve the Prisoner's Dilemma. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 236–240. Lawrence Erlbaum Associates, 1987.

[Fukuda & Shibata 94] T. Fukuda and T. Shibata. Fuzzy-Neuro-GA Based Intelligent Robotics. In J.M. Zurada, R.J. MarksII, and C.J. Robinson, editors, *Computational Intelligence: Imitating Life.* IEEE Press, 1994.

[Goldberg & Deb 91] D.E. Goldberg and K. Deb. A Comparative Analysis of Selection Schemes used in Genetic Algorithms. In G.J.E. Rawlings, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[Goldberg & Richardson 87] D.E. Goldberg and J. Richardson. Genetic Algorithms with Sharing for Multimodal Function Optimization. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum Associates, 1987.

[Goldberg 89a] D.E. Goldberg. Genetic Algorithms and Walsh Functions: Part I, a Gentle Introduction. *Complex Systems*, 3:129–152, 1989.

[Goldberg 89b] D.E. Goldberg. Genetic Algorithms and Walsh Functions: Part II, Deception and its Analysis. *Complex Systems*, 3:153–171, 1989.

[Goldberg 89c] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley Publishing Company, 1989.

[Goldberg 91a] D.E. Goldberg. Construction of high-order deceptive functions using low-order Walsh coefficients. Technical Report 90002, Illinois Genetic Algorithms Laboratory (IlliGAL), 1991.

[Goldberg 91b] D.E. Goldberg. Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking. *Complex Systems*, 5:139–167, 1991.

[Goldberg *et al.* 89] D.E. Goldberg, B. Korb, and K. Deb. Messy Genetic Algorithms: Motivation, Analysis and First Results. *Complex Systems*, 3:493–530, 1989.

[Goldberg *et al.* 90] D.E. Goldberg, K. Deb, and B. Korb. Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale. *Complex Systems*, 4:415–444, 1990.

[Goldberg *et al.* 91] D.E. Goldberg, K. Deb, and B. Korb. Don't Worry, Be Messy. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 24–30. Morgan Kaufmann Publishers, 1991.

[Goldberg *et al.* 92] D.E. Goldberg, K. Deb, and D. Thierens. Toward a Better Understanding of Mixing in Genetic Algorithms. Technical Report 92009, Illinois Genetic Algorithms Laboratory (IlliGAL), 1992.

[Goldberg *et al.* 93]	D.E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. Technical Report 93004, Illinois Genetic Algorithms Laboratory (IlliGAL), 1993.
[González & Herrera 96]	A. González and F. Herrera. Multi-Stage Genetic Fuzzy Systems Based on the Iterative Rule Learning Approach. Technical Report DECSAI-96128, Dept. of Computer Science and A.I., University of Granada, 1996.
[Grefenstette 86]	J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16:122–128, 1986.
[Grosso 85]	P.B. Grosso. *Computer Simulation of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model.* Unpublished PhD thesis, University of Michigan, 1985.
[Gruau 93]	F. Gruau. Genetic Synthesis of Modular Neural Networks. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 318–325. Morgan Kaufmann Publishers, 1993.
[Herrera *et al.* 95]	F. Herrera, M. Lozano, and J.L. Verdegay. A Learning Process for Fuzzy Control Rules using Genetic Algorithms. Technical Report DECSAI-95108, Dept. of Computer Science and A.I., University of Granada, 1995.
[Hoffmann & Pfister 94]	F. Hoffmann and G. Pfister. Automatic Design of Hierarchical Fuzzy Controllers Using Genetic Algorithms. In *Proceedings of the Second European Conference on Intelligent Techniques and Soft Computing (EUFIT'94)*, pages 1516–1522, 1994.
[Hoffmann & Pfister 96]	F. Hoffmann and G. Pfister. Evolutionary Learning of a Fuzzy Control Rule Base for an Autonomous Vehicle. In *Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems*, pages 1235–1240, 1996.

[Holland 75]	J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, first University of Michigan edition, 1975.
[Holland 92]	J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, first MIT Press edition, 1992.
[Holland 95]	J.H. Holland. *Hidden Order.* Addison-Wesley Publishing Company, 1995.
[Holland *et al.* 86]	J.H. Holland, K.J. Holyoak, and R.E. Nisbett. *Induction: Processes of Inference, Learning and Discovery.* MIT Press, 1986.
[Hwang & Thompson 94]	W. Hwang and W.E. Thompson. Design of Intelligent Fuzzy Logic Controllers Using Genetic Algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1383–1388, 1994.
[Ichimura *et al.* 95]	T. Ichimura, T. Takano, and E. Tazaki. Reasoning and Learning Method for Fuzzy Rules Using Neural Networks with Adaptive Structured Genetic Algorithm. In *Proceedings of the Third European Conference on Intelligent Techniques and Soft Computing (EUFIT'95)*, pages 3269–3274, 1995.
[Ishibuchi *et al.* 94]	H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka. Acquisition of Fuzzy Classification Knowledge Using Genetic Algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1963–1968, 1994.
[Ishigami *et al.* 94]	H. Ishigami, Y. Hasegawa, T. Fukuda, and T. Shibata. Automatic Generation of Hierarchical Structure of Fuzzy Inference by Genetic Algorithm. In *Proceedings of the IEEE International Congress on Neural Networks*, pages 1566–1570, 1994.
[Jones 95]	T.C. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* Unpublished PhD thesis, University of New Mexico, Albuquerque, 1995.
[K. Deb 92]	D.E. Goldberg K. Deb. Sufficient Conditions for Deceptive and Easy Binary Functions. Technical Report 91009, Illinois Genetic Algorithms Laboratory (IlliGAL), 1992.

