Information Systems for Tactical Decision Making

Thesis submitted in accordance with the

requirements of the University of Liverpool

for the degree of Doctor in Philosophy by

Andrew Fairley.

September, 1994.

ORIGINAL COPY TIGHTLY BOUND

Information Systems for Tactical Decision Making

Andrew Fairley

Abstract

This dissertation describes pragmatic research into the application of Evolutionary Algorithms (EAs) to the development of military tactics. The motivation for this work arose out of problems associated with manually generating knowledge bases for military Command and Control systems. Conventional methods, whereby knowledge is elicited from military experts, are inherently time consuming and costly. Therefore, a method for automatically acquiring the necessary knowledge would be immensely beneficial. An EA, employed as part of a machine learning system, is one approach which, it has been suggested, can perform this task.

The primary aim of this work is to produce a genetics-based machine learning system (GBML) which is capable of discovering tactics. There are two traditional approaches to GBML - at the rule-level, to search for useful rules thereby generating successful plans, and at the plan-level. Unfortunately, both approaches have inherent weaknesses. However, an alternative method which has the potential to eradicate those weaknesses is to hybridise the rule and plan-level approaches. This idea forms the outline for the main system design.

For the detailed system design, a bottom-up approach is adopted. Firstly, the rule discovery component of the system, the EA, is investigated. This investigation looks at aspects of an EA which are likely to be relevant to its effectiveness in a machine learning environment. These include Lamarckian operators and variable mutation rates, both of which are shown to significantly improve the effectiveness of the EA when compared to more traditional methods.

Secondly, credit assignment is investigated by analysing the effects that parallel rules, that is, rules which are active simultaneously, have on the relative strength of the rules in the extant rule base. A significant distortion in relative strength is found to result from the presence of such rules and two methods for combatting this distortion are demonstrated.

The results from these individual assessments are combined to form the detailed design for the hybrid genetic learning system, which also incorporates co-evolving agents to model the two combatants and a repair mechanism to alleviate problems of forgetfulness. The individual components of the system are all shown to be beneficial to the system but unfortunately, system performance is poor for all but the simplest problems. This is mainly due to problems encountered whilst learning at the rule-level, and in particular, problems associated with credit assignment.

List of Contents

List of Figures

Page No.

List of Tables

Page No.

List of Abbreviations

ABBA	Amended Bucket Brigade Algorithm
AG	Action Group
AHC	Adaptive Heuristic Critic
AI	Artificial Intelligence
AUV	Autonomous Underwater Vehicle
BA	Backward Averaging
BBA	Bucket Brigade Algorithm
CA	Collision Avoidance
C^2	Command and Control
CCS	Component Classifier System
CLP1	Combined Learning Problem 1
CP1	Classification Problem 1
DM	Diversity Mutation
DNA	Deoxyribonucleic acid
EA	Evolutionary Algorithm
EBL	Explanation-Based Learning
EGT	Evolutionary Game Theory
ES	Evolutionary Strategy
ESLR	Evolutionarily Stable Learning Rule
ESS	Evolutionarily Stable Strategy
GA	Genetic Algorithm
GBML	Genetics-Based Machine Learning
HCI	Human-Computer Interaction
HDCS	Hawk-Dove Classifier System
IC	Internal Condition
IL	Inductive Learning
ILU	Identifier Look-Up

IM	Internal Message
KB	Knowledge Base
KBS	Knowledge-Based System
ML	Machine Learning
OR	Operational Research
PLEA	Plan-Level Evolutionary Algorithm
PSP	Profit Sharing Plan
PSPV	Profit Sharing Plan with Variance
RaM	Rule and Message
RLEA	Rule-Level Evolutionary Algorithm
RPS	Relative Payoff Sum
RSIS	Remainder Stochastic Independent Sampling
SAGA	Strategy Analysis by Genetic Algorithm
SCS1	Simple Classifier System 1
SCS-EA	Simple Classifier System with Evolutionary Algorithm
SEA	Simple Epochal Algorithm
SP1	Sequential Problem 1
TTP	Two Tanks Problem
XOR	Exclusive-OR

Preface

This dissertation describes a considerable amount of original work aimed at producing a genetics-based machine learning (GBML) system for discovering military tactics. The work can be divided into four main subject areas:

(a) Evolutionary Algorithms - novel alternatives to current implementations are suggested and investigated using a standard Operational Research problem, the knapsack problem. The topics investigated include methods for maintaining capacity constraints, alternative crossover operators and non-standard mutation rates;

(b) credit assignment - problems caused by the presence of simultaneously active rules are investigated and a number of amendments to current techniques to alleviate these problems are proposed and investigated;

(c) classifier systems - a classifier system for a 2-player game scenario and incorporating a number of novel genetic operators is implemented and analysed using a meta-EA to determine useful operator combinations;

(d) a hybrid genetic learning system - a system combining the two traditional approaches to GBML, that is, the Michigan and Pittsburgh approaches, is implemented. This system includes a number of innovative features, of which two of the most important are co-evolving agents to model the two combatants, and a repair mechanism to alleviate problems of forgetfulness.

The work reported in this dissertation has contributed towards the production of 7 papers. The first of these [Fairley and Yates, 92], presented at the 34th Annual Conference of the Operational Research Society, gives experimental results from a test-EA when applied to the knapsack problem. The next three papers concern credit assignment in a simple classifier

system. The first two of these, [Fairley and Yates, 93], presented at an International Conference on Neural Nets and Genetic Algorithms, University of Innsbrück, and [Yates and Fairley, 93], presented at the 5th International Conference on Genetic Algorithms, University of Illinois, both describe results from experimental investigations into certain problems associated with credit assignment, whilst [Yates and Fairley, 94], presented at a Workshop on Evolutionary Computing, University of Leeds, describes the outcome of a theoretical investigation in credit assignment. Also presented at the same workshop was [Fairley and Yates, 94b] which describes some initial results from a hybrid genetic learning system. The remaining two papers have a greater military bias to them. The first of these, [Virr, et al, 93], published in the Journal of Naval Science, describes much of the background to this research, whilst [Fairley and Yates, 94a] gives a review of all the major applications of EAs to the development of military tactics, together with some directions which future work may take.

Acknowledgements

Many friends and colleagues have helped me throughout my work on this thesis. I am deeply indebted to them all. In particular, my sincere gratitude goes to my supervisor, Des Yates, for his patience and guidance throughout. I would also like to extend my thanks to Ray Paton and Colin Reeves, who have been unstinting in their advice and practical help. Moreover, at my sponsoring body, the Defence Research Agency, I would like to thank Len Virr, for instigating the work, and Adrian Brown, for being of immense help in the final months.

On a personal level, I would like to give a mention to my parents for their patience and great moral support throughout, and for her loving support and kindness, my fiancee, Susan, to whom this work is dedicated.

1. Introduction

1.1 Background to research

At any level of military combat, whether it involves the commander of a fleet or a pilot in a dog-fight, the underlying features of the tactical decision making process are essentially the same. The tactical decision making task can be divided into two main phases. The first, *data fusion*, involves assembling, in real-time, the information from all available sources (channels) to form an accurate *tactical picture* (world model) of the extant situation within the scope of the problem [Byrne, et al, 89]. The channels supply both 'real-time information' received by sensors, and 'non-real-time information', such as intelligence reports, geographical data and weather forecasts. The second phase, *situation assessment*, entails the intelligent appraisal of the tactical picture, and the consequent selection of an appropriate action. To facilitate selection, a tactical plan must be available. The process of tactical decision making, therefore, can be viewed as is depicted in Figure 1.1.

Unfortunately, this process is fraught with problems. These problems are mainly caused by the stress placed upon the human decision maker due to the gravity of the situation, and such stress can affect both his analysis of the tactical picture and his ability to correctly apply the tactical plan. Moreover, several perceptual decision making idiosyncrasies affect the process including self-fulfilling predictions, habit, over confidence (as well as a lack of confidence in some scenarios), gamblers fallacy and panic. As a result of these problems, considerable technological effort has been directed at the design of *Command and Control Systems* which attempt to ease some of the problems caused by the cognitive limitations of the human decision maker.

Command and Control (C^2) concerns '*the application of human intelligence to the management of resources in a dynamic (military) environment*' [p 1, Miles, 88], and systems designed to facilitate C^2 are amongst the most complex and large scale real-time resource

management systems known to man [Harris, 88]. Most practical C^2 systems are, however, only *action information systems* - they concentrate solely on the acquisition of information, data processing /communication and the physical *Human-Computer Interaction* (HCI) aspects whilst ignoring the decision making process itself. Thus, in general, tactical decision making remains the domain of the human expert in the scenario. Nonetheless, in recent years, a number of so-called *expert systems* such as BATTLE [Walker and Miller, 86] and BATES [Rackman, 90] have been developed in which an in-built knowledge base is employed to provide guidance for the human in respect of what action(s) should be taken. However, the knowledge (in the form of tactics) that is encapsulated within one of these expert systems must first be acquired.

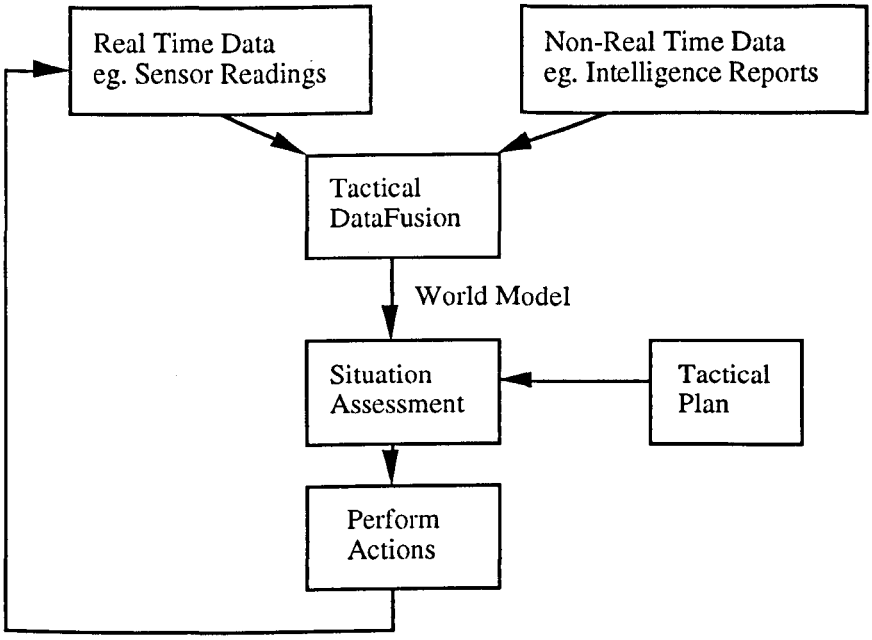


Figure 1.1: Tactical Decision Making

In the knowledge-based approach, a necessary prerequisite is possession of the relevant knowledge. [It should be noted that this contrasts with the so-called *behaviour-based approach* [Manderick, 91] in which patterns of behaviour emerge from the way in which simple

behaviours interact in complex ways.] Traditionally, knowledge acquisition has been the province of the knowledge engineer who, by means of verbal exchange with experts in a given field, would elicit the facts and heuristics used by those experts for solving problems. This process, *transference of knowledge*, is subject to two main problems whose effects have led [Lenat, 83] to refer to the process as 'the bottleneck of knowledge acquisition'. The first problem is the difficulty experienced by an expert in articulating what he/she knows, and the second is the impedance caused by the mismatch between the concepts and vocabulary of the expert and those of the knowledge engineer. There are two possible approaches to solving these problems. The first involves widening the channel between the expert and the knowledge base by, for example, instituting an appropriate natural language interface. Unfortunately, with this approach, the expert still has difficulty in verbalising his heuristics - the informal judgemental rules which guide him. This difficulty is especially acute for problems in such fields as combat management since, in the absence of any substantial underlying theory, many of the rules described by an expert are likely to rely on the expert's subjective opinions. The second approach entails eliminating both the expert and the knowledge engineer completely, and investing the expert system with the ability to 'discover' the required knowledge for itself.

In this thesis, this approach, that of designing a system to automatically generate knowledge bases, is investigated. As this type of research falls into the category of *Artificial Intelligence* (AI), it is appropriate to begin by defining some of the approaches to this problem which have been adopted within AI.

1.2 Artificial Intelligence for Command and Control

Artificial Intelligence concerns understanding and creating human facilities which are regarded as intelligent, and since C² involves the application of human intelligence to the management of resources in a dynamic environment, AI is an appropriate vehicle for investigating the processes involved within C². Since intelligence itself is not well defined, the term AI is not

well defined either. However, there are a large number of researchers spread across many disciplines who consider themselves to be members of the AI community, and the two broad objectives of this community are:

(a) to model human intelligence and investigate how the human brain works. This area of research is known as *Cognitive Simulation* and examples of systems designed to perform this task include those reported in [Klahr, et al, 87];

(b) to design systems which assist or replace humans where they are too expensive, inefficient, slow or the task too dangerous.

Most work in AI (including that reported here) has concentrated on the second of these objectives[1] and typical problems for which systems have been designed to acquire knowledge include game playing [Winston, 77], natural language [Feigenbaum, 63] and learning [Kodratoff and Michalski, 90]. However, despite research in AI having been underway for nearly four decades, nobody has yet constructed a system which possesses 'intelligence'.

Early research in AI made little headway into the problem of designing systems which have the ability to automatically acquire knowledge, and consequently two methodologies were established as potential 'shortcuts' to building such systems. These were:

(a) design a system to understand natural languages such as English, and then "feed" the system with knowledge from sources such as books and human experts. This approach, known as *Understanding Natural Language*, is more radical than simply widening the interface between the expert and the knowledge base since it avoids the need for an expert altogether;

[1]It is the belief of many authors (see, for example, [Miles, 88]) that by concentrating on the design of systems which can replace humans, the first objective, that of discovering how the human brain works, may be attained as a by-product.

(b) design a system which has the ability to *learn* solutions to a wide variety of problems and then place it in a 'problem environment' where it would be hoped that the system would learn solutions and thus, acquire knowledge. This approach is referred to as *Machine Learning*.

A small amount of progress has been made with machines for understanding natural languages and a number of primitive natural language interfaces (see, for example, [Bolc, 80]) have been designed, but these are far from being practical means for generating intelligence. Even less progress has been made with respect to Machine Learning, but in recent years, the field has enjoyed a significant revival due to advances in computing technology permitting the use of previously unexploited techniques. For example, little was made of the early work in linear perceptrons because of theoretical limitations, but in the past decade or so, this work has resurface as connectionist networks with hidden units able to compute and learn non-linear functions.

This resurgence of activity has led to a wide range of machine learning techniques being developed, and different authors have suggested a number of ways to categorize these, including:

(a) the 'techniques perspective' on learning - for example, is the technique used to improve the performance of an existing system, acquire knowledge directly from its environment, or build theories to describe and explain complex phenomena;

(b) its knowledge representation scheme;

(c) its learning paradigm.

In this work, it is appropriate to use the most common of these methods of categorization,

namely classifying systems by their learning paradigm, since the group of techniques that are to be investigated fit neatly into a single category.

Four main categories of learning paradigm exist [Carbonell, 89], each of which employs a very distinct approach to learning[2]. These are:

(a) Analytic Learning

When presented with a problem, an analytic learning system recalls problems that it has encountered previously, and whose characteristics bear a strong resemblance to those of the problem at hand. This triggers retrieval of behaviour that was either appropriate for solving, or known to be appropriate for solving, the 'past problems'. The behaviour, thus retrieved, is then adapted to meet the demands of the new problem. Analytic learning techniques are appropriate when a rich underlying domain theory for the problem type is available together with a number of exemplars (perhaps only one) each consisting of a problem instance and its respective solution. Examples of such techniques include *Explanation Based Learning* (EBL) [Gervasio and DeJong, 89] and multi-level chunking [Rosenbloom and Newell, 87];

(b) Inductive Learning

Inductive Learning techniques require an external teacher or environment to generate sequences of problem instances. The system uses its encapsulated knowledge to try to solve each problem instance in turn. After each attempt, the teacher/environment assesses the attempt as being either successful or not, as the case may be. The assessment is then used by the system to update its knowledge in such as way as to enable it to reproduce all, but only, those problem solving episodes which have previously been deemed successful;

[2]A number of categorization schemes could have ben adopted but the one given here has been chosen because the group of techniques that are the main focus of this thesis fit neatly into a single category in this scheme.

(c) Selectionist Learning[3]

Techniques in this category invest a system with learning capabilities by using simulations of processes that occur in nature as agents in the learning process. For example, inasmuch as genetic processes are instrumental in the evolution of species which are better suited to living in their allotted environments, *Genetic Algorithms*, [Holland, 75], and *Evolution Strategies*, [Schwefel, 75], which seek to simulate the underlying natural genetic processes, attempt to evolve solutions that are more apt for a specific problem. Again, *Immune Networks* [Farmer et al, 86] [Perelson, 89] [Bersini and Varela, 90], which mimic the human immune system, attempt by analogy to make use of the immune systems's 'pattern recognition capabilities';

(d) Connectionist Learning

Techniques in this category are founded on models of the structure of neurons in the human brain. Neural Networks, which are typically adopted in the connectionist learning approach, employ a network (in the graph theoretic sense, see [Boffey, 84], for example) to model a set of interconnected neurons. Given a sequence of training examples, a learning algorithm, such as *back propagation*, can then be used to adjust the weights in the network, so that the network will recognize (distinguish by its output) various classes of pattern supplied as input. Other techniques in this category involve, for example, the use of Hopfield nets, [Hopfield, 82], and Boltzmann machines, [Hinton and Sejnowski, 86].

With regard to military combat scenarios, there is no body of theory that dictates the manner in which the conflict can be successfully concluded for either combatant, that is, the *domain*

[3]Although [Carbonell, 89] uses the term 'Genetic Algorithms' to cover this learning category, it is more appropriate to use the term 'Selectionist Learning' [Manderick, 91] as this also encompasses Evolution Strategies (ESs), immune networks and the like.

theory is weak. Moreover, although some people do have experience in certain conflict situations, such experiential incidents are likely to be of limited value since detailed information about the incidents is unlikely to be available, and such information as exists would tend to be subjective rather than objective. Therefore, as far as C2 systems are concerned, these two facts effectively preclude the use of the analytic learning paradigm. However, they also indicate that the examples used by the machine learning technique that is ultimately adopted will almost certainly be simulated examples, and thereby, suggest that an inductive learning technique might be appropriate. Nonetheless, one of the requirements of an efficacious tactical plan is that it be *up-to-date* - that it should take account, whenever possible, of all relevant features of the extant real world. New weaponry is constantly being evolved and to keep pace with this evolution, tactical plans, if they are to be effective, must similarly evolve. Genetic Algorithms and Evolution Strategies (the generic term for these is *Evolutionary Algorithms* (EAs)) are particularly adept at discovering (generating) solutions to problems, and supported by an appropriate infrastructure to facilitate such discovery (see later), appear to be most appropriate vehicles for use in modifying and updating tactical plans.

Therefore, this research aims to explore the application of Evolutionary Algorithms to the field of Machine Learning. A number of paradigms for so-called *Genetics-Based Machine Learning* (GBML)[4] exist, including *Classifier Systems* (see [Holland, 86b] [Holland, 87], for example), which encapsulate the capabilities of both inductive and selectionist learning into a single learning system, and *Genetic Programming* [Koza, 91] [De Garis, 90]. However, before these approaches can be described and their relative merits discussed, it is necessary to look at the cornerstone of these approaches, an Evolutionary Algorithm, and the research that has been invested in this ever-increasingly popular topic.

[4]The reference to 'genetic' in GBML is a result of the original work, and much of the current work in the field, being based on GAs as opposed to alternative forms of EA.

1.3 An introduction to Evolution

Evolutionary Algorithms constitute a group of heuristic search methods that attempt to mimic the processes that underlie evolution. In order to understand the mechanisms that exist within a EA and why they are necessary, it is first useful to have some understanding of how evolution occurs in nature. This, at first may appear to be a futile exercise since most people have a clear perception of what evolution is, and how it works. However, despite being one of the most powerful theories science has ever known, it is also one of the most misunderstood and so this section is designed to give a cursory introduction to the subject and to lay some common myths to rest. Moreover, this discussion will introduce many of the important terms that are used both in this thesis, and by the EA community at large.

In order to describe the processes that occur at a macro-evolutionary level, it is first necessary to understand the genetic make-up of a living organism. Every organism is a mixture of characteristics which are determined by the *genes* in its *chromosomes*. Genes lie at specific positions or *loci* along a chromosome, with the different versions that a gene can take being called *alleles*. Each gene produces differences in the set of characteristics associated with an entity. For example, for certain types of pea, a single gene determines blossom colour. Moreover, groups of genes can jointly determine a given characteristic, as is demonstrated by eye colour in humans which is determined by a pair of genes. Together the structure of the genes forms the *genotype*, the result of which, in terms of its behavioral and physiological properties, is called the *phenotype*. This information is summarised in Figure 1.2.

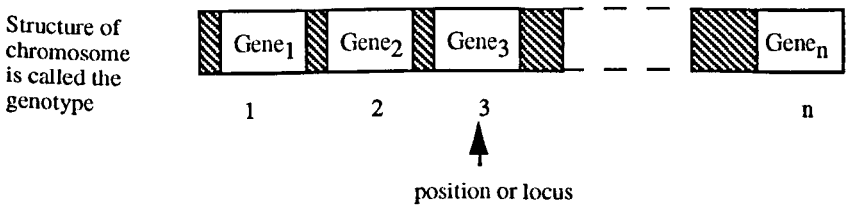


Figure 1.2: Structure of a chromosome

Genes are composed of *DNA* - sequences of *nucleotides* - which are themselves made up of *nucleic acids*, the basic genetic unit. There are 4 kinds of nucleic acid: adenine (A), guanine (G), cytosine (C) and thymine (T) and all living organisms use this same "genetic alphabet" of A, G, C and T. However, there is a fair amount of redundancy in this genetic code. For example, triplets of nucleotides form codons. Thus, there are 64 (4^3) possible codons but only 20 or so amino acids and so a number amino acids are coded by several codons. Moreover, the genes in a chromosome do not form a continuous string.

Moving up from the genetic level, the level of the *organism* is reached, and subsequently, the *population* (species) level. It is at this population level that evolution is most commonly observed, although it should be noted that evolution can also occur at the genetic and organism levels. However, the levels of selection remain a burning issue within theoretical biology.

Evolution is defined as being a change in the gene pool of a population over time, where the term 'gene pool' is used to represent the set of all genes in a species or population. This change may either qualitative, quantitative or both. To bring about this change, there must be evolutionary mechanisms present which can introduce (and also reduce) genetic variation in the gene pool. These mechanisms include:

(A) Natural Selection

Natural Selection (commonly referred to , somewhat inaccurately, as "survival of the fittest") is the only non-random mechanism of evolution and is the differential reproductive success of pre-existing classes of genetic variants in the gene pool. This means that genotypes which tend to be highly 'fit' (in an evolutionary sense) will, on average, be more represented in subsequent populations than those genotypes which are less 'fit'. Despite it being non-random, however, Natural Selection is not guided in any way either; it is simply an effect which allows populations of organisms to adapt to their environment;

(B) Genetic Recombination

Genetic Recombination is one of the mechanisms evolution employs to add new alleles and combinations of alleles to the gene pool, thus increasing the extant genetic variation. The most common recombination method is a process called *Crossover* (*meiosis*) which occurs when two organisms mate. In crossover, chromosomes from the two parent organisms are aligned and the DNA of the chromosome broken in several places. At these 'break' or *Crossing* points, strands from one chromosome are joined with the remaining section of the other chromosome (and vice versa), generating two new chromosomes, each with a mix of alleles from both parent chromosomes. A simplified version of this process is depicted in Figure 1.3. Moreover, crossover can occur within genes as well as between genes, thus giving rise to new allele values as well as new combinations of alleles;

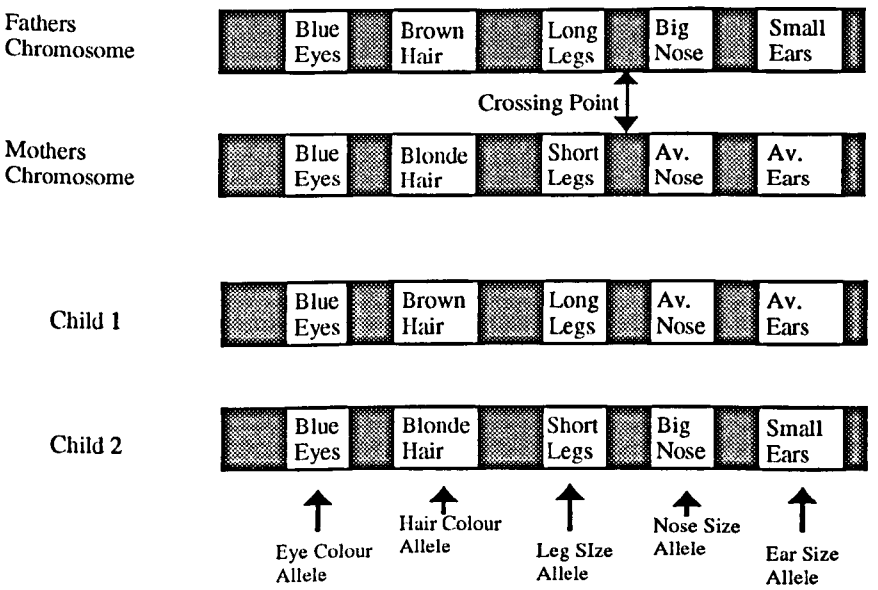


Figure 1.3: Genetic recombination - Crossover

(C) Mutation

Mutation is another evolutionary mechanism used to increase the genetic variation within a population. A mutation is a transformation of one piece of genetic material into another, and can occur at several levels within the genetic hierarchy. For example, entire genes, lengths of DNA, or simply single nucleotides can be mutated to add such variation into the population.

Given these three mechanisms, the evolutionary process is essentially very simple. A population of organisms interact with their environment. The relative prosperity of each individual in the environment determines the individuals *reproductive success* - a guide to the number of times an individual will mate and thus, the number of offspring it will produce. When individuals mate (that is, when genetic recombination takes place) offspring are produced which contain genes from both parents, thus making more likely the prospect of beneficial genes being represented in greater numbers in subsequent generations.

The classic example, often cited in the literature (see [Dobzhansky et al, 77], for example), of evolution working in practice is that of the pepper moth, *Biston betularia*, which was found near urban areas of England in the late 18th Century. Originally, the majority of these moths were lightly coloured to provide camouflage against its natural habitat, lichen; the remainder of the moths being dark in colour. However, with the advent of the Industrial Revolution, exhaust materials such as soot from factories began to turn the lichen black. This had the effect that the moths which were lightly coloured became more susceptible to predation by birds and hence, their numbers sharply declined. However, the environmental change meant that the dark coloured moths were more suited to the new environment, and thus increased in number until eventually they, rather than the lightly coloured moths, were the more common variety.

Unfortunately, evolution is not as simple as this example would appear to show, and is instead subject to several effects which can produce unexpected outcomes. These effects include:

(a) Genetic Drift

Genetic Drift is an evolutionary effect caused by a binomial sampling error during the selection process, and can result in the premature loss of a particular allele type from the gene pool. For example, if a gene in an organism has two possible allele types, α and β, and individuals with the α gene tend, on average, to be twice as successful as those possessing the β gene, then at any time during the evolutionary process, it would be expected that the ratio of α genes to β genes in a population of such organisms would be 2 to 1. However, it is unlikely that a given population will contain exactly two thirds α genes and one third β genes since there will be a 'sampling error' due to the finite population size. Subsequent generations will also tend to have α and β genes in the 2 to 1 ratio but variance can cause significant changes to this ratio over time. Should the selection process ever result in a population containing all α, then any future change would be impossible if selection and recombination (crossover) were the only evolutionary mechanisms. However, this is where the mutation operator is particularly useful since it is able to rectify the situation, through a chance event, by mutating one of the α's to a β. Nonetheless, it should be noted that mutation does not on its own prevent genetic drift from occurring - it merely has the ability to rectify the situation - and that the single β gene produced by the mutation is likely to be subsequently lost once more in genetic drift when further generations are produced. However, there will be occasions when mutations do survive into subsequent generations and regain their correct proportion in the gene pool. This is especially important if, for example, the environment were to change and the β genes to become preferable to the α genes;

(b) Hitch-hiking

The *hitch-hiking effect* occurs when a deleterious gene becomes closely linked to a beneficial one and increases its representation in the gene pool as a result of "riding on the coat tails" of the beneficial gene to which it is linked. The most common cause of this effect is, as a result of the linear ordering of chromosomes, genes which are far apart (that is at opposite ends of the chromosome) have a much larger chance of being separated when a recombination takes place than genes which are close together. Thus, genes which are located close to beneficial genes, may spread through the population regardless of their own utility, and the closer the linkage between the genes, the more significant the effect;

(c) Epistasis

Epistasis is a phenomenon which derives from the complex, non-linear way in which the genes in a chromosome interact. The result of this is that the effect of an allele can strongly depend upon what other alleles are present, and thus, it is possible that almost identical genotypes can have very different phenotypes [Spiessens, 88]. Because of epistasis, the primary aim of the evolutionary mechanisms described earlier is to generate co-adapted sets of alleles which combine well to significantly augment the fitness of the whole genotype;

(d) The Baldwin effect

The *Baldwin effect* [Hinton and Nowlan, 87] occurs when a phenotype adapts over the period of its lifetime (that is, it 'learns'), thus altering the fitness landscape of the corresponding genotype. Consequently, the selection process favours those genotypes which learnt over their lifetime and hence, the adaptations become genetically fixed.

It is pertinent to mention here an alternative theory on evolution which, at first, may appear to be the same as the Baldwin effect - namely *Lamarckism.* Lamarckism [Gorczynski and Steele, 81] is the name given to the ideas proposed by the French

biologist Jean Baptiste Lamarck (1744-1829) who suggested that characteristics acquired during the lifetime of an individual could be inherited by its offspring. The example Lamarck gave to support his hypothesis was that a blacksmith's son could inherit his father's strong, muscular frame, and so become a successful blacksmith himself. Unfortunately, there are many biological flaws in the Lamarckian theory and hence, it has been generally accepted as being non-feasible [Maynard-Smith, 89].

At first sight, Lamarckism and the Baldwin effect appear to be one and the same. However, this is not so since Lamarckian theory requires the adapted phenotype to change the genotype, whilst the Baldwin effect requires only the environment to change (to favour organisms which learn certain traits).

The success of evolution in developing the wide range of robust organisms that exist in the world today led researchers to investigate whether mechanisms such as Natural Selection and genetic recombination could be employed by artificial systems (that is, computer programs) to generate solutions to real-world problems.

One of the first conscious efforts to mimic evolutionary processes to generate optimum solutions was carried out by the German scientist Rechenberg in 1964 at the University of Berlin. This resulted in an approach called an *Evolution Strategy* (ES) [Schwefel, 75] which, in its simplest form, is a mutation-selection scheme. Not long after this, and independent of the work carried out in Berlin, researchers in the United States also started to investigate evolutionary computing schemes. One of the first techniques to emerge from this work was that of Fogel, Owens and Walsh [Fogel, *et al*, 66] which was used to perform simple symbol prediction tasks. Also around the same time, John Holland and his students at the University of Michigan were applying genetic-like operators to artificial problems in adaptation. However, it wasn't until the publication in 1975 of Holland's classic book, *Adaptation in Natural and Artificial Systems* [Holland, 75], that the research was exposed to a wider audience and the term *Genetic Algorithm* was first thrust upon the world at large.

1.4 Evolutionary Algorithms

1.4.1 Overview

In a strict interpretation, the term *genetic algorithm* (GA) refers specifically to the model proposed in [Holland, 75]. However, within the context of this thesis, a more general term will be used, namely that of an *Evolutionary Algorithm*. An EA is any population-based computational model that employs selection and recombination mechanisms to generate new solutions to the problem at hand.

For an EA to be applied to a particular problem, two elements are essential:

(a) An encoding scheme

This provides a representation (encoding) of every possible solution to the problem and is analogous to a chromosome in natural systems. Typically, an encoding is a binary string although many other types of encoding have been used within the EA community including Lisp programs [Koza, 91], semantic networks [Forrest, 91] and real-number strings [Bäck, 91]. However, it should be noted that no one technique works best for all problems, and the most suitable representation to employ will be dependent heavily upon the problem being investigated.

(b) An evaluation (fitness) function

This is analogous to the rôle of the environment in Natural Selection and assesses each solution in terms of its 'utility' or 'fitness'.

With these two elements in place, an EA proceeds by mimicking the application of the evolutionary processes described earlier to a population of solutions. This it achieves by iteratively performing the following sequence of operators (also depicted in Figure 1.4) until some appropriate terminating criterion is achieved.

(1) *Initialise*

Generate an initial population of solutions, P.

(2) *Evaluation*

Determine the fitness of each member of P using the fitness function.

(3) *Selection*

Select a subset of P to form a *mating pool*, M, using fitness as the underlying selection criterion; the greater the fitness of a member, the more likely it is to be selected.

(4) *Recombination*

Select pairs of solutions from M and *mate* them to produce an offspring population, O. The mating process includes applying a number of recombination mechanisms such as crossover and mutation, which mimic the natural genetic processes. The EA terminology for these recombination mechanisms is *genetic operators*.

(5) *Repeat*

Replace the original population, P, by the offspring population, O, and return to step (2). Upon termination, the best solution generated from amongst the various populations (generations) is adopted as the solution to the original problem.

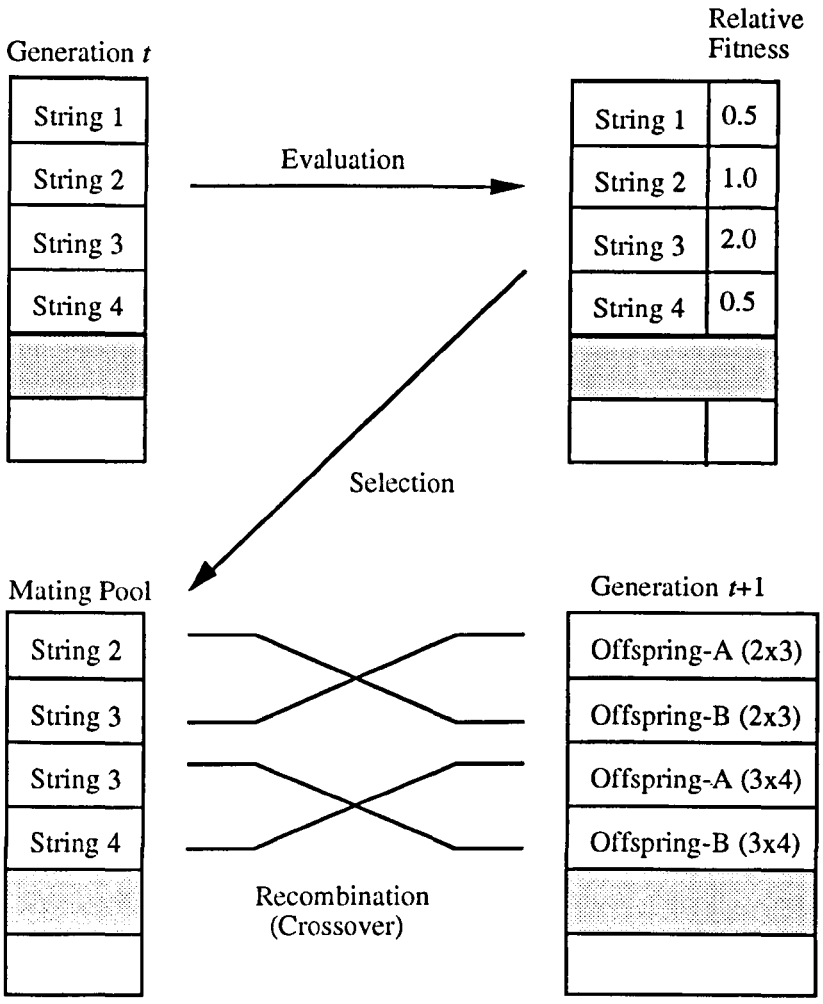


Figure 1.4: Main cycle of an Evolutionary Algorithm

Although this algorithmic description encompasses the majority of EAs, the varying background of many of the researchers in the field has meant that many EA models have been proposed. The following is a list of the most significant of these variants of the standard model.

(1) Initialisation Schemes

There are two basic approaches to generating the initial population (that is, generation

1). The most common method is to use a random number generator to create random solutions. This approach is useful because it is both simple and a good test of the robustness of the algorithm at solving the problem. However, where the search space is particularly large, it may take the algorithm a painstakingly long time to achieve an optimal solution. Moreover, particularly complex problems for which non-binary representations are typically used, are sometimes such that the initial solutions may not be good enough for evolution to even 'get off the ground'. In such circumstances, it will be preferable if any *a priori* knowledge can be incorporated into the algorithm by seeding the initial population with known solutions. A number of studies (see, for example, [Grefenstette, 87b]) have shown that this approach can improve the performance of the EA in many circumstances, but it can also have a detrimental effect in that the search performed by the EA will have an initial bias thereby making it more difficult for the EA to find an optimal solution.

(2) Reproduction Schemes

The EA outlined above is often referred to as a *traditional* or *generational* EA because of the way in which each generation is replaced on every iteration of the algorithm. Many variations of this basic algorithm exist but one of the most radical and widely studied is the *incremental* or *steady-state* EA. In this, on each iteration, only a small number of individuals (often only one) are replaced. Typically, during each iteration, two of the best solutions are mated with one of the resulting offspring replacing one of the poorer solutions in the population. Several comparisons of generational and incremental EAs have been performed (see, for example, [Fairley and Yates, 92] and [Whitley, 89]), with the general conclusions being that the incremental approach converges far quicker than a generational EA, but it is also more likely to converge to a sub-optimal solution.

(3) Selection Schemes

The degree to which the selection scheme favours the best solutions is known as the

selective pressure and is critical to the success of the algorithm. The difficulty in choosing an appropriate selection scheme is known as the *exploitation-exploration problem* [Goldberg, 89a] since an EA is looking to exploit the best solutions in its search for better solutions, whilst simultaneously trying to do the opposite and maintain diversity within the population. There exists a variety of selection algorithms, of which the two most common are:

(a) roulette wheel (fitness proportionate) selection;

(b) rank-based selection.

In the traditional EA outlined previously, members of the population are selected in proportion to their fitness relative to the rest of the population. This is known as *Roulette wheel* selection. [Baker, 87] proposed a number of alternative roulette wheel schemes and demonstrated that implementation details can significantly affect their performance. For example, if the probability of selecting an individual (that is, its relative fitness divided by the population size) remains constant throughout the selection process, then significant deviations from the expected number of selections for a particular individual can occur.

The main drawback to roulette wheel selection is that it is afflicted by difficulties of scaling. For example, in problems where the pay-off function is sparse (that is, few of the encodings correspond to valid solutions), it is easy for a single 'super-individual', which has a fitness well-above the average, to rapidly dominate a population and thus cause premature convergence. To overcome this problem, [Baker, 85] suggested a *ranking* approach in which a population is ordered according to fitness and selection probabilities assigned to each member which are inversely proportional to rank. There are two usual approaches to relating selection probability to rank. In the first, *linear ranking*, the best and worst individuals are assigned typical fitnesses of 2 and 0 respectively, with all intermediate individuals being assigned a fitness which is

interpolated within this range. The alternative to this is exponential ranking and in this, each individual is assigned a fitness of $(1- \delta)^{R-1}$ where δ is a small value (typically 0.01) and R is the rank of the individual.

However, as with most issues associated with EAs, the choice of selection scheme is not clear cut. Several trials (see, for example, [Hancock, 94] and [Whitley, 89]) have shown that exponential ranking tends to produce superior results, whereas trials with problems which have noisy evaluation functions have shown that roulette wheel selection may be more appropriate [Fogarty, 93].

(4) Genetic Operators

Arguably the most important aspect of an EA's operation is the discovery of new, high utility solutions. This task falls to the genetic operators, and in particular, the crossover operator, of which there are a number of possible implementations. A crossover operator which selects a single crossing point about which to exchange information is referred to as a *1-point* crossover operator [Holland, 75]. In the search for improvements to the standard EA, this operator was quickly joined by more general versions, the *2-point* and *n-point* crossover operators [Goldberg, 89a]. Unfortunately, all of these operators suffer from problems of *positional bias* - the distance between two bits affects their chances of remaining together [Eshelmen, et al, 89]. To overcome this problem, Syswerda [Syswerda, 89] proposed an alternative operator, *uniform crossover*, in which each bit in an offspring has an equal chance of coming from either parent, thereby reducing the positional bias. Trials recorded in [Syswerda, 89] showed that uniform crossover can generally outperform both 1-point and 2-point crossover, although once again, there are a number of occasions where it will be preferable to use 1-point or 2-point crossover (see [Fairley and Yates, 92] for a more in-depth discussion of the issues).

In recent years, more interest has been focussed on recombination operators for

encoding schemes wherein the simple forms of crossover detailed above produce *lethal* offspring (that is, offspring which do not encode a valid solution). One such problem which has received a great deal of interest in the EA community is the Travelling Salesman Problem (TSP) [Boffey, 84] and a number of operators have been specifically designed for the various encoding schemes associated with this.

1.4.2 How EAs work - the Schema Theorem

To facilitate a theoretical analysis of an EA, [Holland, 75], introduced the important concept of a *schema* (plural, *schemata*). A schema is a subset of A, the set of all possible string configurations (solutions) which have a number of common alleles. For example, given the two strings:

0 1 0 1 0

1 1 0 0 0

then an example of a schema common to both strings is *10*0 (where the symbol * represents either 0 or 1), whilst 01*** is only represented in the first string.

Since each string can contain either the actual value or a * at any position, then it can hold up to 2^n distinct schemata, where n is the length of the string. Therefore, whenever a solution is evaluated, information is received relating to all 2^n schemata contained therein. Scaling up, it is clear that for a population size of M, as many as $M2^n$ schemata may be evaluated per generation, although analysis [Goldberg, 89a] has shown this more likely to be $O(M^3)$ due to overlapping schemata. It is this ability of an EA to process a large number of schemata which is the key to its success and it was deemed so important that Holland gave it a special name - *Intrinsic Parallelism*[5].

The success of an EA is therefore based around its ability to (a) sample a large number of schemata using crossover and mutation, and (b) employ selection to increase or decrease the number of instances of each schemata in proportion to its relative fitness. This is described in

[5]Also referred to as *Implicit Parallelism*

the following theorem which describes the propagation of schemata from one generation to the next and has proved to be the cornerstone of EA theory.

Given the following terms:

(a) l(S), the length of schema S (the number of positions between the first and last non-* positions);

(b) k(S), the order of schema S (the number of non-* positions in S);

(c) N(S, *t*), the number of instances of schema S at time *t*;

(d) f(S, *t*), the fitness ratio of schema S at time *t* (ie. the ratio of the average fitness of S to the average population fitness);

(e) P[S,*t*], the probability that schema S will be represented in the population at time *t*,

then The *Schema Theorem*[6] states that:

Given an EA with probabilities of employing crossover and mutation of P_c *and* P_m *respectively, the expected number of representatives of schema S at time t+1, E(S, t+1), is given by*:

$$E(S,t+1) \geq \left\{1 - \frac{P_c l(s)}{n-1}(1 - P[S,t]) - P_m k(S)\right\} f(S,t) N(S,t)$$

The main consequence of this theorem is that, provided the fitness ratio is greater than 1, then short, low-order schemata are favoured over long, high-order schemata with the same fitness ratio. Thus, it may be concluded that an EA is likely to construct solutions by combining short, low-order schemata together to build a solution in a hierarchical manner. This idea has had such a significant impact on EA theory that it was given a special name by [Goldberg, 89a] - the *building-block hypothesis*.

---

[6]The version of the Schema Theorem given here is taken from [Reeves, 93]. As Reeves notes, this statement of the theorem is slightly different from that sometimes found in that the left-hand-side is explicitly stated in terms of *expectation*, a fact which is often insufficiently stressed.

1.4.3 Recent advances in EA theory

Since the original work by Holland, several tools for theoretically analysing an EA have been proposed. One of the most important of these was *Walsh functions* [Beauchamp, 75]. In one of their first applications to EAs, Walsh functions were employed by [Goldberg and Rudnick, 91] to produce an expression relating to the variance of schema fitness. However, in recent years, most attention has been paid to their application to the analysis of so-called *deceptive problems* [Goldberg, 87] where the building-block approach leads the EA away from an optimal point and towards sub-optimal solutions. An alternative method of analysing EAs which has received some attention of late is the model proposed by [Vose and Liepins, 91]. The motivation for this model was the failure of the schema theorem to account for the fact that the representation of schemata in a population is impossible to determine unless the composition of the population itself is known. To address this problem, Vose and Liepins used population vectors (as opposed to schemata) and Markov Models to analyse the operation of an EA in terms of moving between points in a 'population space' [Nix and Vose, 92]. This form of analysis is still in its infancy, but initial results using simple EAs [Vose, 92] have shown great promise.

1.4.4 Current research in EAs

The explosion in research into EAs which occurred in the late 1980's has led to a vast literature covering all aspects of EAs from applications to theory. However, to even touch upon this work would take up more space then can be accommodated in this thesis. Instead, this section gives a brief review of some of the directions which research in EAs is currently taking. These directions are defined by the so-called 'Hot Topics' which have emerged in recent years and initial investigation of which show promise. Four of the most significant of these are:

(1) Self Adapting Mechanisms

One of the problems central to the use of EAs is that many of the control parameters such as operator rates, population size and the representation itself, have to be

determined by the analyst at the outset. In the absence of strong predictive theories which specify suitable values for such parameters, this process is ad hoc and tends to rely upon the subjective view of the analyst. However, poor choice in parameter settings can lead to dramatically low system performance. For example, a mutation rate which is too high can be too disruptive to the EA's search and lead to the search degenerating into little better than a random walk. Conversely, too small a mutation rate can lead to 'lost allele values' being regenerated too infrequently. Unfortunately, parameter optimisation is not a straightforward task because many parameters are interdependent and the relationship between them is complex. For example, an appropriate mutation rate may depend upon a number of factors including string length, population size and crossover rate. As it is not uncommon for an EA to contain in excess of 10 to 20 such parameters, each of which may have a large number of plausible values, trial and error optimisation is therefore out of the question.

One method for determining parameter settings which has been used successfully within the EA community is that of using a *meta-EA* [Grefenstette, 86] [Freisleben and Härtfelder, 93] [Mercer and Sampson, 87]. In a meta-EA, the parameters of the original EA are encoded in string form; a group of bits representing each particular parameter. The meta-EA maintains a population of such strings and finds the fitness of each of these by running a test-EA for a number of generations and using the fitness of the best member found during that run as its fitness. The meta-EA then recombines the best settings and repeats the process for a number of generations. The main drawback to this approach is the time it requires. Even a simple EA may take several minutes to evaluate a particular parameter setting and consequently, it may take of the order of hours to evaluate a single generation of settings. Therefore, for many problems, this is clearly an unsuitable method.

An alternative approach which appears to be much more apt is the inclusion of 'self-adapting mechanisms' in the genetic operators themselves. Such mechanisms can use the state of the extant system or its current performance to adapt the operation

of its genetic operators to suit the situation. For example, [Bäck, 92], based on his work with ESs, investigated the use of an adaptive mutation rate and demonstrated that environment-dependent self-adaptation of appropriate settings for the mutation rate is possible in EAs.

(2) Use of Lamarckian Operators

The Lamarckian theory on inheritance of adapted phenotypes was discussed briefly in Section 1.2 where it was mentioned that the theory had been largely discarded by biologists. However, despite its uncertain biological foundations, the theory has proved useful in EA research wherein system designers can implement features that nature cannot, thereby enhancing the learning process. A number of such Lamarckian mechanisms have been implemented including Grefenstette's use of a hybrid approach [Grefenstette, 87c], wherein individuals go through a learning stage with associated representational changes before their fitness is evaluated, and Davidor's variable crossover probability [Davidor, 91], wherein the operator usage rate varies according to phenotypic performance.

(3) Use of Parallel Architectures

EAs have a natural inherent parallelism, and this fact has led to a large number of projects concentrating on the implementation of EAs on both fine-grained and course-grained parallel computers. Parallelisation has an obvious advantage in that the speed for execution of an EA can be dramatically increased thereby enabling more complex EAs to be run for more generations on harder problems. However, there is also another, less obvious benefit, namely that, by splitting the population into a number of individual sub-populations (the term used by biologists is *demes*), and only occasionally allowing migration between these sub-populations, the population can maintain greater diversity and hence avoid problems associated with premature convergence. Again, research into the use of parallel architectures is still in its early

stages with most of the work being concentrated on the use of improved architectures in order to increase speed. Whilst research which takes account of the development of faster, more efficient architectures is clearly an important part of this, there remains a great deal to be discovered by further investigations of the evolutionary effects of parallel architectures.

(4) Co-evolutionary Systems

Systems which involve two or more interacting populations aimed at evolving solutions to different problems have been a very recent but exciting development in the EA community. One of the first people to develop such a system was Hillis [Hillis, 91] when he implemented an EA to optimise sorting networks [Bose and Nelson, 62]. Hillis found that when he used a traditional EA approach to this problem, evaluating a solution was difficult because, unless each solution was evaluated over every possible problem, after only a few generations, most individuals could successfully solve most problems and consequently, there was insufficient selective pressure to gain improvement. To overcome this problem, Hillis implemented a second population which consisted of problems (ie. sequences) which the host population had to sort. By judging each of these problems on how many networks they managed to find faults in, it was observed that, as the host population evolved towards better sorting networks, the problem or *parasite* population co-evolved to produce sequences which provided a better test set for the improved networks.

Hillis's work has opened up a vast number of opportunities for exploiting the properties of co-evolving systems with more recent work on the subject including that of [Husbands and Mill, 91] which investigated co-evolutionary mechanisms for planning and scheduling, [Angeline and Pollack, 93], which investigated the use of fitness functions that are dependent on the constituents of the population, and [Ikegami and Kaneko, 91]. However, it remains to be seen whether they can have a significant impact upon the rest of EA research.

1.5 Genetics-Based Machine Learning

1.5.1 Overview

As mentioned earlier, the particular application of EAs which is investigated in this thesis is that of the development of military tactics. This can be considered as a machine learning problem and there has been a broad interest in the use of EAs in such problems for a considerable time, so much so that a separate research field, known as Genetics-Based Machine Learning (GBML) has developed which focusses solely on this topic.

The theoretical foundation for GBML systems was laid by Holland [Holland, 68] and resulted in the development of the first functional GBML system - the *classifier system* CS-1 [Holland and Reitman, 78]. Classifier systems are highly parallel, message passing, rule-based systems and are the most common vehicle for the implementation of EAs in a machine learning environment.

1.5.2 Classifier Systems

1.5.2.1 Knowledge Representation

Classifier systems encode their knowledge using one of the most common representation schemes used in AI - *Production Rules*. Production Rules have two parts, both of which are represented as lists. The first part, the *conditional part* or *antecedent*, consists of a series of conditions, whilst the second part, the *action part* or *consequent*, consists of one or more actions which are to be performed if and when the conditions in the conditional part are satisfied. Thus a rule has the following form:

Rule: IF (AND condition_1, condition_2,..., condition_n) THEN PERFORM (action_1,.., action_m)

In the context of a military problem, each condition in such a rule will relate to specific information acquired in the data fusion phase, such as :*distance to target, speed of target, target bearing*; and each action specifies a 'real' action that should be taken, of which: 'increase speed

5 knots', and 'turn to bearing 270°' might be examples. Any plan can be represented as a set of such rules whose conditions cover all of the possible situations that may occur during combat. Such a rule set is called a *Knowledge Base* (KB).

Within a classifier system, the production rules and the KB are referred to as *classifiers* and the *classifier store* respectively. Each classifier is a fixed length string of k, say, characters. The first $n<k$ characters of a classifier form the conditional part with the remaining $m = k-n$ characters forming a *message* which indicates the action(s) to be performed when the conditional part is satisfied. Also, associated with each classifier is a *strength*, a value which indicates the usefulness of the classifier in the given environment.

Messages are the basic tokens of information exchange within a classifier system. Each character in a message is one from a finite alphabet, A. Typically, A is chosen to be the binary alphabet $\{0,1\}$. The antecedent of a classifier is a simple pattern recognition device whose underlying alphabet is A together with a wildcard[7] character, #, which matches any character in A. For example, if A is $\{0,1\}$, then the antecedent 01#0 will match both 0100 and 0110.

Three main components constitute a classifier system, namely:

(1) a rule and message system;

(2) a credit assignment scheme;

(3) an evolutionary algorithm.

In essence, the rule and message system interfaces with a representation of the extant environment - the *world model*. It accepts information from the world model (via the input interface), it utilises this information in combination with the rule base to select the appropriate action, and it effects the action (via the output interface) thereby causing the world model to be updated. There is no learning inherent in this component of the classifier system. Learning is achieved by *credit assignment*. It is the credit assignment scheme which distinguishes between good and bad rules, thereby enabling the rule and message system to *learn* which actions are appropriate in which circumstances. The EA component is the mechanism by which rules in

[7]In the remainder of this thesis, the term 'wildcard' will be used to represent the wildcard character, '#'.

the KB can be replaced by new and improved rules.

A slightly more detailed schematic representation of a classifier system is given in Figure 1.5. The boxes in the diagram labelled 'Credit Assignment' and 'Rule Discovery' correspond respectively to the credit assignment scheme and the evolutionary algorithm.

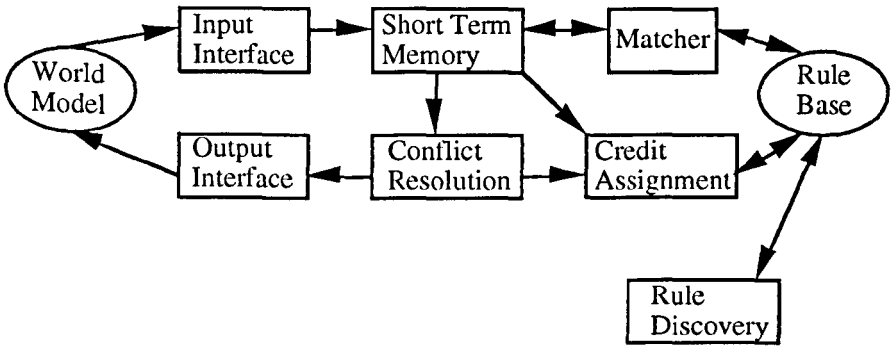


Figure 1.5: Main cycle of a classifier system

1.5.2.2 The Rule and Message System

During an *episode* (the time period spanning a conflict situation), the rule and message (RaM) system is triggered each time a change, not induced by the RaM system itself, is detected in the world model. When triggered (and this includes the occasion of the inception of an episode), the RaM performs the sequence of operations given below.

(1) The current state of the world model is translated by an *input interface* into a special message called the *environmental* or *detector* message. The message consists of m characters, and each character or group of characters, is an encoding of some relevant property of the world model.

(2) The environmental message is passed to the *message list*, where it is stored. The message list is a section of short-term memory capable of holding a small number, $P>1$ say, of messages. In addition to the environmental message, the message list may also

hold the messages corresponding to the most recently 'active' of the classifiers (see (4) below).

(3) Each message held in the message list is compared with the conditional part of each and every classifier in the classifier store. When a message matches the conditional part of a given classifier, that classifier becomes a candidate to send its message to a newly constructed message list.

(4) If there are P or fewer candidate classifiers, each classifier passes its message to the message list thereby updating it. If, however, there are $C>P$ candidate classifiers, a problem arises: the message list is too small to hold the messages from all C classifiers. Although several methods do exist for resolving the problem, the most common is to require the C classifiers to enter into an 'auction'. In the auction, each of the classifiers 'makes a bid', the value of which is a function of the classifier's strength, and those classifiers which make the P highest bids send their messages to the message list. The classifiers which are successful in passing their messages to the message list are said to be 'active'.

(5) Steps (3) and (4) are repeated iteratively until the messages in the message list indicate that actions should be performed to update the world model (that is, one or more suggested courses of action have been derived in response to the externally 'triggered' change in the world model). If the courses of action suggested by the messages accord, the appropriate action(s) are effected, and the world model correspondingly updated. Otherwise, some mechanism for resolving the conflict is called into play. Although a number of different approaches exist, the most common is that of holding an auction between the competing courses of action. The size of the bid made by a competitor is typically calculated as a function of the strengths of the classifiers which advocate it; the highest bid dictating the course of action ultimately taken to update the world model.

1.5.2.3 Credit Assignment

The problem of determining (learning) an appropriate plan of action to achieve some desired goal in a tactical situation is made more difficult by what is called the *Credit Assignment Problem.* Consider an episode involving a long and complex sequence of decisions. The problem of accurately and definitively labelling each decision in the episode as being either good or bad defines the credit assignment problem. An example which illustrates the problem well is that of a game of chess which 'black' loses after 50 or 60 moves. It is quite possible that only a handful of black's moves were poor, but isolating them precisely in a non-trivial task.

An approach which attempts to solve the credit assignment problem for classifier system is provided by the *Bucket Brigade Algorithm* (BBA) [Holland, 85]. Essential to the operation of this algorithm is an *external critic* which, at the end of an episode, judges the level of performance of the plan employed with respect to the outcome of the episode, and determines a corresponding pay-off, P say, which takes the form of a real number. The algorithm assigns credit to classifiers that are active during an episode in the following way:

(1) each time a classifier, C, succeeds in its bid to send its message to the message list (C becomes active), C's strength is reduced by C_{bid}, a function of the value of the bid made by C in the auction. The value C_{bid}/n is then added to the strength of each of the n classifiers, active on the previous iteration, whose messages match C's antecedent. Thus, C may be viewed as rewarding those classifiers that were active on the previous iteration which were instrumental in giving it the chance to become active.

(2) the value P/A is added to the strength of each of the A classifiers that were active at the end of the episode.

When the BBA is employed in just a single episode, the credit assigned to all but the classifiers that were active at the end of the episode may not be wholly appropriate to the outcome of the

episode. However, when employed in a sequence of episodes, the pay-offs awarded by the external critic will ultimately filter back to the *stage setting* classifiers, awarding successful classifiers in line with successful episodic outcomes, and *vice versa*. The algorithm therefore facilitates the *learning* of successful rules by reinforcement.

1.5.2.4 Rule discovery using Evolutionary Algorithms

The rules which comprise a KB can be viewed as a population to which an EA, utilising the strength of individual rules to guide its search, can be applied. An EA used in a classifier system strongly resembles those described earlier. However, one important point which is raised by [Holland, 86a] is that in situations for which the classifier system has a well-practiced, effective set of classifiers, only the worst classifiers should ever be replaced. Holland called this concept *gracefulness* and since an incremental EA is more suited to this type of operation, then it is predominantly used with respect to classifier systems.

Classifier systems of the form detailed above have been implemented for a number of machine learning problems including the multiplexor problem [Wilson, 87a] [Quinlan, 88], letter recognition [Frey and Slate, 91], gas pipeline control [Goldberg, 83] and the ANIMAT problem [Booker, 82] [Wilson, 87a] [Wilson, 87b]. Moreover, classifier systems have been successfully implemented to run on a parallel computer [Robertson, 87].

1.5.2.5 Current techniques and research in Classifier Systems

Research in the field of classifier systems can be divided into three categories corresponding to the three constituent components of a classifier system, namely system architecture (RaM), credit assignment and the EA.

System architecture has probably been the least investigated aspect of classifier systems and few extensions have been proposed. An exception is the proposal made by Forrest [Forrest, 85] in her Doctoral thesis for which she implemented a compiler to translate a relatively complex, high-level language, KL-ONE, used for representing semantic networks,

into the standard classifier form described earlier. By successfully achieving this mapping, Forrest was able to show that classifier systems can emulate the more complex models traditionally used in AI. Other researchers have also investigated the connection between classifier systems and other AI learning paradigms. For example, both [Spiessens, 88] and [Belew and Gherrity, 89] have compared classifier systems with connectionist architectures, whilst [Davis, 88, 89b] actually provides detailed procedures for converting a classifier system into a neural network, and vice versa. A classifier system/neural network hybrid has also been investigated by [Ball, 91, 93] and shown to be effective for a simple control problem. Another particularly interesting extension to the basic system architecture was also proposed by [Zhou, 90] in which a long-term memory was incorporated into his system to avoid problems wherein the system deletes previously useful rules.

In contrast to system architecture, credit assignment has received a great deal of interest from within the EA community, especially with respect to deficiencies in the BBA. One of the most widely studied limitations of the BBA is the difficulty it has in propagating strength backwards from the reward state to 'stage setting' classifiers when the sequence of classifiers is long. Results produced by Wilson [87c] and Riolo [89a] suggest that the number of iterations necessary to reinforce the first classifier in a chain of length n is of the order of $10n$, which is impractical for many problems. Several solutions have been suggested to alleviate this problem. One of the first solutions suggested was to employ *bridging classifiers* [Holland, 85]. A bridging classifier is a classifier which remains active over a number of time steps and therefore, by continuously paying and receiving credit to and from itself, it facilitates the transfer of strength to stage setting classifiers. [Riolo, 89a] demonstrated that manually injected bridging classifiers could help alleviate the problem of long chains but no situation has been found wherein they can develop spontaneously. An alternative approach to the same problem is that of a *hierarchical* credit assignment scheme [Wilson, 87c] [Wilson and Goldberg, 89] [Bruderer and Shevoroshkin, 93], in which, by placing classifiers in a hierarchy, bucket brigade chains can be kept short, thus facilitating the propagation of strength. This concept was implemented by [Shu and Scheaffer, 91] by grouping classifiers into

'families' and higher-order structures such as 'communities of families'. Initial results with this system were encouraging but no further work on the subject has been reported to date.

Another aspect of credit assignment that has received a great deal of attention is that of the ability of the BBA to support the formation of *default hierarchies* [Holland, 85] [Goldberg, 89a] - plans in which general classifiers (those with many #'s) cover the general conditions, whilst more specific classifiers cover the exceptions. [Riolo, 87b, 89b] showed that viable default hierarchies can be maintained once appropriate classifiers have been generated, and [Goldberg, 89a] showed that such structures can be formed through the application of the EA. However, the work of [Wilson, 89] appears to contradict this, showing that the bidding process suggested by Goldberg does not encourage the formation of default hierarchies. Moreover, Wilson went further by adding that the bidding process tends to favour more general rules (since they have less to pay out per activation than specific rules), resulting in a detrimental bias in the search of the EA. This effect was also observed by [Robertson and Riolo, 88].

As a result of the various problems associated with the BBA, a number of alternative credit assignment schemes have been proposed. Several of these belong to a class of so-called *epochal schemes* - schemes where a record of all the classifiers which have been active during an episode is maintained and at the end of the episode, each one is paid a constant fraction of the environmental reward. Examples of such schemes include the *epochal algorithm* [Holland and Reitmen, 78] and the *Profit Sharing Plan* (PSP) [Grefenstette, 88]. In his RUDI system, Grefenstette demonstrated that the PSP can learn the expected system reward more accurately than the BBA but is generally not as successful at measuring dynamic coupling between rules - a necessity if useful classifier chains are to be developed. [Liepins, et al, 89, 91] proposed another two alternative credit assignment schemes - Backward Averaging (BA) and an Adaptive Heuristic Critic (AHC) [Sutton, 88] - but results achieved with these two schemes showed little or no improvement over the BBA. Finally, it is worth mentioning another scheme which has received not insubstantial interest in recent years, a technique adapted from Q-learning and dynamic planning [Sutton, 90] [Watkins, 89]. Q-learning has many similarities

to the way in which the BBA operates but also takes into account extra information such as the expected future reward. Therefore, because a system using Q-learning has more information available, system performance should be expected to be better. However, results produced by [Roberts, 89] on an ANIMAT-type problem showed that results produced with Q-learning are, at best, only comparable with the BBA, and in many cases are in fact worse.

Due to the problems encountered with credit assignment, the rule discovery aspect of the system has not been widely studied since the EA cannot be analysed without the involvement of credit assignment. One of the few studies in this area was that of [Booker, 85] in which a number of modifications including a restricted mating policy and a *crowding algorithm* [Goldberg, 89a] were shown to be beneficial to the EA. [Holland, 87] also suggested a number of novel operators including a *triggered coupling* operator to link more tightly, pairs of rules which are used on consecutive time steps. This operator was implemented by [Robertson and Riolo, 88] for a letter sequencing problem with the results showing a significant improvement in system performance. However, these studies aside, the operators used by the majority of authors are simply those associated with the standard EA, namely crossover and mutation.

1.5.3 Other approaches to GBML

The approach to rule discovery which is adopted within a classifier system is referred to as the *Michigan approach*[8]. A few years after the Michigan approach was first proposed, an alternative implementation for GBML was suggested by [Smith, 80] which resulted in the development of his LS-1 system for playing the card game poker. Since then, the approach Smith adopted has been commonly referred to as the *Pittsburgh* or *Pitt approach*. Like the Michigan approach, it too uses production rules to represent knowledge. However, rather than each rule in a single rule base being an individual of the population to which the EA is applied, the Pitt method maintains a number of plans and it is a plan that is treated as a member of the

[8]Named after the University where the approach was pioneered.

EA population.

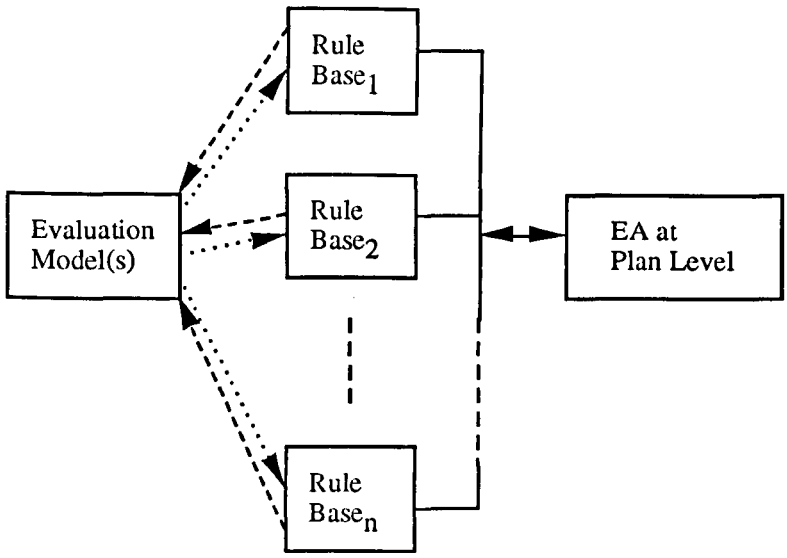
A system adopting the Pitt approach typically operates by running each of its plans a number of times on the simulation test model in much the same way as a classifier system runs its single plan. This is achieved either by running the plans on a number of different simulation models in parallel, or by running each plan, one at a time, on a single simulation model. However, a major difference between the Pitt and the Michigan approach is that rather than ascribing a utility value to each rule in a plan, the Pitt approach assigns a utility value to each plan, and by so doing by-passes the credit assignment problem of trying to discover which rules are most useful. Periodically, the EA is invoked which causes the most successful plans to be passed forward to a mating pool and genetic operators to be applied to select members of the pool, thereby producing a new population of plans (the next generation). Subsequently, this new population is tested and the cycle repeated until some criterion for termination is met. By operating in this manner, the EA is said to be working at the *plan level.* A schematic showing the operation of a system using the Pitt approach to GBML is shown in Figure 1.6.

The Pitt approach appears to present fewer problems for GBML than do classifier systems. However significant among those problems which have been described in the literature are:

(1) extensive computational resources are needed to evaluate the potentially large number of rule sets;

(2) a single fitness value (derived in some way from the fitness values of the constituent rules) is used to represent the quality of an entire plan;

(3) additional mechanisms are required to construct new rules in the various plans.



Key:

- - - ► Suggested actions passed from Rule Bases to Evaluation Model

···· ► Model returns utility of the action

——► EA at plan level generates next generation of Rule Bases

Figure 1.6: Pittsburgh Approach

The first of these problems can be addressed with the aid of parallel processing facilities; each plan being 'run' as a separate process. Whilst means for overcoming the other two problems do exist, their mere existence serves to indicate that, despite deficiencies in respect of rule co-operation, application of an EA at the rule level does have certain advantages over its application at the plan level.

Since the design of LS-1, there have been surprisingly few implementations of the Pitt approach. One of the few which systems which is described in the literature is the SAMUEL learning system [Grefenstette et al, 90] [Grefenstette, 91a] which is used for the development of military tactics - precisely the same problem domain that this work aims to investigate. SAMUEL has been shown to work reasonably well for a number of military problems

including the evasive manoeuvres problem [Gordon and Grefenstette, 90] [Schultz and Grefenstette, 90], the cat and mouse problem [Grefenstette and Ramsey, 92] and dogfighting [Grefenstette, 91b], although the problems are very simple in nature. Moreover, SAMUEL has been invested with other AI learning paradigms, such as Explanation-Based Learning [Gervasio and DeJong, 89] and Anytime Learning [Dean and Boddy, 88], and has also been extended to cope with multiagent environments [Grefenstette, 93] where several agents learn solutions simultaneously.

1.6 The proposed approach

The above discussion has shown that the Michigan and Pittsburgh approaches both have relative strengths and weaknesses. Therefore, it would appear that a *hybrid* system which attempts to capitalise on the strengths of both approaches by harnessing the two in a single system, may prove to be most advantageous. For example, the deficiency in the Pittsburgh approach of only having a single fitness value to represent the utility of an entire plan will be reduced by using rule strengths derived from the Michigan part of a hybrid system. Furthermore, the additional mechanisms required by the Pitt approach to construct new rules will not be needed if an EA at the rule level is also used. Undoubtedly, [pp 304, Goldberg, 89a] had this view in mind when he wrote "*we should encourage parallel experimentation with both types of systems [Michigan and Pittsburgh] because each has something to contribute to GBML understanding*".

Consider a set of plans made available to such a hybrid system. By associating one classifier system with each and every plan, and running the totality of classifier systems concurrently, the independent development of the plans is facilitated via application of the EA at the rule level. If, periodically, operation of the classifier systems is suspended and then interaction between the individual plans effected by applying the EA at the plan level, it may be possible to achieve the desired advantages of both approaches to EA application. Thus, a

classifier system employing dual application of EAs (at the rule and plan levels) is proposed as the most appropriate way for the machine learning of military tactics. An outline schematic representation of this hybrid architecture is given in Figure 1.7.

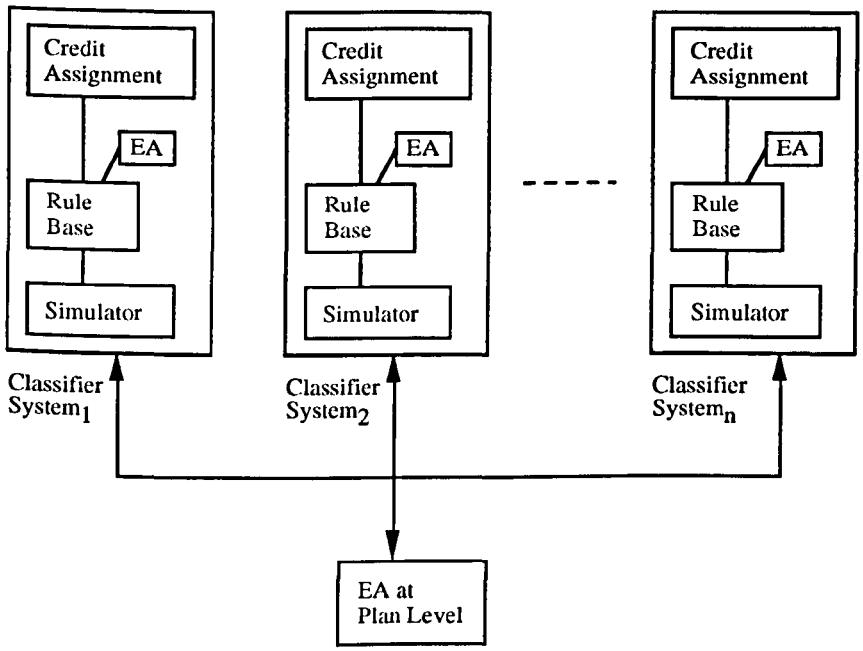


Figure 1.7: The proposed hybrid architecture

The aim of the remainder of this research is to develop, investigate and implement such an architecture. Although the outline architecture for the system has been decided upon, virtually all aspects of the detailed system design, including knowledge representation, genetic operators and credit assignment, remain to be resolved. In this research, it is intended that all these implementation details be investigated by experimenting with alternative approaches to the various issues and determining which is/are 'best'.

1.7 Design strategy

To support the approach to the research which is described above, the system is to be designed and built in a step-wise, bottom-up fashion in which the lowest level units are first designed and constructed, and then subsequent layers are added, one at a time, until the whole system is complete. A suitable decomposition of the system architecture for the purpose of a bottom-up design is shown in Figure 1.8.

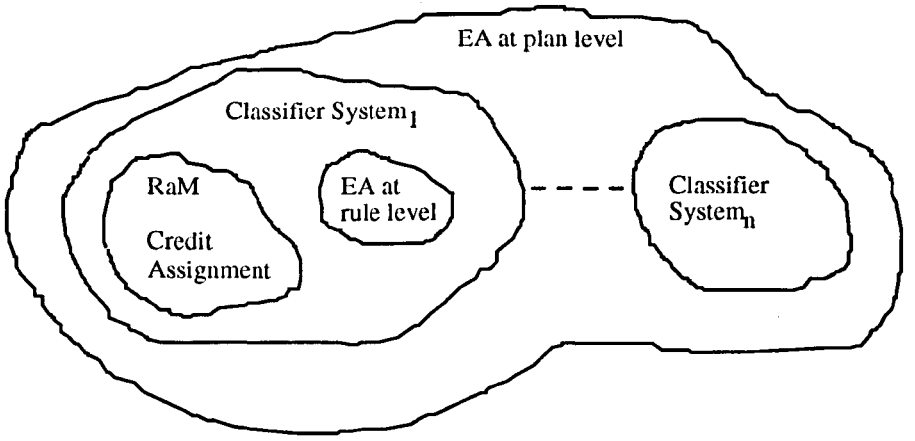


Figure 1.8: Levels of the proposed system

At the heart of the system is a series of classifier systems, each of which can be broken down into 3 component parts; the rule and message (RaM) system, the credit assignment algorithm and the rule-level evolutionary algorithm (RLEA) Using a simple test problem, the RLEA can be tested independently of the rest of the system to investigate possible improvements to its effectiveness. Moreover, a module combining both the RaM and the credit assignment algorithm can also be investigated independent of the other parts of the system. Moving up a level, the two components can then be combined to form a complete classifier system which can then be analysed to determine if the findings of the investigations into its component parts are maintained when combined. Subsequent to this, a series of such classifier systems can be

constructed and connected together via an EA at the plan level (PLEA), thus resulting in the desired system architecture.

1.8 About this document

The contents of the remainder of this thesis are as follows. Chapter 2 describes the implementation of a simple EA used to investigate aspects which are important in a machine learning environment. Chapter 3 consists of two studies investigating credit assignment in a simple classifier system. In the first study, a classifier system is analysed from a theoretical perspective to determine the sort of results that a classifier system should produce, whilst in the second study, a simple classifier system is implemented and the aspects of the credit assignment algorithm which most affect system performance investigated. Chapter 4 then adds the EA investigated in chapter 2 to the simple classifier system of chapter 3 to investigate whether such a system is capable of learning suitable rules for real-world military problems. Chapter 5 then details the final system design, whilst chapter 6 describes the experimental results derived from this system. Finally, chapter 7 describes the conclusions from this research and discusses those directions which future work in this area might take.

2 Implementation of a simple Evolutionary Algorithm

2.1 Introduction

In the previous chapter, it was decided that a step-wise, bottom-up approach would be adopted for the design of the proposed hybrid GBML system. One of the main components of this system is the evolutionary algorithm (EA) and it is the aim of this chapter to investigate a number of the key aspects which affect its effectiveness. This will be achieved by comparing the schemes which are generally used in EA applications with some novel alternatives to see if the effectiveness of the algorithm can be improved. In particular, since the primary use of the EA in the intended system is to discover useful rules as part of a classifier system architecture, the features investigated here relate to aspects which may be significant to the rule discovery process.

Section 2.2 introduces the test problem upon which the investigation will be based - the *knapsack problem* - and defines the precise form of the problem used for the trials. Although not usually specified by many authors, a number of the small design decisions associated with the EA can have a significant bearing upon its effectiveness, and so section 2.3 is used to give a detailed description of the algorithm used in the trials, as well as defining the measures of effectiveness which are employed to ascertain the relative merits of the alternative schemes. Section 2.4 then gives the details of the alternative schemes proposed in this study, together with the results from the experiments used to evaluate them.

2.2 Definition of the test problem

As previously stated, this thesis aims to investigate a GBML system for the development of military tactics. Thus, the most appropriate vehicle to use to analyse the EA part of a GBML

system would be a tactical problem itself. Unfortunately, such a problem would require other features of a GBML system to be present (the credit assignment algorithm, for example) and these would unduly complicate any analysis of the EA. Moreover, any conclusions would be system specific and may not necessarily hold should the detailed design of the proposed system change.

Therefore, to investigate the EA, it was decided that a simpler test problem was required, and preferably one which requires as little problem specific knowledge as possible. Two such problems suggest themselves:

(a) a function optimisation task;

(b) the knapsack problem.

Function optimisation is the traditional way to investigate features of an EA and a number of test functions have been developed to serve this purpose (see, for example, [De Jong, 75]). However, because the payoff returned from standard function optimisation tasks is always fixed, it is not possible to investigate one of the factors that plays a key role in a GBML system, namely the effect of the quality of the payoff on the performance of the algorithm. The knapsack problem, however, does include a mechanism by which the effect of environmental payoff can be investigated, namely the requirement for some means of enforcing capacity constraints. For example, by increasing the value of the penalty in the knapsack evaluation function, the effect of sparse payoff information on the performance of the EA can be determined.

In addition to this, the knapsack problem has a particularly simple encoding scheme in which solutions map very naturally as binary strings. Thus, the problems that tend to occur with, say, function optimization where some binary to decimal encoding scheme is required, are avoided. Moreover, unlike other OR problems such as the Travelling Salesman Problem [Michalewicz, 92], no problem specific operators are required. However, if they are to be investigated, problem specific operators can easily be implemented for the knapsack problem,

thus making it very suitable for comparing the effectiveness of problem specific and standard operators, such as crossover.

Finally, the knapsack problem is also useful because:

(a) it is well understood but has a non-trivial solution (that is, it is NP-hard);

(b) several investigations of EAs have already been performed using the knapsack problem (see, for example, [Goldberg and Smith, 87] [Goldberg, 89a] and [Syswerda, 89]) and the operators / mechanisms analysed therein (eg. Uniform Crossover [Syswerda, 89]) can be used to extend the analysis performed here.

Thus, because of its generic nature, it was decided that the knapsack problem was a very appropriate option for analysing a number of the features of an EA which will effect a GBML system.

In the problem, a set of n items is available to be packed into a knapsack. However, there are N capacity constraints, that limit the number of items which the knapsack can hold. Each item i, i=1..n, has a value v_i and uses up w_{ij} units in respect of the jth capacity constraint, j=1,...,N. The objective of the problem is to determine the subset I of items which should be packed in order to maximise the value of the contents of the knapsack without violating any of the capacity constraints.

This is written mathematically as:

$$\max \sum_{i=1}^{n} v_i x_i$$

such that

$$\sum_{i=1}^{n} w_{ij} x_i \leq K_j, j = 1..N$$

where

$$x_i = \begin{cases} 1 \text{ if item } i \text{ is included} \\ 0 \text{ otherwise} \end{cases}$$

K_j is the capacity associated with constraint j

This problem is a particularly appropriate application to which an EA can be applied because of the way in which solutions can be encoded very naturally as binary strings of length n, where bit i represents the ith item and is set to 1 if i is included in the solution and 0 otherwise.

2.3 Test Regime

As [Cohoon et al, 87] point out, the basic EA approach is easily understood but there is no canonical "pseudo-code" version in published accounts and hence, almost all EA implementations are different. Although seemingly unimportant in themselves, the small, "obvious" design decisions which are not usually specified in reports by many authors, can have a major effect on how successfully an EA 'solves' a problem and hence influence any results which are reported. For this reason, the following is a complete description of the EA used in this chapter in respect of the knapsack problem.

Each trial consists of running the EA on a population, S, of size *pop_size* until a number of knapsack evaluations, *num_evals*, have been performed, where *num_evals* is some multiple of *pop_size* (so that the number of generations is an integer), and every *pop_size* evaluations represents a generation. At the start of a trial the population is initialized randomly using a random number generator and then the main EA cycle commences. In this, each member, S_i, of the population is evaluated; any constraint violation being subjected to a penalty function as described in Experiment 1 (section 2.4). The relative fitness of each population member is then calculated and used by the selection algorithm to select a mating pool, M, also

of size *pop_size*.

The selection algorithm used is the *Remainder Stochastic Independent Sampling* (RSIS) scheme suggested by [Baker, 87]. Crossover is then employed in respect of the members, M_i, of the mating pool, thus selected, with a probability P_c (the crossover rate), of application. In this, M_i and another string S_j, selected at random from S, mate using 1-point crossover, leading to two offspring, C_1 and C_2, from which one is selected at random and used to replace M_i in the mating pool (the other child is discarded). If crossover is not applied to M_i, and this occurs with a probability of 1-P_c, then M_i remains unchanged. The mutation operator is then applied to every member of the mating pool by inverting each of its bits with a probability P_{mut} (the mutation rate). The resulting mating pool then becomes the next generation and the cycle repeated.

Each experiment or trial reported in this chapter involved 10,000 problem evaluations with various performance measures (see below) being recorded. Each trial was repeated 10 times with different initial populations on a testbed of 5 knapsack problems (outlined in Table 2.1 and detailed in Appendix 1) of varying degrees of difficulty.

Table 2.1: Test bed of knapsack problems

Problem Number	Number of Items	Number of Constraints	Search Space
1	15	10	32768
2	20	10	1.0 x 10e6
3	28	10	2.7 x 10e8
4	39	5	5.5 x 10e11
5	50	5	1.1 x 10e15

As research into the applications of EAs has grown, several performance measures [De Jong, 75] [Goldberg, 89a] have been developed in order to evaluate EA performance. The three most common of these are the *on-line*, *off-line* and *best-so-far* measures.

(i) *On-line performance*

This is a statistic devised by De Jong in his dissertation [De Jong, 75] and is used to gauge ongoing performance. It is calculated as follows:

$$\omega^{online}(T) = \frac{1}{T}\sum_{t=1}^{T}\omega(t)$$

where $\omega^{online}(T)$ is the on-line performance after T function evaluations and $\omega(t)$ is the function value on trial t.

(ii) *Off-line performance*

This is another of De Jong's measures and is used to gauge convergence among a population. It is calculated by:

$$\omega^{offline}(T) = \frac{1}{T}\sum_{t=1}^{T}\omega^{*}(t)$$

where $\omega^{*}(t)$ is the best $\{ \omega(1), \omega(2), .. , \omega(t) \}$

(iii) *Best-so-far*

This measure is self-explanitory and is the usual measure of performance when evaluating any iterative heuristic technique.

The three statistics were recorded during each of the 10 trials, and of these, the best and the average for each measure was found.

2.4 Experimental Results

2.4.1 Default Settings

The experiments reported here investigate elements of each of the three constituent components of an EA, namely Evaluation, Selection and Recombination. Although a great number of features could have been chosen for investigation, five were deemed to be particularly useful because they investigate aspects which may prove significant to the rule discovery process in a GBML system.

Firstly, as identified in Section 2.2, it is important to investigate the effect of the quality of the payoff on the performance on the algorithm. Consequently, Experiment 1 investigates alternAtive methods for maintaining the knapsack capacity constraints. Another aspect likely to be important because of the need to maintain a high level of on-line performance in a GBML system is the selection of partners for mating. This is investigated in Experiment 2. Finally, the use of tailored versions of the standard genetic operations and their rates of operation are both likely to be important in discovering appropriate new rules for the proposed system and these are investigated in Experiments 3, 4 and 5.

Thus, the five experiments reported in theis chapter are:

(1) maintenance of capacity constraints in the knapsack problem (evaluation);

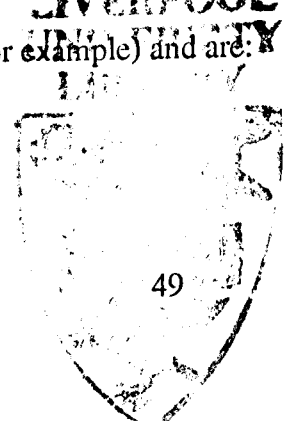
(2) selection of partners for mating (selection);

(3) alternative crossover operators (reproduction);

(4) variable mutation rates (reproduction);

(5) other genetic operators (reproduction).

The default settings for the three main parameters were chosen to be largely in line with those in general use ([Goldberg, 89a], for example) and are:



Population Size:	50
Crossover Rate:	0.75
Mutation Rate :	0.001

2.4.2 Experiment 1: An investigation into the maintenance of capacity constraints

One difficulty associated with applying EAs to constrained optimisation problems, such as the knapsack problem, is dealing with infeasible solutions. There are two basic approaches to this problem in general use. One is to modify the genetic operators so that only feasible solutions are produced. A simple operator of this type for the knapsack problem could involve removing items (that is, changing 1's to 0's) at random from infeasible solutions until all constraints are satisfied, whilst a more complex operator could remove the items with, say, the least value relative to the weights of the constraints, until all constraints are satisfied. The second approach is to introduce a penalty function into the cost function so that an infeasible solution is penalised and its value degraded. Despite the penalty function being critical to the EA's search capability, its design in the past has tended to be *ad hoc*. In early attempts at solving constrained optimisation problems using EAs, the penalty was made harsh so that the EA would avoid the infeasible regions of the solution space. However, when such a scheme is employed, the EA receives little information from the infeasible solutions upon which to base its search. For example, it may be that a number of invalid solutions exist for which a single mutation will produce not only a valid solution, but an optimal solution, and yet, despite this, a 'harsh' penalty function will give the EA misleading search information, and make it difficult for the EA to find the optimal solution. Conversely, the penalty function must not be 'over-tolerant' since infeasible solutions possessing a sufficiently large initial utility will maintain a high utility after application of the penalty with the consequence that the population may end up being dominated by a large number of high-fitness, infeasible solutions.