[Kargupta 97] H. Kargupta. SEARCH, Computational Processes in Evolution, and Preliminary Development of the Gene Expression Messy Genetic Algorithm. *Complex Systems*, 11:233–287, 1997.

[Karr 91] C.L. Karr. Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 450–457. Morgan Kaufmann Publishers, 1991.

[Kinnear 94a] K. Kinnear. *Advances in Genetic Programming*. MIT Press, 1994.

[Kinnear 94b] K. Kinnear. Alternatives in automatic function definition: A comparison of performance. In K. Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994.

[Knjazew & Goldberg 00] D. Knjazew and D.E. Goldberg. OMEGA - Ordering Messy GA: Solving Permutation Problems with the Fast Messy Genetic Algorithm and Random Keys. Technical Report 2000004, Illinois Genetic Algorithms Laboratory (IlliGAL), 2000.

[Kosko 92] B. Kosko. *Neural Networks and Fuzzy Systems*. Prentice-Hall International, 1992.

[Kosko 94] B. Kosko. *Fuzzy Thinking*. HarperCollinsPublishers, 1994.

[Koza 92] J. Koza. *Genetic Programming: On the Programming of Computers by the Means of Natural Selection*. MIT Press, 1992.

[Koza 94] J. Koza. *Genetic Programming II*. MIT Press, 1994.

[Lee & Takagi 93] M.A. Lee and H. Takagi. Dynamic Control of Genetic Algorithms Using Fuzzy Logic Techniques. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 76–83. Morgan Kaufmann Publishers, 1993.

[Li & NG 95] Y. Li and K.C. NG. Genetic Algorithm Based Techniques for Design Automation of Three-Term Fuzzy Systems. In *Proceedings of the*

Sixth International Fuzzy Systems Association World Congress (IFSA'95), volume 1, pages 261–264, 1995.

[Lindley 87] D.V. Lindley. The Probability Approach to the Treatment of Uncertainty in Artificial Intelligence and Expert Systems. *Statistical Science*, 2(1):17–24, 1987.

[Linkens & Nyongesa 96] D.A. Linkens and H.O. Nyongesa. Learning systems in intelligent control: an appraisal of fuzzy, neural and genetic algorithm control applications. *IEE Proceedings of Control Theory Applications*, 143(4):367–386, 1996.

[Liska & Melsheimer 94] J. Liska and S.S. Melsheimer. Complete Design of Fuzzy Logic Systems Using Genetic Algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1377–1382, 1994.

[Macready & Wolpert 95] W.G. Macready and D.H. Wolpert. What Makes an Optimization Problem Hard? Technical Report SFI-TR-95-05-046, The Santa Fe Institute, Santa Fe, New Mexico, 1995.

[Michalewicz 92] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

[Minsky 85] M. Minsky. *The Society of Mind.* Simon & Schuster, first Touchstone edition, 1985.

[Mitchell & Forrest 93] M. Mitchell and S. Forrest. Genetic Algorithms and Artificial Life. *Artificial Life*, 1993.

[Mitchell 95] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1995.

[Mühlenbein & Schlierkamp-Voosen 94] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm bga. *Evolutionary Computation*, 2:335–360, 1994.

[Mühlenbein 92] H. Mühlenbein. How Genetic Algorithms Really Work I: Mutation and Hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature - PPSN II*, pages 15–25. Elsevier Science Publishers, 1992.

[Ng & Li 94] K.C. Ng and Y. Li. Design of Sophisticated Fuzzy Logic Controllers Using Genetic Algorithms. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1708–1712, 1994.

[Nordin & Banzhaf 95] P. Nordin and W. Banzhaf. Complexity Compression and Evolution. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 310–317. Morgan Kaufmann Publishers, 1995.

[Parmee 96] I.C. Parmee. The development of a dual-agent strategy for efficient search across whole system engineering hierarchies. In *Proceedings of the 4th International Conference on Parallel Solving from Nature. Lecture Notes in Computing 1141*, pages 523–532. Springer-Verlag, 1996.