A utility function for the knapsack problem which employs a simple penalty function is shown below. The penalty function involves the product of a constant penalty coefficient, P,

and a function of the error terms, e_j.

$$\max \sum_{i=1}^{n} v_i x_i - P \sum_{j=1}^{N} e_j^2$$

where

$$ej = \begin{cases} K_j - \sum_{i=1}^{n} w_{ij} x_i \text{ if } \sum_{i=1}^{n} w_{ij} x_i > K_j \\ 0 \text{ otherwise} \end{cases}$$

However, even for simple penalty functions like this, selecting the value of the penalty coefficient is difficult and, more often than not, to be on the safe side, the value will be made large, thus making the penalty function harsh. However, as [Richardson, et al, 89] note, graded (variable) penalties are better than constant harsh ones. Such a variable penalty scheme is the lagrangian relaxation heuristic used by [Klincewicz and Luss, 86], on the capacitated facility location problem with single-source constraints. This entails calculating a Lagrangian multiplier, λ_j, for each constraint in the problem as follows:

$$\lambda_j = \text{Max}\left(\lambda_j - \frac{\alpha(V_{upper} - V_{lower})e_j}{\left(\sum_{j=1}^{N} e_j^2\right)^{\frac{1}{2}}},\ 0 \right)$$

where α is a positive scalar (0.25 had previously been used with success),

V_{upper} is the best known upper bound on the solution, found by

$$V_{max} = \sum_{i=1}^{n} v_i$$

V_{lower} is the best known lower bound, which is set to the best feasible solution to date.

A third method of dealing with constraints, and one not previously used in connection with

EAs, is that of alternating the evaluation function every g generations between optimising the value of a solution regardless of constraint violations, and minimising the constraint violations regardless of the objective function's value. By doing this, it would be hoped that only feasible, high strength solutions will survive over a number of generations.

In this experiment four methods of maintaining the capacity constraint were compared. These were:

(i) incorporation of a genetic operator into the evaluation function to remove items from an invalid solution until it is valid. The items to be removed were selected by removing those with least relative value where

$$Relative\ Value_i = \frac{v_i}{\sum_{j=1}^{n} w_{ij}}$$

(ii) employing a constant penalty function of the form:

$$V_i = \sum_{i=1}^{n} v_i x_i - 1000 \sum_{j=1}^{N} e_j^2$$

where V is the fitness of a solution and e_j is the error term;

(iii) using a variable penalty function with the penalty coefficients found by using the Klincewicz and Luss method, and α set to 0.25;

(iv) alternating the evaluation function between optimising the solution value and minimising constraint violations. A range of values of g was tried with the results presented here being those derived for the period that was generally found to work best, namely alternating every generation (ie. g=1).

The results derived from performing the aforementioned trials using, in turn, each of these four

schemes are shown in Tables 2.2 to 2.4, where the figures in parenthesis represent the percentage difference from the highest value of the four methods.

Table 2.2: Experiment 1 - Best of Best-so-far

Problem No.	1	2	3	4	5	Av. error
Fixed Penalty Coefficient	4015 (0.0)	6110 (0.2)	12270 (1.1)	10336 (0.4)	15946 (1.1)	0.6
Variable Penalty Coefficient	4015 (0.0)	5990 (2.2)	12310 (0.7)	10347 (0.3)	16117 (0.0)	0.6
Oscillating Evaluation Fn.	4015 (0.0)	6090 (0.5)	12190 (1.7)	10378 (0.0)	15770 (2.2)	0.9
'Item Removal' Operator	4015 (0.0)	6120 (0.0)	12400 (0.0)	10285 (0.9)	16020 (0.6)	0.3

Table 2.3: Experiment 1 - Best On-line

Problem No.	1	2	3	4	5	Av. error
Fixed Penalty Coefficient	3668 (1.3)	5152 (11.1)	10581 (10.7)	9085 (0.5)	13624 (8.1)	6.3
Variable Penalty Coefficient	3717 (0.0)	5120 (11.8)	10503 (11.5)	8928 (2.3)	14092 (4.5)	6
Oscillating Evaluation Fn.	1699 (118.8)	2606 (119.7)	4934 (137.4)	4310 (111.9)	6549 (124.8)	122.5
'Item Removal' Operator	3672 (1.2)	5726 (0.0)	11715 (0.0)	9134 (0.0)	14724 (0.0)	0.2

Table 2.4: Experiment 1 - Best Off-line

Problem No.	1	2	3	4	5	Av. error
Fixed Penalty Coefficient	4003 (0.2)	6012 (1.7)	12080 (2.0)	10215 (0.0)	15676 (1.8)	1.1
Variable Penalty Coefficient	3987 (0.6)	5965 (2.5)	12130 (1.5)	10177 (0.4)	15955 (0.0)	1
Oscillating Evaluation Fn.	3980 (0.8)	5988 (2.1)	11939 (3.2)	10131 (0.8)	15646 (2.0)	1.8
'Item Removal' Operator	4012 (0.0)	6116 (0.0)	12316 (0.0)	10100 (1.1)	15874 (0.5)	0.3

The results show that for all three performance measures, incorporating problem specific knowledge in the form of an operator to correct invalid solutions, namely the 'Item Removal'

operator, can significantly improve performance.

The results also show that there is little difference between the use of constant and variable penalty functions - the former producing slightly better on-line performance and the latter producing a slightly better off-line performance.

The reason for this is as follows. The harsh fixed penalty keeps the EA away from the infeasible region of the search space by severely penalising any infeasible solutions. On the other hand, the variable penalty function allows infeasible solutions, especially those close to the infeasible/feasible border, to gain some credit and therefore, these solutions have a chance to affect further generations. Consequently, the population will tend to have a greater number of infeasible solutions and so on-line performance will suffer. However, the areas searched by the EA with this added information from infeasible solutions should be more fruitful and therefore produce better solutions and lead to an increase in off-line performance.

For all performance measures, the alternating evaluation function scheme generally produced the poorest results. This is due to a number of reasons, the main one being that the alternating function does not allow useful building blocks to be gradually developed. Instead, a building block which is useful under one of the functions may be replaced on a subsequent generation by an inferior building block which is better for the alternative evaluation function.

2.4.3 Experiment 2: An investigation into the selection of partners for mating

In the EA used for these trials, a mating pool is generated in the reproduction stage and it is from this that one of the two parent strings for each crossover is selected, the second parent being selected randomly from the original population. An alternative to this which is also in common use is to select the second parent, as well as the first, from the mating pool, thus giving a greater bias towards the exploitation of useful solutions. Somewhat surprisingly however, there appears to have been little work into any other schemes besides these two for the selection of the second parent. In this section, two alternatives for selecting the second parent, both of which use ideas from partner selection in nature, are suggested and investigated.

The first alternative scheme is derived from the common dominance hierarchy set up within herds of animals such as mountain sheep, in which the "best" male is found through a series of competitions (such as the ramming of heads until the opponent submits). The animal which proves dominant then has the first rights to any reproductively active female. This *herd leader* approach can be implemented easily in an EA by always mating the first parent with the generation's current best member. Another novel approach which is similar to a cross between the herd leader and ranking methods, is based on a rare method of sexual selection found in several unrelated species called the *lek* [Gould, 82]. In a lek community, males initially gather to form an integrated matrix of tiny individual territories which are painstakingly acquired and militantly defended. Subsequently, mating success is strongly biased toward the one male in the centre, which generally performs 50-75% of the copulations. The immediate neighbours perform the remaining matings, while any other males do not mate at all. Within the context of this study, this scheme is implemented by performing herd leader mating for 50-75% of the time and on the other occasions, mating takes place with either the second or third best members of the population (equal probability of either occurring).

Four trials were performed (using the same test problems as in Experiment 1) in which the second parent in every mating was, in turn, selected as follows:

(a) randomly from the original population;

(b) stochastically from the original population using relative fitness as the guide for selection;

(c) using the herd leader strategy;

(d) using the lek method.

The results from these trails are shown in Tables 2.5 to 2.7.

Table 2.5: Experiment 2 - Best of Best-so-far

Problem No.	1	2	3	4	5	Av. error
Random	4015 (0.0)	6110 (0.0)	12270 (0.4)	10336 (1.7)	15946 (1.3)	0.68
Relative Fitness	4015 (0.0)	6100 (0.2)	12170 (1.2)	10418 (0.9)	15962 (1.2)	0.7
Herd Leader	4015 (0.0)	5920 (3.2)	12320 (0.0)	10449 (0.6)	16155 (0.0)	0.76
Lek Method	4005 (0.2)	6090 (0.3)	11770 (4.7)	10507 (0.0)	15865 (1.8)	1.4

Table 2.6: Experiment 2 - Best On-line

Problem No.	1	2	3	4	5	Av. error
Random	3724 (7.0)	5187 (14.6)	10637 (13.3)	9193 (11.7)	13817 (13.6)	12.0
Relative Fitness	3863 (3.2)	5693 (4.4)	10996 (9.6)	9374 (9.5)	14211 (10.5)	7.4
Herd Leader	3988 (0.0)	5651 (5.2)	12047 (0.0)	10201 (0.6)	15701 (0.0)	1.2
Lek Method	3922 (1.7)	5946 (0.0)	11441 (5.3)	10267 (0.0)	15415 (0.9)	1.8

Table 2.7: Experiment 2 - Best Off-line

Problem No.	1	2	3	4	5	Av. error
Random	4003 (0.2)	6012 (0.8)	12080 (1.7)	10215 (2.2)	15676 (2.6)	1.5
Relative Fitness	4009 (0.1)	6061 (0.0)	11998 (2.4)	10353 (0.8)	15844 (1.5)	1.0
Herd Leader	4013 (0.0)	5771 (5.1)	12288 (0.0)	10398 (0.4)	16079 (0.0)	1.1
Lek Method	3997 (0.4)	6063 (0.0)	11696 (5.1)	10439 (0.0)	15795 (1.8)	1.5

Table 2.5 shows that for the best-so-far performance measure, there is little to choose between methods a, b and c. The lek method appears to perform rather badly compared to the others but this may be largely attributed to the relatively poor results it produced for problem 3 - otherwise, its performance was largely in line with the other three methods. Not surprisingly, similar conclusions are also reached when gauging performance by the off-line results (Table

2.7). However, the on-line performance figures (Table 2.6) paint a very different picture with the two alternative suggestions, namely the herd leader and lek methods, both demonstrating far superior performance to the two standard schemes - this being largely a result of the exploitative rather than explorative nature of the herd leader and lek methods.

Thus, the results have shown that for the problems used in these trials, significant gains in on-line performance were gained from adopting alternative selections schemes, without detriment to the off-line performance of the EA.

2.4.4 Experiment 3: An investigation into alternative crossover operators

Within the EA community, a great deal of attention has been paid to the design of alternative genetic operators which improve the effectiveness of an EA and since crossover is the most important genetic operator, this has been the focus of much of the interest.

As mentioned in Chapter 1, probably the most promising variant of the standard 1-point crossover to have been suggested is Uniform crossover [Syswerda, 89]. In some cases though, a simple operator such as 1-point crossover is preferable. Such a situation occurs when the bits of a problem encoding are ordered so that related bits are physically grouped together, as in Grefenstette's *Clustering* operator [Grefenstette, 87c] where cooperative rules are set together to facilitate rule discovery in a classifier system. In such a situation, 1-point crossover is likely to leave favourable values together whilst uniform crossover would most likely destroy favourable groupings. However, one problem with 1-point crossover is that on some occasions, the crossing point may be selected such that the two segments exchanged between the parents are identical, thereby producing offspring which are no different from their parents. For example, given the two parents below, if a crossing point of less than 8 or greater than 20 is chosen, then the offspring are identical to the parents and the operation is said to be 'wasted'.

Parent A: 00101110100100111010100101

Parent B: 00101110001011011010100101

One approach for overcoming this problem which is suggested and investigated here is to use what will be referred to as the *string-of-change* method[7]. In this, the Exclusive-OR (XOR) operation is applied to the two parent strings producing a third string, s. If there are fewer than two non-zero digits (that is, '1's') in s, then irrespective of where the crossing point p is chosen, the two offspring will be identical to the parents. However, if s contains two or more non-zero digits, then a crossing point selected between the first and last non-zero digits will ensure that the offspring are different to their parents.

Proof:

If p is selected to the left of the first '1' in s, then all bits in parent 1 to the left of p must be identical to the equivalent bits in parent 2. Although it is traditional within EA applications to exchange the sections to the right of the crossing point, exactly the same offspring are produced by exchanging the sections to the left. Therefore, the crossover is effectively exchanging two identical sections and so fails to produce new offspring. Similarly, if p is selected after the last '1' in s, then the crossover will also fail to produce any new offspring. However, if p is selected between the first and last non-zero digits in s, then there must be at least one bit different between either of the sections to be exchanged, and so 'new' offspring must be produced.

For example, given the two parents A and B, s would be

000000001011111001100000000

and so a crossing point selected between the ninth and nineteenth positions will result in offspring which are different to both A and B. The following analysis [Fairley and Yates, 92] shows that for strings of length greater than 10, this modification reduces the probability of a 'wasted' crossover to virtually zero.

[7] Since performing the work reported here on the string-of-change method, earlier work by Booker and reported in [Booker, 87] on a similar approach, the *reduced surrogate* method, has been discovered.

Given two parent strings S and T, of length l, and a randomly selected crossing point, p, the probability that p leads to a 'wasted' crossover is the probability of all bits to the left of p being identical in both parents, plus the probability of all bits to the right of p being identical, less the probability of both occurring. That is,

$$\left(\frac{1}{2}\right)^{p}+\left(\frac{1}{2}\right)^{l-p}-\left\{\left(\frac{1}{2}\right)^{p}\left(\frac{1}{2}\right)^{l-p}\right\}$$

Therefore, the probability that any particular crossover is wasted is

$$\frac{1}{l-1}\sum_{p=1}^{l-1}\left[\left(\frac{1}{2}\right)^{p}+\left(\frac{1}{2}\right)^{l-p}-\left\{\left(\frac{1}{2}\right)^{p}\left(\frac{1}{2}\right)^{l-p}\right\}\right]$$

Conversely, if the crossing point is selected with the string-of-change method, the probability that any crossover is wasted is only

$$\left(\frac{1}{2}\right)^{l}(l+1)$$

Figure 2.1 shows how these two probabilities compare for strings of lengths up to 30 bits.

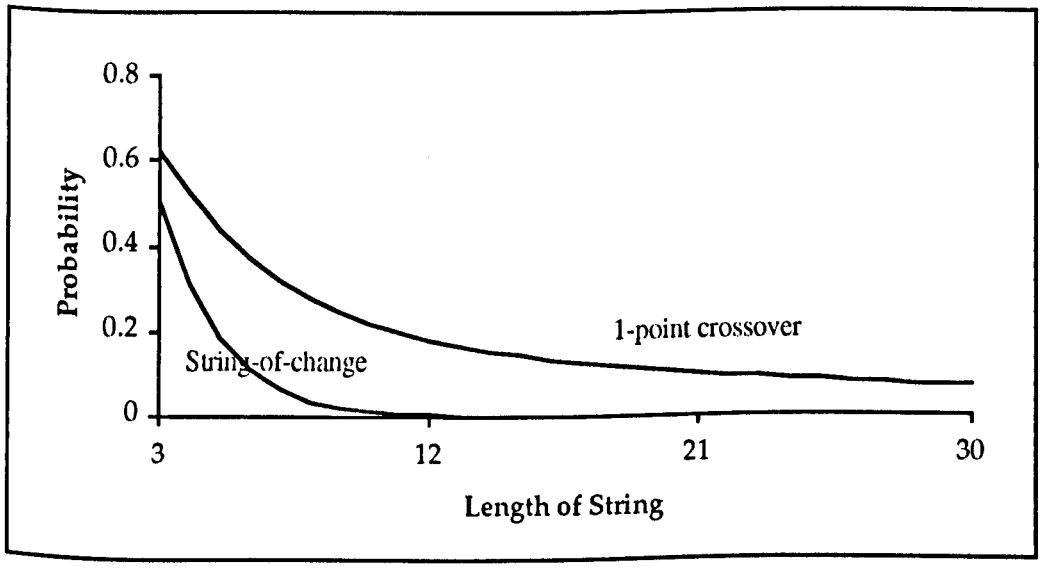


Figure 2.1: Probability of a 'wasted crossover'

Although variations on existing operators may improve the search of the EA, the operation

performed remains the same irrespective of the environment. Consequently, there are likely to be occasions when the operator is inappropriate. For example, a crossover operator which causes a great deal of disruption in the population (that is, it exchanges many sections) is likely to be unsuitable in the first few generations when it is necessary to exploit useful solutions to generate the building blocks from which useful solutions can be created. Conversely, an operator which causes little disruption is likely to be unsuitable for later generations when the population has almost converged and there is little extant variation. Thus, what is required are *adaptive operators* which are able to change their *modus operandi* to be appropriate for the state of the population upon which they operate.

Adaptive operators have been previously suggested on a number of occasions. One idea investigated by [Sirag and Weisser, 87] employed ideas from Simulated Annealing [Reeves, 93] to select appropriate crossing points, whilst another scheme, suggested by [Schaffer and Morishima, 87], involved encoding the crossing points at the end of the members which they produced. In this second scheme, it would be hoped that as particular crossing points become more beneficial, this would be reflected in the fitness of the offspring they produce and so the crossing points would be passed on to further generations. However, it is the author's belief that this method is likely to suffer from the common EA problem of getting stuck at local optima since crossing points producing offspring which would move the search away from a local peak would not be as fit as those which produced no change and hence, the useful crossing points would die off.

A similar idea to that of Schaffer and Morishima is proposed here. An operator which encodes crossing points but maintains these in a separate population structure and uses a different means of evaluation is suggested. This operator will be referred to as the *Lamarckian crossover* operator to reflect the current proliferation of the term Lamarckian in the EA community in describing features of an EA which use environmental information.

To implement the Lamarckian crossover operator, two distinct populations of strings are maintained. One of these is the solution population, whilst the other, the *template population*, is a population of crossover templates, as described in [Syswerda, 89]. After both

populations are randomly initialised, the EA proceeds as usual but when crossover occurs, a template is selected at random from the template population to perform the crossover. Each template has as its measure of fitness a *diversity rating* which is used to assess how successful that template has been at producing offspring which are different from their parents. This is very different from the traditional idea, such as that used by Schaffer and Morishima, of using the fitness of the offspring relative to their parents to judge the success of a crossover. This diversity rating, set to 1.0 for every template during the initialisation process, is then updated after crossover by using

$$\textit{diversity rating after crossover} = (\, c \,.\, \textit{diversity rating before crossover}\,) + \textit{success of crossover}$$

where *success of crossover* is the percentage of bits in the offspring which are different to those of its most similar parent, and c is a constant in the range $0 \leq c < 1$ used to give a weighting to the previous results. In common with other weighting factors used in other areas of EA research, such as the bid co-efficient in a classifier system, c was set to 0.9 (that is, 90%).

This process continues for n generations (n=5 was used with success) of the solution population until the template population itself is subjected to the selection and mating stages of an EA. The selection process uses the diversity rating to represent each template's fitness; the greater the diversity rating of a template, the more likely it is to be selected for reproduction, whilst the reproduction phase employs standard mutation (probability = 0.001) and 1-point crossover (probability = 0.75) to generate new templates. After mating, the diversity ratings of the new templates are reset to 1.0 and the whole process repeated. By performing crossover in this way, it is envisaged that, as the search of the EA focuses upon profitable areas of the search space, the crossover templates will *co-evolve* to bias crossover away from those areas, and thus avoid premature convergence.

The trials that were performed in Experiments 1 and 2 were repeated using, in turn, the following five crossover operators:

(a) 1-point crossover;

(b) string-of-change crossover;

(c) 2-point crossover;

(d) uniform crossover;

(e) Lamarkian crossover.

The results from these trials are shown in Tables 2.8 to 2.11.

Table 2.8: Experiment 3 -Best of Best-so-far

Problem No.	1	2	3	4	5	Av. error
1-point	4015 (0.0)	6110 (0.0)	12270 (0.9)	10336 (0.2)	15946 (1.5)	0.5
String-of-change	4015 (0.0)	6060 (0.8)	12200 (1.5)	10352 (0.0)	15917 (1.7)	0.8
2-point	3885 (3.3)	5900 (3.6)	12330 (0.4)	10232 (1.2)	15856 (2.1)	2.1
Uniform	4015 (0.0)	6030 (1.3)	12185 (1.6)	10267 (0.8)	16193 (0.0)	0.7
Lamarckian	4015 (0.0)	6050 (1.0)	12380 (0.0)	10329 (0.2)	16160 (0.2)	0.3

Table 2.9: Experiment 3 -Average Best-so-far

Problem No.	1	2	3	4	5	Av. error
1-point	4003 (0.0)	5740 (3.9)	11896 (1.2)	10124 (0.9)	15374 (2.1)	1.6
String-of-change	3984 (0.5)	5816 (2.5)	11822 (1.9)	10176 (0.4)	15541 (1.0)	1.3
2-point	3718 (7.7)	5415 (10.1)	11402 (5.6)	9879 (3.4)	14782 (6.2)	6.6
Uniform	3973 (0.8)	5834 (2.2)	11898 (1.2)	10136 (0.8)	15380 (2.1)	1.4
Lamarckian	3959 (1.1)	5963 (0.0)	12044 (0.0)	10215 (0.0)	15700 (0.0)	0.2

Table 2.10: Experiment 3 - Best On-line

Problem No.	1	2	3	4	5	Av. error
1-point	3724	5187	10637	9193	13817	
	(0.0)	(3.2)	(1.2)	(0.6)	(4.1)	1.8
String-of-change	3719	5259	10353	9143	14032	
	(0.1)	(1.8)	(4.0)	(1.1)	(2.5)	1.9
2-point	3544	5280	10717	9246	14387	
	(5.0)	(1.4)	(0.5)	(0.0)	(0.0)	0.5
Uniform	3692	5150	10343	9175	13857	
	(0.9)	(4.0)	(4.1)	(0.8)	(3.8)	2.7
Lamarckian	3690	5354	10767	8977	14026	
	(0.9)	(0.0)	(0.0)	(3.0)	(2.6)	1.3

Table 2.11: Experiment 3 - Best Off-line

Problem No.	1	2	3	4	5	Av. error
1-point	4003	6012	12080	10215	15676	
	(0.0)	(0.2)	(0.8)	(0.3)	(2.1)	0.7
String-of-change	4001	5936	12097	10248	15673	
	(0.1)	(1.5)	(0.6)	(0.0)	(2.2)	0.9
2-point	3809	5851	11490	9973	15535	
	(5.1)	(3.0)	(6.2)	(2.8)	(3.1)	4.0
Uniform	3997	6018	11994	10210	15947	
	(0.2)	(0.1)	(1.5)	(0.4)	(0.4)	0.5
Lamarckian	4004	6027	12179	10176	16013	
	(0.0)	(0.0)	(0.0)	(0.7)	(0.0)	0.1

The tables show that for all measures of effectiveness, the Lamarckian operator produces by far the best results, whilst the 2-point crossover operator tends to produce the worst performance of the five operators tried. Table 2.8 appears to indicate that of the other three operators, 1-point crossover produces the highest best-so-far performance. However, when the average rather than the best, best-so-far performance is calculated (Table 2.9), it shows that the string-of-change method is superior. As for the other performance measures, Tables 2.10 and 2.11 show that uniform crossover tends to produce the worst on-line, and the best off-line performance, whilst the two forms of 1-point crossover (simple and string-of-change) produce roughly equivalent on-line and off-line results.

These results demonstrate that an adaptive crossover operator, in the form of a population of co-evolving templates, can significantly improve the performance of an EA

compared to more traditional operators due to its ability to avoid premature convergence. Such adaptive operators can be especially useful in GBML systems where they can be used to guide the generation of new rules, rather than relying on purely random changes. Unfortunately, such an operation is much more expensive in terms of both time and space, and so in some cases, it may be more appropriate to use one of the simple operators. Out of these, 2-point crossover produced good on-line results but was poor in other performance measures whilst both string-of-change and uniform crossover generally improved upon 1 point crossover with regard to best-so-far and off-line performance.

2.4.5 Experiment 4: An investigation into mutation rates

2.4.5.1 Some alternative mutation schemes

Like crossover, mutation has been widely investigated by many researchers in the EA community. However, because of its very simple nature, the mutation operator itself has not received much attention, except in situations where tailored operators are required for specific encodings (eg. finding a mutation operator which produces a legal tour in the Travelling Salesman Problem [Michealewicz, 92]). Instead, most research has concentrated on finding optimal mutation rates (see, for example, [Fogarty, 89b], [Grefenstette, 86]). Finding such rates is often overlooked but is an important part of a EA because a rate which is too low would lead to no new variants being introduced, whereas too high a rate would lead to a loss of adaptation. Therefore, some intermediate value that is an optimum for the problem, in the sense of enabling solutions to evolve rapidly without imposing an excessive level of deleterious mutation, is necessary. In practice, such a level is found by trial and error but becomes redundant every time either the problem, or the EA itself, is changed. Some research has derived general formulae for automatically generating mutation rates (see, for example, [De Jong, 75], [Harvey, 92] and [Schaffer, et al, 89]) but such formulae provide only rough guidelines and are unlikely to be useful for more than a small number of problems.

An alternative which has many attractions because of its affinity to natural events is the use of a mutation rate which is dynamic and evolves throughout the run of an EA. To date, only a small number of studies have investigated adaptive mutation rates in an EA. These studies include the work of [Coyne, 92], in which the mutation rate was steadily increased from a small base value to a maximum after a set number of generations, [Reeves, 92], in which the mutation rate was set to be inversely proportional to the population diversity, and [Bäck, 92], in which mutation rates are incorporated in the genetic representation of the individuals. Further insight into adaptive mutation rates can also be gained from the work on ESs by the team of scientists based at the University of Dortmund in which the variance in the algorithm (as determined by the mutation rate) is varied according to the 1/5 success rule [Bäck, et al, 90]. However, because of the lack of work in this field, it was felt that a more extensive investigation would be profitable. On account of this, six alternative schemes were devised. These are:

(a) *On-line mutation*

With this method, the mutation rate is dependent upon the change in on-line performance between generations, and is calculated by

$$P_{mut} = C_1 \left(\frac{\omega^{on}(n+1)}{\omega^{on}(n)} \right)^{-2}$$

where $\omega^{on}(n)$ is the on-line performance after generation n, C_1 is a small constant and the power of 2 was adopted to place a greater emphasis on the value of the relative on-line performances. The mutation rate will therefore be at its highest value of C_1 when there is no change in on-line performance, while any improvement will reduce P_{mut} in proportion to the improvement in performance. The rationale for using this operator is that when there is little or no improvement in the on-line performance of the EA, it is an indication that there needs to be a fresh injection of new alleles into the population and so the mutation rate should be increased. This is typical of the situation wherein the population has converged on a small number of

solutions. However, when on-line performance is rising (that is $\omega^{on}(n+1) > \omega^{on}(n)$), mutation does not need to occur at such high levels and so P_{mut} is reduced to below the value of C_1.

(b) *Off-line mutation*

This is the same as the above on-line mutation method except that off-line performance, ω^{off}, is used to generate the new mutation rates instead of ω^{on}, and the rational for its use is analogous to that for on-line mutation.

(c) *Diversity Mutation 1* (DM1)

For this and the next two operators (DM2 and DM3) a value called the *diversity measure*, D, is used. This is calculated at the end of each generation but prior to mutation being performed, and is the total number of loci within the population which are not 'fixed', where a 'fixed locus' implies that every solution in the population has the same allele value at that locus. For diversity mutation method 1 (DM1), the diversity measure is used in the following formula to generate the appropriate mutation rate to be applied at the end of each generation:

$$P_{mut} = C_2(N - D)$$

where N is the number of items in the knapsack (ie. the string length) and C_2 is a small constant. The rational for using this operator is that it would be desirable for the mutation rate to increase as the population diversity decreases. Consequently, with DM1, the mutation rate increases linearly with a fall in diversity.

(d) *Diversity Mutation 2* (DM2)

For the DM2 method, the mutation rate is inversely proportional to the diversity measure,with the formula being:

$$P_{mut} = \frac{C_3 N}{D}$$

where C_3 is a small constant. This operator has the same effect as DM1 on the mutation rate except that the rate increases exponentially with a decrease in diversity rather than linearly. Moreover, the minimum value of P_{mut} is C_3 with DM2, rather than zero, which is the case with DM1.

(e) *Diversity Mutation 3* (DM3)

For the DM3 method, the diversity measure is used as follows:

$$P_{mut} = C_4 D^{C_5}$$

where C_4 as a small floating-point constant, and C_5 is a small integer constant. The rational for using this operator is speculative - to see if there is any benefit in a mutation rate which acts in the opposite manner to DM1 and DM2 by maintaining a high initial mutation rate and lowering it as diversity decreases.

(f) *Individual Mutation*

This scheme, which has many similarities to that investigated in [Bäck, 92], is based on the way that mutation occurs in some natural systems. In these, the mutation rate is determined by the enzymes that replicate and repair DNA, and as enzymes themselves are coded by genes, they too are subject to selective pressure and recombination operators in the same way as the rest of the chromosome. Therefore, rather than having a single global parameter which is applied to the population as a whole, each individual with this method is subject to mutation at its own individual rate. This mutation rate is passed on with an individual if the individual is selected for the mating pool, and should that individual be subsequently subject to crossover with another member, then either (a) the new child inherits a mutation rate somewhere between the mutation rates of the 2 parents, or (b) the child inherits, with equal probability, the mutation rate of either parent. These will be referred to as methods IM1 and IM2 respectively.

It is thus hoped that those individuals which have both a good phenotypic value and a

profitable mutation rate will become prolific within the population, and therefore, the need to have a global mutation rate would be avoided.

2.4.5.2 Results

(a) On-line and Off-line mutation

The experiments described earlier were repeated using the two mutation schemes which employ on-line and off-line performance to derive the mutation rate, and the average value of the mutation rate for each method was determined. These are shown in Figure 2.2.

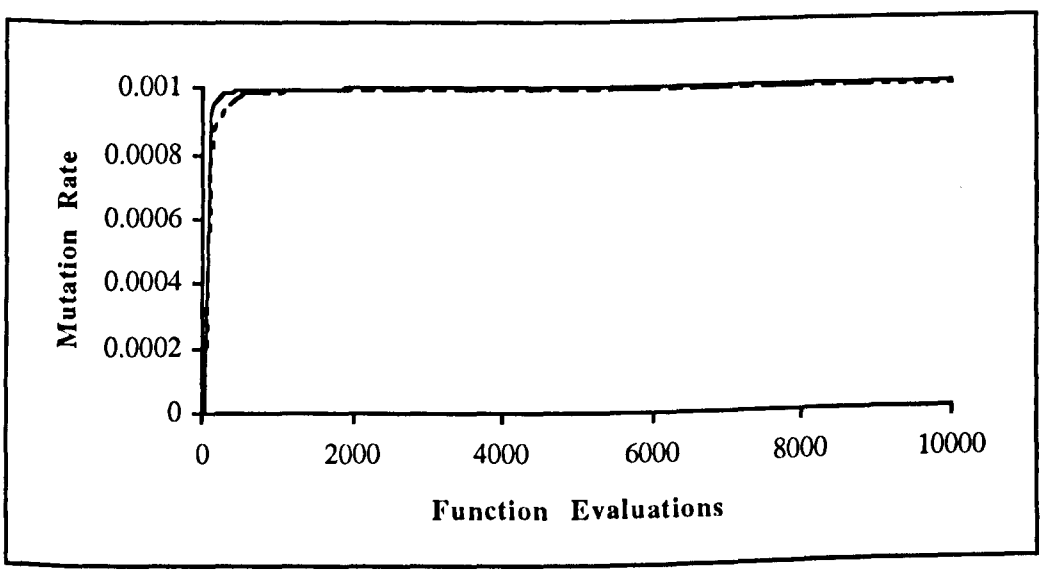


Figure 2.2: Variation in mutation rate for on-line (dotted) and off-line (solid) mutation using C_1=0.001 on problem 5.

As this graph shows, the mutation rate under both schemes rapidly converges (within 5-10 generations) to the value of the constant C_1. This is because after the first few generations, the difference in both on and off-line performance between generations is fairly minor. Hence, the ratio of performance measures is approximately unity and so the mutation rate quickly settles at C_1.

Predictably, when results using both of these schemes were compared, with a fixed

mutation rate of 0.001, there was very little difference, with the fixed scheme showing slightly better off-line performance (due to the greater number of mutations early on) and slightly worse on-line performance compared to the two performance related schemes.

(b) *Diversity Mutation 1*

The observed values for the mutation rates for various values of C_2, again averaged over 10 trials of problem 5, are shown in Figure 2.3.

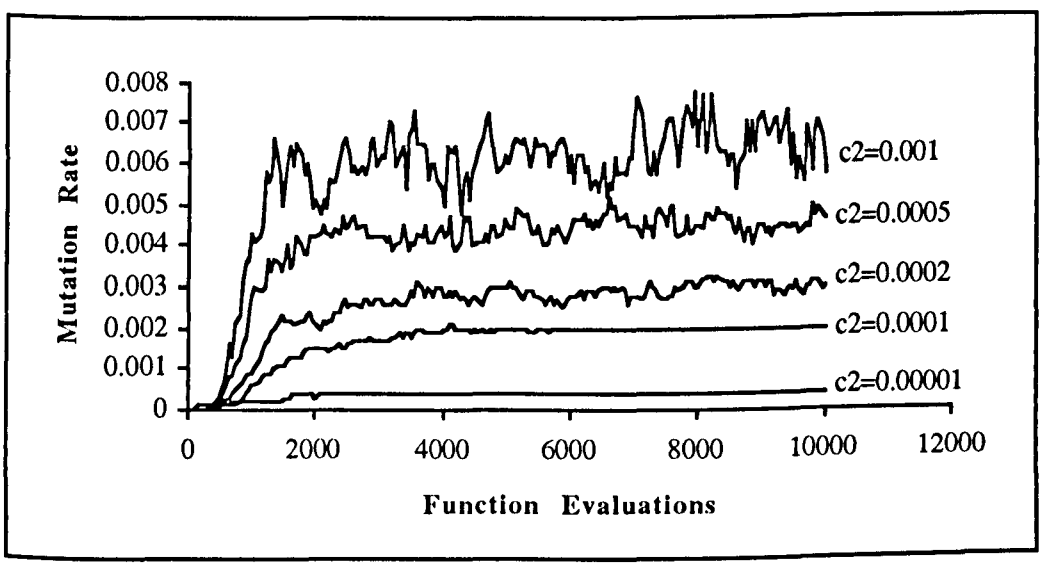


Figure 2.3: Variation in mutation rate of DM1 for different values of C_2

As the graphs in Figure 2.3 show, for the first 5-10 generations no alleles are fixed and hence the mutation rate is zero. The next few generations show a rapid rise in mutation rate due to several loci becoming fixed, before oscillating about an equilibrium level.

If the level at which the mutation rate reaches an equilibrium state can be formulated and its dependent variables found, then its behaviour can be better understood. An approximation to the equilibrium rate can be found if we assume that when the constant C_2 is very small, then the population will be very close to total convergence, ie. almost every allele value at each locus will be the same. Therefore after mutation,

Number of fixed loci, $f = N - N.n.P_{mut}$

$$= N(1 - n.P_{mut}) \quad (1)$$

Also,

$$Diversity, D = N - f$$

$$f = N - D$$

and $$P_{mut} = (N - D)\ C_2 \qquad \text{(from definition of } P_{mut})$$

$$D = N - (P_{mut} / C_2)$$

$$f = P_{mut} / C_2 \quad (2)$$

At equilibrium, (1) and (2) are equal, so

$$N(1 - n.P_{mut}) = P_{mut} / C_2$$

$$P_{mut} = (N.C_2) / (N.C_2.n+1) \quad (3)$$

This formula is correct in cases where C_2 is very small and only a very small number of loci on average are changed every generation. This is demonstrated by comparing the bottom graph in Figure 2.3 (C_2=0.00001, n=50 and N=50) to the predicted value - both being identical at P_{mut}=0.0005. However, when the equilibrium mutation rate is large enough to produce several mutations per generation, the equilibrium diversity is larger and it would be expected that half of the non-fixed loci would survive to the next generation, and so the mutation rate need only be approximately half of that predicted, namely

$$P_{mut} = (N.C_2) / 2(N.n.C_2 + 1)$$

Table 2.12 shows the expected values of P_{mut} against those observed when N=50 (problem 5) and n=50.

Table 2.12: Comparison of observed and predicted equilibrium mutation rates for DM1

C2	Prob. Mutation	Observed Value
0.0001	0.002	0.002
0.0002	0.003	0.003
0.0005	0.006	0.005
0.001	0.007	0.007

As the results show, the formula gives a good approximation to the expected equilibrium mutation levels for various values of C_2. The formula also allows the effects of varying the two other factors which combine to give the value of P_{mut}, namely the string length N and the population size n, to be predicted. According to the formula, P_{mut} should increase with increases in N but decrease with increases in n. These observations were found to be correct when the values of n and N were varied, the results being shown in Figures 2.4 and 2.5.

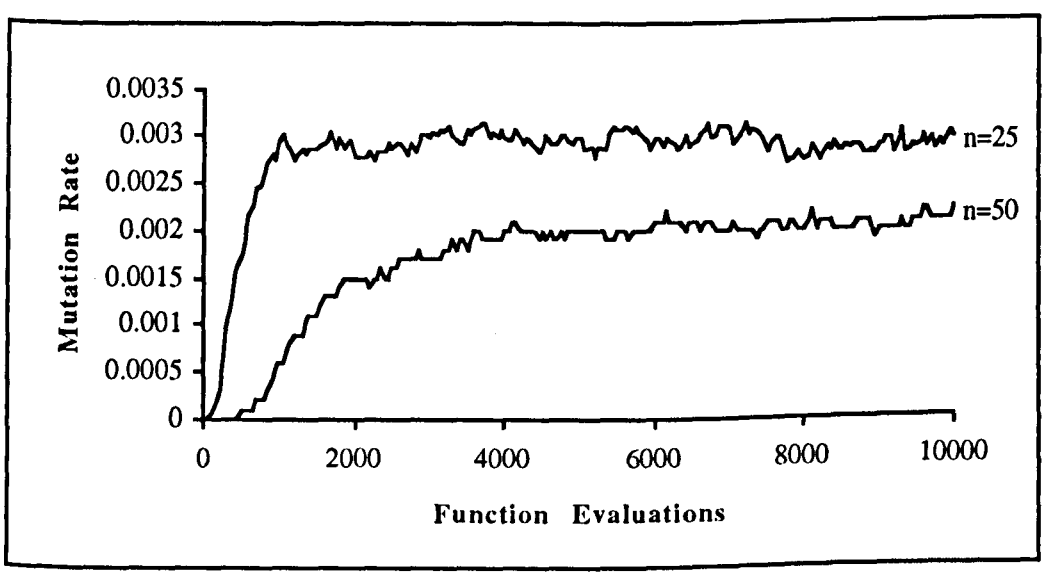


Figure 2.4: Variation in mutation rate for DM1 for values of population size, n

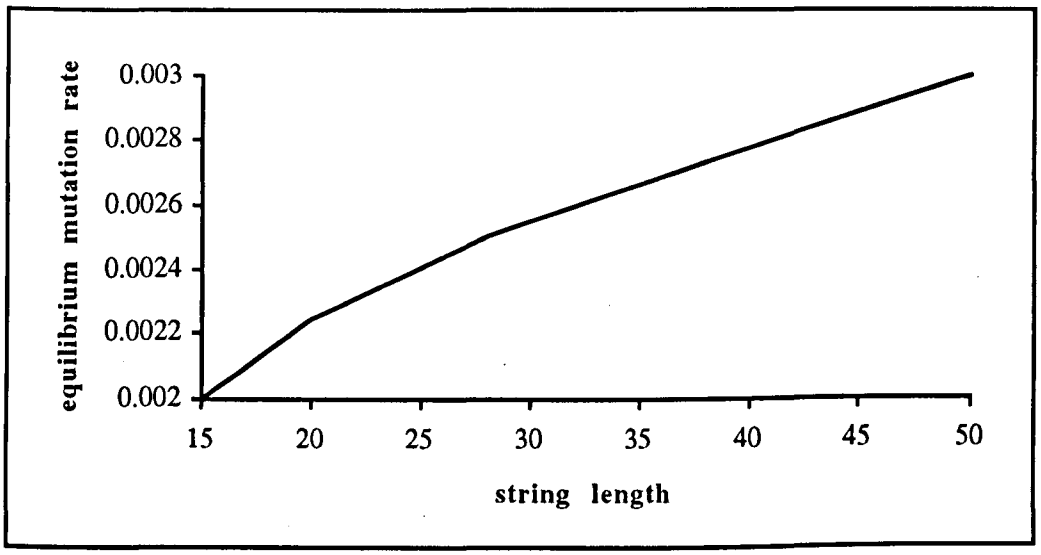


Figure 2.5: Steady state mutation rate for DM1 for values of string length (problem size)

The performance of DM1, relative to that of a fixed mutation rate, can be determined by comparing the results using various values of N, n and C_2 for DM1 with fixed rates set to the equivalent equilibrium levels. However, this puts the results of the DM1 method at a disadvantage since DM1 will be making fewer mutations overall. Therefore, it was felt that the best way to compare the two schemes was to run an EA using DM1 and record the actual number of mutations made. This value, averaged over 10 trials, then gives a fairly good estimate of the average number of mutations performed on any one trial and from this, the average mutation rate to be used in a fixed scheme can be determined and the results of the fixed and DM1 methods compared.

Twenty five sets of results were compared (5 values of C_2 for each of the 5 problems with N constant at 50) using the 6 measures of effectiveness previously described. The results showed that there was no significant improvement in using DM1 compared to a standard fixed mutation rate.

(c) *Diversity Mutation 2*

The family of mutation rates generated by varying the value of C_3 and keeping N and n

constant (both at 50) are given in Figure 2.6 below.

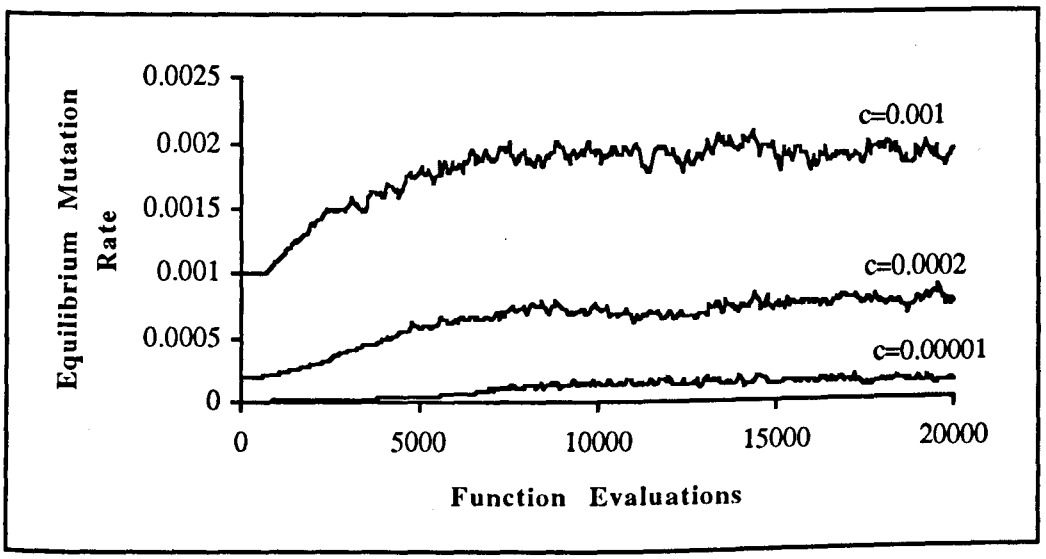


Figure 2.6: Variation in mutation rate for DM2 for values of C_3

As the graph shows, with this method there is initially a mutation rate of C_3 since $P_{mut}=N.C_3/D$ and during the initial generations $D=N$. As some of the loci become fixed, the diversity will drop and the mutation rate will subsequently rise. However, this is a much more gradual increase than with the DM1 method since each drop in diversity only increases the mutation rate by $100/N$ percent, and hence, the time taken to reach an equilibrium level is significantly longer.

To find an expression to give an approximate value for the equilibrium mutation rate for DM2, using analogues to that of DM1,

$$f = N (1 - n. P_{mut})$$

but this time, $D = N. C_3 / P_{mut}$ (from definition of mutation rate)

therefore, $f = N (1 - (C_3 / P_{mut}))$

At equilibrium,

$$N (1 - n.P_{mut}) = N (1 - C_3 / P_{mut})$$

$$n.P_{mut} = C_3 / P_{mut}$$

$$P_{mut} = \left(\frac{C_3}{n}\right)^{\frac{1}{2}}$$

Again, in practice only half of this is needed and so

$$P_{mut} = \frac{1}{2}\left(\frac{C_3}{n}\right)^{\frac{1}{2}}$$

and Table 2.13 shows the expected values of P_{mut} against those observed when N=50 (problem 5) and n=50.

Table 2.13: Comparison of observed and predicted equilibrium mutation rates for DM2

C3	Mutation Probability	Observed Value
0.0001	0.0007	0.0005
0.0002	0.001	0.0008
0.0005	0.0016	0.0013
0.001	0.0022	0.002

As the results show, the formula gives a fair approximation to the expected equilibrium mutation levels for various values of C_3. One interesting factor of the formula is the implication that the equilibrium rate is dependent only upon C_3 and population size, n, and is independent of the string length N. Figure 2.6 has already shown that the equilibrium mutation rate increases as C_3 increases. Moreover, Figure 2.7 demonstrates that, as predicted, the mutation rate falls as the population size rises, whilst Figure 2.8 shows that there is no direct relationship between N and mutation rate, once again, as predicted.

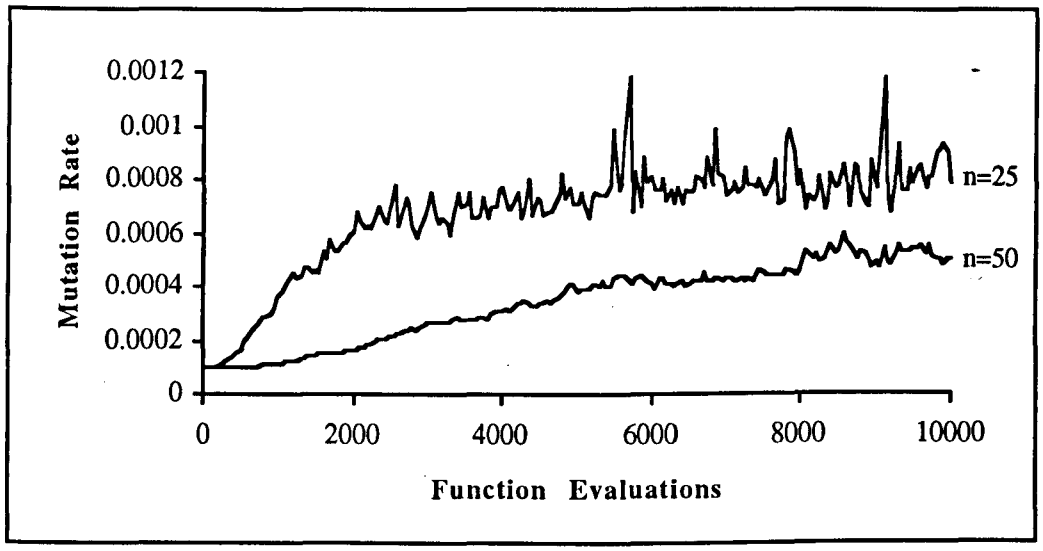


Figure 2.7: Variation in mutation rate for DM2 for values of population size, n

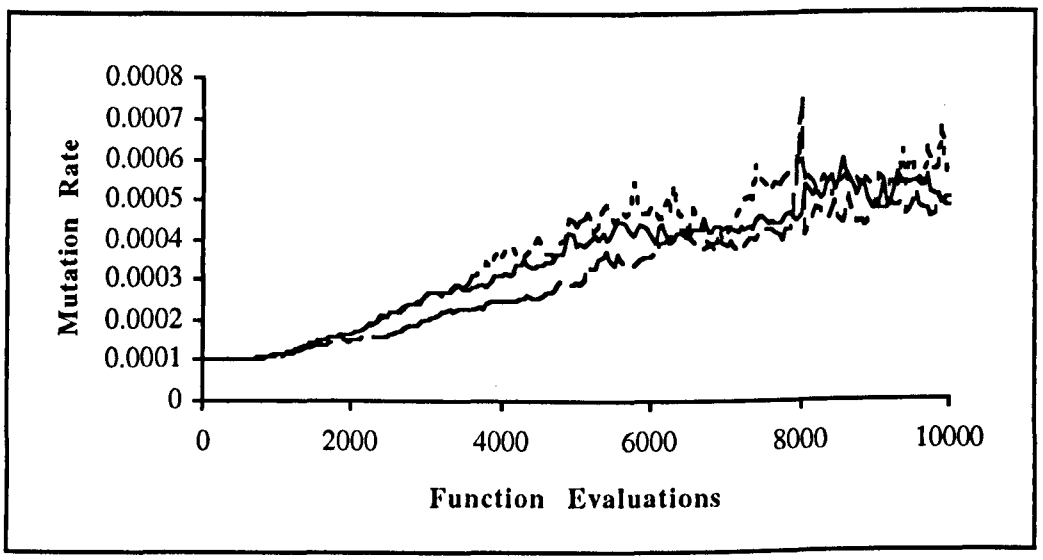


Figure 2.8: Variation in mutation rate for DM2 for values of string length, N

(solid⇒N=50, dotted⇒N=39, dashed⇒N=28)

Using the same method as was used to compare DM1 with a fixed mutation rate, DM2 was also compared to a fixed rate. However, with this set of 25 results, a slight improvement upon the results achieved with the equivalent fixed mutation rates was observed.

(d) *Diversity Mutation 3*

The family of curves generated by varying C_4 and C_5 are shown in Figures 2.9 and 2.10 respectively.

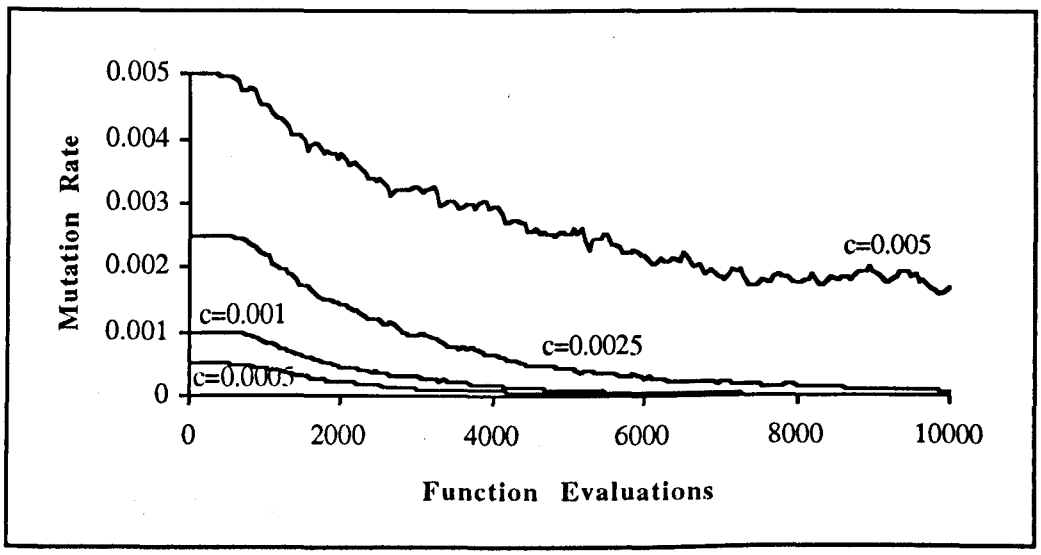


Figure 2.9: Variation in mutation rate for DM3 for values of C_4 with N=50, n=50, C_5=2.

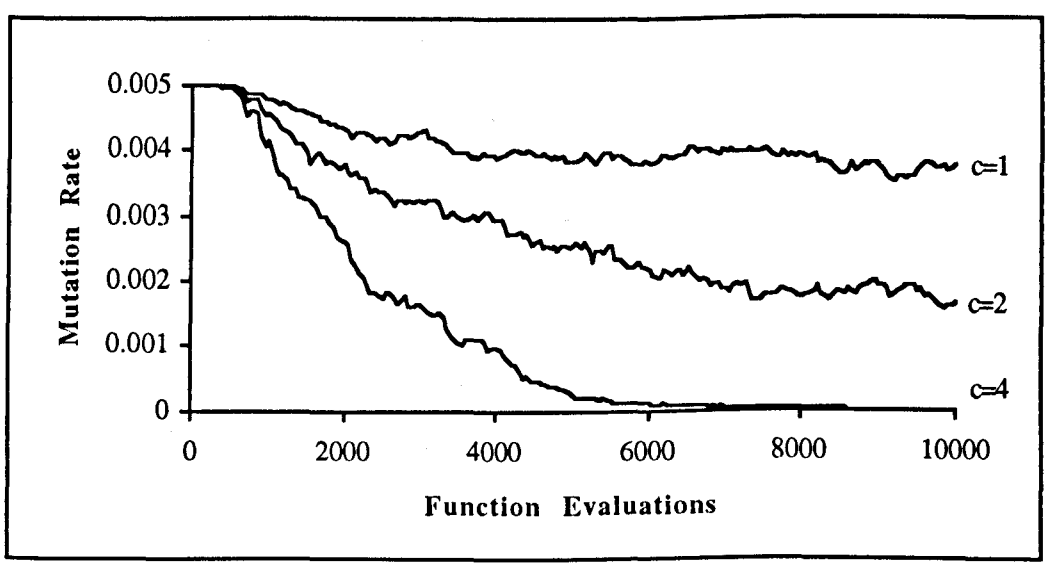


Figure 2.10: Variation in mutation rate for DM3 for values of C_5 with N=50, n=50, C4= 0.0001 / 50 C_5-1

These two graphs demonstrate that the mutation rate starts off at a value of $C_4.N$ (or $C_4.N^{C_5}$ if C_5 is unadjusted) and continues at this level until the population diversity starts to fall (that is, some of the loci become fixed). At this point the mutation rate falls in proportion to the value of C_5, before, once again, levelling off at an equilibrium level.

Results comparing DM3 with a fixed rate show that DM3 produces slightly worse results. Thus, the initial hypothesis that the mutation rate should rise, not fall, as the population converges, is confirmed.

(e) *Individual Mutation Rates*

The mutation rates averaged over 10 trials of problem 5 using both IM1 and IM2 are shown in Figure 2.11. In both experiments the initial mutation rate for each individual was generated randomly with normal distribution between 0.0 and 0.01.

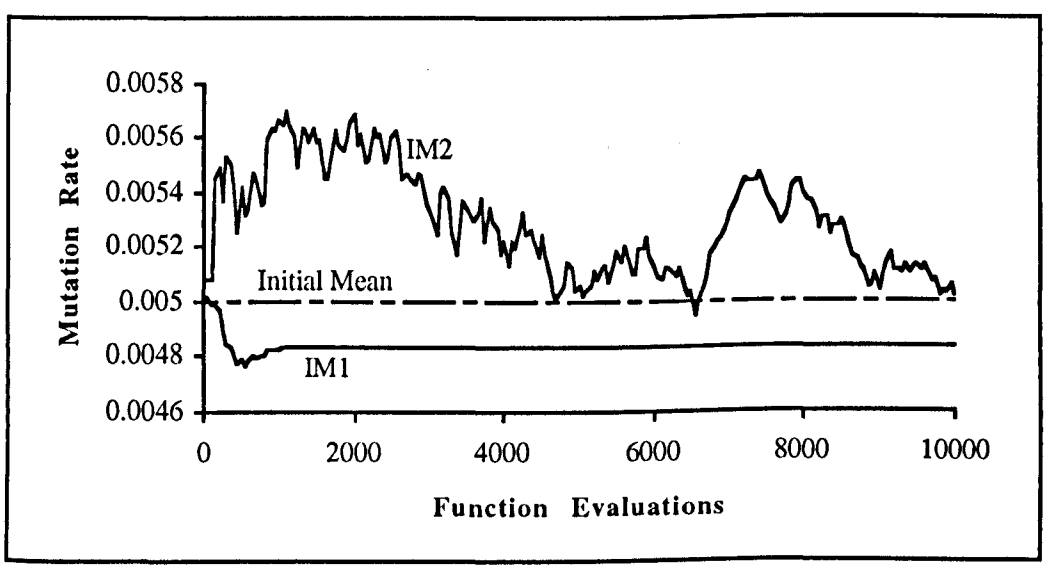


Figure 2.11: Average mutation rates when using 'individual mutation' with initial mean = 0.005

The mutation rates shown in Figure 2.11 are typical of those generated using a number of different distributions of initial mutation rates. In all cases the final rate, not surprisingly, was

very close to the mean initial value (0.005 in Figure 2.11). This is mainly due to the fact that for the small mutation rates used (of the order of 1 in 1000), there is very little selective pressure between even relatively different mutation rates, and so the rate is virtually ignored in terms of selection. For the first crossover method, this simply means that because the mutation rate is chosen to be the average of the two parent rates, the overall rate rapidly converges to the initial average. However, with the second scheme, where one or the other of the parent rates survives, the hitch-hiking effect takes place and the population tends to converge on the rate which is associated with the first profitable solution - it is only when the average rate over several trials is calculated that the mutation rate appears to be close to the mean.

2.4.5.3 Summary of Mutation Rate Experiments

The results from this series of experiments suggests that using either on-line or off-line performance to vary the mutation rate during a trial is not worthwhile since despite being computationally expensive to calculate, within a short number of generations, the rate settles at a constant level. Of the diversity schemes, both DM1 and DM3 performed poorly. The failure of DM1 was due to a zero mutation during the first 5-10 generations, whilst the poor performance of DM3 was due to insufficient mutation occurring as the population neared convergence. Also, assigning mutation rates to each individual proved unsuccessful since the problems were not of sufficient size for the various mutation rates to force any selective pressure towards an optimal rate. However, the performance of the DM2 operator did improve upon that of the standard scheme. Moreover, it appears to be useful in another respect since it adapts to a change in population size which may be advantageous if the population size fluctuates.

2.4.6 Experiment 5: An investigation into other genetic operators

As mentioned in Chapter 1, of the other genetic operators which are applied to GA's, *inversion* is the most common. However, it is difficult to use the standard form of inversion with the knapsack problem because the change in the order of the items caused by inversion

would make crossover difficult to carry out, in much the same way that special operators have to be designed for the TSP to make crossover possible. Some experiments have been performed [Fairley and Yates, 92] on inversion-like operators which perform an inversion of part of the genotype therefore creating a new solution. However, whilst results with this operator were quite promising, the operator is not really an inversion operator but more like a modified mutation operator where several adjacent bits may be mutated together. Here, an operator which is more akin to an inversion operator is proposed. This operator, called *shuffle*, when called every n generations, randomly generates a new order for the problem description (ie. objective function and constraints) and adjusts the encoding scheme accordingly, thus leaving the phenotype of each population member unchanged.

The results comparing an EA using the shuffle operator every 10 generations to one without the operator are shown in Tables 2.14 to 2.16.

Table 2.14: Experiment 5 - Best of Best-so-far

Problem No.	1	2	3	4	5	Av. error
Without Shuffle	4015 (0.0)	6110 (0.0)	12270 (0.3)	10336 (0.5)	15946 (1.0)	0.36
With Shuffle	4015 (0.0)	6100 (0.2)	12310 (0.0)	10387 (0.0)	16103 (0.0)	0.04

Table 2.15: Experiment 5 - Best On-line

Problem No.	1	2	3	4	5	Av. error
Without Shuffle	3724 (0.0)	5187 (0.8)	10637 (1.7)	9193 (0.0)	13817 (1.3)	0.76
With Shuffle	3579 (4.1)	5227 (0.0)	10815 (0.0)	9046 (1.6)	14003 (0.0)	1.14

Table 2.16: Experiment 5 - Best Off-line

Problem No.	1	2	3	4	5	Av. error
Without Shuffle	4003 (0.2)	6012 (0.9)	12080 (0.7)	10215 (0.4)	15676 (1.2)	0.68
With Shuffle	4010 (0.0)	6066 (0.0)	12166 (0.0)	10251 (0.0)	15869 (0.0)	0.0

The results show that with respect to best-so-far and off-line performance, the EA using the shuffle operator produced by far the better results. The on-line performance tended to show a slight reduction in performance with this new operator, although this may be largely attributed to the very poor on-line performance achieved with the shuffle operator on problem 1. Otherwise, the on-line performance achieved with the shuffle operator was largely in line with that achieved without it.

2.5 Conclusions

This chapter has presented the results from applying several experimental EA's to a series of knapsack problems. Although the results obtained are specific to the knapsack problems investigated in this chapter, the following conclusions may be suggested:

(a) based on the impressive performance of the REMOVE_ITEM operator, operators which rectify or improve invalid/poor solutions appear, under certain conditions, to be useful additions to an EA. In relation to GBML, this fact can be used in the situation where no rules match the extant state of the world model. For example, this result adds weight to the argument for altering existing rules rather than creating new rules at random, in the 'no match' situation.

(b) in these trials, the use of Lamarckian operators has been shown to be beneficial to the search of the EA and such operators may prove useful in guiding the generation of new rules as opposed to employing purely random techniques.

(c) incorporating environmental information into operator rates as well as their operation appears to, under certain conditions, improve system performance. This can be

investigated further by incorporating such features in the operator rates of the proposed GBML system.

(d) the results with the SHUFFLE operator demonstrate that using problem specific operators can, under certain conditions, be beneficial to the search of the EA. Thus, the use of operators specific to the representation scheme of the proposed system should also be considered in the design stage.

(e) the results with the alternative partner selection schemes showed that schemes such as the Herd Leader and Lek methods can, in some cases, significantly improve the performance of the EA

3. Credit assignment in a simple classifier system

3.1 Introduction

In the previous chapter, one of the main components of a classifier system, the evolutionary algorithm, was implemented and a number of alternative designs investigated to determine if the effectiveness of the EA applied to the knapsack problem could be improved in any way. Following on from that work, the aim of this chapter is to investigate the other central component of a classifier system, the credit assignment algorithm, to see if the performance of this too can be improved.

Since they are usually geared towards practical applications, classifier systems in general, and credit assignment algorithms in particular, have received very little attention from a theoretical standpoint. Therefore, in an attempt to redress the balance, section 3.2 presents a study in which a relatively new branch of mathematics called *Evolutionary Game Theory* (EGT) is used to analyse the evolution of a hypothetical rule set.

The remainder of this chapter is devoted to investigating some of the practical problems associated with credit assignment in classifier systems. To enable this investigation to be performed, a simple classifier system, SCS1, was implemented. Moreover, the problem upon which the investigation would be based was chosen to be a simple 2-player military game called the *Two Tanks Problem* (TTP). Section 3.3 contains both a functional description of SCS1 and a definition of the version of the TTP used for this investigation, together with some results from a study investigating a particularly important problem associated with classifier systems, that of assigning credit to simultaneously active rules. This work is then continued in section 3.4 where alternative credit assignment schemes are suggested as potential solutions to the problems identified. The relative merits of the new schemes are then investigated with the consequent conclusions being reported in section 3.5.

3.2 Evolutionary stability in simple classifier systems

3.2.1 The need for some background theory

To date, Classifier Systems have proved to be a most popular vehicle for the application of Evolutionary Algorithms in the field of machine learning (see, for example, [Booker, et al, 89]). However, despite such widespread popularity, the vast majority of research involving classifier systems has focussed either on specific applications (see, for example, [Goldberg, 83] and [Wilson, 87a]), or on means for improving the performance of certain of their constituent elements, see [Yates and Fairley, 93], [Booker, 89] and [Riolo, 89a]. Arguably the most notable exception to this is the work performed by [Holland, 86a] which attempts the construction of a mathematical framework for learning in classifier systems. Notwithstanding, many of the more 'global' questions regarding classifier systems remain unanswered. One such question is: does a classifier system optimise the overall effectiveness of its plan, or the effectiveness of the individual rules which constitute the plan - at the possible expense of the plan itself? In the search for a suitable method with which to answer this and related questions, one approach which emerged as extremely promising was that of *Evolutionary Game Theory*.

3.2.2 Evolutionary Game Theory

The branch of mathematics known as Evolutionary Game Theory (EGT) [Maynard-Smith, 86] is a relatively recent development, the motivation for which was the need for a theory which would "model evolutionary processes in populations of interacting individuals and explain why certain states of a given population are - in the course of the selection process - stable against perturbations induced by mutations" [p 1, Bomze and Potscher, 89]. Based upon pre-programmed behavioral policies called *strategies*, EGT differs from classical game theory in that the effectiveness of strategies are measured not in terms of *utility*, but *fitness*, and *natural selection*, rather than *rationality*, is used as a means for selecting strategy.

In EGT, a population consists of individuals possessing but a single inherited trait, a *pure strategy*, which is an in-built collection of behavioral characteristics selected from a finite

set of such strategies. The environment in which the individuals interact takes the form of a set of contests, in each of which, two randomly selected individuals from the population compete with one another, each playing its respective strategy. Each individual also has an associated 'fitness measure' which represents that individuals effectiveness in the environment.

Evolution in a population occurs as follows. After a contest between a pair of individuals has been concluded, the outcome results in the rewards (payoffs), P and P′, reflecting the success of the respective strategies adopted, being made to the two combatants. After a large number of such contests, an individual's average payoff corresponds to its utility (fitness), and therefore, the relative fitness of each individual can be calculated. Using natural selection to replicate the individuals in proportion to their fitness, the next generation of individuals is generated, and by repeating this sequence, the population evolves over time.

EGT provides a means of analysing evolving populations and can be used to determine whether there exists a state wherein the distribution of the various strategies amongst the population is in equilibrium. If such an equilibrium state does exist, the population is said to be *evolutionarily stable* and the collection of strategies is said to constitute an *Evolutionarily Stable Strategy* (ESS).

The concept of an ESS may be clarified by the following example, the *Hawk-Dove Game*, taken from [Dawkins, 76]. In this game, the objective of each contest between randomly selected pairs of individuals from a population is to take control of territory. There are only two strategies that can be adopted by any one of the population of identical individuals. These two strategies are called *hawk* and *dove*. In a contest, any individual adopting the hawk strategy will, at the risk of injury, attack its opponent in order to gain territory. However, an individual adopting the dove strategy will not risk a confrontation, and so, in the face of an aggressive opponent (that is, one playing a hawk), will retreat and remain unscathed. If two hawks meet, they will fight and one will emerge the winner, whilst the other will sustain injury. If two doves meet, each will evade the other until ultimately, one wins by, almost accidentally, being in control of the territory.

After a confrontation, both contestants are given a 'reward' consistent with their

performance. There is a reward of 50 points for gaining territory, 0 for ceding ground immediately but remaining uninjured, -10 for being involved in a protracted confrontation and -100 for losing a fight.

Given such a scenario, what strategy should an individual adopt? The answer to this question depends upon the strategy adopted by the rest of the population. For example, given a population consisting entirely of doves, the average payoff for each member is 15 points (since in each contest, one dove will gain 50 points for winning the territory but lose 10 for being involved in a protracted confrontation, while the other will lose 10 overall, and thus the average pay-off is (40+(-10))/2). Now, if just one of the population were to adopt the hawk strategy, it would be awarded 50 points after every confrontation, thereby increasing that individual's reward above the mean for the population. If then other members of the population learn from this example, the hawk strategy would spread through the population. However, as the number of hawks grows, their average fitness would decrease since, on some occasions, hawks would meet each other, and one of them would lose 100 points.

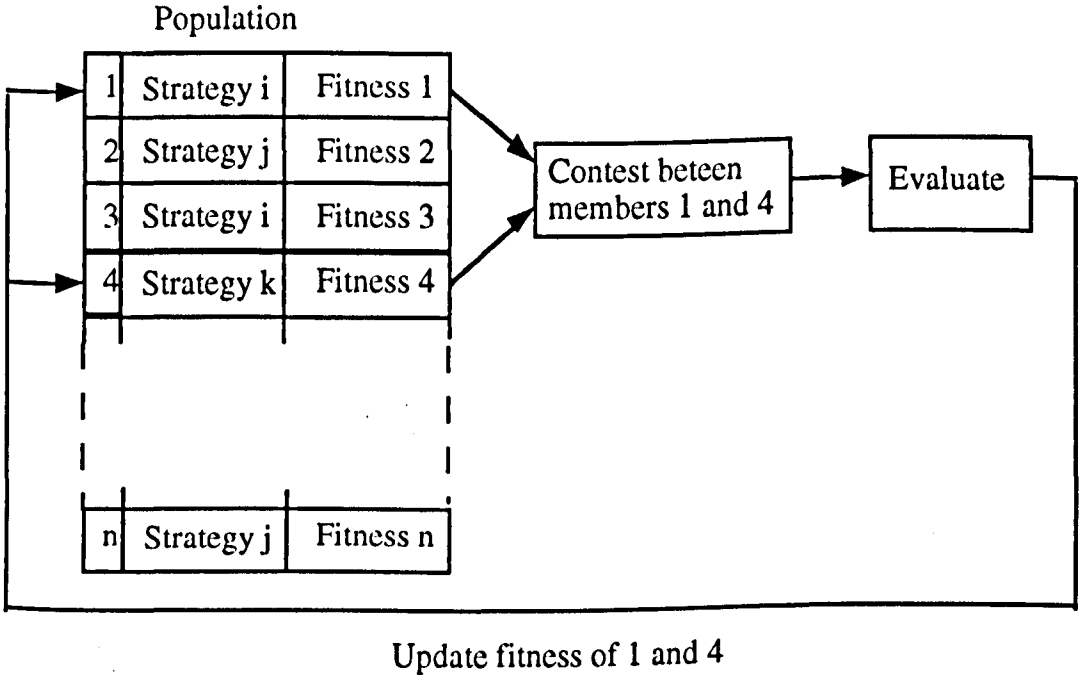


Figure 3.1: Main cycle of an evolutionary game

In order to find the ratio of hawks to doves in this population at which stability is reached, let x be the probability that a member of the population plays the hawk strategy and $(1-x)$ the probability that he plays a dove. At equilibrium, the average pay-off for playing hawk, P_{hawk}, equals the average pay-off for playing dove, P_{dove}, and

$$P_{hawk} = (x(-25) + (1-x)50)/2, \text{ and } P_{dove} = (x(0) + (1-x)15)/2.$$

Therefore, at equilibrium, $(50 - 75x) = (15 - 15x) \Rightarrow x = 7/12$, and so stability is only reached when the population is divided in a ratio of 7/12 hawks to 5/12 doves. At this ratio, any mutant individual, that is one which changes its strategy from hawk to dove or vice versa, will receive a lower average pay-off than the population mean and so the population is evolutionarily stable and forms an ESS.

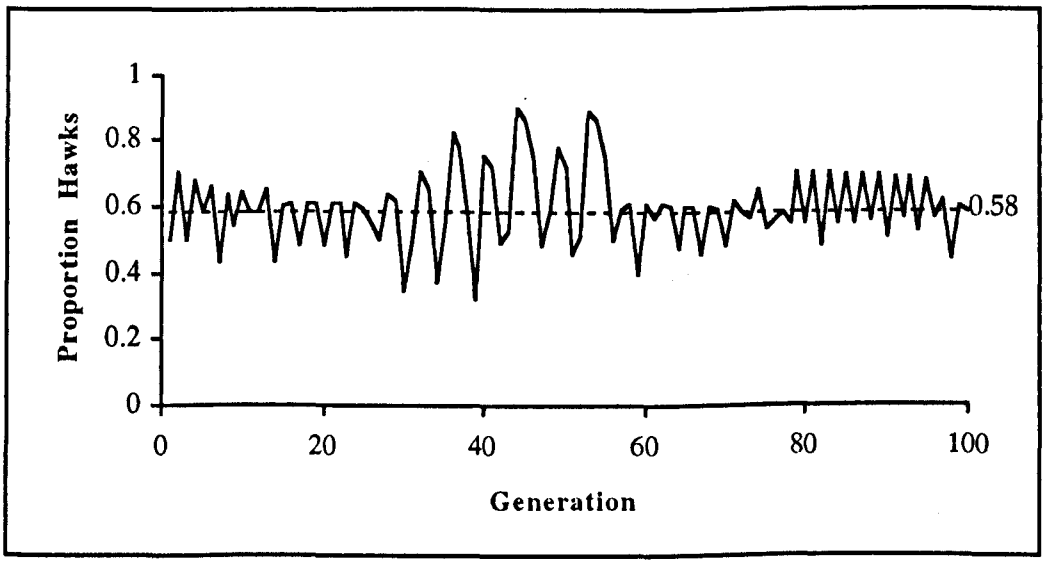


Figure 3.2: Results when simulating the Hawk-Dove Game

To investigate this result, a 'simple' experiment was performed. In the experiment, a population of 500 individuals was allowed to evolve over 100 generations. The initial population was divided equally into hawks and doves, and in every generation, each individual competed against all others five times (giving a total of over six hundred thousand contests per

generation). Natural selection was mimicked using roulette wheel selection, and the 500 individuals thus selected were used to form the succeeding generation. Figure 3.2 depicts the number of hawks in each of the 100 generations of the single simulation run. The results clearly demonstrate that the ratio of hawks to doves oscillates about the predicted theoretical equilibrium value, the oscillations being due, in the main, to the sampling errors in respect of the (small) population size.

It is important to note that the above does not imply that an ESS is optimal in terms of the expected reward for individual members of the population. For example, at the evolutionary stable state, the average reward for each of the population's members is 6.25 which is less than the 15 point reward for each member of a population consisting entirely of doves. Instead, the evolutionary stable state is more like a pareto optimum for which the success of one individual depends upon that of the others.

To be capable of evolving into such a stable state, a system must have some mechanism by which it can adapt its response to become more profitable. Such a mechanism in EGT is called a *Learning Rule*. A learning rule is a meta-rule which specifies the probabilities with which an individual will adopt one of a set of strategies as a function of the strategies it has adopted in the past and the results of adopting those strategies. An important question relating to learning rules is: does there exist a learning rule that will always cause an initially naive plan to adapt, in time, towards an ESS for the game which it is playing? [Harley, 81] proved that such a rule for ESSs is itself evolutionarily stable since a system adopting this could not be invaded by a mutant utilising an alternative learning rule. He called such a rule an *Evolutionarily Stable learning rule* or ES learning rule.

3.2.3 Evolutionary Stability and Classifier Systems

Although the above discourse on the hawk-dove game suggests that an evolutionary stable state is achieved if 7/12ths of the population play hawk and the remainder play dove, without loss of generality, an identical ESS can be achieved if a single individual adopts each strategy in the

appropriate ratio (that is, a mixed strategy of playing a hawk on 7 out of 12 occasions and a dove on the other 5). This is so for any individual competing with either a single random opponent who is also playing the same mixed strategy, or a set of opponents, each playing a fixed (pure) strategy. Note that this situation is equivalent to a classifier system playing its respective strategies (represented by its rules) in proportion to their respective strengths. For example, the ESS in the hawk-dove game could be represented by the plan, P_{HDG}, containing the two rules:

P_{HDG}: Rule 1: IF opponent is either Hawk or Dove THEN play_hawk *Strength*=0.5833

Rule 2: IF opponent is either Hawk or Dove THEN play_dove *Strength*=0.4177

By representing strategies in this form, EGT can be applied to classifier systems to determine whether the strength of the rules (classifiers) approach evolutionary stability. Indeed, as [Maynard Smith, 82] points out, the idea of a strategy is the same as that of a behavioral phenotype and as such can be applied equally well to any kind of phenotype such as the growth form of a plant, the age at first reproduction, or the relative numbers of sons and daughters produced by a parent".

However, for a classifier system's plan to change, there must be some mechanism by which the strengths of its rules can be adjusted - that is, there must be a learning rule. Moreover, if the system is to reach an ESS, then this rule must be an ES learning rule. In a classifier system, the means by which rule strength is usually adjusted in order to adapt the system's response, is the Bucket Brigade Algorithm and thus, if the system's rule set is to move towards evolutionary stability, the BBA must be an ES learning rule.

3.2.4 The Bucket Brigade Algorithm as an ES Learning Rule

[pp 671, Harley, 81] suggests that, "*it is unlikely that any animal has a learning rule which provides it with an unbeatable strategy for every game that it might encounter. Even an extremely simple frequency independent game such as the two-armed bandit requires a*

dynamic programming algorithm and a very large memory to solve for the best sequence of behaviours." Instead it is thought that some animals have a simple learning rule which allows them to do well in a wide range of situations. To determine whether such a learning rule is in fact an ES learning rule, Harley detailed five properties that it should possess. These properties are based on several assumptions about both the system and the game, of which the most important are:

(a) each game considered actually possesses an ESS, which in principle is capable of being learnt;

(b) although the reward (pay-off) may change in time, it does so sufficiently slowly that learning rules can establish stable frequencies of behaviours;

(c) the pay-off, $P_i(t)$, that a behaviour B_i receives at time t is non-negative and evaluated in units of fitness. The pay-off at time t for any behaviour B_j not used at time t is zero;

(d) the learning rule defines the probability of displaying each of the possible behaviours at each time step as a function of the previous pay-offs.

Given these assumptions, the five properties of an ES learning rule as specified by Harley are:

(1) it must have the property that, after a long series of plays, the probability of selecting action A is equal to the total payoff for playing A divided by the total payoff so far. Harley called this the *relative payoff sum* or *RPS property*;

(2) the ES learning rule will never completely abandon any action, K, nor will it ever fix the behaviour so that it always performs K. This is necessary because circumstances (and hence pay-off) may change;

(3) the ES learning rule will attach greater significance to more recent pay-off information. Again, this is necessary because circumstances may change;

(4) the ES learning rule will be a function of prior expectation of pay-off;

(5) the ES learning rule will also be a function of actual pay-off.

By investigating the BBA in respect of these five properties, it is possible to determine whether the BBA can indeed be classed as an ES learning rule.

Property 1:

For the purpose of this proof, consider a classifier system, M, which employs the simplest form of BBA, namely:

$$S_i(t+1) = S_i(t) + P_i(t) + R_i(t) - B_i(t)$$

where $S_i(t)$, $P_i(t)$, $R_i(t)$ and $B_i(t)$ denote respectively the strength of, the pay-off made to, the payment received from other classifiers, and the bid made by rule i at time t.

Given a sufficiently long time, the strength of a rule, X_n, which is used at the end of a sequence of actions, will be proportional to its payoff, P. As this rule pays a proportion of its strength to its immediate predecessor, X_{n-1}, in the sequence, then the strength of X_{n-1} will also be proportional to P. Repeating this argument, it can be seen that the strength of the first active rule in the chain, X_1, will also be proportional to P. In M the probability of selecting a rule is its relative strength divided by the sum of the strengths of all other candidate rules (that is, its relative pay-off) and hence, the Relative Payoff Sum property is satisfied.

Property 2:

Assuming that pay-off is always non-negative, then a classifiers strength only decreases through taxation and bidding. As both of these quantities are only small percentages of a classifiers strength, then strength can only asymptotically approach zero, and so its probability will never be exactly zero. Correspondingly, as no classifier has zero strength, then no classifier can have a probability of 1.0 of being

employed and hence fixation also can never occur.

Property 3:

The taxation element of the BBA causes a rule's strength to decay exponentially over time, and as such, causes greater significance to be placed on more recent pay-off information.

Property 4:

All classifiers are each assigned an initial strength for use with the BBA and therefore, any subsequent selection is a function of these initial values.

Property 5:

The fact that the actual pay-off a classifier receives is one of the parameters in the updating equation shows that the BBA is indeed a function of actual payoffs.

Thus, the BBA satisfies Harley's five properties for an ES learning rule. Of course, the BBA is not the only ES learning rule. Much more sophisticated and efficient rules are likely to exist for many problems. However, as the effectiveness of an ES learning rule is improved, its generality is reduced and its usefulness, in a general learning environment wherein a variety of different problems are encountered, would be limited.

3.2.5 HDCS: A Classifier System for the Hawk-Dove Game

If the BBA is, for a specific application, an example of an ES learning rule, then it should be possible to discuss a plan developed by a classifier system in terms of evolutionary stability. To determine whether a classifier system using the BBA does indeed exhibit such a characteristic, a simple classifier system for the hawk-dove game, HDCS (Figure 3.3), was developed.

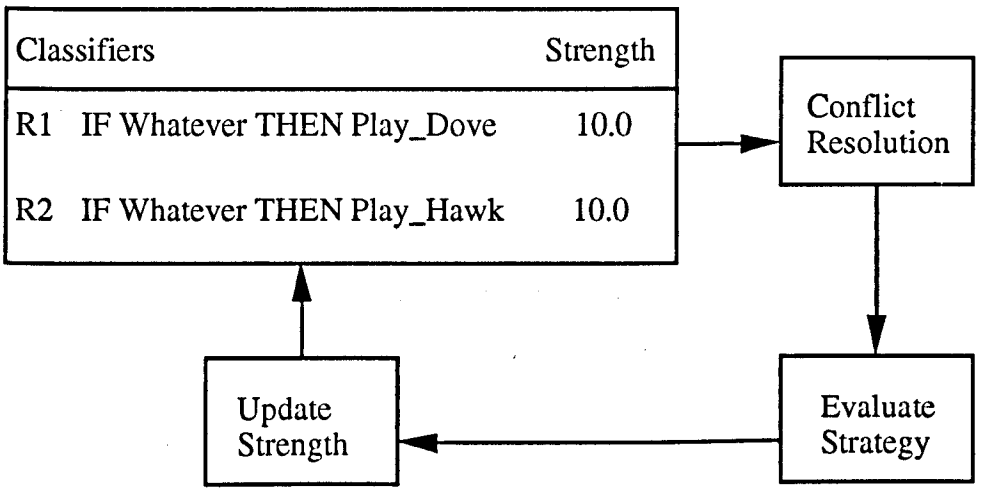


Figure 3.3: Outline of HDCS, a classifier system for the Hawk-Dove Game

As a rule base, HDCS was invested with the plan P_{HDG} described earlier, and both rules were assigned an initial strength of 10.0. Since HDCS's environment would always be the same, and only a single action would ever be performed on any problem solving cycle, those of its features which made use of internal and environmental messages, such as matching and message passing were rendered superfluous and were therefore omitted. This situation also obtained for the system's evolutionary algorithm.

HDCS was applied to 1000 instances of the hawk-dove game. At the start of each game instance, one of P_{HDG}'s two rules was selected stochastically in accordance with their respective strengths. When selected, the rule, R say, paid 10% of its strength (equivalent to its bid) to the environment. The action corresponding to R was then adopted in contests with 50 opponents whose respective strategies were selected in proportion to the strengths of the two strategies in the extant plan. For example, if the extant relative strength of rule 1 (play a hawk) were 0.6, then the system would adopt this rule on approximately 60% of occasions, and in the 50 consequent contests, the opponent would play a hawk in approximately 30 of them and dove in the remainder. The value of the pay-offs awarded at the end of each game were those detailed in section 3.2.2 but with each value being supplemented by 100 points in order to ensure that all pay-offs were non-negative (as demanded by Harley, see section 3.2.4). (It

should be noted that only the pay-off to the 'system player' needs to be taken into account.) Games wherein both protagonists played a hawk or both played a dove, were decided randomly.

The relative strength of rule 1 of P_{HDG} (that is, play hawk) was recorded at the conclusion of every tenth game instance. Figure 3.4 is a plot of the values derived against number of game instances. It can be seen from the plot that the strength of the two rules oscillates about the ESS equilibrium point. Correspondingly, given the finiteness of the population size and the number of game instances, it is not unreasonable to conclude that HDCS is playing the ESS for the problem.

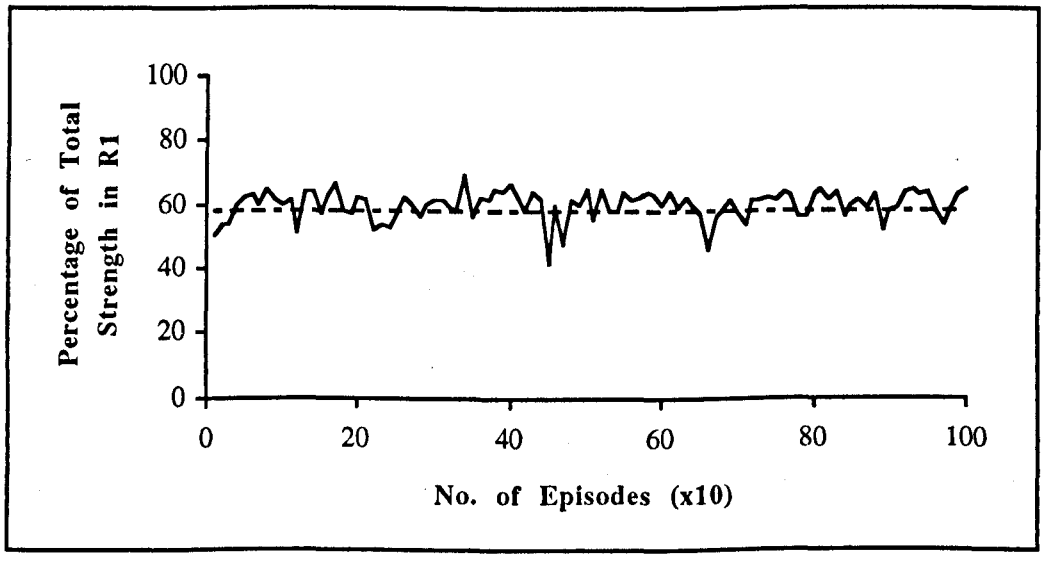


Figure 3.4: Results of Hawk-Dove Game using HDCS

Thus, it has been proved that, provided the assumptions detailed in section 3.2.4 obtain, the Bucket Brigade Algorithm constitutes an ES learning rule, and that the plan of a classifier system which utilises the BBA will evolve in such a way as to constitute an ESS for the problem to which it is applied. It should be noted however, that this result is valid only for a 'static' plan, that is, one which is not changed as the result of applying an evolutionary algorithm. For, in a classifier system's regime, the evolutionary stability built up in the system's extant plan will likely be disrupted by the change in constitution of the plan that will

almost certainly be the result of applying the evolutionary algorithm. Notwithstanding, the characteristics of a plan in respect of evolutionary stability, both in between consecutive applications of an EA, and as ultimately developed, will, if the assumptions of 3.2.4 are valid, tend to conform to that of an ESS.