[Parmee 97] I.C. Parmee. Strategies for the Integration of Evolutionary/Adaptive Search with the Engineering Design Process. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 453–478. Springer-Verlag, 1997.

[Parmee 99] I.C. Parmee. Exploring the Design Potential of Evolutionary Search, Exploration and Optimisation. In P.J. Bentley, editor, *Evolutionary Design by Computers*, chapter 5, pages 119–143. Morgan Kaufmann Publishers, 1999.

[Perry 84] Z.A. Perry. *Experimental Study of Speciation in Ecological Niche Theory Using Genetic Algorithms.* Unpublished PhD thesis, University of Michigan, 1984.

[Quinlan 92] J.R. Quinlan. *C4.5.: Programs for machine learning.* Morgan Kaufmann, 1992.

[Radcliffe & George 93] N.J. Radcliffe and F.A.W. George. A Study in Set Recombination. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 23–30. Morgan Kaufmann Publishers, 1993.

[Radcliffe & Surry 92] N.J. Radcliffe and P.D. Surry. Genetic Set Recombination. In D. Whitley, editor, *Foundations of Genetic Algorithms 2.* Morgan Kaufmann, 1992.

[Radcliffe & Surry 94] N.J. Radcliffe and P.D. Surry. Co-operation through Hierarchical Competition in Genetic Data Mining. Technical Report EPCC-TR94-09, Edinburgh Parallel Computing Centre, University of Edinburgh, 1994.

[Radcliffe 92] N.J. Radcliffe. Non Linear Genetic Representations. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature - PPSN II*, pages 259–268. Elsevier Science Publishers, 1992.

[Radcliffe 95] N.J. Radcliffe. GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms Final Report. Technical Report EPCC-AIKMS-GA-MINER-REPORT 1.0, Edinburgh Parallel Computing Centre, University of Edinburgh, 1995.

[Rechenberg 73] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Frommann-Holzboog, 1973.

[Rosca & Ballard 94] J.P. Rosca and D.H. Ballard. Genetic Programming with Adaptive Representations. Technical Report 566, University of Rochester, Computer Science Department, 1994.

[Rosenberg 67] R.S. Rosenberg. *Simulation of Genetic Populations with Biochemical Properties.* Unpublished PhD thesis, University of Michigan, 1967.

[Rosenman & Gero 99] M. Rosenman and J. Gero. Evolving Designs by Generating Useful Complex Gene Structures. In P.J. Bentley, editor, *Evolutionary Design by Computers*, chapter 15, pages 345–364. Morgan Kaufmann Publishers, 1999.

[Ross & Ballinger 93] P. Ross and G.H. Ballinger. PGA, the Parallel Genetic Algorithms testbed. ftp.dai.ed.ac.uk, 1993. Department of Artificial Intelligence, The University of Edinburgh.

[Ross 94] P. Ross. Basics of Genetics. Department of Artificial Intelligence, The University of Edinburgh, 1994.

[Ross 96] P. Ross. About PGA v3.0. Department of Artificial Intelligence, The University of Edinburgh, 1996.

[Russo 98] M. Russo. Fugenesys— a fuzzy genetic neural system for fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, 6(3):373–388, 1998.

[Sakurai *et al.* 94] M. Sakurai, Y. Kurihara, and S. Karasawa. Color Classification Using Fuzzy Inference and Genetic Algorithm. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, pages 1975–1978, 1994.

[Schaffer & Morishima 87] J.D Schaffer and A. Morishima. An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 36–40. Lawrence Erlbaum Associates, 1987.

[Schaffer *et al.* 92] J.D. Schaffer, D. Whitley, and L.J. Eshelman. Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art. In L.D. Whitley and J.D. Schaffer, editors, *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE Computer Society Press, 1992.

[Schwefel 75] H.P. Schwefel. *Evolutionsstrategie und numerische Optimierung.* Unpublished PhD thesis, Technical University of Berlin, 1975.

[Schwefel 81] H.P. Schwefel. *Numerical Optimization for Computer Models.* John Wiley & Sons, 1981.

[Shaefer 87] C.G. Shaefer. The Argot Strategy: Adaptive Representation Genetic Optimizer Technique. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 50–58. Lawrence Erlbaum Associates, 1987.

[Shaffer 94] J.D. Shaffer. Combinations of Genetic Algorithms with Neural Networks or Fuzzy Systems. In J.M. Zurada, R.J. MarksII, and C.J. Robinson, editors, *Computational Intelligence: Imitating Life.* IEEE Press, 1994.