In relation to the proposed GBML system, this result means that a classifier system architecture employing a BBA for credit assignment is, in itself, unlikely to derive rule strengths which accurately reflect the true utility of rules. Instead, rule strengths are likely to conform to an ESS and useful rules may not be readily identifiable. This observation affected the design of the proposed system by suggesting that additional mechanisms are likely to be needed if the system is to successfully learn useful plans for all but the simplest problems. Such an additional mechanism is an EA at the plan level and this, together with a number of other mechanisms is discussed in more detail in Chapter 5.

3.3 Practical problems with credit assignment

3.3.1 Parallel rule activation

Of the many difficulties associated with the 'standard' BBA, only two, the allocation of credit to stage setting classifiers where long chains are involved [Riolo, 87a, 89a] and the formation of default hierarchies [Riolo, 87b, 89b], have received much attention from within the EA community. One of the outstanding problems which appears to be very poorly investigated is that of the formation of *parasite*[8] rules when several rules are active simultaneously. (For the purpose of this work, simultaneously active rules will be referred to as *parallel* rules). Parasite rules are rules which, although detrimental in their effect, receive a high degree of environmental payoff by virtue of being active on the same occasions as useful rules. Therefore, due to the parasite rule 'stealing' some of the reward which should have gone to the

[8][Westerdale, 91] uses the term 'Cannibal' rather than parasite, whilst Goldberg refers to them as 'freeloaders'.

useful rule, the strength of that useful rule is lower than it should have been (possibly leading to its deletion from the rule base) whilst that of the parasite is inflated, resulting in it being employed more often than it should with the consequence being a reduction in the performance of the classifier system. Several authors (see, for example, [Wilson, 89] and [Westerdale, 89]) have briefly alluded to this problem but to the authors knowledge, no detailed analysis of the subject has been performed. Since at any one time there are likely to be a fairly large number of classifiers active simultaneously, it was felt that this was a very important problem for the credit assignment algorithm to deal with and so a detailed investigation into the effect of parallel rules on BBA effectiveness was instigated.

This investigation was carried out by analysing the effect on the BBA of three of the most common types of parallel rule, namely:

(a) *duplicate rules*;

(b) *subsuming / subsumed rules*;

(c) *equivalent rules*.

These three types of rule, referred to collectively as *redundant*[9] rules, are defined as follows: Given a plan containing 2 rules, A and B, with identical actions, then:

A is a *duplicate* of B if the condition parts of A and B are identical.

A *subsumes* B if A and B are not duplicates, B is active $\Rightarrow$ is active but A is active does not $\Rightarrow$ is active. If A subsumes B, then A is called the *subsuming* rule and B the *subsumed* rule.

[9]The term 'redundant' is borrowed from [Westerdale, 89] where it is used to mean mostly redundant, not necessarily completely redundant.

A and *B* are said to be *equivalent* if *A* and *B* are not duplicates and *A* is active $\Leftrightarrow$ *B* is active.

The nature of duplicate and subsuming rules is obvious but as that of equivalent rules may not be, the following example is given to help clarify the definition. Given a plan for an aircraft on a bombing mission, two possible rules (represented here in a high level representation for clarity) are:

Rule A: IF radar working AND time to target < 2 mins
THEN set course for target

Rule B: IF no crew members injured AND time to target < 2 mins
THEN set course for target

Here the only difference in the two rules are the conditions 'radar working' and 'no crew members injured'. These conditions will both be active for the majority of the time, and even when one is not active, such as when an aircraft takes a direct hit, the other is also unlikely to be active. Therefore, over the limited number of trials which a classifier system would have to evaluate the two rules, they are likely to be active on identical occasions. Hence, although the two rules are clearly different, they are said to be equivalent.

In order to investigate the effect that these three types of rule have upon the distribution of credit in a classifier system, a simple test system, SCS1, was developed. Moreover, the test problem upon which the investigation would be based was chosen to be a simple 2-player military game called the *Two Tanks Problem* (TTP). This problem, and the architecture of the test system used to manipulate it, are described below.

3.3.2 The Two Tanks Problem

In its most general form, the problem is set in three dimensions and requires a tank, *T1*, armed with but a single gun, to destroy another tank *T2* possessing identical capabilities. In the simplified version of the problem adopted for the work reported here, the setting was restricted to a 'one-dimensional board' composed of an infinite strip of identical contiguous squares. Movement of the tanks was also restricted to one square backwards or forwards per 'move' and gun movement to increasing/decreasing elevation by one position or notch - equivalent to increasing/decreasing the gun range by one square. In addition, all other complicating features of the more general problem, such as limits on the ammunition available and weather conditions, were removed from consideration.

An episode of the TTP commences with the two tanks in their initial positions, placed randomly at a distance of $10 \leq x \leq 20$ units apart. The tanks then take alternate moves (*T1* takes the first), and on each of its moves, a tank may:

- move forward or backward 1 square;
- fire;
- do nothing;
- increase or decrease gun elevation by 1 position or notch.

The episode ends when one of the two tanks hits its opponent (achieved by firing when the inter-tank gap equals the range of the guns), or when t moves (where t is a preset maximum for the length of an episode) have been made without a hit being registered.

Given this general framework, two slightly different TTPs were used for testing various problems associated with parallel rule activation. The first of these, TTP1, was designed to enable a random strategy to have a fairly high probability of achieving its goal, thus allowing the initial population to gain some credit from the environment. In contrast to this, the second test problem, TTP2, was designed such that a random strategy has a negligible chance

of being successful, with the consequence being that the system would initially have to discover useful rules with very little environmental information to go on. The two problems are defined as follows:

(a) Test Problem 1 (TTP1)

In this, the firing range of *T1* is set to $x \pm i$, where i is selected at random to be either 1, 2 or 3, and *T2* is given an identical range. The maximum number of moves available during any one episode is set to 60;

(b) Test Problem 2 (TTP2)

For this problem, the firing range or *T1* is set to x-2, and the maximum number of moves allowed during an episode is set to 5.

3.3.3 System Architecture

Figure 3.5 shows the central part of SCS1, the rule and message (RaM) system. This is very similar to that of a standard classifier system described in Chapter 1. The RaM operates cyclically with each cycle being equivalent to a move in the TTP.

At the start of each cycle, the *world model* is inspected. The world model is one of the problem dependent parts of the system and is simply a list of all the parameters which fully describe the state of a game (episode). For example, for the TTP this may contain the distance between the two tanks, their respective ranges, and so on. A detector converts this information into an *environmental message* which can then be used by the rest of the system. This environmental message is a six bit binary string with each bit corresponding to a different property of the world model. Bits 1 and 2 (the leftmost bits) are set to 00 to identify the message as coming from the environment, with the remaining bits being set as follows:

bit 3 = 1 if *T2* altered elevation, and 0 otherwise;

bit 4 = 1 if *T2* moved, and 0 otherwise;

bit 5 = 1 if *T2* did nothing, and 0 otherwise;

bit 6 = 1 if *T2* fired, and 0 otherwise.

After reading the environmental message, the RaM adds it to the systems internal message list where it replaces the previous environmental message (if one is present).

On all but the first cycle, also present on the message list will be *internal messages* which were posted by rules that were active on the previous cycle. Internal messages facilitate inter-classifier communication, and as such, provide a one move memory for the system. They specify an action that may be performed by *T1*, and as with environmental messages, they are six bits long. The first two bits are set to 01 to identify the message as being internal, bit 3 is redundant and always set to zero, and bits 4 to 6, which represent the six possible actions, are set as follows:

001 - move forward 1 square;

010 - move back 1 square;

011 - fire;

101 - increase elevation by 1 position;

110 - decrease elevation by 1 position;

000 - do nothing.

The messages on the message list are then compared against the conditional parts of the classifiers in the extant rule base using the MATCH routine. The classifiers used by SCS1 have the standard <condition part>:<action/message part> form, where the <condition part> consists of 1 or 2 conditions relating to one or both types of message. A condition is represented as a string of length six over the alphabet {0, 1, #}, where # is a wildcard ('don't care') symbol.

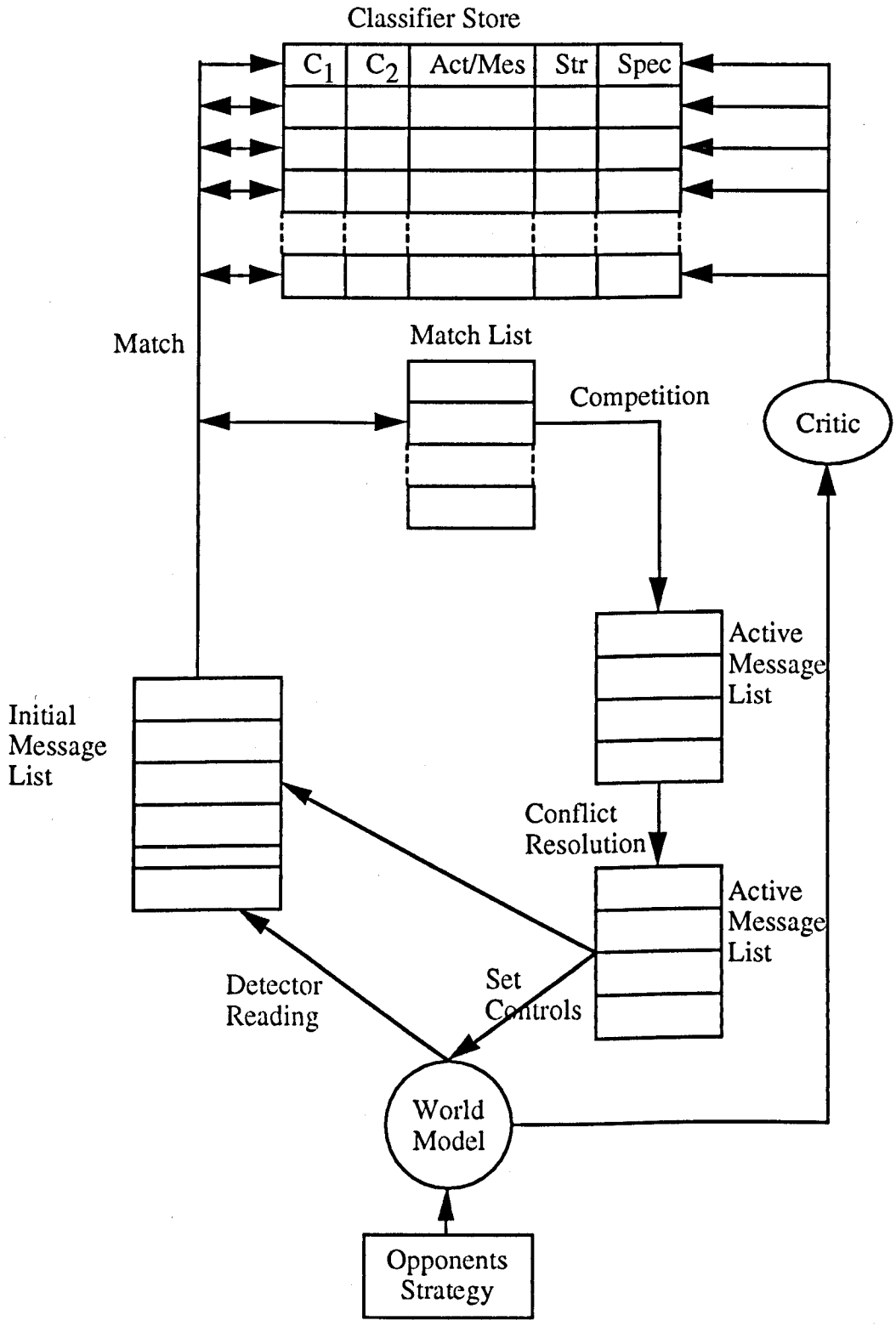


Figure 3.5: Rule and Message (RaM) system in SCS1

The <action/message part> is an internal message which specifies the action to be performed when the condition(s) in the <condition part> is/are satisfied. Two examples of rules of this form are 000001:010110 and 000100, 010001:010011 and these should be interpreted respectively as:

IF *T2* fired THEN decrease elevation;

IF *T2* moved AND *T1* advanced THEN fire.

In the matching phase, a classifier is 'matched' if, for each condition, *C*, in its <condition part>, there exists a message, *M*, on the message list, such that $C_i = M_i$ or C_i = #, where subscript i denotes bit i, i=1, ..., 6. A list is kept of all matching classifiers and upon completion of the matching phase, a competition or *auction* is held to reduce their number to the size of the message list. This is achieved by first calculating a bid, bid_j, for each competing classifier, j, as follows:

$$bid_j = 0.01 * b * specificity_j * strength_j$$

Here, b - the *bid coefficient*, is a constant, $specificity_j$ is a measure of j's generality (calculated as the percentage of non-wildcard characters in j's conditions), and $strength_j$ is the systems current estimate of j's utility. n classifiers, where n is the size of the message list, are then selected for inclusion in the message list. This selection is either stochastic with each classifiers selection probability being proportional to its bid, or deterministic, with the choice of selection scheme depending upon a system parameter, AUCTION_SCHEME.

Some of the selected (*posted*) messages may specify actions which conflict, and correspondingly, a process of *conflict resolution* is performed in order to ensure that the messages sent to the output interface (these specify the action *T1* is to perform) are consistent.

This involves calculating the cumulative bid, ξ, associated with each different action specified by the messages on the message list, and selecting that action whose associated value of ξ is largest. Again, this selection process may be stochastic or deterministic, and is determined by the parameter CONF_RES_SCHEME. Subsequently, all messages whose actions are not consistent with the one chosen are deleted from the message list, while those which remain repay their 'suppliers' by paying their bid to the classifiers which sent the messages which matched their conditions, as in the explicit Bucket Brigade Algorithm. (In the special case of the environmental message being responsible for activating a classifier, payment is made to the environment.) As well as paying their bid, those classifiers which remain after conflict resolution also pay a *bid tax* for making their bid. Moreover, all classifiers, irrespective of whether they are active or not, pay a life tax (fixed at 1% of their strength) to the system, thereby reducing the strength of rules which are never active.

After an action for *T1* has been selected, it is passed to a control setting routine which decodes the message and subsequently updates the world model. *T2* then makes its move according to a fixed strategy encoded in the OPPONENT'S STRATEGY module and the world model is correspondingly updated. In the experiments described here, the opponents strategy remained fixed at one of firing on every move. The time step counter is then incremented by one and this marks the end of a cycle.

This cycle is repeated a number of times until the end of the episode is reached. For the TTP, this occurs either when one of the tanks is hit or when the number of cycles (time steps) has passed a fixed maximum. Upon the end of the episode being reached, a further problem dependent routine, the CRITIC, is invoked, and this assigns a reward, R, to those classifiers active at the end of an episode. The value of R was set as follows:

$$R = \begin{cases} 10.0 & \text{if T1 wins} \\ Q * 10.0 & \text{if neither tank wins before time elapsed} \\ 0.0 & \text{if T2 wins} \end{cases}$$

where Q is a constant called the *draw rating* whose value, unless otherwise stated, was set to 0.5.

3.3.4 An Investigation into the effect of redundant rules on the bucket brigade algorithm

3.3.4.1 Default settings

In section 3.3.1, it was noted that little work had been carried out to determine the effect of redundant rules upon the BBA. The experiments reported in this section aim to investigate these effects in relation to each of the three types of redundant rule identified in section 3.3.1 (namely duplicate, subsuming and equivalent rules). The experiments were carried out using SCS1 with the default parameter settings shown in Table 3.1.

Table 3.1: Default parameters for SCS1.

Parameter	Value
Initial Rule Strength	10.00
Bid Co-efficient	0.10
Bid Tax	0.00
Life Tax	0.01
Payoff	10.00
Draw Rating	0.50
Message List Size	6
Auction Scheme	Deterministic
Conf Res Scheme	Deterministic

3.3.4.2 Experiment 1: Tests involving duplicate rules

To serve as a control experiment, SCS1 was invested with a simple plan, P_1, containing but the single rule, A:- 00#### : 010011, (fire on every move) and applied to a test set S of 60 instances of test problem TTP1. The resulting value of A's strength was found to be Σ = 0.6247.

P_1 was then replaced, in turn, by plans P_2,P_3,...P_8, and the experiment repeated under

conditions of no competition, that is, using a message list sufficiently large to hold all simultaneously active classifiers. Here, P_i denotes a plan containing i copies of A alone. Each experiment gave analogous results, namely and perhaps not surprisingly, that in using plan P_j, each of the j copies of A acquired a strength of Σ/j, the combined strength of the copies equating with the strength of A in the control experiment.

Although indicating that the presence of duplicate rules can significantly distort rule strength, the above experiment is somewhat unrealistic in that it did not involve the element of competition which normally arises as a result of both finite message list size and other active classifiers. Correspondingly, to investigate the influence of message list size, SCS1 was invested with plan P_6, a message list of size k and, for k=2, ..., 5, applied to the problem set S (the cases k=1 and k=6 replicate the control experiment and that involving P_6 with no competition respectively).

It had been anticipated that, with a message list size of k, (6-k) of the classifiers would achieve a strength of zero and each of the remaining classifiers a strength of Σ/k. Such proved to be the case. Thus, it can be said that reducing the size of the message list does reduce the distortion in rule strength caused by duplicate rules. Unfortunately, however, distortion will not be eliminated completely unless a message list of size one is adopted - and this would be too limiting for many applications.

In respect of competition from other classifiers, a variety of different plans were tested using the problem set S. Typical of the results obtained are those which were derived from the plan P_k^* containing the 7 rules given in Table 3.2 together with k duplicate copies of A, k=0, ..., 9. These results, obtained under conditions of no competition in respect of message list size, are depicted in Figure 3.6.

*Table 3.2: The Plan P_k**

No.	Classifier	Rule	
		Condition	Action
1	00#### : 010011	whatever T2 did	fire
2	01#### : 010011	whatever T1 did	fire
3	00#0##, 01#011 : 010101	T2 didn't move AND T1 fired	increase elevation
4	00#0##, 01#011 : 010110	T2 didn't move AND T1 fired	reduce elevation
5	01###0 : 010011	T1 didn't fire	fire
6	01##0# : 010011	T1 didn't fire	fire
7	000010 : 010011	T2 did nothing	fire

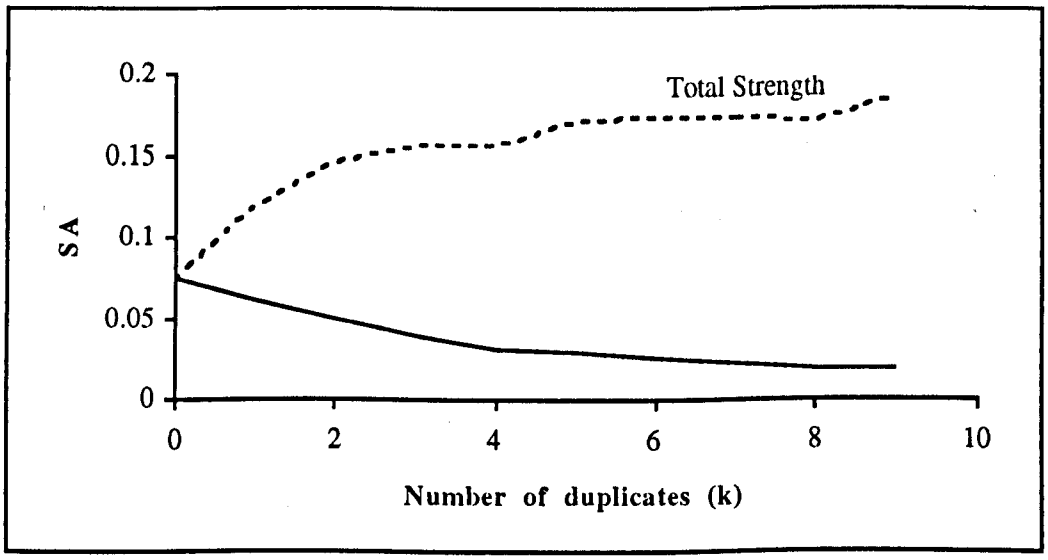


*Figure 3.6: Total strength allocated to duplicates of A in P_k**

In every case (k=1, ..., 9), the strength of each copy of A in P_k* was found to be the same and reduced below that of rule A, Σ^* say, in the control experiment (k=0). However, the extent of the reduction was greater than that observed in the experiment using the corresponding plan P_k. Thus, σ_k, the combined strength of the individual copies of A in P_k*, falls below Σ^* (σ_k does increase with increasing k but never reaches Σ^*). This is explained as follows. When only A and its k duplicates are active on a given cycle, each of these rules receives $1/(k+1)$th of

the pay-off. However, rules other these will also be active on some cycles, and hence the pay-off will be divided equally between A, its duplicates, and these other active rules. On such cycles, the rules differing from A are, in terms of the effects of pay-off, acting as if they were duplicates of A!

The distortion of rule strength deriving from the presence of duplicated rules within a plan is undesirable. Fortunately, this problem can easily be circumvented by checking any newly generated rule against the extant rule-set prior to that rule's assumption into the set. More problematic, however, is dealing with new rules which turn out to subsume, be subsumed by, or be equivalent to, rules already present in a plan.

3.3.4.3 Experiment 2: Tests involving equivalent rules

Consider two equivalent rules, R_1 and R_2, in some plan. Let S_i, $spec_i$, and $payment_i$ denote respectively the strength and specificity of rule R_i, and the payment made to R_i after a successful bid, i=1, 2. During an episode which lacks competition, R_1 is active $\Leftrightarrow$ R_2 is active and in the steady state, $payment_1 = payment_2$, hence:

$$b * spec_1 * S_1 = b * spec_2 * S_2$$

and thus

$$S_1 / S_2 = spec_2 / spec_1 \qquad \text{(Eq. 3.1)}$$

This simple analysis suggests that the ratio of the strength of two equivalent rules might be expected to be the inverse of the ratio of their specificities. To investigate this, a simple plan, denoted P_E, was applied to the problems in test set S, and using the default parameters (Table 3.1) but with the length of each episode constrained to L cycles, L=1, 2, ..., 200. P_E contained the three rules:

X :- 01#0## : 010011

Y :- 010#11 : 010011

Z :- 000010 : 010011

Here, rules *X* and *Y* are equivalent and rule *Z* is only active on the first cycle of an episode.

The results derived using P_E are depicted in Figure 3.7, a plot of relative strength (S_X/S_Y) versus *L*. The shape of the graph does not accord with the above analysis but may be understood if it is compared with the graph of Figure 3.8 which was obtained by repeating the experiment but allocating no pay-off at the end of any episode. From these graphs it may be readily concluded that it is system pay-off which most radically influences the ratio of the rule strengths and that, as a result of the effects of taxation, its influence increases with the length of episode such that $S_X/S_Y \rightarrow 1$.

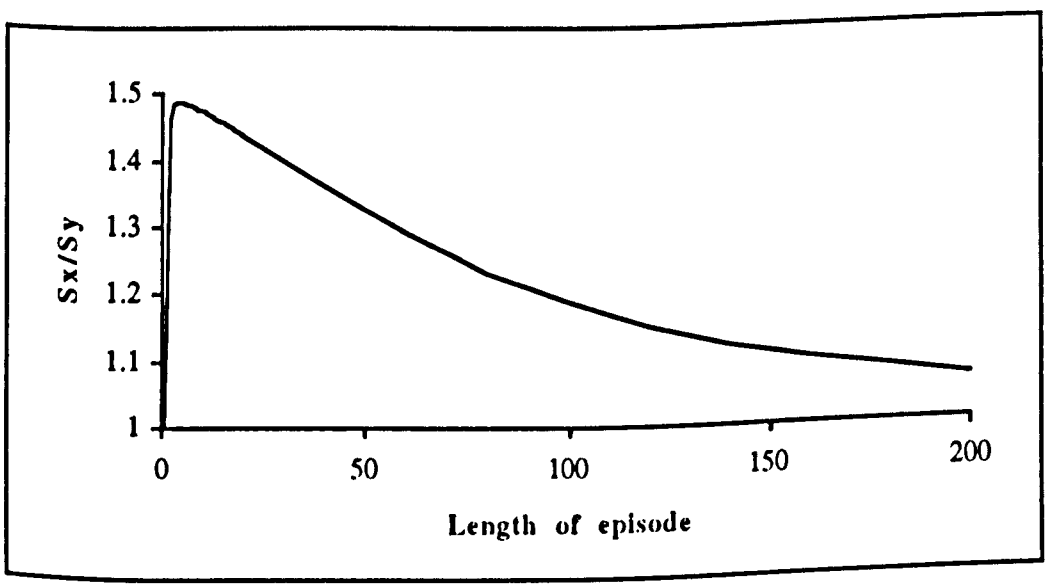


Figure 3.7: Ratio of S_X to S_Y for the plan P_E

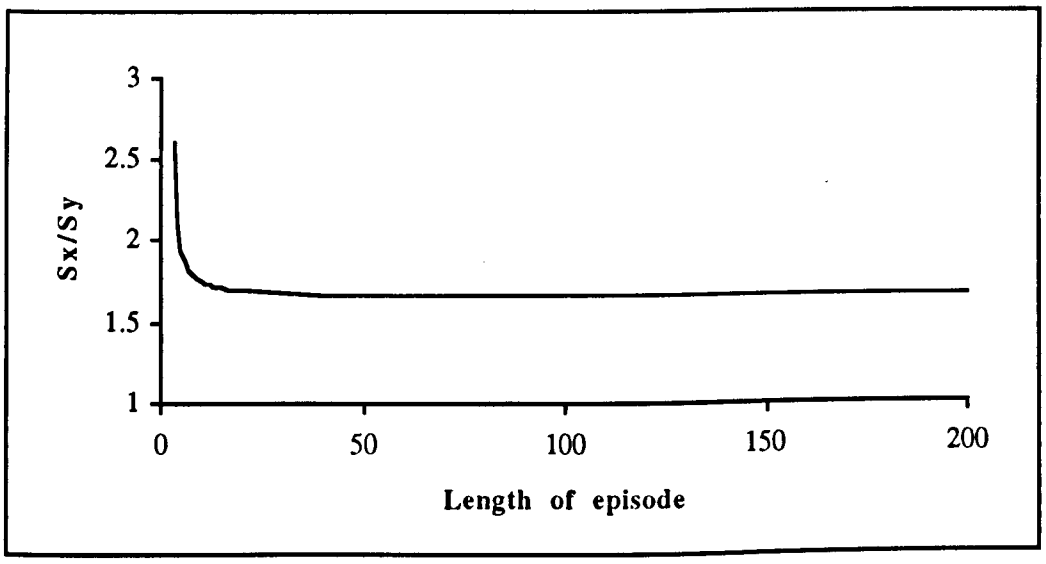


Figure 3.8: Ratio of S_X to S_Y when payoff=0.0

3.3.4.4 Experiment 3: Tests involving subsuming rules

Defining the nature of the influence of coexistent subsumed and subsuming rules on the 'true strength' of rules in a plan is a far more difficult task than the analogous ones involving duplicate and equivalent rules. The reason for this is the potential for complex relationships to exist between several rules, each pair of which is linked by subsumption. However, by looking at fairly simple cases, we can get some idea of how the BBA distributes strength between them.

To serve as a control experiment for subsuming rules, SCS1 was invested with the plan P_S which contains no simultaneously active classifiers. Table 3.3 shows P_S and the cycles (moves) during an episode when each rule is active.

Table 3.3: The plan P_S

No	Classifier	Action	Active Cycles
1	000010 : 010011	fire	1
2	010011 : 010110	move forward	2,4,6,...
3	010110 : 010011	fire	3,5,7,...

As there is no competition between any of the rules in P_S, there is only one possible action sequence, namely "Fire, Forward, Fire, Forward, Fire ..." which continues unchanged until either the end of the episode is reached or the opposing tank is hit.

P_S was applied to the problems in test set S using the default parameters (Table 3.1) and the results shown in Table 3.4.

Table 3.4: The plan P_S after application to test set S

No	Classifier	Abs. Strength	Rel. Strength
1	000010 : 010011	4.19	0.07
2	010011 : 010110	15.6	0.28
3	010110 : 010011	36.25	0.65
	No. Wins 26		
	No. Draws 34		

To investigate how the BBA distributes strength between rules linked by subsumption, P_S was in turn replaced by plans P_{T1}, P_{T2}, and P_{T3}, all of which perform the same action sequence as P_S and are shown in Tables 3.5 to 3.7.

Table 3.5: The plan P_{T1}

No	Classifier	Action	Active Cycles
1	000010 : 010011	fire	1
2	010011 : 010110	move forward	2,4,6,...
3	0#0#10 : 010011	fire	1,3,5,7,...

Table 3.6: The plan P_{T2}

No	Classifier	Action	Active Cycles
1	0#0#10 : 010011	fire	1,3,5,7,...
2	010011 : 010110	move forward	2,4,6,...
3	010110 : 010011	fire	3,5,7,...

Table 3.7: The plan P_{T3}

No	Classifier	Action	Active Cycles
1	000010 : 010011	fire	1
2	010011 : 010110	move forward	2,4,6,...
3	010110 : 010011	fire	3,5,7,...
4	0#0#10 : 010011	fire	1,3,5,7,...

In P_{T1}, R_3 from P_S is generalised so that it is also active on the first cycle, therefore subsuming R_1. P_{T2} has R_1 generalised so that it subsumes R_3, and to P_{T3}, a further rule R_4 is added which subsumes both R_1 and R_3.

Tables 3.8 to 3.10 show the strengths of the rules in plans P_{T1}, P_{T2} and P_{T3} when run on the same test set, S, as P_S.

Table 3.8: The plan P_{T1} after 60 episodes

No	Classifier	Abs. Strength	Rel. Strength
1	000010 : 010011	1.74	0.03
2	010011 : 010110	11.06	0.22
3	0#0#10 : 010011	37.04	0.74
	No. Wins 26		
	No. Draws 34		

Comments: When P_{T1} is compared to P_S, rule 1 has had its strength roughly halved, R_2 has had a slight reduction in strength (due to it receiving less from R_3) and the generalised R_3 shows a slight increase in strength.

Table 3.9: The plan P_{T2} after 60 episodes

No	Classifier	Abs. Strength	Rel. Strength
1	0#0#10 : 010011	21.09	0.4
2	010011 : 010110	14.23	0.27
3	010110 : 010011	18.03	0.34
	No. Wins 26		
	No. Draws 34		

Comments: When P_{T2} is compared to P_S, R_2's strength shows a slight reduction but that of R_3 has been approximately halved.

Table 3.10: The plan P_{T3} after 60 episodes

No	Classifier	Abs. Strength	Rel. Strength
1	000010 : 010011	1.94	0.04
2	010011 : 010110	13.59	0.26
3	010110 : 010011	17.93	0.34
4	0#0#10 : 010011	19.56	0.37
	No. Wins 26		
	No. Draws 34		

Comments: When P_{T3} is compared to P_S, the strengths of both R_1 and R_3 are approximately half of their value before R_4 was introduced.

All three sets of results show that strength is shared between subsuming rules for the period that they are simultaneously active (cf. equivalent rules). For example, in P_{T3} (Table 3.10) the strengths achieved by R_1 and R_3 are approximately half of those achieved by the corresponding rules in P_S. Moreover, the sum of these two 'missing halves' is approximately equal to the strength achieved by R_4, the subsuming rule.

3.3.4.5 Discussion of Results

The results in sections 3.3.4.2 to 3.3.4.4 have demonstrated that when a rule R_A in some plan is supplemented with either duplicate, equivalent or approximately equally effective subsuming rules (call these *analogues* of R_A for ease of exposition), the 'true strength' of R_A is distorted to the extent that it is shared with its analogues. Such distortion may well lead to a reduction in a classifier system's capability to produce an effective plan. However, it is not only a distortion of absolute strength that is observed, the strength of R_A relative to that of other rules in the plan will also be affected. This in turn can affect both the performance of the basic system and also that of the EA, as shown by the following example.

Consider a plan P_I which contains the two rules R_X and R_Y. In a particular state of the game, R_X and R_Y are found to be the only active rules, and in steady state, it is found that their strengths are 7.0 and 5.0 respectively. After several generations, the EA selects R_X for genetic recombination, and the resultant offspring, R_X^*, an analogue of R_X, is added to P_I. Since it has been shown that analogues tend to share strength, then the new steady state strengths of the rules would be 3.5, 3.5 and 5.0 for R_X, R_X^* and R_Y respectively. If the message list size is effectively infinite (a size of 3 would be sufficient in this case) then no competition takes place at the auction stage and the performance of the system would be largely unaffected. However, a message list size of any less than this could lead to problems. For example, a message list size of 2 would usually lead to R_Y being selected along with one of R_X and R_X^*, with the subsequent conflict resolution phase most likely to select R_Y, the worst of the three rules. Even if any such problems are avoided, there may still be difficulties when the genetic operators are applied, especially if, as suggested in Chapter 1, an incremental rather than a generational EA is used. For example, when reproduction occurs using an incremental algorithm, the member to be deleted is more likely to be either R_X or R_X^*, despite both being better than R_Y.

As this, albeit rather contrived example shows, the manner in which the BBA shares strength between analogues can cause a number of problems in a classifier system and so some way is needed of reducing this effect. With regard to this, [pp 354, Westerdale, 91] notes that "*if two classifiers have similar effects, we want the reward scheme to reward the one with the lower availability*[10] *less. However, in designing such a reward scheme we would prefer to avoid introducing detrimental biases*". Correspondingly, a number of alternative credit assignment schemes - ones which induce a relative strength of other than unity between a rule and its analogue - were investigated to determine whether their use might reduce distortion of rule strength.

[10][Westerdale, 91] uses the term 'availability' rather than 'strength'

3.4 Alternative credit assignment schemes

3.4.1 Variants of the bucket brigade algorithm

[Wilson, 89] was one of the first to investigate variants of the BBA. Whilst maintaining the standard scheme for inter-rule payment, he focused attention on environmental pay-off, but unfortunately, the informal experiments he performed on the alternative schemes considered, revealed none as being any more effective than the standard. More recently, [Fairley and Yates, 93] and [Yates and Fairley, 93] report the results from various alternatives to the standard BBA when used with the TTP. These involve dividing the inter rule payment (I) and environmental payoff (E) in ways other than equally between active classifiers. Of these alternatives, the only one which produced a significant difference from the standard BBA was one in which both I and E were divided in proportion to the bids of the receiving classifiers. Call this the Amended BBA, ABBA1. Figure 3.9 shows the distribution of strength for this scheme between the two equivalent rules, X and Y (from the plan P_E) when applied to the test set S, while Table 3.11 shows the distribution between the subsuming rules in P_{T3} when using the same test set.

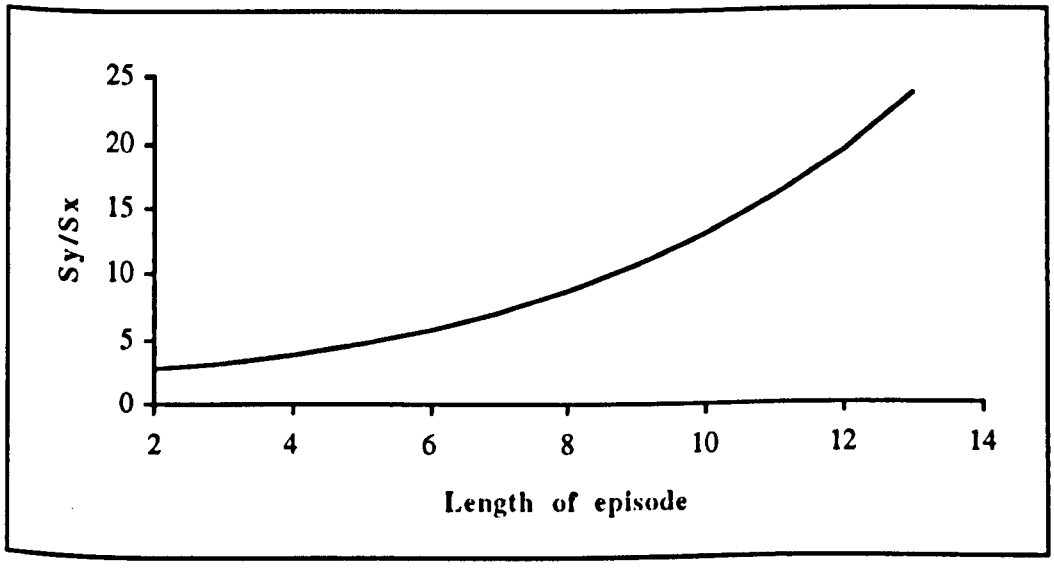


Figure 3.9: Ratio of S_Y to S_X using ABBA1

Table 3.11: The plan P_{T3} using ABBA1

No	Classifier	Abs. Strength	Rel. Strength
1	000010 : 010011	0.00	0.00
2	010011 : 010110	14.08	0.27
3	010110 : 010011	18.45	0.35
4	0#0#10 : 010011	20.59	0.39
	No. Wins 26		
	No. Draws 34		

As Figure 3.9 shows, with ABBA1, all the strength between equivalent rules goes to the more specific rule. This is due to the more specific rule having a larger bid (since *bid = strength * specificity*) and as the strengths of equivalent rules have been shown to be almost equal, then the more specific rule will receive exponentially more payment as both the length of episode and the number of episodes increases. However, with subsuming rules, the situation is more problematical since the strengths of two subsuming rules are not usually equal. For example, looking at Table 3.11, R_1 and R_3 are both more specific than the subsuming rule R_4, but the strength of R_1 is zero and that of R_3 is still at its original level. The reason for these results is as follows. Although the specificity of R_1 is larger than that of R_4, its strength is much lower and hence its bid is lower, leading to an exponential fall in payments to R_1. However, when dealing with R_3, although once again its specificity is larger than that of R_4 and its strength is lower, this time the strength differential is only slight and acts to counterbalance the difference in specificity, thus leading to results analogous to those found using the standard BBA wherein strength is shared. This can be verified if the results in Table 3.11 are compared with those achieved by plan P_{T2} using the standard BBA (Table 3.9) where rules 1, 2 and 3 correspond to rules 4, 2 and 3 in P_{T3} respectively.

The opposite effect to that achieved by ABBA1 can be observed if, rather than dividing payments in proportion to the bids (that is, *strength * specificity*), they are divided in proportion to *strength / specificity*, thus favouring less specific (more general) rules. This

scheme, ABBA2, was applied to the same test set as used for ABBA1, and the analogous results for equivalent and subsuming rules can be seen in Figure 3.10 and Table 3.12.

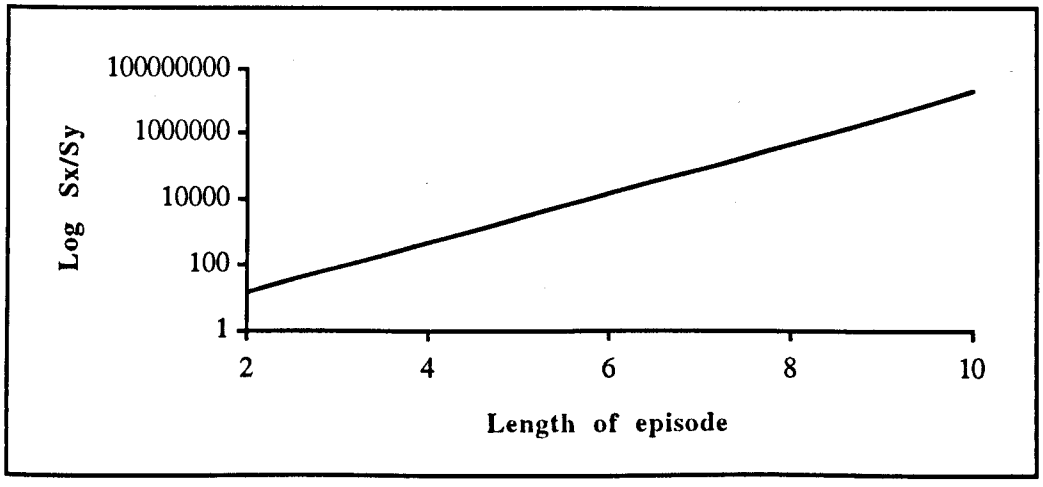


Figure 3.10: Ratio of S_X to S_Y using ABBA2

Table 3.12: The plan P_{T3} using ABBA2

No	Classifier	Abs. Strength	Rel. Strength
1	000010 : 010011	0.00	0.00
2	010011 : 010110	11.61	0.23
3	010110 : 010011	0.00	0.00
4	0#0#10 : 010011	38.49	0.77
	No. Wins 26		
	No. Draws 34		

As expected, Figure 3.10 shows the more general of the two equivalent rules receiving exponentially more strength as the time period progresses, and Table 3.12 shows R_4 gaining all the strength at the expense of rules 1 and 3 due to both its lower specificity and its greater strength.

To investigate whether adopting either of these new schemes could give results that improve upon those derived using the standard BBA, each new scheme was in turn applied to the plan P_k*(Table 3.2) on a more extensive test set, S' containing 300 problems of type TTP2.

Tables 3.13 to 3.15 show the plans developed by each of the three schemes, and Figure 3.11 shows the respective performances during the development of each of the three plans. The parameters used were the same as the default set (Table 3.1) except that both the auction and conflict resolution schemes used were stochastic so that most of the possible sequences of rules would be sampled.

Table 3.13: using the standard BBA

No.	Classifier	Abs. Strength	Rel. Strength
1	00#### : 010011	3.34	0.33
2	01#### : 010011	3.34	0.33
3	00#0##, : 01#011 : 010101	0.03	0.00
4	00#0##, : 01#011 : 010110	0.00	0.00
5	01###0 : 010011	0.00	0.00
6	01##0# : 010011	3.37	0.33
7	000010 : 010011	0.00	0.00

Comments: As expected, when using the standard BBA, strength is shared between the two default rules, R_1 and R_2, as they are almost equivalent. Thus, previous results (section 3.3.4.3) indicating that rule strength is shared between equivalent plans are confirmed in the context of a more complex plan.

Table 3.14: I and E divided in proportion to the bid (ABBA1)

No.	Classifier	Abs. Strength	Rel. Strength
1	00#### : 010011	0.34	0.18
2	01#### : 010011	0.00	0.00
3	00#0##, : 01#011 : 010101	0.75	0.41
4	00#0##, : 01#011 : 010110	0.00	0.00
5	01###0 : 010011	0.00	0.00
6	01##0# : 010011	0.76	0.41
7	000010 : 010011	0.00	0.00

Comments: Again, as predicted by the earlier experiments, the strength accorded to the two default rules is lodged in only one of them, in this case R_1, because it makes a slightly larger

profit by being active on the first move of an episode. Furthermore, two of the more specific rules (rules 3 and 6) have achieved a much larger percentage of the total strength (82%), thus making the plan more successful than one employing only the default rules.

Table 3.15: I and E divided in proportion to Strength/Specificity (ABBA2)

No.	Classifier	Abs. Strength	Rel. Strength
1	00#### : 010011	3.73	0.11
2	01#### : 010011	0.00	0.00
3	00#0##, : 01#011 : 010101	3.56	0.10
4	00#0##, : 01#011 : 010110	0.00	0.00
5	01###0 : 010011	0.00	0.00
6	01##0# : 010011	26.81	0.79
7	000010 : 010011	0.00	0.00

Comments: Again, the distribution of strength is in agreement with predictions from earlier experiments, with all the strength accorded tot he two default rules residing in only one of them. Once again, a high percentage of the total strength (89%) is lodged in the more specific rules (rules 3 and 6), thus making the plan more successful than the plan derived using the standard BBA.

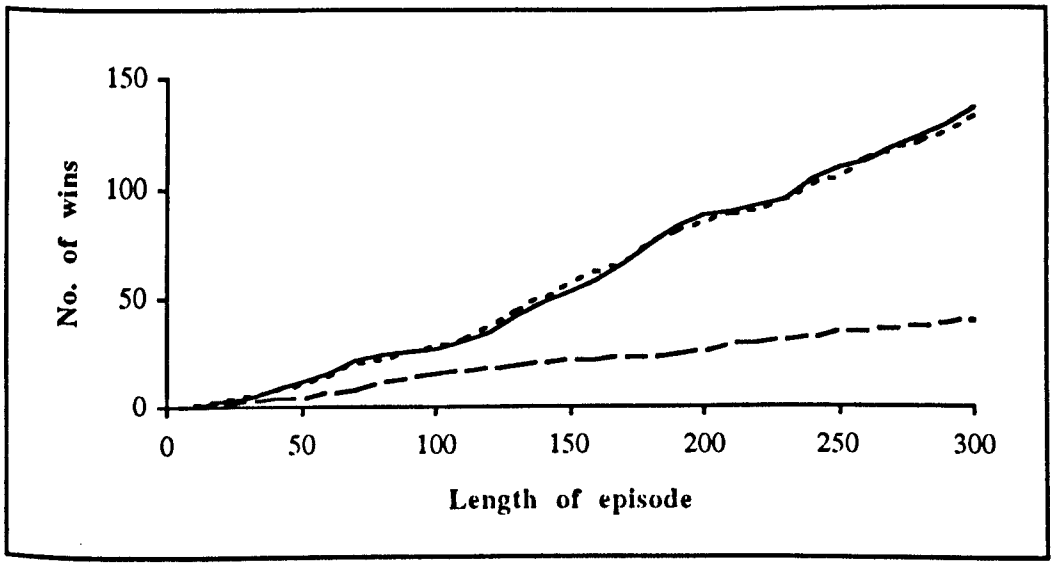


Figure 3.11: Performance of the standard BBA (long dash), ABBA1 (solid) and ABBA2 (short dash) on test set S'

The results depicted in Figure 3.11 show that the plans developed by both of the variants of the standard BBA perform significantly better (approximately 3 times better) than that developed by the standard scheme, Moreover, both variants appear equally suitable for the particular problem tried, namely TTP2, with neither out-performing the other.

3.4.2 An adaptive Bucket Brigade Algorithm

With the success of both ABBA1 and ABBA2, one question which suggests itself is "which one is the more suitable for rule discovery in a classifier system?" To help answer this, it is necessary to analyse how an ideal plan will evolve in a classifier system.

Assuming that the initial population is generated randomly, the first few generations are likely to perform fairly poorly but after a while, it would be expected that a small number of reasonably effective rules would emerge. However, these rules are likely to be useful in only a small percentage of their activations. For example, a suitable rule for a particular plan may be 'if (*range*, 10, 20) then *fire*' and after a number of generations, a rule such as 'if (*range*, 18, 32) then *fire*', which works effectively on 20% of occasions, may be generated. In such circumstances, it would not be unreasonable to favour a credit assignment scheme which has a bias towards specialization. Clearly, ABBA1 is suitable for use in this situation.

With the bias provided by ABBA1, more specific rules such as 'if (*range*, 18, 20) then *fire*' will be favoured, and so the number of instances where the useful rules perform poorly is likely to be reduced. Consequently, it would be expected that system performance would rise. As the performance rises, it may be preferable to start to generalise some of the effective rules to cover a greater number of situations. For example, by generalising 'if (*range*, 18, 20) then *fire*', it is possible to reach the goal state of 'if (*range*, 10, 20) then *fire*'. Clearly, in these circumstances, ABBA2 will be the more suitable scheme.

Consequently, an *adaptive BBA* which utilises aspects of both ABBA1 and ABBA2 could be extremely beneficial in much the same way that an adaptive mutation rate was shown to be beneficial when solving the knapsack problem (Chapter 2). For an adaptive scheme, some measure of system performance, ϑ, is required. One example of such a measure for the

TTP is a time weighted average of the number of successful episodes during a run. By subsequently normalising this value to a real number on the range -1.0 to +1.0, where -1.0 represents 100% success and +1.0 represents zero success, an adaptive scheme can be achieved by dividing both *I* and *E* in proportion using the following formula:

$$strength * (specificity)^{\vartheta}$$

With ϑ initially set to +1.0, the scheme will act like ABBA1 and give preference to more specific rules. Then, as the generations progress and system performance improves, ϑ will be reduced and the adaptive scheme will change to give a greater bias towards general rules. However, if at any stage overgeneralisation occurs, the performance level will fall with the consequence that, once again, more specific rules will be favoured.

Unfortunately, to evaluate such a scheme, the rule discovery element has to be incorporated into the system. This is the task of the evolutionary algorithm which is the subject of the next chapter. Therefore, a comparison of this adaptive scheme with the other BBAs is left until then.

3.4.3 Other schemes

In Chapter 1, it was mentioned that although the BBA is the most commonly used credit assignment scheme in classifier systems, a number of alternative schemes exist to perform the same task. These include the *epochal algorithm* [Holland and Reitmen, 78], the *Profit Sharing Plan* [Grefenstette, et al, 90] and *Backward Averaging* [Twardowski, 93]. All of these have both advantages and disadvantages when compared to the BBA, with the general conclusion being that no one method is likely to be best for all possible problems. However, a number of authors (see, for example, [Grefenstette, 88] and [Liepins, et al, 89, 91]) have suggested various hybrid schemes in which the advantages of both the BBA and one or more of these alternative schemes are combined to form a superior algorithm.

To investigate whether any of these other algorithms could be hybridised in a beneficial manner with the variants of the BBA described earlier, a series of simple experiments similar to those of the previous sections were performed. Six alternative schemes were implemented in these experiments and compared to the BBA, ABBA1 and ABBA2 algorithms respectively.

(a) Simple Epochal Algorithm (SEA)

In epochal credit assignment algorithms, the system maintains a record of each rule which has been active during a problem solving episode (*epoch*) and upon conclusion of the episode, each activated classifier is awarded a fixed percentage of the environmental pay-off, P. In the simple epochal algorithm (SEA) implemented here, the strength of each rule is updated at the end of an episode using the following equation:

$$S_i'(\alpha) = S_i(\alpha)(1-c) + cP$$

where $S_i(\alpha)$ and $S_i'(\alpha)$ represent the strength of rule i before and after episode α respectively, and c, is a constant (set to 0.1) which has the same role as the bid coefficient in the BBA. [Grefenstette, 88] showed that such a scheme computes a time weighted estimate of the expected external pay-off for each rule and that this is useful for conflict resolution.

(b) Backward Averaging (BA)

Backward Averaging is probably the most common epochal-type algorithm to be used in classifier systems (see, for example, [Holland and Reitman, 78], [Liepins et al, 91] and [Twardowski, 93]). In the simple version of the algorithm adopted here, a *trace* parameter, x_i, is incorporated into the updating equation of the simple epochal scheme to place a greater emphasis on more frequently used rules. On each move (time step) of

an episode, the trace parameter is updated as follows:

$$x_i(t+1) = (1-\lambda)x_i(t) + \lambda q$$

where q=1 if rule i is active at time t, and 0 otherwise, and λ is the *trace decay rate*, which is again analogous to the bid co-efficient and set to 0.1. This trace parameter is subsequently used in the following epochal updating equation:

$$S_i'(\alpha) = S_i(\alpha)(1-c) + x_i cP$$

(c) Profit Sharing Plan with Variance (PSPV)

The simple epochal scheme described in (a) was employed by Grefenstette in an early study of credit assignment [Grefenstette, 88] in which he referred to the scheme as the *Profit Sharing Plan* (PSP). In more recent work, [Grefenstette, et al, 90] employ an updated version of the algorithm which includes the variance associated with the estimated payoff so that conflict resolution has a measure of confidence in the estimate, as well as the estimate itself, to base its decision on. This scheme, which will be referred to as the *Profit Sharing Plan with Variance*[11] (PSPV), maintains a mean, μ_i, and variance, υ_i, for each rule i, as well as the strength, S_i, and updates each of these at the end of an episode as follows:

$$\mu_i'(\alpha)=\mu_i(\alpha)(1-r) + rP$$

$$\upsilon_i'(\alpha)=\upsilon_i(\alpha)(1-r) + r(\mu_i'(\alpha) - P)^2$$

[11][Grefenstette, et al, 90] continue to refer to this as the PSP, despite it being quite different from the original version.

$$S_i'(\alpha)=\mu_i'(\alpha) - \sqrt{(\upsilon_i'(\alpha))}$$

where r, is a constant called the *psp rate* (again set to 0.1).

(d) Hybrid BA-ABBA1

A hybrid of the BA and BBA was originally proposed by [Liepins, et al, 91] but the only investigation to date into the effectiveness of the scheme was performed by [Twardowski, 93] on the pole-balancing problem, where results were very promising. The version implemented here is identical to that of both Liepins and Twardowski except that it employs ABBA1, rather than the standard BBA. The scheme operates in two distinct phases:

(i) during an episode, the scheme assigns strength in precisely the same manner as the ABBA1 algorithm;

(ii) upon completion of the episode, the strength of each rule is updated using the BA updating equation:

$$S_i'(\alpha) = S_i(\alpha)(1-c) + x_i cP$$

where x_i is the trace parameter.

(e) Hybrid PSPV-ABBA1

A direct hybrid of the BBA and PSPV algorithms is not possible because the BBA updates the strength of each rule using the previous strength, whilst the PSPV simply ignores this and replaces the strength by a value calculated from the mean and variance. However, a hybrid scheme is proposed here in which, during an episode, the strength of each rule is updated using ABBA1, and upon completion of the episode, μ_i and υ_i

are calculated (as in the PSPV) with the strength of each rule updated using the following equation:

$$S_i'(\alpha) = S_i(\alpha)(1-r) + r\,[\,\mu_i'(\alpha) - \sqrt{(\upsilon_i'(\alpha))}\,]$$

(f) Hybrid BA-PSPV-ABBA1

Finally, a hybrid of the BA, PSPV and ABBA1 algorithms was implemented by adding the trace parameter associated with the BA algorithm into the updating equation of the PSPV-ABBA1 hybrid, as follows:

$$S_i'(\alpha) = S_i(\alpha)(1-r) + x_i r\,[\,\mu_i'(\alpha) - \sqrt{(\upsilon_i'(\alpha))}\,]$$

The performance of the 6 schemes when applied to 300 instances of test problem TTP2 are shown in Figures 3.12 and 3.13. Figure 3.12 shows the results for the three epochal algorithms by themselves (that is, methods a, b and c). In each case, the plans developed were more successful than those of the standard BBA but were not as good as those produced by either ABBA1 or ABBA2. Of the three algorithms, due to the additional information provided by the variance, the PSPV method produced the best results. Of the other two schemes, rather surprisingly, the BA algorithm produced the worst results. However, the reason for this can be identified by analysing the plan produced by the BA algorithm (Table 3.16). This shows that, due to the BA algorithm favouring frequently used rules, the two default rules (rules 1 and 2) gain a considerable percentage (49%) of the total strength and hence, lead to a reduction is system performance.

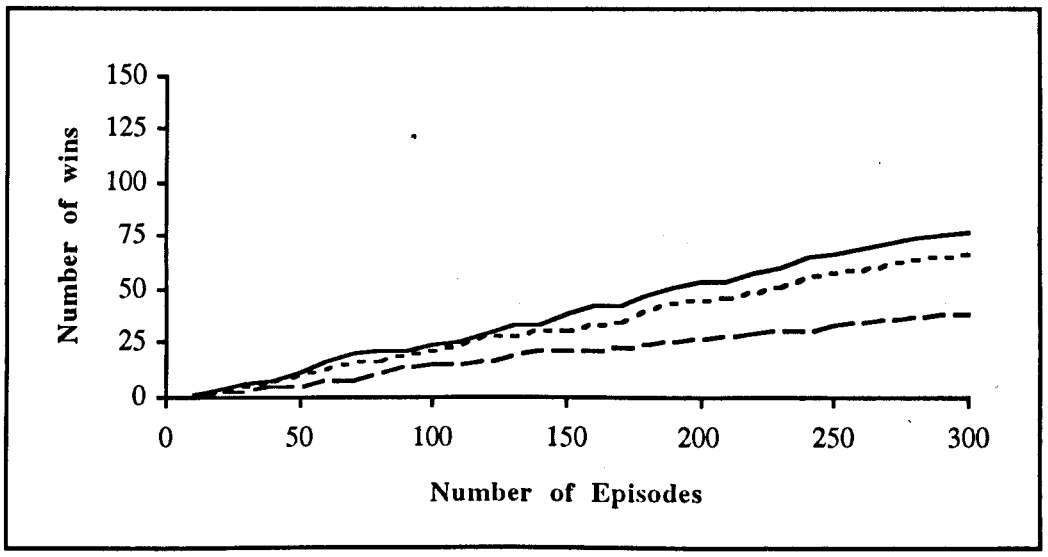


Figure 3.12: Performance of SEA (short dash), BA (long dash) and PSPV (solid) algorithms on test set S'

Table 3.16: Plan developed by BA algorithm

No.	Classifier	Abs. Strength	Rel. Strength
1	00#### : 010011	0.09	0.29
2	01#### : 010011	0.07	0.22
3	00#0##, : 01#011 : 010101	0.06	0.19
4	00#0##, : 01#011 : 010110	0.00	0.00
5	01###0 : 010011	0.00	0.00
6	01##0# : 010011	0.06	0.22
7	000010 : 010011	0.02	0.08

Figure 3.13 shows the equivalent results for the three hybrid schemes. In each case, the hybrid schemes produced better results than the stand-alone epochal algorithms but could only produce results comparable with those of the stand-alone ABBA1 and ABBA2 algorithms.

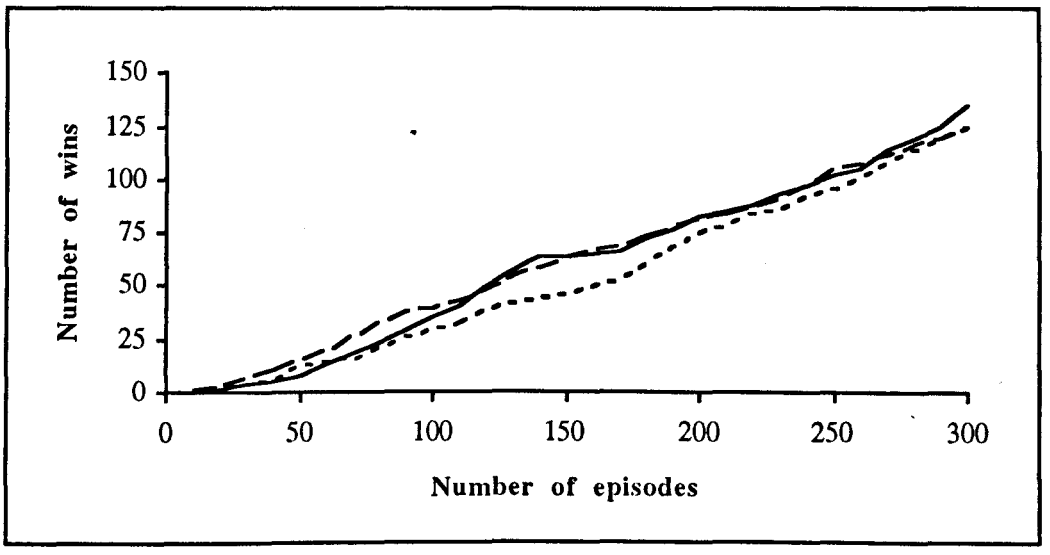


Figure 3.13: Performance of three hybrid schemes on test set S'

3.5 Conclusions

The work reported in this chapter has investigated one of the main components of a classifier system, the credit assignment algorithm - firstly from a theoretic viewpoint and then from a practical perspective.

The vehicle used for the theoretical investigation was a branch of mathematics known as Evolutionary Game Theory, and analysis with this showed that, under certain mild assumptions, the strengths of rules in a classifier system using the Bucket Brigade Algorithm will tend to conform to that of an Evolutionary Stable Strategy. The main consequence of this work is that if a plan were to develop in the manner predicted by the theory, then the plan may not end up maximising the expected payoff to the system but rather develop towards a pareto optimal state. Thus, additional mechanisms, such as the EA at the plan level, are likely to be needed if the proposed system is to successfully learn useful plans for all but the simplest of problems.

The remainder of the chapter addressed some of the practical problems associated with credit assignment, with particular attention being paid to the problems caused by parallel rules. Results from a test system applied to the two tanks problem showed that such rules can have a

significant impact upon the distribution of strength within a classifier system, with potentially disastrous consequences for system performance. A number of variants of the standard BBA were suggested as possible solutions to this problem, of which two in particular, those for which the payoff is divided in proportion to (a) the bid (ABBA1) and (b) strength/specificity (ABBA2), were shown to significantly improve system performance. A number of epochal schemes were also investigated, as were hybrids of the bucket brigade and epochal algorithms, but whilst the hybrid schemes showed comparable performance, none were able to surpass the performance of the stand-alone ABBA1 and ABBA2 schemes.

The main limitation of the above work is that all trials were performed using a 'static plan', that is, one in which the rule base remained constant. To investigate the relative merits of the proposed credit assignment algorithms on an evolving rule base, the test system needs to be extended to include a rule discovery element, that is, an evolutionary algorithm. The effects of credit assignment due to the presence of an EA are the subject of the next chapter, where some of the conclusions reached in Chapter 2 will also be investigated in the context of machine learning. Moreover, the addition of the EA allows the adaptive BBA which was suggested in section 3.4.2, to be investigated to determine if it too is of any benefit to a classifier system.

4. Rule Discovery in a Classifier System

4.1 Introduction

The work presented in the previous two chapters investigated some of the key features which affect the performance of the evolutionary and credit assignment algorithms respectively. The aim of this chapter is to combine the findings of these two investigations to produce a fully operational classifier system capable of learning solutions to simple 2-player games. Building this system serves two main purposes. Firstly, the system can be used to determine if the conclusions derived from the two separate investigations obtain when the two components are combined. Secondly, since such a system forms the main constituent of the hybrid learning system proposed in Chapter 1, then by analysing the effectiveness of this *component classifier system*, some of the shortcomings in the design of the hybrid system can be identified at an early stage and the system architecture corrected.

Section 4.2 describes the system employed to carry out this work, SCS-EA (an extension of the SCS1 system); particular attention being paid to the genetic operators suggested for the rule discovery phase. Section 4.3 then describes a technique not previously used in connection with classifier systems, a *meta-EA*, which it is suggested can be employed to discover combinations of genetic operators that work well for different classes of problem. Thereafter, sections 4.4 and 4.5 respectively describe the test problems used for this work and the results from applying the meta-EA to the problems.

4.2 System Architecture

4.2.1 Modifications to SCS1

The aim of this chapter is to describe the production of a fully operational component classifier system capable of learning solutions to 2-player military problems, and subsequently to use this system to investigate some of the key aspects affecting rule discovery in a machine learning environment. Since SCS1 was specifically designed for use in a 2-player game scenario (more explicitly, the two tanks problem) and had also been used successfully in the work in the previous chapter on credit assignment, it was decided to continue to use it as the basis for this investigation. However, to make the distinction between this and the previous version, the updated system was renamed SCS-EA.

Besides the obvious addition of an evolutionary algorithm, some minor amendments had to be made to the system design to enable it to be employed in an investigation of rule discovery. The only alteration made to the RaM part of SCS1 was in respect of the critic. This change was made because it was felt that the simple tri-partite critic employed by SCS1 was inadequate for all but the simplest problems. Consequently, a number of alternative critics, each one designed with a specific test problem in mind, were generated. These are described in sections 4.4.2 to 4.4.4 together with the corresponding test problems.

For credit assignment, four of the most promising schemes suggested by the work of the previous chapter were employed. These were the ABBA1, PSPV and hybrid ABBA1-BA-PSPV schemes, together with the adaptive BBA suggested in section 3.4.2. Selecting which scheme to use on any system run was governed by the parameter CRED_ASSIGN_TYPE, which is assigned an integer value in the range 1 to 4 corresponding to each of the four schemes employed.

4.2.2 Initialisation

When the outline of a standard EA was described in Chapter 1, it was noted that there are two methods of generating the initial population for an EA in common use - randomly or seeding

using *a priori* knowledge. The former option has the advantage that it provides a good test of the robustness of the system, whilst the latter option may improve the speed with which the system develops a good solution (although this may be at a cost of biasing the initial population). There are, however, various hybrids of these two approaches (see, for example, [Michalewicz, 92] on approaches to generating initial populations for the TSP). Moreover, for machine learning systems, there is also another option referred to as *Adaptive Initialisation*. In this scheme, each bit in the conditional part of a classifier is initialised to a 'wildcard' symbol and the action/message part to an action selected at random from the set of those available. Such a scheme results in all classifiers initially matching all possible situations, thereby causing the system to effectively employ a random walk strategy. Systems adopting this initialisation scheme (see, for example, SAMUEL ([Grefenstette, et al, 90] [Grefenstette, 91a])) are required to employ *specialization* operators, which use the outcomes from the sequences of actions to reduce the generality of the rules in an appropriate manner. However, if the rule discovery element is not based around such operators, then this is a poor way to generate the useful building blocks which the EA needs to work effectively.

Therefore, given that (i) the purpose of this work is to test the robustness of the learning system, (ii) an external expert to suggest any *a priori* knowledge was unavailable, and (iii) the EA to be employed was of a standard nature (that is, one not specifically designed to work with adaptive initialisation), it was decided that random initialisation of the population was the most suitable method. However, features of both the other two initialisation methods are required to be incorporated in SCS-EA because of the need to avoid generating a particular class of rule, referred to here as *lethals*. Two types of lethal classifier can be generated in SCS-EA.

(a) *Illegal classifiers*

Illegal classifiers can arise as a result of using a binary alphabet to represent non-binary variables. In such representations, problems surface when the number of values a variable can take is not an exact power of 2. For example, in the representation

scheme adopted by SCS-EA, the rightmost 3 bits in the action/message part are used to specify the action to be effected, for which there are six candidates. This leaves 2 of the 8 possible representations (111 and 100) with no action to specify. Therefore, a randomly generated action may, when decoded, suggest an action which doesn't exist, and consequently cause a system error.

(b) *Contradictory classifiers*

Contradictory classifiers are more problematical than illegal classifiers, and can arise when the conditional part of a classifier contains one or more conditions which can never be matched. For example, the environmental message used by SCS-EA has bit 4 set to 1 if the opponent moves (either forward or backward) and bit 5 set to 1 if the opponent does nothing. Clearly, if in a classifier, both bits are set to 1, then no message can ever satisfy its conditional part, thus making the classifier redundant.

Clearly, illegal classifiers must be avoided at all costs. The solution adopted in SCS-EA was to regard the 3 action-specifying bits as a unit. Then, given a list of the n available actions along with their respective binary codes, a 'random' action can be generated by selecting a random number, r, $1 \leq r \leq n$, and copying the 3 bit code corresponding to the rth action into the action specifying bits of the new classifier. By generating an action/message part in this manner, the system is incorporating some *a priori* knowledge (in the form of knowledge of the available actions) into a generally random process.

Unfortunately, because of the complex relationships that can exist between variables, there is no simple syntactic check to avoid generating contradictory classifiers. Moreover, complex relationships can develop in even the simplest TTPs. For example, in the TTP used thus far, a tank may be able to move forward and increase its gun elevation simultaneously, or move forward and fire simultaneously, but may not be able to increase its gun elevation and fire together.

Producing contradictory classifiers as a result of genetic recombination cannot be

avoided. Fortunately though, the problem can be somewhat eased when generating the initial population. Although at first sight this may not appear to be of much use, it is in fact vital to the operation of the system because the EA needs sufficient 'grist-for-the-mill', that is, sufficient numbers of building blocks in the initial population upon which to base its search. Without these, the search of the EA is unlikely ever to get 'off the ground'. For SCS-EA the problem is eased by proceeding as follows:

(a) when generating the *internal* condition, a valid action is selected at random and its code placed in bits 1..3 of the condition. Then, with a probability of 0.33, each bit is mutated to a wildcard. This process will ensure that every internal condition will match at least one internal message;

(b) when generating the *environmental* condition, a relatively high bias is given towards generating wildcards (a probability of a 0.5 was used with success). Whilst not eradicating the possibility of generating contradictory classifiers altogether, this does make it significantly less likely that they will be generated. Even if a small number are generated, due to the life taxes to which they are subject, their strengths should steadily decline and ultimately they will be deleted when the EA is invoked. It should be noted that this approach has much in common with the adaptive initialisation approach but does not require specific operators to generate the necessary 'grist-for-the-mill'.

One final point regarding the initialisation process concerns the presence of duplicate rules. The work presented in the previous chapter demonstrated that duplicate rules can have a detrimental effect on system performance and so, it was decided to ensure that duplicate rules could not exist within the plan. In SCS-EA, this is achieved by employing a syntactic monitor to detect the presence of duplicate rules, not only during the initialisation phase but also after the application of genetic operators, and, where necessary, to delete all but one of the duplicates.

4.2.3 Main cycle of the EA

Once the initial population has been generated, the main cycle of the EA can commence. For machine learning applications, the generational EA investigated in Chapter 2 is not generally desirable because a high level of on-line performance is usually required to facilitate the learning process. Instead, as [Holland, 86] suggests, new rules should be introduced gradually so that they can be tested alongside the existing proven rules. This is precisely the manner in which an incremental EA operates. Moreover, without the addition of a number of special mechanisms, useful rules may be lost in the selection phase of a generational EA but, with an incremental EA, only relatively poor rules are ever deleted from the rule base. Consequently, it was decided to employ an incremental rather than generational algorithm. Although the operation of such an algorithm is different from that used for the trials described in Chapter 2, the suggestions made at the end of those trials (Section 2.6) still maintain because (a) the trials were mainly analysed at a high level (for example, the net effect upon the EAs search of a particular mechanism - did it favour exploitation or exploration?), and (b) the conclusions derived were of a very general nature (for example, 'incorporating environmental information into operator rates appears, under certain conditions, to benefit system performance').

The evaluation task is performed using one of the four available credit assignment algorithms. To enable the rule strengths to approach their steady-state values, a number of episodes have to be run to make sure that all rules are evaluated sufficiently. This number of episodes is called the *period* of the EA and great care has to be taken when selecting its value; too short a period will mean that rules will not be properly evaluated, with the consequence that the search of the EA may degenerate into little better than a random walk, whilst a period which is too long may make the learning process unbearably slow. Unfortunately, there are no guidelines for selecting suitable periods for classifier systems and so the only way in which it can be determined is through trial and error.

One further point relating to the period is that it is likely that the rule strengths of the initial population will take significantly longer to approach their steady-state values than those

of subsequent generations. Consequently, it was decided to incorporate into the EA another parameter, the *threshold*, to enable a longer period to be set aside for the initial population to be evaluated. Thus, for example, when employing an EA with a threshold of 50 and period of 5, the system would run for 50 episodes before the first application of the selection / recombination phases, with every 5th episode after that being the cue for the selection / recombination phases to be invoked.

After each evaluation phase, the fittest members of the population are selected as arguments for the various genetic operators employed by SCS-EA. The selection scheme employed by SCS-EA was chosen to be a weighted ranking mechanism because of its particular suitability for use with an incremental EA. In this scheme, the population is ordered in terms of fitness (worst member at position 1, best member at position n, where n is the population size) and then a member selected using the following formula [Reeves, 91] (shown graphically in Figure 4.1)

$$i = 1 + \text{ROUND}\ [\ (\ 0.5\ \sqrt{(\ 1 + 4n(n\text{-}1)\ rand\ )}\ )\ \text{-}0.5\ ]$$

where i is the rank of the member selected, n is the population size and *rand* is a random number such that $0 \leq rand < 1$. Thus, for a population size of 20, the average value for i (when rand=0.5) is 14 (that is, the 7th best member).

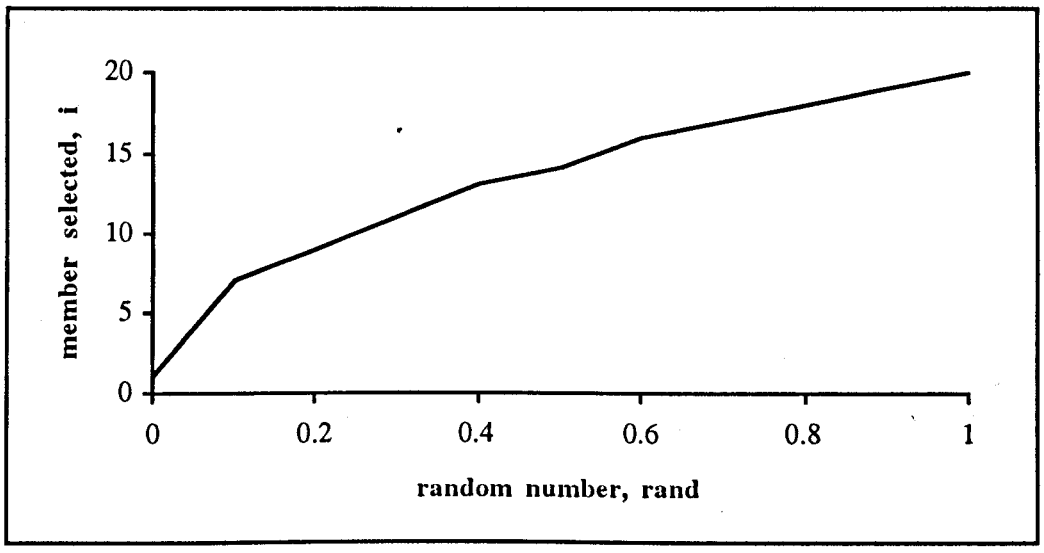


Figure 4.1: Distribution of the member selected, i, for different values of 'rand' when employing the weighted ranking scheme

4.2.4 Genetic operators

SCS-EA employs five types of genetic operator to facilitate the rule discovery process. These are:

(a) NO_MATCH;

(b) crossover operators;

(c) mutation operators;

(d) coupling operators;

(e) NEW_RULE.

The first of these operators, NO_MATCH, is essential to any classifier system which incorporates a rule discovery element, and is employed to generate a matching rule for the cases where no extant rule matches the current internal and environmental messages. The crossover and mutation operators are employed to perform the standard genetic operations of the EA, whilst the last two operators, the coupling operators and NEW_RULE, are employed to determine if, as suggested by the trials in Chapter 2, the addition of problem-specific operators

can improve the performance of the EA.

(a) NO_MATCH

In the most common approach to generating a matching rule where no extant rule matches the current messages, typically referred to as *best match*, each rule is compared with the current set of internal / environmental messages and the rule with the least number of non-matching positions selected. All non-matching bits in the rule are then set, with equal probability, to either the wildcard symbol or the equivalent bit in the corresponding internal/environmental message. Not only is this a simple and effective approach to the problem but is also consistent with the results reported in section 2.4.2 which demonstrated that amending extant members of the population, rather than generating completely new solutions, is likely to be a profitable approach to rule discovery.

Alternative schemes do exist including an adaptive No Match operator [Grefenstette, et al, 90] which generates a completely general rule to match the extant state of the message list but, as with the adaptive initialisation scheme, this operator requires specialization operators to generate subsequent, more useful rules.

A further point relating to the NO MATCH operator is that one of the rules in the extant rule base has to be removed to accommodate the newly generated rule. SCS-EA employs a REMOVE operator to achieve this, which itself employs one of two alternative schemes to select the rule to be removed. In the simplest scheme, one of the rules in the bottom 25% of the population is selected at random. However, because of the need to maintain a co-adapted population, it was decided to employ a second scheme which uses a *crowding mechanism* [De Jong, 75] to select the rule to be deleted. In a crowding mechanism, x members of the population are selected at random from the population and the one with the minimum strength selected. This process is then repeated y times, to end up with y rules which are candidates for deletion (no rule is selected more than once). The values x and y are referred to as the *crowding sub-population* and *crowding factor* respectively. Each of these y rules is then compared to the new rule to be added to the rule base, and the one which has the fewest non-

matching bits selected as the rule to be deleted. In the experiments reported here, both the crowding sub-population and crowding factor were set to 3 - a value which has been used successfully in a number of previous studies (see, for example, [Goldberg, 89a]).

A system parameter, CROWDING, which is set by the user at run-time, is employed to determine which of the two deletion schemes should be used on a particular run.

(b) Crossover operators

The experiments investigating the use of Lamarckian operators (section 2.4.4) demonstrated that incorporating environmental information into the crossover operator can, under certain conditions, significantly improve the rule discovery process. During the operation of a classifier system, two situations wherein environmental information may be useful are:

(i) after the occurrence of a special event during the course of an episode;

(ii) the end of an episode when its outcome can be used to guide the genetic operators.

Therefore, two crossover operators were designed for use in SCS-EA, one for each of these situations. These are called *triggered* and *end-of-episode* crossover respectively.

(i) *TRIGGERED CROSSOVER*

TRIGGERED CROSSOVER is employed whenever the message list, consequent upon conflict resolution, contains more than one message. In practice, this is likely to be almost every time step since several general rules specifying the same action are likely to be active simultaneously on many of the time steps, and if allowed, would produce a turn-over of rules so great that no rule would be able to develop its strength towards its steady-state value. This would effectively reduce the search of the EA to little better than a random walk. Consequently, the system was invested with another parameter, *trig_probability*, to limit the use of TRIGGERED CROSSOVER. Given that the number of time steps involved in an episode will be of the order of 10, and about one

new rule every 10 episodes would seem appropriate, the value of *trig_probability* was set to 0.01.

When the operator is invoked, two of the message list's m messages are selected at random, and the respective rules, R_1 and R_2, which posted the messages, selected as the parents for the crossover. Due to its success in trials with the knapsack problem, it was decided to employ the *string-of-change mechanism* (section 2.4.4) in the crossover operator. The string-of-change is calculated for R_1 and R_2 by concatenating the internal and environmental conditions with the action/message part and, as discussed earlier, treating the 3 action specifying bits as a unit to avoid generating illegal classifiers. An example of this operation is shown below.

Parent A	00###0,	01010#:	010011
Parent B	00#100,	010#0#:	010001
String-of-change	000110,	000100:	000 1

If the string-of-change has fewer than two 'ones' in it, then no matter where the crossover point is selected, no new rules will be generated and so the operation is abandoned with no further attempt made to select new parents. Otherwise, a crossing point is selected using the string-of-change, the crossover performed, and one of the offspring selected at random.

Before this new rule can be added to the rule base, a monitor is employed to compare it with all extant rules to ensure that it does not duplicate a rule. If it is found to duplicate an extant rule, then a further operator, CROSS MUTATION is employed to repeatedly mutate it until it no longer duplicates an extant rule. Not all the bits in a classifier are mutable; the 2-bit tag at the start of each 6-bit message is deemed immutable (because a change to it would make the classifier illegal), whilst the three action specifying bits are treated as a unit. Thus, of the 18 bits in a rule, only 10 are mutable, as follows:

Example string:	00#10#,	0111#0:	010111	
Mutable bits:	**1234	**5678	**9 10	(*=immutable bit)

(ii) *END OF EPISODE CROSSOVER*

Given that n is the period of the EA, then this form of crossover is employed periodically at the end of every n^{th} episode subject to the constraint that the number of episodes is greater than the threshold of the EA. Two parents, R_3 and R_4, are provisionally selected according to the weighted ranking scheme outlined earlier, and a string-of-change produced. If this string-of-change contains at least two 'ones' then the crossover is performed and one of the offspring (selected at random) added to the rule base. Once again, the member it replaces is selected using REMOVE. However, if the string-of-change contains fewer then two 'ones', then two new provisional parents are selected, again according to rank, and the process repeated until a suitable string-of-change is produced. Then, and only then, are the parents mated.

Once again, a monitor is employed to check that the new offspring generated by this process does not duplicate an extant rule, and if necessary, the CROSS MUTATION operator employed to generate a non-duplicate.

(c) **mutation operators**

SCS-EA employs two other mutation operators besides CROSS MUTATION.

(i) *STANDARD MUTATION*

The first of these mutation operators, *STANDARD MUTATION*, is similar to the mutation operator used in traditional EA applications. It is applied at the end of an episode (immediately after END-OF-EPISODE CROSSOVER) and operates by mutating each bit with a probability P_{st_mut}. The mutated version of the rule then

replaces the original version. This form of mutation is not suitable for application to profitable rules since valuable information could be lost. Consequently, STANDARD MUTATION is only applied to the worst 25% of classifiers.

A further feature of this operator is that the mutation probability, P_{st_mut}, is *adaptive*. This feature was added because of the success of the diversity-based mutation operators described in section 2.4.5. However, calculating diversity can be quite processor-intensive and, if carried out on a regular basis, may have a significant impact on the running time of the system. Fortunately, a 'rough' first approximation to population diversity can be derived from the number of attempts taken by the END OF EPISODE CROSSOVER operator to find suitably diverse parents. For example, if the operator finds appropriate parents on its first attempt, then the rule base is likely to be quite diverse and hence, will not require much mutation. However, if several attempts are required to find a suitable string-of-change, then this is an indication that the diversity in the rule base has fallen, and so a greater level of mutation is needed. Consequently, the mutation rate, P_{st_mut}, for this operator is calculated as follows:

$$P_{st\_mut} = (n+1) \cdot C_1$$

where C_1 is a small constant (set to 0.001 to be largely in line with the mutation rates investigated in Chapter 2) and n is the number of attempts made by the END OF EPISODE CROSSOVER operator to find a suitable string-of-change.

As with the crossover operators, before the new rule is added to the population, a monitor is invoked to ensure that it does not duplicate an extant rule. If it does, the rule is subjected to further mutation until a non-duplicate rule is derived.

(ii) *CLONAL MUTATION*

STANDARD MUTATION is applied to only the worst 25% of classifiers. Although

some of these poor classifiers may be only a single mutation away from being useful, in general, the most profitable mutations will be those involving useful rules. Consequently, SCS-EA is invested with a second mutation operator, *CLONAL MUTATION*, which makes an exact copy of a profitable rule, x say, mutates this duplicate, and then replaces a poor rule, and not x, by the mutated version, therefore minimising the loss of valuable information.

As with STANDARD MUTATION, CLONAL MUTATION, is invoked at the end of an episode and is applied with a probability P_{rr}, the *replication rate*, set to 0.01. Rules which are candidates for application of the operator are those which have made a *profit*, that is, those which have strength greater than 10.0. Each candidate is subjected to a single mutation, the mutant is checked using a monitor to ensure that it does not duplicate an extant rule, and is then added to the population in place of a rule selected using REMOVE.

(d) **coupling operators**

The problem of reinforcing long chains of rules has been well documented (see, for example [Riolo, 87a, 89a]). One method that has been suggested which may facilitate the development of long chains is to create tighter links between consecutive rules. Consequently, [Holland, 86b] suggested a COUPLING operator wherein the internal condition of a rule, R_2, active at time t, is mutated to be more akin to the internal message, M, posted by the rule R_1 which was active at time t-1. Two occasions on which such an operator may prove useful are:

(i) after a rule has made a profit;

(ii) at the end of a successful episode (that is, one in which the system achieved its goal) when the sequence employed has been proved to be useful.

The second of these is the most obvious time to couple two rules together but is rather limiting as it only applies to the last two rules in a sequence. Nevertheless, two such rules which have been shown to be successful, should be closely linked together. Consequently, this form of coupling is called TIGHT COUPLING. When, however, coupling is performed when a rule makes a profit, there is only a possibility that the rule has been involved in a successful sequence. Consequently, with this form of coupling, the link between the two rules should not be made as closely and so this form of coupling is called LOOSE COUPLING. The precise form of these two operators is as follows.

(i) *LOOSE COUPLING*

Before an episode, a record of the current strength of each rule is recorded. If at any time during the episode the strength of a rule becomes larger than its initial strength, then it is deemed to have made a profit and is thus subjected to LOOSE COUPLING. In this, a single wildcard in the conditional part of the rule is mutated to the corresponding bit in the internal / environmental message that was active on the previous time step. If there are no wildcards in the conditional part, then the rule must already be maximally specific and hence, there is no need to apply the operator.

Before the newly generated rule can be added to the population, a monitor is once again employed to ensure that the new rule does not duplicate any extant rules. If a duplicate is found, then further wildcards are mutated until either a rule with no duplicates is generated, at which point it is added to the rule base using REMOVE, or it is found to be maximally specific, in which case the operation is abandoned.

(ii) *TIGHT COUPLING*

This operator is employed at the end of an episode and is applied to the rule used on the final time step of the episode. It operates by mutating between 1 and w wildcards in the internal condition of the rule (where w is the total number of wildcards contained in the internal condition) to the corresponding bits in the internal / environmental messages

that were active on the penultimate time step. Thereafter, the same procedure performed in LOOSE COUPLING is employed.

(e) **NEW RULE**

The NEW RULE operator is similar to the TIGHT COUPLING operator except that it is not limited to the final pair of rules in an episode. It too is invoked at the end of a successful episode and operates as follows. On any time step, t, during an episode, the environmental message, the internal message[12] and the action performed on a particular time step are all recorded with a probability of $1/(t+1)$. Subsequently, should the episode be successful, a new rule is generated from all 3 messages, and, providing it is not a duplicate, added to the rule base; the member it replaces being selected using REMOVE.

4.3 Evaluation method

4.3.1 A meta-EA analysis of operator combinations

Elements of the system which it would be useful to identify are the combinations of genetic operators and credit assignment schemes that tend to produce the best results. However, because of the large number of genetic operators and their inter-dependencies, this is a difficult task. For example, analysing the system with and without one type of crossover may lead to misleading results because its effectiveness may be linked to the degree of mutation used by the system. Moreover, as there are 8 different genetic operators supported by the system and 4 different credit assignment schemes, even neglecting the various parameters such as operator probabilities, there are 1024 possible operator configurations.