[Shimojima *et al.* 95]	K. Shimojima, T. Fukuda, and Y. Hasegawa. Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm. *Fuzzy Sets and Systems*, 71:295–309, 1995.
[Smith & Valenzuela-Rendón 94]	R.E. Smith and M. Valenzuela-Rendón. Introduction to the Special Issue. *Evolutionary Computation*, 2(1):v–vii, 1994.
[Smith 80]	S.F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. Unpublished PhD thesis, University of Pittsburgh, 1980.
[Smolensky 86]	P. Smolensky. Information Processing in Dynamical Systems: Foundations of Harmony Theory. In D.E. Rumelhart, J.L. McClelland, and PDP Research Group, editors, *Parallel Distributed Processing, vol. I.* MIT Press/Bradford Books, 1986.
[Stiny 80]	G. Stiny. Introduction to shape and shape grammars. *Environment and Planning*, 7:343–351, 1980.
[Syswerda 89]	G. Syswerda. Uniform Crossover in Genetic Algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1989.
[Syswerda 91]	G. Syswerda. Schedule Optimization Using Genetic Algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 332–349. Van Nostrand Reinhold, 1991.
[Tang *et al.* 98]	K. Tang, K. Man, Z. Liu, and S. Kwong. Minimal fuzzy memberships and rules using hierarchical genetic algorithms. *IEEE Transactions on Industrial Electronics*, 45(2):162–169, 1998.
[Thierens & Goldberg 93]	D. Thierens and D.E. Goldberg. Mixing in Genetic Algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45. Morgan Kaufmann Publishers, 1993.
[Thierens & Goldberg 94]	D. Thierens and D.E. Goldberg. Convergence Models of Genetic Algorithm Selection Schemes. In Y. Davidor, H.P. Schwefel, and

R. Männer, editors, *Parallel Problem Solving from Nature - PPSN III, Proceedings of the Third International Conference on Evolutionary Computation*, pages 119–129. Springer-Verlag, 1994.

[Thrift 91] P. Thrift. Fuzzy Logic Synthesis with Genetic Algorithms. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 509–513. Morgan Kaufmann Publishers, 1991.

[Topchy *et al.* 96] A.P. Topchy, V.V. Miagkikh, R.N. Kononenko, and A.N. Melikhov. Adaptive Genetic Search for Optimization of Fuzzy and Neuro-fuzzy Systems. In *EvCA-96*, pages 245–253, 1996.

[Valenzuela-Rendón 91] M. Valenzuela-Rendón. The Fuzzy Classifier System: A Classifier System for Continuously Varying Variables. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 346–353. Morgan Kaufmann Publishers, 1991.

[Venturini 93] G. Venturini. A Supervised Inductive Algorithm with Genetic Search for Learning Attribute Based Concepts. In *Proceedings of the European Conference on Machine Learning*, pages 280–296, 1993.

[Viot 93] G. Viot. Fuzzy Logic: Concepts to Constructs. *AI Expert*, pages 26–33, November 1993.

[Wang *et al.* 98] C.H. Wang, T.P. Hong, and S.S. Tseng. Integrating fuzzy knowlege by genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):138–149, 1998.

[Wasserman 93] P.D. Wasserman. *Advanced Methods in Neural Computing.* Van Nostrand Reinhold, 1993.

[Watson & Parmee 97] A.H. Watson and I.C. Parmee. Steady State Genetic Programming with Constrained Complexity Crossover Using Species Subpopulation. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, 1997.

[Watson & Pollack 99] R.A. Watson and J.B. Pollack. Incremental Commitment in Genetic Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99*, volume 1, pages 710–717. Morgan Kaufmann Publishers, 1999.

[Weinberg 70] R. Weinberg. *Computer Simulation of a Living Cell.* Unpublished PhD thesis, University of Michigan, 1970.

[Whitley 89] D. Whitley. The G*ENITOR* Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[Wilson & Goldberg 89] S.W. Wilson and D.E. Goldberg. A Critical Review of Classifier Systems. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255. Morgan Kaufmann, 1989.

[Wolpert & Macready 95] D.H. Wolpert and W.G. Macready. No Free Lunch Theorems for Search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, New Mexico, 1995.

[Zadeh 65] L.A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.

[Zadeh 94] L.A. Zadeh. Fuzzy Logic, Neural Networks, and Soft Computing. *Communications of the ACM*, 37:77–84, March 1994.

[Zhang & Mühlenbein 93] BT Zhang and H. Mühlenbein. Genetic Programming of Minimal Neural Nets Using Occam's Razor. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 342–349. Morgan Kaufmann Publishers, 1993.

[Zhang & Mühlenbein 95] BT Zhang and H. Mühlenbein. Balancing Accuracy and Parsimony in Genetic Programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[Zimmerman 94] H.J. Zimmerman. Hybrid Approaches for Fuzzy Data Analysis and Configuration Using

Genetic Algorithms and Evolutionary Methods. In J.M. Zurada, R.J. MarksII, and C.J. Robinson, editors, *Computational Intelligence: Imitating Life*. IEEE Press, 1994.