Evaluating these manually is infeasible and so some automated search technique is required to find an optimal or near optimal operator configuration. Clearly, an EA itself is such

[12]If more than one are active, then the first message on the message list is selected.

a technique for this problem. As mentioned in Chapter 1, several studies (see, for example, [Grefenstette, 86] and [Freisleben and Härtfelder, 93]) have already been performed using one EA to optimize the parameter settings for another. Such 'parameter-finding' GA's are termed *GAs for GAs* or *meta-GAs*. As the number of operators in SCS-EA is much larger than in a traditional EA, the parameters associated with the operators were not investigated (as this would make the search space too large). Instead, the system was assigned default parameters for the operators (based on the authors experience of similar parameters) and the meta-EA was employed to look for useful combinations of operators.

4.3.2 Operation of the meta-EA

The problem to which the meta-EA was applied was one of discovering which combinations of operators and credit assignment schemes work well for various problems. Therefore, the encoding scheme for this meta-EA was a binary string in which each bit in the string corresponds to an operator from the set of those available to SCS-EA. As there are 8 operators (each of which can be used or unused) and 4 credit assignment schemes available to SCS-EA, then the bit-string used to represent an operator configuration consists of 10-bits where a 1 in position i means that the i^{th} operator in the list of operators (see Tables 4.1 and 4.2) is included in the configuration described by the string, whilst a 0 implies that it is not present in the configuration. The ordering of the operators within the encoding scheme was selected randomly.

Table 4.1: Encoding Scheme for Meta-EA

Bit No.	Operator / Parameter
1	TIGHT COUPLING
2	LOOSE COUPLING
3	NEW RULE
4	TRIGGERED CROSSOVER
5	END OF EPISODE CROSSOVER
6	CLONAL MUTATION
7	STANDARD MUTATION
8	CROWDING
9, 10	CREDIT ASSIGNMENT SCHEME

Table 4.2: Encoding scheme for bits 9 and 10

Code		Credit Assignment Scheme
0	0	ABBA1
0	1	Adaptive BBA
1	0	PSPV
1	1	ABBA1-BA-PSPV Hybrid

Each configuration in the population is evaluated by running the system for 10,000 episodes with that configuration, and recording the on-line performance of the system. The value of the on-line performance at the end of the 10,000 episodes is then used to represent the fitness of the configuration.

One problem with this evaluation phase is the length of time it takes. The 10,000 episodes take approximately 5 minutes to run and so only a relatively small number configurations can be evaluated. Consequently, the meta-EA must converge fairly rapidly and so an incremental algorithm was employed, the population size being 20.

After evaluating the first generation, the whole population is ordered (in terms of fitness) and two parent configurations selected according to the weighted ranking scheme described in section 4.2.3. Standard 1-point crossover is then applied to the two parents and one of the two resulting offspring selected at random. Mutation is then applied to this offspring, with a probability of 0.04 of changing any particular bit. The new configuration is subsequently evaluated and added to the population in place of a member selected at random from the worst five configurations. The population is subsequently re-sorted and the process repeated a further 399 times, giving a total of 420 evaluations (corresponding to approximately 35 hours of processing time).

4.4 Test Problems

4.4.1 Problem design

When designing the problems to investigate appropriate operator combinations / credit assignment schemes, it was decided to pay particular attention to the type of rule that would be

required for 2-player military games. Analysing the scenarios showed that the rules required fall into two broad categories.

(i) *stimulus-response rules* - those in which an action is performed in response to the environment (world model) being in a certain state.
eg. IF *distance to target* = *range* THEN *fire*

(ii) *coupled rules* - those in which the action to be performed is determined by the previous actions.
eg. IF *previous action was fire* THEN *retreat*

Correspondingly, three types of learning task were designed to investigate which operator combinations are particularly useful for discovering these types of rule:

(1) *classification tasks* - used to investigate the system's ability to discover stimulus-response rules;

(2) *sequential tasks* - used to investigate the system's ability to discover coupled chains of rules;

(3) *combined classification and sequential tasks.*

A small number of problems were developed for each of these cases. Examples of problems for each type of learning problem are given below.

4.4.2 Classification Problem 1 (CP1)

In this problem, each episode lasts for a single time step and involves the system selecting the correct response (out of 8 available responses) to one of 16 randomly generated 4-bit inputs.

Table 4.3: Classification Problem 1

Inputs	Desired Output
0000	001
0001	001
0010	001
0011	001
0100	001
0101	001
0110	001
0111	001
1000	101
1001	101
1010	101
1011	101
1100	011
1101	011
1110	110
1111	010

Due to there being no sequence involved with this problem, the internal message of every rule was initialised to 01#### to match every message. The critic employed for this problem was very straightforward; +10 when an input is correctly classified and 0 otherwise.

4.4.3 Sequential Problem 1 (SP1)

In this problem, a version of the TTP, the objective is to perform the 3 actions 'move forward', 'increase gun elevation' and 'fire' in that order, given no environmental information (that is, the environmental message is constant). Thus, a solution to this problem must necessarily involve the generation of coupled sequences of rules. The critic for this was slightly more complex than that used for Classification Problem 1, in that +10 is awarded for a successful sequence, +5 if the first 2 actions are correct, +2.5 if only the first action is correct, and 0 otherwise.

4.4.4 Combined Learning Problem 1 (CLP1)

The scenario for this problem, a relatively complex version of the TTP, is depicted in Figure 4.2.

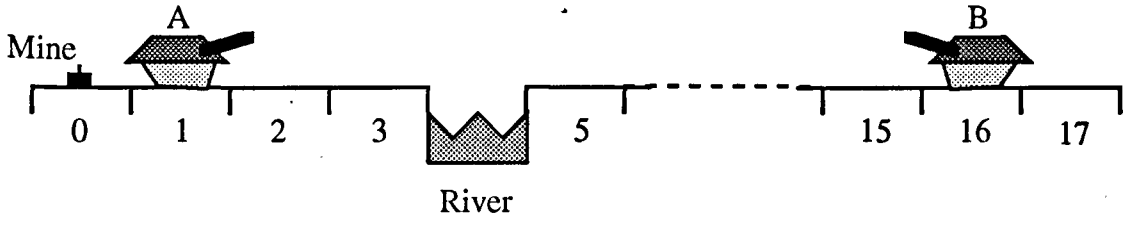


Figure 4.2: Initial positions in CLP1

At the start of the game, the two tanks are set 15 spaces apart, the gun elevation set to 3 ($1 \leq$ *gun elevation* ≤ 5), the range set to 11 ($9 \leq$ *range* ≤ 13) and the ammunition level (that is, the number of times a tank can fire) set to 2. Each tank has the same six moves (section 3.3.2) available to it as in previous problems and the maximum number of moves that Tank A (the learning agent) can take is 9.

The only environmental information Tank A receives as to the state of the game is a binary encoding of the gap between the two tanks. Due to the positions of the mine and the river, A cannot move further than 2 places forward, or any further back than its original position. Two direct hits on B are required to destroy it and as the ammunition level is limited to 2, for A to win the game, it is necessary for both its shots to hit. Tank B's strategy is to do nothing until it is hit for the first time, at which time it either (a) moves forward one space on the next move and waits there, or (b) moves back one space, and then moves forward on the next move ((a) and (b) are randomly selected with equal probability). On all subsequent moves, B does nothing. Therefore, the optimal strategy for A is to move forward twice and increase its gun elevation twice in the first four moves (in any order) and then fire on the fifth move. This will result in B being hit for the first time. Then, if B subsequently moves forward (detected by the gap changing to 12) A should either decrease the elevation of its gun or move back one space, before firing on the subsequent move. Alternatively, if B moves backward (detected by the gap changing to 14), then A should do nothing for one move and then, as B comes back into range, fire on the subsequent move.

This is a useful scenario for investigating the formation of both stimulus-response rules and coupled rules. For the first five moves; there is no environmental knowledge and so, for A to perform the correct actions, the system needs to generate an appropriate set of coupled rules. Subsequently (when B is hit), the system needs to generate two appropriate stimulus-response rules - one which specifies a move forward if the environmental message is 001100 (gap = 12), and another which specifies 'do nothing' should the environmental message become 001110 (gap = 14).

Due to this problem being much more difficult than either CP1 or SP1, the critic employed was made much more complex so as to provide the system with as much information as possible. The critic operates by evaluating each move made by T1 during the course of an episode and maintains a running total of the values returned. This total is adjusted (see below) and then awarded as the payoff to T1. The ratings for the various moves available to T1 are as follows:

(a) forward - moving forward is beneficial and thus scores +1, unless it results in T1 falling in the river, in which case it scores -3;

(b) backward - if T2 moves forward on the previous move (as it does after it is hit for the first time), then this is a good move and so is awarded +1. However, on any other occasion, moving backward is a poor move and so it is awarded -1;

(c) fire - if the shot misses, then this is assigned -1, but if it hits, then the total is incremented by 9 points;

(d) increase elevation - this is always beneficial and so is assigned +1;

(e) decrease elevation - see move backward;

(f) do nothing - if T2 moved backward on the previous move, then this is a good 'move' to make and is thus assigned +1, otherwise it is a poor move and is assigned -1 points.

If, at the end of an episode, the total score is less than zero, then the critic awards zero points, otherwise it awards 10 x (total score/2).

4.5 Results

The results generated by the 3 problems described above were found to be generally representative of the problem categories from which they were derived. Therefore, although only the results from 3 problems are shown here, these were verified by the results of other similar problems.

4.5.1 Classification Problem 1

Table 4.4 shows the final state of the meta-EA population after the 420 configuration evaluations (the solution with the highest utility is at position 20).

Table 4.4: Meta-EA population for CP1

Rank	Bits									
(20=best)	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	1	0	1	0	1
2	1	1	0	0	1	1	0	1	0	1
3	1	1	0	0	1	1	0	0	0	1
4	1	1	0	0	1	1	0	1	0	1
5	1	1	0	0	1	0	0	0	0	1
6	1	0	0	1	0	1	0	1	0	1
7	1	1	0	0	1	1	0	1	0	1
8	1	1	0	0	1	0	0	0	0	1
9	1	0	1	1	0	1	0	1	0	1
10	1	0	0	1	1	0	0	0	0	1
11	1	0	0	1	0	1	0	1	0	1
12	1	0	1	1	0	1	0	1	0	1
13	1	1	0	0	0	1	0	1	0	1
14	1	1	0	0	1	0	0	0	0	1
15	1	0	1	1	0	1	0	1	0	1
16	1	0	1	1	0	1	0	1	0	1
17	1	1	0	0	1	1	0	1	0	1
18	1	0	1	1	0	1	0	1	0	1
19	1	1	0	0	1	1	0	0	0	1
20	1	1	0	1	1	1	0	1	0	1

As the results demonstrate, not all the operators are beneficial to the learning of stimulus-response rules. For example, STANDARD MUTATION (represented by bit 7) tends to

produce poor results, as does the NEW RULE operator (bit 3). Since this problem has no sequential nature to it, one aspect of the results which at first sight appears a little curious is the success of the two coupling operators (bits 1 and 2). However, their success can be explained by the fact that after a successful classification, one or more bits in the internal condition of the successful rule(s) (originally set to 01####) is/are mutated by the operators to match more closely the internal message posted at that time. Since the internal message is always the same, this does not facilitate the classification process. However, it does increase the specificity of the successful rule, thus making more likely its selection on future occasions.

Another important feature of these results is that the entire population adopted the adaptive BBA as its credit assignment algorithm (that is, bits 9 and 10 were set to 0 and 1 respectively), thus giving an initial indication that this scheme is particularly useful in classifier systems.

Figure 4.3 shows the performance of SCS-EA over the 10,000 problem evaluations when employing the fittest operator combination. This demonstrates that the system is capable of learning an appropriate set of stimulus-response rules in a relatively short period of time. Moreover, it demonstrates that once the system has discovered an appropriate set of rules, it is capable of maintaining a high level of performance.

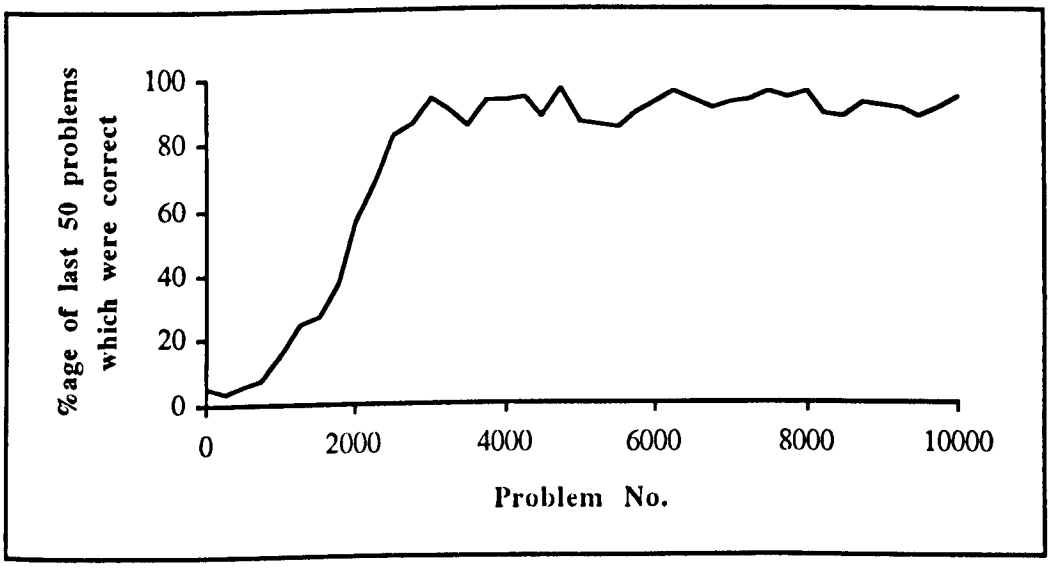


Figure 4.3: Performance of SCS-EA (employing the 'optimal' operator configuration) when applied to CP1

4.5.2 Sequential Problem 1

Table 4.5 below shows the meta-EA population after the 420 configuration evaluations involving the test problem SP1.

Table 4.5: Meta-EA population for SP1

Rank	Bits									
(20=best)	1	2	3	4	5	6	7	8	9	10
1	1	1	1	0	0	0	0	1	1	1
2	1	1	1	0	1	1	1	1	1	0
3	1	1	1	0	0	0	0	1	1	1
4	1	1	1	0	0	1	0	1	1	1
5	1	1	1	0	0	0	0	1	1	1
6	1	1	0	0	0	1	0	1	1	1
7	1	1	1	0	0	0	0	1	1	1
8	1	1	1	0	0	0	1	1	1	1
9	1	1	1	0	0	0	0	1	1	1
10	1	1	1	0	0	0	0	1	1	1
11	1	1	1	0	0	0	0	1	1	1
12	1	1	1	0	0	1	0	1	1	1
13	1	1	1	0	0	0	0	1	1	1
14	1	1	1	0	1	0	0	1	1	1
15	1	1	1	0	0	0	0	1	1	1
16	1	1	1	0	0	0	0	1	1	1
17	1	1	1	0	0	1	0	1	1	1
18	1	1	1	0	0	0	0	1	1	1
19	1	1	1	0	0	0	0	1	1	1
20	1	1	1	0	0	0	0	1	1	1

Not surprisingly for a problem in which the objective is to discover sequences of coupled rules, both the two coupling operators and the NEW RULE operator were adopted by almost every member of the population. Moreover, the results demonstrate that for this particular type of problem, the crossover and mutation operators (bits 4 to 7) were not beneficial to the rule discovery process. The reason for this is that the two coupling operators, together with NEW RULE, were sufficient in themselves to discover an effective plan. Consequently, the only effect of either crossover or mutation would be to disrupt this plan, thereby reducing system performance.

In respect of the credit assignment algorithm, the results show that for this type of

problem, the PSPV-BA-ABBA1 hybrid scheme was the most successful out of the 4 alternatives. The lack of success of the adaptive BBA in this environment can be attributed to the fact that once a set of successful, specific rules have been generated, the adaptive BBA would attempt to generalise them and this can only reduce system performance.

Figure 4.4 shows the performance of SCS-EA when employing the fittest operator configuration over 10,000 type SP1 problems. Once again, the results demonstrate that the system is capable of discovering an appropriate set of rules to solve the problem at hand - in this case, the type of rule being coupled rules. Note that for this operator configuration, the system manages to maintain a 100% success level.This is entirely due to the fact that the system does not employ crossover or mutation, and thereby avoids generating potentially erroneous rules once the optimal rule set has been developed.

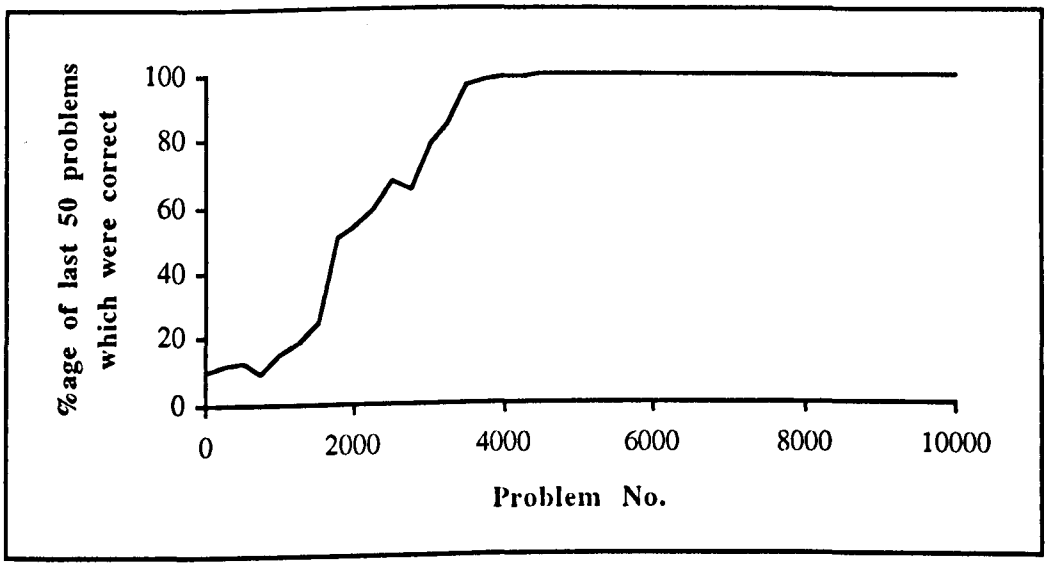


Figure 4.4: Performance of SCS-EA (employing the 'optimal' operator configuration) when applied to SP1

4.5.3 Combined learning problem 1

Table 4.6 shows the population generated by the meta-EA when applied to CLP1.

Table 4.6: Meta-EA population for CLP1

Rank	Bits									
(20=best)	1	2	3	4	5	6	7	8	9	10
1	1	1	0	1	1	1	1	1	1	1
2	1	1	0	0	1	0	0	0	0	1
3	1	0	1	1	1	1	0	0	0	1
4	1	1	0	1	1	1	1	1	0	1
5	1	1	0	0	1	0	0	1	0	1
6	1	0	1	1	0	0	0	1	0	1
7	1	0	0	1	1	1	1	0	0	1
8	1	1	1	0	0	0	1	1	0	1
9	1	1	0	1	1	1	1	0	0	1
10	1	0	1	1	1	0	0	1	0	1
11	1	1	1	1	0	0	1	1	0	1
12	1	1	1	0	1	1	1	1	0	1
13	1	0	1	1	1	1	0	0	0	1
14	1	1	0	1	1	1	1	1	0	1
15	1	1	1	0	1	1	1	1	0	1
16	1	1	0	1	0	1	1	1	0	1
17	1	0	0	1	1	1	1	1	0	1
18	1	1	0	0	1	0	0	0	0	1
19	1	1	1	1	1	1	1	1	0	1
20	1	1	0	1	1	1	0	1	0	1

The results show that the best operator configuration found for SCS-EA when applied to CLP1 was identical to that for CP1, thus demonstrating that whilst their use may be detrimental to results for a certain class of problems (namely simple sequential learning tasks), both crossover and mutation are important elements in the rule discovery aspect of the system. However, whilst the results show that both crossover operators are beneficial to the system, only one of the mutation operators (CLONAL MUTATION) is clearly beneficial - the other operator, STANDARD MUTATION, generally being neither beneficial nor detrimental to the system. Moreover, the results once again demonstrate that an adaptive BBA appears to be the most useful of the four credit assignment schemes available.

Figure 4.5 shows the performance of SCS-EA over 10,000 CLP1 problem evaluations

when employing the fittest operator combination. From this, it can be clearly seen that even when employing a near-optimal operator configuration, SCS-EA fails to find a set of rules which can consistently solve the problem. However, the system does manage to successfully 'solve' this relatively difficult problem on about 25% of occasions, thus demonstrating that the system is able to discover many of the rules necessary for a successful solution.

There are a number of reasons why the system only solves the problem on about 25% of occasions. For example, the system is constantly generating new rules to be tested and some of these will be poor, thereby reducing system performance. Another reason is that no attempt has been made to try to improve the parameter settings for the various operators. The values employed were only estimates (based on the authors experience of similar parameters) and it is likely that there is some scope for improvement with regard to these parameter values. However, a more pressing problem is that for this problem, sub-goals are rewarded by the critic. This rewarding of sub-goals is necessary for relatively difficult problems like CLP1 because the probability of solving the problem by chance is so low that the system would have too few chances to reinforce any successful sequences which it may generate. This was demonstrated when the critic was replaced by one employing a simple scheme which rewarded 100 points for a victory and 0 otherwise. In trials with this critic, it was found that the system failed to win a single game in 10,000 attempts. However, given that sub-goals are to be rewarded, then the system will often just solve the sub-goal (for which it will receive credit) rather than the ultimate goal (that is, hitting the opponent twice in this case).

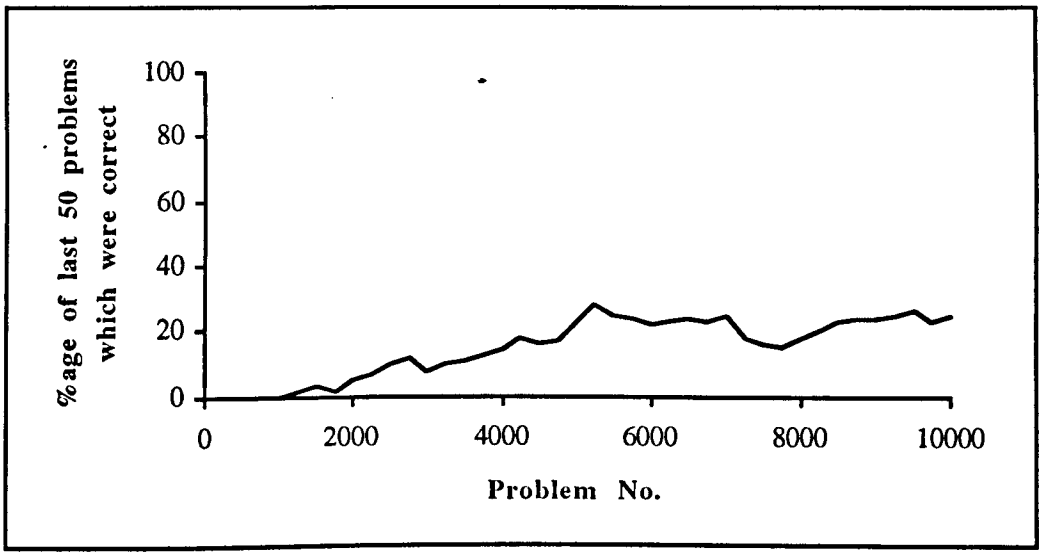


Figure 4.5: Performance of SCS-EA (employing the 'optimal' operator configuration) when applied to CLP1

4.6 Conclusions

In this chapter, a number of genetic operators for use in classifier systems have been proposed and investigated using a meta-EA to determine which operator combinations tend to produce the best results.

The results show that not all the operators are necessary and in some situations, it may be a case of "too many cooks spoiling the broth". For example, generating too many new rules will not give sufficient opportunity for the strength of either new or extant rules to settle to an equilibrium level where they can be compared and the best selected. Consequently, having two forms of COUPLING operator plus the NEW RULE operator is, as demonstrated by the results in Tables 4.4 to 4.6, excessive for all problems except a purely sequential learning problem like SP1. However, most problems are likely to require some sequence of rules to be generated and so one or possibly two of these three operators are likely to prove useful. Both forms of crossover were generally found to be useful but having two forms of mutation was found to be excessive. Instead, it would appear that employing only CLONAL MUTATION is sufficient.

Not surprisingly, the crowding mechanism was found to generally produce better results than a simple 'delete a poor rule at random' strategy. Finally, the high percentage of solutions which adopted the adaptive BBA as their credit assignment scheme give a strong indication that this is the most appropriate credit assignment scheme to employ in classifier systems, although for some problems (for example, simple sequencing problems) a PSPV-BA-ABBA1 hybrid proved to be more suitable.

The results presented in this chapter have demonstrated that meta-GAs can be employed to facilitate the fine tuning of a classifier system as well as more standard EA applications. However, the time taken by the system to perform the analysis is a major limiting factor in the number of trials that can be performed. For example, it would be interesting to determine if significantly different results are produced by the meta-EA when a measure other than on-line performance is used as the fitness of an operator combination. Unfortunately, due to the length of time taken to perform a single run, further investigations such as this could not be carried out.

5. The SAGA[13] system

5.1 Introduction

In Chapter 1, it was suggested that a hybrid architecture combining aspects of both the Michigan and Pittsburgh approaches is likely to be the most appropriate architecture for developing military tactics. This resulted in an outline system design being proposed (Figure 1.7). The work in Chapters 2, 3 and 4 then investigated some of the key features which would effect the performance of such a system. This work has been combined and resulted in a more detailed system design. The system has been given the name the *SAGA* system - an acronym for Strategy Analysis by Genetic Algorithm - the reference to *genetic* rather than *evolutionary* being a result of historical reasons.

The aim of this chapter is to describe the architecture of the SAGA system, and the various operators and algorithms it employs. In particular, attention is paid to why, in the course of system development, various design decisions were taken, and to the major factors that influenced its implementation.

Section 5.2 gives an overview of the system architecture, whilst section 5.3 discusses the requirements of a suitable knowledge representation for real-world problems, and describes a *symbolic* scheme which satisfies those requirements. In section 5.4 the operation of the central part of the SAGA system- the *Component Classifier System* - is detailed with particular emphasis being placed upon the innovative credit assignment algorithm / genetic operators employed by the systems to enhance its general problem solving ability. Finally, section 5.5 details the higher (plan) level operation of the system and its interaction with the lower level component classifier systems.

---

[13]Not to be confused with the Species Adaptation Genetic Algorithm developed by Harvey [Harvey, 92].

5.2 Overview of the SAGA system

At the end of Chapter 1, an outline design was proposed for the system deemed to be most suitable for learning military tactics. This consisted of a hybrid architecture in which a series of *Component Classifier Systems* (CCSs) operate in parallel, each one learning independent of the others. Periodically, an EA at the plan (Pittsburgh) level selects the best rules/plans from the set of classifier system plans and recombines these to form a new set of plans. These are then divided amongst the set of classifier systems and the process repeated until some terminating criteria is reached.

As a result of the experiments performed throughout this research, the outline system design has remained largely unchanged, with the exception of one minor alteration. The success of systems employing co-evolutionary techniques (eg. [Hillis, 91], [Ikegami and Kaneko, 91]) and the need to model the opponent have suggested that it may be beneficial to allow the opponent to co-evolve with the learning agents. Consequently, two populations of co-evolving CCS are employed in the SAGA system - one for the learning agents and one for their adversaries. Each CCS is paired with a single CCS from the opposing population and as such, co-evolve in an environment in with each combatant is constantly trying to beat its opponent. Such co-evolutionary behaviour is often referred to as the *Red Queen's race*[14][Rennie, 92], and is similar to the relationship between co-evolving predators [Sumida and Hamilton, 94]. Consequently, the two populations of CCSs in the SAGA system are called the *Predator A* and *Predator B* populations, with the Predator A and Predator B CCSs playing tanks *T1* and *T2* respectively.

Figure 5.1 is an outline schematic of the SAGA system. In the system, each CCS has essentially the same architecture as the SCS-EA system employed in the previous chapter, although there are some important differences, most notably in the genetic operators used for

[14]The term *Red Queen's race* derives from the Red Queen in Lewis Caroll's *Through the Looking Glass* who says, "Now, here, you see, it takes all the running you can do, to keep in the same place."

rule discovery These differences are discussed in section 5.4. Furthermore, as well as interacting with a plan in the opposing population during an episode, each plan also interacts with other plans from the same population via the genetic operators at the plan level. These are discussed in greater detail in section 5.5.

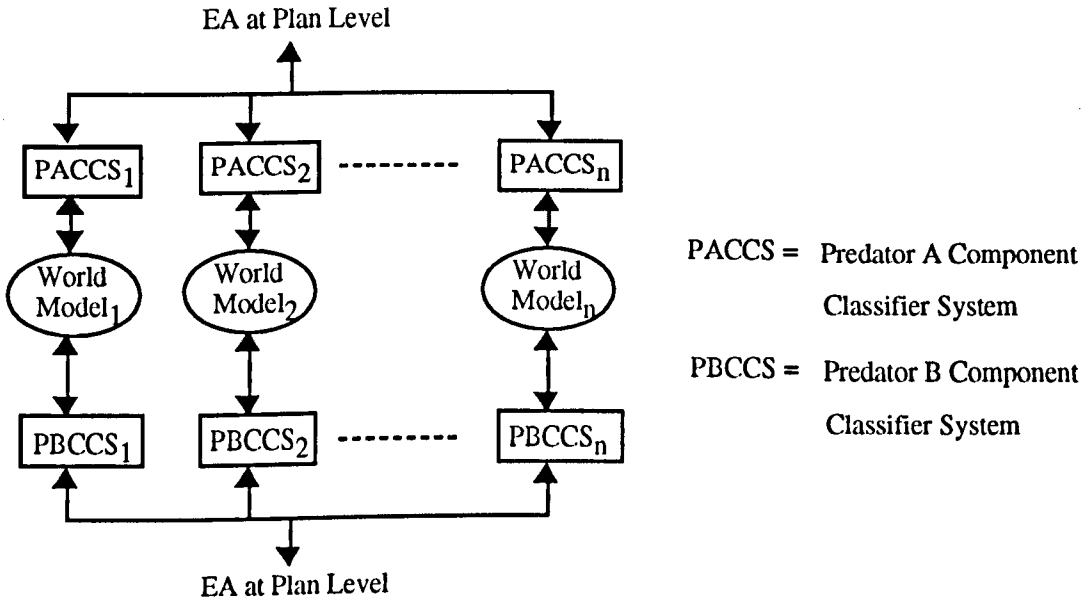


Figure 5.1: Outline of the SAGA system

5.3 Knowledge Representation

5.3.1 Problems with the representation used so far

The fixed length strings over the alphabet {0,1,#}, employed by the SCS-EA system to encode its knowledge, have a number of limitations. These include:

(a) *comprehensibility* - rules encoded in such a simple alphabet are difficult to understand without the use of a relatively complex interpreter;

(b) *fixed length* - real-world problems have a large number of entities associated with them, and since fixed length rules require space for conditions relating to each one of these entities, then the rules are likely to be very long;

(c) *accuracy* - the accuracy of values represented in a binary form is limited by the number of bits allocated for each entity.

5.3.2 A symbolic, variable length rule format for real-world problems

In view of these three limitations, it was decided that a more comprehensible, accurate and manageable knowledge representation scheme would be required for the SAGA system. To make the rules more comprehensible, the most suitable approach is to employ a *high-level* or *symbolic* representation scheme, similar to that employed in SAMUEL [Grefenstette, et al, 90]. Such a scheme not only makes the rules easier to interpret, but also facilitates the development of the system by making test data easier to devise / express and analyse. Furthermore, a symbolic representation scheme also enables *floating-point* numbers to be incorporated into the system, thus resolving many problems regarding the accuracy of rules. Moreover, representing conditions and actions in a high-level language has the consequence that rules are no longer forced to have a specifically ordered, fixed-length format. A system using such a representation is able, by virtue of the variable length format, to discount irrelevant conditions (that is, those which match for all possible values) thereby making rules more concise, and speeding-up the matching phase by obviating non-essential comparisons.

In respect of the structure of the rules to be employed in the SAGA system, it was felt that because it is the most suitable way of encoding generative rules of behaviour, the basic production rule form was still the most appropriate representation scheme for the system. However, a small number of changes to the format of a rule were necessary to allow symbolic, variable-length rules to be implemented. These changes resulted in rules having the following general form:

Rule *x*: **IF** γ_x *IC* **THEN** α_x *IM*

where γ_x is a set of conditions relating to the extant state of the world model, *IC* is the internal

condition, α_x is a set of actions specified by the rule and *IM* is the internal message. In each rule, *IC*, which relates to the extant state of the internal message list, must always be present. The number of conditions in set γ_x can range from zero, γ_x is the empty set, to *max*, for which value there is a condition for every problem entity. The action part of rules also have varying size and consist of a (possibly null) set of non-conflicting actions, but each rule must possess an internal binary message, *IM*, which is posted to the message list whenever the rule is activated.

Both conditions and actions in this format are represented by *tuples* similar in form to the *entity-attribute-value triples* employed by many Knowledge Based Systems [Bench-Capon, 90]. An example of such a triple for a tank (the entity) having a speed (the attribute) of 2 km/h (the value) might be (*Tank*, *Speed*, 2). However, to simplify matters, in the SAGA system, a triple's entity and attribute parts are combined and represented by a single *identifier*. Thus, in respect of the previous example, the simpler *identifier-argument tuple* (*speed_of_tank*, 2) is obtained.

In order to cover the wide variety of variables that need to be modelled in any problem domain, the representation employs 6 different identifier types. Details of each of these types, along with the structure that the respective conditions and actions take, are given below.

(a) **Linear**

Description: Linear tuples are used to represent identifiers which have real scalar values, such as the speed of a tank and the distance to a target. For simplicity, an integer value in the SAGA system is also represented by a real value with a zero decimal part.

Condition: (Identifier, lower bound, upper bound)

eg. (*speed_of_tank*, 100.0, 200.0)

Action: (Identifier, value)

eg. (*new_speed*, 10.0)

(b) **Cyclic**

Description: Cyclic tuples are a special form of linear tuple which are used to represent entities whose values are 'cyclical in nature', such as bearings.

Condition: (Identifier, lower bound, upper bound)

The lower bound in cyclic conditions may be greater than the upper bound. For example, (*bearing*, 300.0, 90.0) which is satisfied if $300.0 \leq bearing < 360.0$ or $0.0 \leq bearing \leq 90.0$.

Action: (Identifier, value)

eg. (*set_bearing*, 100.0)

(c) **Tree Structured**

Description: Tree structured tuples are used for conditions/actions in which the domain of the identifier can be represented by a hierarchy. For example, Figure 5.2 gives an example of how the *weather* may be represented for a particular problem. Here, each value which *weather* may adopt is represented by a *node* in the tree.

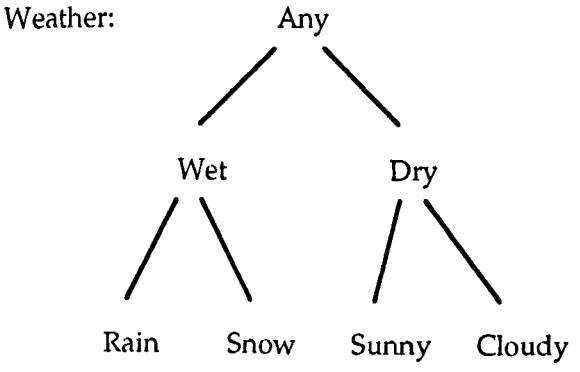


Figure 5.2: Tree Structured representation of weather

Condition: (Identifier, node_name)

This condition will match for any value in the subtree of 'node_name'.

For example, (*weather*, wet) is satisfied if the weather is either raining or snowing.

Action: (Identifier, node)

(d) **List**

Description: List tuples are used to represent identifiers which may assume one of several values from a non-hierarchical domain.

Condition: (Identifier, argument 1, argument 2, ... , argument *n*)
where *n* is the number of arguments.
eg. (*tank_type*, MkI, MkIII)

Action: (Identifier, argument)
eg. (*tank_type*, MkIII)

(e) **Boolean**

Description: Boolean tuples are used for identifiers which can assume the values TRUE or FALSE.

Condition: (Identifier, True/False)
eg. (*first_move_of_game*, TRUE)

Action: (Identifier, True/False)
eg. (*set_alarm*, FALSE)

(f) **Pattern**

Description: Pattern tuples are used mainly for internal conditions and messages. However, if an environmental identifier arises for which a binary string is the most appropriate representation, then the system can also cope with this.

Condition: (Identifier, *m*) where *m* is a message over the alphabet {0,1,#}.
For example, the condition (*Internal*, 0#01) will match if the internal

message list contains either 0001 or 0101.

Action: (Identifier, *M*) where *M* is a binary string.

To ensure that the arguments of a condition or action are legal, the SAGA system maintains an *Identifier Look-Up* (ILU) *table* which has a record of the legal range (*domain*) of each problem identifier. The values which are required to define the domain of each identifier type are shown in Table 5.1, along with some suitable examples.

Table 5.1: ILU table entries

Type	ILU table entries	Examples
Linear	Minimum legal value Maximum legal value	Speed: 0 to 25m/s
Cyclic	Minimum legal value Maximum legal value	Bearing: 0 to 360°
Tree	Complete Tree Description	Weather: See Figure 5.1
List	All list entries	Tank type: MkI, MkII, MkIII, MkIV
Pattern	Number of bits in message	Internal: 4
Boolean	Null	In_range:

To enhance the expressive power of the rules in the SAGA system, two further features were added to the basic form of a condition:

(a) conditions may be negated by placing a NOT before the tuple;

(b) instead of fixed values, identifiers themselves may be used as the arguments of a condition.

The reason for the inclusion of the first of these features is simply for efficiency of representation but that for the latter is less obvious and therefore, the following example is used to demonstrate how the use of *variable* arguments can significantly enhance the expressive

power of the SAGA system.

Given a fairly simple TTP in which A knows the distance to B, then A's best strategy is to elevate its gun until B is within range, and to subsequently fire. Consequently, if A's firing range is limited to between 15 and 20 units and if conditions are limited to containing only fixed bounds, then the best rule set that can be derived to fire at the appropriate time may look something like:

$Rule_1$: IF (*dist to target*,15,15) (*range*,15,15) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_2$ IF (*dist to target*,16,16) (*range*,16,16) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_3$: IF (*dist to target*,17,17) (*range*,17,17) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_4$: IF (*dist to target*,18,18) (*range*,18,18) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_5$: IF (*dist to target*,19,19) (*range*,19,19) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_6$: IF (*dist to target*,20,20) (*range*,20,20) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_7$: IF (*dist to target*,16,16) (*range*,15,15) (*Internal*, ####) THEN (*Incelev*, TRUE) (0010)

$Rule_8$: IF (*dist to target*,17,17) (*range*,15,16) (*Internal*, ####) THEN (*Incelev*, TRUE) (0010)

$Rule_9$: IF (*dist to target*,18,18) (*range*,15,17) (*Internal*, ####) THEN (*Incelev*, TRUE) (0010)

$Rule_{10}$:IF (*dist to target*,19,19) (*range*,15,18) (*Internal*, ####) THEN (*Incelev*, TRUE) (0010)

$Rule_{11}$:IF (*dist to target*,20,20) (*range*,15,19) (*Internal*, ####) THEN (*Incelev*, TRUE) (0010)

This is clearly a space inefficient way to represent an effective strategy. However, by permitting identifier names as condition arguments, the optimal strategy for the problem can be represented more efficiently by:

$Rule_1$: IF (*dist to target, range, range*) (*Internal*, ####) THEN (*Fire*, TRUE) (0001)

$Rule_2$: IF NOT (*dist to target, range, range*) (*Internal*, ####) THEN (*Inc elev*, TRUE)(0010)

Not only do such rules make the solution more concise and easier to understand, but with

fewer rules to discover, the problem should be easier for the system to solve and hence take fewer generations of the EA.

Many more extensions to the knowledge representation scheme are possible including the use of compound expressions (such as 'distance to target + 1') as condition arguments, and multiple ranges in linear/cyclic conditions. However, the above provided a scheme that is sufficiently powerful but nevertheless not too complex.

By employing a variable length format for rule representation, it was anticipated that many of the problems relating to rule length would be resolved. However, having a variable length format has the consequence that when genetic operators are applied, the following two problems may arise:

(a) the conditional part of a rule may become overspecified;

(b) the action part of a rule may contain conflicting actions.

To avoid the conditional part becoming overspecified, a 'condition checking' mechanism, CONDITION_CHECK, was introduced, to be invoked whenever a condition is added to a rule. The mechanism tests for the existence of more than one occurrence of an identifier in the conditional part of the transformed rule and removes all but one of these occurrences (selected at random) should more than one be present. Preventing a rule from containing conflicting actions is, however, more problematical. To alleviate this problem, during its initialisation (see section 5.4.1), the SAGA system arranges together actions which conflict and places them in sets or *groups* within a data structure called an *Action Group* (AG) *table*. Subsequently, whenever the action part of a rule is altered, a relatively simple process called the ACTION_CHECK mechanism is invoked. This mechanism ensures that no two actions are in the same group, thus avoiding conflicting actions. For example, given a simple TTP in which a tank has three possible actions available to it ('fire', 'move forward' and 'do nothing'), then if it cannot 'fire' and 'do nothing' simultaneously, nor 'move forward' and 'do nothing' simultaneously' (but can 'fire' and 'move forward' together) the AG table that would be set up

to prevent the occurrence of conflicting actions would be that shown in Table 5.2.

Table 5.2 An Action Group table for a simple TTP

Group Number	No. of actions	Action List
1	2	'Move Forward','Do Nothing'
2	2	'Fire','Do Nothing'

Unfortunately, the procedure of placing actions into conflict groups has to be performed manually when the system is initialised, and due to the complex relationships that are likely to exist between the alternative actions, this can be a time consuming and error prone process. If errors do result, then there is no way to predict what will happen, and it is possible that incorrect solutions may arise. Consequently, a thorough check must be performed by the user to avoid errors occurring in this phase.

5.4 Component Classifier System operation

5.4.1 Initialisation

Before its life cycle can commence, the SAGA system must be initialised. This process is divided into two parts. In the first part, system run time parameters are read from an input file. These parameters include operator rates, the number of episodes to be run and the size of the message list. The second part of the initialisation process involves setting up the problem specific knowledge, including the ILU table and the initial rule bases. Depending on the run-time parameter *random_rule_base*, the set of rule bases are either read from an input file (thereby allowing the seeding of knowledge into the initial population) or randomly generated (thereby providing the sternest test for the system). If random initialisation is selected, then a rule base is generated in such a way that it contains a set number of rules, *max_no_rules*, with each rule having:

(a) i conditions, $1 \leq i \leq I$, where I is the number of identifiers represented in the ILU table. Each condition has an identifier randomly selected from the ILU table and bounds which are randomly selected from the legal range of the identifier;

(b) a single randomly selected action.

A final piece of data which the system requires is the type of strategy which the opponent is to play. Here, depending upon the parameter *opponent_type*, the opponent may play either a co-evolving predator strategy or one of a number of fixed strategies which are built into the problem dependent part of the system.

5.4.2 Main Cycle

Once the SAGA system has been initialised, the predator A CCS population is run sequentially - one episode per CCS before returning to the first CCS and repeating the process until a fixed number of episodes (*num_episodes*) have been performed.

In each CCS, an episode starts with the world model being initialised, and the message list being reset so that it only contains the single 4-bit message '1111', which the system recognises as being derived from the system itself and not from a classifier. The following operations are then performed cyclically until some problem specific terminating criteria is reached.

(a) *Read Environment*

At the start of the main operating cycle, the extant state of the world model is recorded in the environmental message, *E*. This is simply a list of identifier-value pairs - one for each problem identifier.

(b) *Match*

The Match phase is employed to determine which rules have their conditional parts satisfied by *E*. As well as having all of its conditions satisfied, a rule also has to have a complete set of executable actions to progress to the next stage of the system cycle. For example, a rule which suggests raising the elevation of a gun which is already at the maximum elevation will fail to match.

If no matching rules are found, then the operator NO_MATCH is employed to generate one. NO_MATCH operates by selecting a candidate rule, R_s, and then mutating it so that (i) all of its conditions are satisfied by *E*, and (ii) all of its actions can be performed. To select R_s, NO_MATCH first determines if there are any rules which have a complete set of executable actions. If there are, then R_s is selected to be the best matching rule, that is, the one with the highest percentage of matching conditions. However, if no rule with a complete set of executable actions exists, then R_s is selected as the best matching rule regardless of its actions, and a special flag, *new_action_flag*, set to TRUE. The conditions in R_s are then mutated, depending on the type of a condition, as follows:

(i) a non-matching linear or cyclic condition tuple has one of its bounds replaced by the value of its identifier in *E*. For example, if the speed of a tank is 10, and R_s contains the condition (*speed*, 20, 30), then this condition is mutated to (*speed*, 10, 30);

(ii) in a tree structured condition tuple, the argument is mutated to the lowest common ancestor of both the argument of the tuple and the value of the identifier in *E*. For example, if the weather is raining (as recorded in *E*) but R_s contains the condition (*weather*, snow), then the lowest common ancestor of both 'rain' and 'snow' is 'wet' and so the condition is mutated to (*weather*,

wet);

(iii) non-matching list condition tuples simply have the value of the corresponding identifier in E added to the list (or removed from the list if the condition is negated);

(iv) since they cannot be generalised, non-matching boolean condition tuples must be removed altogether from the conditional part of R_s to enable the rule to match;

(v) all bits of a non-matching pattern tuple which do not match the corresponding bits in E are mutated to #.

After mutating the conditional part, the action part of the selected rule is mutated if the *new_action_flag* is set to TRUE, or with a small probability, P_{NA}, if *new_action_flag* is set to FALSE. By incorporating this small random probability P_{NA} into the mutation process, potential problems wherein the operator always generates a rule with the same action to a particular situation are avoided. This mutation process involves deleting the current set of actions in a rule and replacing them with a single randomly selected action.

(c) *Competition*

If more than one rule is active after the match phase, then the SAGA system is required to select which actions to perform out of those advocated. To achieve this, the system employs two levels of competition - one at the rule level and the other at the action level. The first level of competition, the *rule auction*, is employed only if there are more matching rules than the message list can accommodate. This auction consists of a

roulette wheel selection of m rules (where m is the size of the message list) based on the bid from each active rule, where the bid for a rule i is calculated by:

$$bid_i = specificity_i \,.\, b \,.\, strength_i$$

where b is the bid-coefficient. Unfortunately, the calculation of the specificity of a rule for the SAGA system is not as simple as for standard classifier systems wherein it is simply the percentage of non-# symbols in the rule. Instead, a different method of calculating specificity is required for each data type and these are detailed below.

(i) For linear/cyclic condition tuples, the specificity is calculated by

$$1 - \left[\frac{\text{range of condition}}{\text{domain of identifier}} \right]$$

Thus, the specificity of the condition (*speed*, 10, 20), where the domain of *speed* is between 0 and 100, is 1-(11/101) = 0.89.

(ii) For tree structured condition tuples, the specificity is calculated as follows.

Let the tree be T.

Let node in question be d.

Then,

$$\text{specificity of } d = \begin{cases} \left(\dfrac{n - m_d}{n} \right) & \text{if } n > 1 \\ 0 & \text{otherwise} \end{cases}$$

where n is the number of nodes in the tree T and m_d is the number of successors of d which are leaf nodes.

Thus, a tree condition with the root node as its argument has a specificity of zero, whilst a condition with a leaf node has a specificity of unity.

(iii) For list condition tuples, the specificity is calculated by

$$1 - \left[\frac{\text{no. of arguments in condition - 1}}{\text{no. of members in domain - 1}} \right]$$

Thus, the specificity of a condition with only a single argument is unity, whilst that for a maximally specific condition (that is, one containing the full compliment of arguments for that condition) is zero. [The 'minus 1' in the numerator is necessary so that when there is a single argument, the fractional part is zero. Similarly, the 'minus 1' in the denominator is required so that the fractional part is unity when the number of arguments in the condition equals the number of members in the domain.]

(iv) For boolean condition tuples, the specificity is always 0.5.

(v) For pattern condition tuples, the specificity is calculated in the same manner as traditional classifier system, namely

$$1 - \left[\frac{\text{no. of non-\# symbols}}{\text{length of message}} \right]$$

After calculating the specificity of each condition, the specificity of a rule can be calculated by summing the specificities of all its conditions.

The second level of competition, *conflict resolution*, takes place between the various actions suggested by the remaining active rules. This operation is a 2-stage process in which first of all, the active rule with the highest bid is selected deterministically. Thereafter, all rules which have one or more actions which conflict with the selected action are removed from the active list, thereby leaving a set of non-conflicting actions which are to be performed.

(d) *Update strengths*

The next phase of the CCS cycle involves updating the strength of each rule. This is

carried out in accordance with the credit assignment algorithm described in the next section.

(e) *Update World Model*

The non-conflicting actions which remain after the competitive stages of the CCS cycle are then passed to the world model which is accordingly updated.

(f) *Opponents Move*

If a pre-programmed strategy is being employed by the 'opponent', then the actions are selected according to the strategy and the world model correspondingly updated. However, if a co-evolving CCS is being used to represent the opponent, then a cycle of the predator B CCS is executed. This cycle is precisely the same as the cycle of the predator A CCS, with the resulting actions from the predator B CCS also being used to update the world model.

This marks the end of the main cycle of a CCS. This cycle is then repeated until some terminating criteria, for example, a tank being hit, is reached.

5.4.3 Credit Assignment

The results presented in the previous chapter from the meta-EA analysis of the SCS-EA system demonstrated that, in general, an adaptive BBA produces the best results. However, in some cases (eg. simple sequential problems), a PSPV-BA-ABBA1 hybrid algorithm appeared to be more suitable. Consequently, for credit assignment in the SAGA system, it was decided to combine the best features of both algorithms. This resulted in a *PSPV-adaptive BBA* hybrid scheme which operates as follows.

During an episode, the strength of each rule is updated as in the adaptive BBA scheme. In this, the strength paid to the r rules active on the previous time step is divided in proportion to the value of

$$b \,.\, strength_i \,.\, specificity_i^{\vartheta}$$

where i is one of the r rules active on the previous time step and ϑ is termed the *constant of adaptation*, which in turn is calculated by

$$\vartheta = -1 \,.\, \left[\left(\frac{current\,form \,.\, 2}{max\,form} \right) - 1 \right]$$

where *current form* is a time weighted average of the payoff awarded to the CCS as a whole (see section 5.5.2 for more details) and *max form* is the payoff the system would receive if it performed with 100% success (that is, the maximum possible value for current form) which is constant for any particular problem. [It should be noted that the SAGA system maintains an array of values for ϑ - one for each CCS.]

Upon completion of the episode, the mean and variance associated with the PSPV algorithm are updated, and subsequently used to update the strength of the rule as shown below:

$$\mu_i'(\alpha) = \mu_i(\alpha)(1-c) + cP$$

$$\upsilon_i'(\alpha) = \upsilon_i(\alpha)(1-c) + c(\mu_i'(\alpha) - P)^2$$

$$S_i'(\alpha) = S_i(\alpha)(1-c) + c\,[\,\mu_i'(\alpha) - \sqrt{(\upsilon_i'(\alpha))}\,]$$

where $\mu_i(\alpha)$, $\upsilon_i(\alpha)$ and $S_i(\alpha)$ are the mean, variance and strength of rule i after episode α respectively, and c is the *psp rate*, which is set to 0.1.

5.4.4 Genetic Operators

With the change in knowledge representation from a simple binary alphabet to a symbolic language, the set of genetic operators (transformation schemes) previously used to investigate

rule discovery (namely, those employed by SCS-EA) needs to be altered to deal with high-level structures. One area of AI research which involves transformation schemes for rules expressed in a symbolic language is that of *Inductive Learning* (IL) [Holland, et al, 86]. Consequently, it was decided that the most suitable approach to the design of the operators to be employed in the SAGA system would be to base the operators on a number of inductive mechanisms. Therefore, many of the operators described in this section are derived form one of the cornerstones of the IL research field, namely Michalski's study of IL mechanisms [Michalski, 83].

The hybrid architecture of the SAGA system allows genetic operators to be employed at two distinct levels - the rule and plan levels. The operators reported below that are applied at the rule level (that is, within a CCS) are divided into two groups according to the time during an episode at which they are used. In the first group, *Triggered* operators are invoked when a special event occurs during the course of an episode, and use details of this special event to guide the genetic transformation. The second group, *End of Episode* operators, are invoked at the end of an episode and use the outcome of the episode to facilitate rule discovery.

5.4.4.1 Triggered Operators

The SAGA system makes use of the following triggered operators:

(a) *CLOSING_INTERVAL*

Genetic operators can be used to advantage when two rules which advocate the same action are active simultaneously. In such circumstances, it is likely that the rules apply to a similar set of situations, and consequently, can be combined. The SAGA system performs the combination according to the relative strengths of the rules.

If both active rules are of high strength, that is their strengths are greater than the average rule strength plus one standard deviation[15], then the system generalises a

[15]The addition of a standard deviation to the selection criteria is so that high strength rules, not merely above average strength rules, are selected.

single condition common to both rules using the operator CLOSING_INTERVAL. One of the two rules is then duplicated and invested with this new condition to create a new rule which is added to the extant rule base. The precise operation of CLOSING_INTERVAL is subject to the data type of the identifier upon which it acts. Table 5.3 gives a brief description of the operation for the 6 data types supported by the system along with suitable examples.

Table 5.3: CLOSING_INTERVAL

Type	Operation	Example
Linear/ Cyclic	Use a bound from one condition to generalise the other condition	(speed, 100, 200) (speed, 150, 250) => (speed, 100, 250)
Tree Structured	Climbing Generalization... change more specific argument to the other argument	(weather, any) (weather, rain) => (weather, any)
List	Add a member of one list to the other list	(Tank Type, MkI, Mk II, Mk III) (Tank Type, MkII) => (Tank Type, MkI, MkII)
Pattern	Change one of the non-# bits to a #	(Internal, ##00) (Internal, 1000) => (Internal, #000)
Boolean	Not Possible	

(b) *OPENING_INTERVAL*

If two simultaneously active rules advocate the same action and both have low strength, then it is possible that there is an element common to both rules which is causing the poor performance. The operator OPENING_INTERVAL adopts this idea to specialise a condition in one of the low strength rules using the corresponding condition in the other rule. Again, the precise operation of the operator varies with the data type of its operand and Table 5.4 gives examples for the 6 data types supported by the SAGA system.

Table 5.4: OPENING_INTERVAL

Type	Operation	Example
Linear/ Cyclic	Use a bound from one condition to specialise the other condition	(speed, 100, 200) (speed, 150, 300) => (speed, 200, 300)
Tree Structured	Descending Specialization... change more general argument to root of subtree not containing the other argument	(weather, rain) (weather, any) => (weather, dry)
List	Remove a member common to both lists from one of the lists	(Tank Type, MkI, Mk II) (Tank Type, MkI, MkIII) => (Tank Type, MkIII)
Pattern	Change a # in one condition to the opposite value of a non-hash in the other	(Internal, 01##) (Internal, #1#0) => (Internal, 11#0)
Boolean	Not Possible	

(c) *CROSSOVER*

There are two types of situation wherein the SAGA system uses the crossover operator to generate a new rule from two parent rules. The first occurs when two high strength rules are active simultaneously. As CLOSING_INTERVAL is also used in this situation, significant disruption may occur in the make-up of the rules. Consequently, the system employs both CROSSOVER and CLOSING_INTERVAL each with a 50% probability. The second situation in which CROSSOVER is used is that in which one of the two active rules has high strength and the other low strength. Here, the resulting rule replaces the rule with lower strength.

5.4.4.2 End-of-Episode Operators

As mentioned earlier, the outcome of an episode can be used to guide genetic operators towards the creation of more profitable rules. If the episode has been a success for a particular CCS in that the actions employed led it to achieving its goal, then it may be profitable to build upon the success by generalising one of the rules used during the episode. Similarly, if the episode has resulted in a CCS not achieving its goal, then at least one of the actions employed was inappropriate and hence, it may be profitable to specialise one of the rules used. Therefore, the SAGA system employs one of the following two types of operators to mutate a single rule, randomly selected from those used during the episode:

(a) *Operators used after a successful episode*

The SAGA system has a set of 5 operators from which one may be selected at the end of a successful episode to generalise one of the rules used. The first three of these are grouped together and are referred to collectively as the *Generalisation Operators*. These have a probability, P_{go} of being selected. The remaining two operators are COUPLING, which has a probability P_{coup} of being selected, and CONSTANT _TO_VARIABLE, which has a corresponding probability P_{cv}. All three probabilities are set at run time and it is a constraint of the SAGA system that only one operator can be used per episode.

(1) *Generalisation Operators*

If the Generalisation Operators are selected, then, depending upon the specificity of the rule, one of the following three operators will be performed:

(a1) *CONJUNCTION_TO_DISJUNCTION*

This operator generalises rules which have a large number of conditions - that is those with more than twice the average number of conditions for a rule. This it achieves by dividing the selected rule's condition set into two subsets at random and creating two new rules, each possessing one of the subsets as its condition set. The action part of the new rules is that of the original with the original rule being deleted. For example, the rule $C_1\ C_2\ C_3\ C_4\ C_5 : A_1$ might be generalised to $C_1\ C_2 : A_1$ and $C_3\ C_4\ C_5 : A_1$.

(a2) *CONDITION_DROPPING*

This operator merely discards a condition from the selected rule in order to form a rule that is slightly more general.

(a3) *EXTEND_BOUNDS*

The aim of this operator is to extend the range of a single condition in the selected rule, and as such, the precise form of the operation performed is subject to the data type of the identifier involved, as shown below:

(i) For linear or cyclic conditions, EXTEND_BOUNDS mutates one of the two bounds (selected at random) so that the range of the condition is increased. However, in such a situation, the amount by which the bound is altered must be determined. For example, considering the condition (*distance_to_target*, 100, 120) with an associated valid range of say 0 to 10000. By what amount should 120 (this has been selected) be altered in respect of the mutation? Certainly one of the factors that must be taken into consideration in such cases is the range of the corresponding variable in the condition. Another factor is the current strength of the rule - if a rule has a high strength, it is likely that it has been successful in the past, and altering the bound too much could severely degrade its performance. However, if a rule has low strength, the opposite applies. Consequently, the SAGA system combines all these factors in a Lamarkian mutation operator which updates the selected bound as follows:

$$\text{New Bound} = \text{Old Bound} \pm \left[\frac{\text{current range} \,.\, C \,.\, R}{\text{relative rule strength}} \right]$$

where C is a constant (set to 0.1) and R is a random number in

the range [0,1).

(ii) For tree structured conditions, the bound of the condition is generalised to either its parent or grandparent (selected at random).

(iii) For list conditions, a randomly selected argument from those not already included is added to the list.

(iv) For pattern (internal) conditions, EXTEND_BOUNDS mutates one of the non-# bits in the argument to a #.

(v) No form of bound extension is applied to boolean conditions as this would lead to the condition being dropped.

(2) *COUPLING*

This operator is analogous to the TIGHT COUPLING operator employed by SCS-EA (section 4.2.4) and is employed to forge tighter links between one of the rules that was active during the successful episode and the rule which activated it. However, by recording a (randomly selected) pair of consecutive rules, this operator is not restricted to the last two rules in the episode, as is the case with TIGHT COUPLING.

(3) *CONSTANT_TO_VARIABLE*

For each pair of variables of the same type in different conditions of a rule which has been active during an episode, this operator first inspects the values that the variables possessed in the world model at the time when the rule was active. If the values are found to be equal then a new rule is created which

expresses this fact. For example, if the rule:

IF ... (*range_of_T1*, 15, 18) (*dist_to_target*, 16, 20) ...
THEN (*Fire*, TRUE) ...

was active during a successful episode, and at that time, the following variable-values were observed in the world model: *range_of_T1* = 17, *dist_to_target* = 17, then the operator CONSTANT_TO_VARIABLE would create a new rule with the conditions:

(*range_of_T1*, *dist_to_target*, *dist_to_target*) AND
(*dist_to_target*, 16, 20)

where the first condition is matched if the range of T1 equals the distance to the target.

One further point worthy of note is that this is the only mechanism employed by the SAGA system to convert fixed bound values into variables and so could prove pivotal to the success of the system in solving some of the problems to which it is applied.

(b) *Operators used after an unsuccessful episode*

When a particular sequence of actions leads to an unsuccessful episode (that is, the CCS failed to achieve its goal), this is likely to be the result of 'overgeneralisation', and therefore, specialising one of the rules used during the episode may be profitable. Consequently, the SAGA system employs a set of *Specialisation Operators* to perform this task. Another possible cause of the lack of success in the episode may be the result of an inappropriate action being applied at a particular time. Therefore, the system also employs an operator, ACTION_CHANGE, in an attempt to 'correct' a poor rule. Both

ACTION_CHANGE and the set of Specialisation Operators are described below.

(1) *ACTION_CHANGE*

This operator is employed when the selected rule has a very low relative strength (that is, its strength is less than the average rule strength minus 2 standard deviations - the 2 being used to signify that the strength has to be well below the average). In such circumstances, the operator will, with equal probability, either replace the whole action part of the selected rule by a single randomly generated action, or mutate the argument in one of the actions to a randomly generated argument.

(2) *Specialisation Operators*

The SAGA system employs three specialisation operators;

(i) *UNCOUPLE*

The aim of this operator is to prevent the same (poor) sequence of rules from being repeated and as such, is the inverse of the COUPLING operator. UNCOUPLE is selected with a probability, P_{uc}, and mutates one of the randomly selected bits in the internal condition of the selected rule to the 'opposite value' (that is, a '0' to a '1' and vice versa) of the corresponding bit in the internal message of the rule which activated it. For example, if the selected rule contained the condition (*Internal*, 0##1) and the message which satisfied it was 0101, then UNCOUPLE may generate the condition (*Internal*, 1##1) by mutating the first bit of the original.

(ii) *CONDITION_ADDING*

This operator is used when the selected rule, chosen randomly from

those used during the episode, has only a small number of conditions (that is, less than the average). The effect of CONDITION_ADDING is to introduce into the rule a new condition which satisfies the recorded environmental state, provided that it is not already present in the rule. The identifier for this new condition is selected at random from those in the environmental message, *E*, but not already in the rule, with the argument of the condition (or both arguments for linear and cyclic conditions) taking the value of that identifier in *E*.

(iii) *REDUCE_BOUNDS*

This operator is employed when the selected rule has quite a large number of conditions (that is, greater than the average). The operator functions by using the state of the world model at the time the rule was active to exclude an unsuccessful value from a condition. As with other operators which mutate conditions, the precise operation of REDUCE_BOUNDS is subject to the data type of the identifiers upon which it acts, and Table 5.5 gives a description of the operator along with suitable examples.

Table 5.5: REDUCE_BOUNDS

Data Type	Operation	Example
Linear/ Cyclic	Compares the value of the identifier in the world model to the two bounds. If it is nearer to the lower bound, then this bound is mutated to the value of the world model+1; otherwise, the upper bound is mutated to the world model value -1.	(bearing, 100, 200) world model: bearing=180 >> condition mutates to (bearing, 100, 179)
Tree Structured	Mutates the condition 's current argument to the other child of its parent	(weather, rain) >>condition mutates to (weather, snow)
List	Removes the value of the world model from the argument list	(tank_type, MkI, MkII) world model: tank_type, MkI >>condition mutates to (tank_type, MkII)
Pattern	(See UNCOUPLE)	
Boolean	Not Possible	

(c) *Other End of Episode Operators*

As well as employing operators which use the outcome of an episode to guide rule discovery, the SAGA system also employs three operators at the end of an episode irrespective of the outcome. These are:

(a) *DELETE*

This operator is used in an attempt to rid the system of old and ineffective rules by removing rules if their strength has fallen below a certain value (the *deletion threshold* - a run-time parameter).

(b) *GARBAGE_COLLECTOR*

This operator inspects every condition (except the internal condition) in every rule and if any condition is found to be maximally general (that is, it matches all possible values for the identifier), then it is deleted.

(c) *INDUCTIVE_RESOLUTION*

Despite having a wide variety of generalisation and specialisation operators available, the SAGA system has no operators which work with boolean conditions. The generalisation operator INDUCTIVE_RESOLUTION was added to the system to redress this balance. Applied at the end of an episode and with a probability, P_{IR}, this operator searches for any two high strength rules in the rule base which have the same actions. If two such rules are found and they both contain the same boolean condition but with opposite values, then the two conditions are removed from the respective rules. For example, given the following two rules:

R_A: IF ... (*engine_functional*, TRUE) .... THEN (*fire*, TRUE)

R_B: IF ... (*engine_functional*, FALSE) ... THEN (*fire*, TRUE)

if both these rules have above average strength, then it would appear to indicate that the '*engine_functional*' condition is irrelevant and can therefore be removed from both R_A and R_B. [It should be noted that although this operator is the boolean equivalent to CLOSING_INTERVAL, it cannot be used when CLOSING_INTERVAL is used since INDUCTIVE_RESOLUTION requires the conditions to be mutually exclusive whilst CLOSING_INTERVAL, being a triggered operator, requires some degree of overlap between the two conditions.]

This final trio of operations brings to 15 the total number of genetic operators supported by the SAGA system. Figure 5.3 (overleaf) depicts the interplay between these operators and the circumstances in which each operator is used.

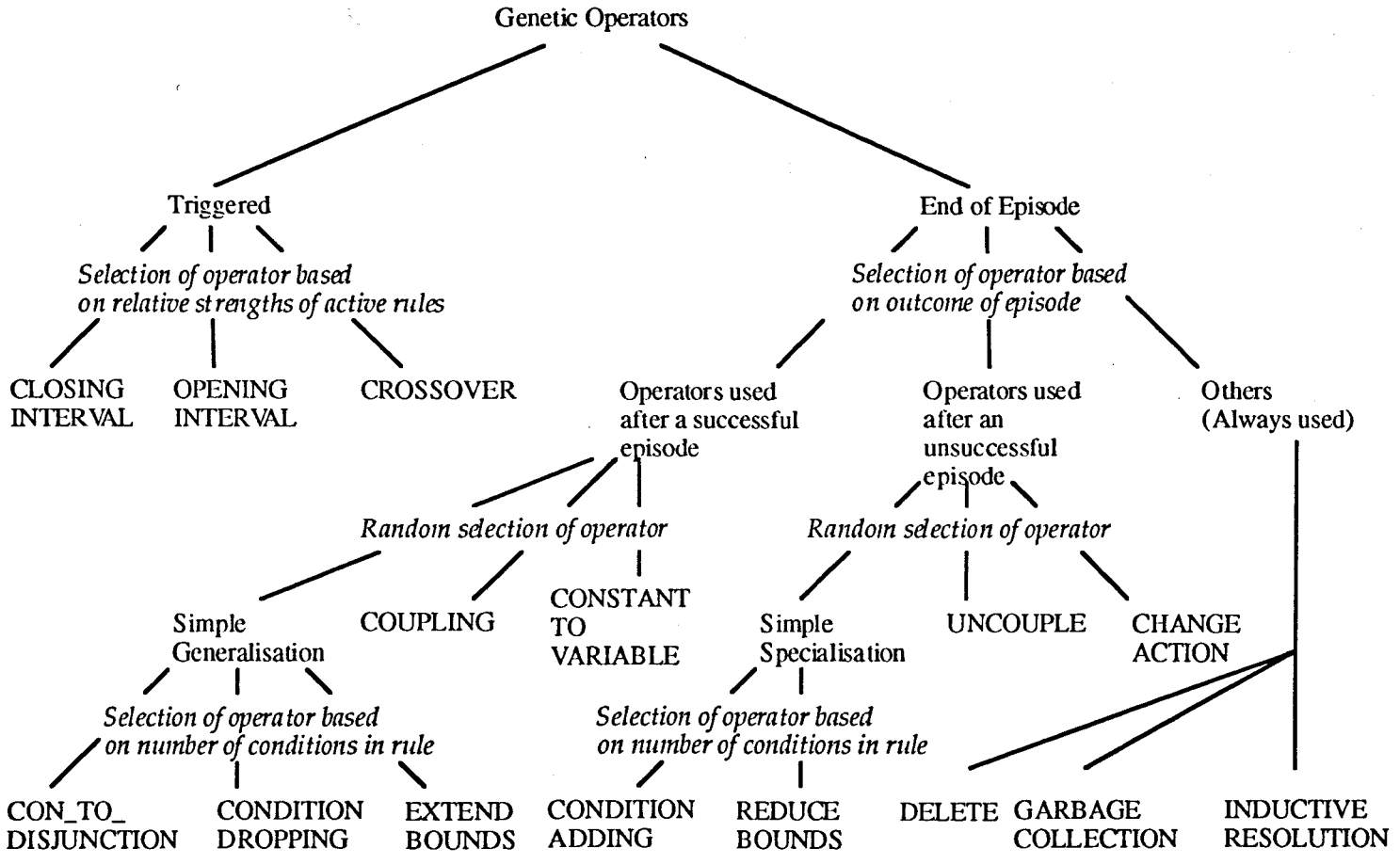


Figure 5.3 Overview of operators employed by SAGA

5.4.5 Repair Mechanism

Due to the limited size of its rule base, a CCS may rapidly forget valuable experience as a result of the deletion of previously useful rules. Two possible solutions to this problem are readily suggested. The first is to consider using a much larger rule base. However, this severely increases the running time of the system without guaranteeing the retention of useful information. The second is to maintain some form of memory. [Zhou, 90] suggests the use of a long term memory of inactive, well-trained rules. However, this will also attract an overhead in terms of time, as well as creating the difficult problem of deciding whether or not a rule has been useful. Another problem with this scheme is that special mechanisms will be required to prevent a number of slightly different versions of a successful rule from dominating the long-term memory.

In the SAGA system, a rather different approach, that of a *Repair Mechanism*, is adopted and the reason for this is as follows. Until recently, it was thought that all mutations occurred at random with respect to their adaptive significance. However, a new class of mutation has recently been reported in unicellular organisms, such as yeast and bacteria, whereby the organism undergoes *directed mutation* [Paton, 92b] to repair "broken genes". The reversion mutation that repairs a gene occurs several orders of magnitude more frequently when the repaired gene is needed than when it isn't. Although the mechanism behind directed mutation is unknown, it has been shown to be under genetic control and not the result of random behaviour like normal mutations. Rather than 'repairing' only selected genes (conditions) as in biological systems, the SAGA system's repair mechanism restores the whole chromosome (rule) to a previous state. It functions by comparing the current strength of each rule with a previous best strength which is stored along with the version of the rule which achieved that strength. If the current strength becomes significantly higher than its previous best, the strength and the corresponding rule are recorded. If however, a rules current strength falls significantly below its previous best, then the rule can be repaired by supplanting the current rule by the earlier superior version. Such an operator is of great advantage in situations for which a generalisation significantly degrades the performance of a rule. The operator does

not however reduce the efficiency of the system because the stored copy is not used during any of the processor intensive phases of system operation such as matching and conflict resolution. The only cost to the system is a doubling of storage in respect of the rules and as storage is unlikely to be a real restriction on most modern machines, then this is not a problem. However, it should be noted that the operations underlying the repair mechanism, unlike those corresponding to such ideas as directed mutation, is fixed and is not influenced by the environment.

5.5 Plan Level Operation

5.5.1 Introduction

The hybrid architecture of the SAGA system is designed such that a second EA, operating at the plan level (as in the Pittsburgh approach), is also employed to facilitate the learning process. This plan-level EA (PLEA) facilitates the learning process by allowing useful rules from different plans to be combined into a single CCS which is better than any of its predecessors.

5.5.2 Main Cycle

The operation at this level is much simpler than that at the rule level; there are no credit assignment algorithms or complex evaluation functions involved. Instead, as was mentioned in relation to the adaptive credit assignment algorithm (section 5.4.3), each CCS has a single fitness measure called its *current form*. This is initially set to zero and is updated at the end of every episode (that is, within the operating cycle of a CCS) using:

$$current\,form = (1\text{-}p)\ current\,form + Pp$$

where p is a constant called the *plan rate* which is analogous to the bid co-efficient and

consequently set to 0.1, and P is the payoff received from the environment at the end of every episode.

A cycle at the plan level simply consists of periodically (every *plan_period* episodes) selecting the best plans (based on their *current form*) and recombining them using a set of plan level genetic operators. No other operations take place at the plan level. The cycle at this plan level, and consequently the system run, terminates when a set number of episodes, *num_episodes*, have been performed.

5.5.3 Plan Level Genetic Operators

The SAGA system employs two genetic operators to recombine rules from different plans - *PLAN_CROSSOVER* and *ELITE_PLAN.*

(a) *PLAN_CROSSOVER*

Crossover at the plan level occurs after roulette wheel selection has been used to generate a mating pool of the most successful plans. With a probability P_{PC}, PLAN_CROSSOVER is applied to each member of the mating pool except the fittest member which is left unchanged. Two versions of the crossover operator exist within PLAN_CROSSOVER, each with a 50% probability of being selected:

(i) *Uniform PLAN_CROSSOVER*

This operation is analogous to the Uniform Crossover operator suggested by [Syswerda, 89] for binary encodings and operates as follows. First, the two parent plans are aligned such that rule 1 in parent A is above rule 1 in parent B, etc. Then, one of the two rule 1's is selected (at random) and allocated to child A - the other rule 1 being assigned to child B. This process is repeated for rules 2 to m, where m is the number of rules in the smaller of the two parent plans. Should the other parent contain a greater number of rules, n say, then the rules $m+1$ to n in this parent are all assigned to one of the two offspring, again

selected at random.

For example, given two parent plans, $Plan_1$ and $Plan_2$,

$Plan_1$: $R_1\ R_2\ R_3\ R_4\ R_5\ R_6\ R_7\ R_8$

$Plan_2$: $R_a\ R_b\ R_c\ R_d\ R_e\ R_f\ R_g\ R_h\ R_i$

the offspring generated by Uniform PLAN_CROSSOVER could be:

$Child_1$: $R_a\ R_b\ R_3\ R_d\ R_5\ R_6\ R_7\ R_h\ R_i$

$Child_2$: $R_1\ R_2\ R_c\ R_4\ R_e\ R_f\ R_g\ R_8$

(ii) One-point PLAN_CROSSOVER

This operator is analogous to the standard one-point crossover operator. However, because the two plans may contain different numbers of rules, the crossing point is selected to be between 1 and m-1, where m is the number of rules in the smaller of the two parents. For example, given the same two parent plans,

$Plan_1$: $R_1\ R_2\ R_3\ R_4\ R_5\ R_6\ R_7\ R_8$

$Plan_2$: $R_a\ R_b\ R_c\ R_d\ R_e\ R_f\ R_g\ R_h\ R_i$

then, the crossing point must be selected to be between 1 and 7. The resulting offspring if position 3 was selected would be:

$Child_1$: $R_1\ R_2\ R_3\ R_d\ R_e\ R_f\ R_g\ R_h\ R_i$

$Child_2$: $R_a\ R_b\ R_c\ R_4\ R_5\ R_6\ R_7\ R_8$

(b) *ELITE_PLAN*

Since the rule level system gives not only the fitness of each plan but also that of each rule, then the system can use this information to create a plan consisting of the best rules from every plan in the population. The operator which the SAGA system employs to do this, ELITE_PLAN, is applied at the end of every episode with a small probability, P_{EP}. It operates by combining into a single plan, all the rules with have a

strength greater than the average strength plus one standard deviation. If more rules are selected than a single plan can accommodate, then the best '*max_num*' rules are deterministically selected to form the new plan. This plan then replaces the plan with the lowest *current form.*

5.6 Conclusions

In this chapter, a functional description of the hybrid genetic learning system proposed in Chapter 1 has been presented. This system has been given the name the SAGA system. One of the major differences between this system and those investigated thus far is the knowledge representation scheme it employs. Since the representation scheme needs to be comprehensive and accurate, it was felt that a symbolic representation scheme in which rules are composed of identifier-argument tuples, would be the most suitable way to represent knowledge in the SAGA system.

The system itself is divided into two levels. At the lower level, there is a series of component classifier systems each of whose architecture is similar to that of the systems studied in the previous two chapters, with the addition of a co-evolving predator CCS to perform the role of the adversary in the 2-player games for which the system was designed. Based on the meta-EA analysis of the SCS-EA system, the credit assignment algorithm employed within a CCS is a hybrid of the PSPV and adaptive bucket brigade algorithms. Moreover, each CCS supports a set of 15 genetic operators whose use depends mainly upon either the occurrence of a special event during the course of an episode, or the outcome of the episode. Finally, one further feature of the system at this lower level is the use of a repair mechanism which it is suggested, may alleviate the problem of 'forgetfulness' in an evolutionary learning system.

At the higher (plan) level, the performance of the system is much simpler; requiring no complex algorithms or credit assignment schemes. Only three operators are supported:

analogues of uniform and one-point crossover and an ELITE_PLAN operator which employs the strengths of the rules derived at the lower level to generate a plan composed entirely of highly fit rules.

Given this description of the SAGA system, it is the purpose of the next chapter to describe an investigation of its effectiveness in the context of a number of simple military problems.

6 Evaluation of the SAGA system

6.1 Introduction

The central aim of this thesis has been to investigate the important features of a evolutionary learning system and to use these findings to construct a system capable of developing tactics for a range of simple military-type problems. The work described in the first four chapters led to the design of a hybrid, co-evolutionary GBML system, the SAGA system, the functional description of which is given in Chapter 5. It is the aim of this chapter to investigate the effectiveness of this system in the context of the problems for which it was designed, and to determine if, indeed, it can be employed in a practical context.

This chapter is organised as follows. Section 6.2 gives details of the numerous parameter settings adopted by the SAGA system for the purpose of performing the evaluation. To verify that the various rule-level operators employed by the system work as anticipated, section 6.3 presents the results from a pre-cursive study investigating their relative benefits, and analyses the roles played by each operator in the rule discovery process. Section 6.4 presents the results derived from the main set of experiments investigating the effectiveness of the SAGA system in respect of the TTP. These experiments investigate the system's performance both against an opponent employing a fixed-strategy and when it operates in co-evolutionary mode. Section 6.5 looks at the application of the SAGA system to two other simple military problems - the *Underwater Warfare Hide and Seek Problem* and the *Collision Avoidance Problem*, and assesses its performance when attempting to solve these. Finally, section 6.6 presents a discussion of some of the key issues arising from the results.

6.2 System parameters

The large number of genetic operators and system mechanisms employed by the SAGA system means that there are many parameters associated with its operation. This, together with the fact that there will be a large degree of interdependency between the operators, means that any attempt to optimize the parameters would be difficult and time consuming. Therefore, the values adopted by the system's parameters are only estimates of suitable values - the estimated values being based on (a) values that are in general use, and (b) the authors opinion derived from experience with similar parameters (for example, when performing the experiments in Chapter 2).

Unless otherwise stated, the values adopted by the parameters employed in the SAGA system are as shown in the following list, together with, where appropriate, a brief explanation or reference to their derivation.

(a) **Parameters associated with Credit Assignment:**

Bid co-efficient	0.1	Used in SCS-EA
Life tax	0.01	Used in SCS-EA
Bid tax	0.1	Used in SCS-EA
PSP rate	0.1	[Grefenstette, 88]
Payoff	(this is problem dependent and is given with the respective problem descriptions in Sections 6.3 and 6.4)	

(b) **Parameters associated with rule-level operators:**

Threshold	100	
Triggering Operators used	TRUE	
Unsuccessful Operators used	TRUE	
Probability (Uncouple),P_{UC}	0.05	An unsuccessful sequence does not always imply that 2 of the moves made

		should not be used together and so this operator should be used sparingly.
P(Generalisation Ops),P_{GO}	0.9	As the Generalisation Operators are only used after a successful episode, they will have a fairly low application rate and so P_{GO} must be set fairly high.
P(Couple),P_{COUP}	0.9	As the problem requires rule sequences to be developed, the COUPLING operator will play an important role in rule discovery and hence, P_{COUP} needs to be set relatively high.
P(Constant_to_Variable),P_{CV}	0.1	A very time-consuming operation and so needs to be used sparingly.
P(Inductive Resolution),P_{IR}	0.1	A very time-consuming operation and so needs to be used sparingly.
Deletion Threshold	0.001	

(c) **Parameters associated with plan-level operators:**

Plan period	10	
P_{PC}	0.7	'Standard' crossover rate
P_{EP}	0.8	An important operation and so should have a fairly high application rate.

(d) **Other parameters:**

Size of message list	5	Used in SCS-EA.
Opponent_type	Problem Dependent	
Initial rule strength	10.0	Used in SCS-EA.

Size of initial rule base	20	Tradeoff between performance and time
Maximum size of rule base	20	taken for both the size of a rule base and
Number of rule bases	10	the number of rule bases.
Random_rule_base	TRUE	Best test for system effectiveness.
Length of Internal message	4	Used in SCS-EA.
Num_episodes	10,000	

The critic employed by the SAGA system is dependent upon the problem to which the system is applied, and hence is described in the relevant problem sections of this chapter. Moreover, for each problem, the value of the constant *max form*, which is used in the calculation of the constant of adaptation, ϑ, is set to the maximum value awarded by the critic.

6.3 An investigation into rule-level operator utility

6.3.1 Experimental details

In virtue of the large number of novel rule-level operators which the SAGA system supports, it was deemed appropriate to perform an investigation into the relative benefits of the operators before proceeding directly to an analysis of the effectiveness of the system. One element of the system which warranted particular attention was the repair mechanism which was introduced to alleviate the problem of forgetfulness which is inherent in classifier systems. The aim of the investigation described below was not to determine absolute levels of performance achieved when employing the novel operators / mechanisms but rather to give a comparison of their relative benefits and to determine their particular roles in the rule discovery process.

As the investigation deals only with learning at the rule level, only a single Component Classifier System (CCS) was considered. Consequently, all the genetic operators which occur at the plan level, such as ELITE_PLAN and PLAN_CROSSOVER, were disabled. Moreover,

the predator B population was disabled, thereby allowing the role of the adversary to be taken by a plan employing a fixed strategy which, for the purpose of this work, involved the opponent, T2, firing on every move. Since credit assignment was also not being studied, a further simplification was made, namely, that of employing only the basic Bucket Brigade Algorithm, in which, given that classifiers i and j are active at times t and t+1 respectively, the strength of classifier i is updated using:

$$S_i(t+1) = S_i(t) + b.spec_j.S_j(t) + P(t) - b.spec_i.S_i(t) - tax.S_i(t)$$

where $S_i(t)$ is the strength of classifier i at time t, $spec_i$ is the specificity of i, and b and tax are the bid and the taxation co-efficients respectively.

The simple version of the TTP used for this investigation involved the two tanks being initially placed randomly at a distance of $19 \leq x \leq 21$ units apart and the range of each gun being set to 15. No obstacles, such as mines or rivers, were present. On each move, T1 had three actions available to it (Move Forward one square, Fire, and Increase Gun Elevation by one notch), and these were subject to the constraints that:

(a) each tank could fire at most (x-18) times;

(b) each gun could only be elevated at most 3 times.

The critic employed by the SAGA system for this game involved awarding 10 points for a victory, 5 points for a draw and no points for a defeat (that is, when T2 destroyed T1).

The investigation took the form of a series of experiments, each examining a different combination of operators. In an experiment, two slightly different versions of the system were used to fully test the range of operators employed by the SAGA system. In the first, the value of the distance from T1 to T2 (*distance_to_target*) was omitted from the environmental message. Therefore, in this version, T1 was effectively blind and had to rely on the state of the internal message list to select an appropriate action (that is, it had to learn a sequence of

actions). However, in the second version, this knowledge was made available to T1, and so it was able to use environmental information, as well as internal messages, to decide upon its actions.

Each experiment consisted of 40 trials of up to 200 games. At the start of each trial, T1 was invested with a plan consisting of six rules. Three of these could only become active on the first move of a game (with each proposing a different one of the three possible actions); the other three being active on all subsequent moves, and again advocating a different one of the three possible actions. In the course of a trial, a subset of the operators listed above were used to modify the plan during and after each episode as appropriate. The success or failure of T1 was recorded at the end of each episode. If it had been successful on 10 consecutive episodes, it was assumed to have found a winning strategy, and the trial was regarded as successful. If, however, T1 had failed to win on any 10 consecutive episodes during a trial, then the trial was deemed a failure. To gauge the success of a specific set of operators, two values were used - the number of successful trials out of the 40 and the average number of episodes it took to find a winning strategy (the *learning rate*).

As the system commenced each trial with a very general plan, the minimum set of operators (*base operators*) that the system in fact would need to enable it to adapt favourably are the specialisation operators (used after an unsuccessful episode) and NO_MATCH. Whilst not essential for learning, but only for reasons of efficiency, the DELETE and GARBAGE_COLLECTOR operators were also included in the set of base operators. Using first the 'blind' version of the system and then the 'sighted' version, bench-mark results were recorded using only these base operators. Subsequently, various combinations of the remaining operators[14] were tested to determine whether they could improve the performance of the system.

[14]INDUCTIVE_RESOLUTION was not considered because no boolean variables were present in the environmental message.

6.3.2 Results

With the large number of run-time parameters, the number of possible operator configurations is very large and so the results reported here are unlikely to be optimal for a CCS in the SAGA system. However, they do reflect the relative performance of each operator in the results derived.

Table 6.1: Results using 'blind' system

Operators	Times Successful (out of 40)	Learning Rate
Base Operators	12	118.7
COUPLING	16	88.2
Repair Mechanism	13	92.0
Triggered Operators	13	116.1
Generalisation Operators	13	90.6
CONSTANT_TO_VARIABLE	12	118.7
COUPLING and Repair Mechanism	17	65.6
COUPLING, Triggered Ops and Repair Mechanism	22	98.1
COUPLING, Generalisation Ops and Repair Mechanism	18	97.2

The first set of results are shown in Table 6.1 and relate to the 'blind' system. When only the base operators are employed, a winning strategy was found on 12 out of 40 trials. The addition of either the repair mechanism, the triggered operators or the generalisation operators led to a very slight improvement but the most considerable increase in performance was observed with the addition of the COUPLING operator. This is what would be expected, for in such a 'blind' system, the opportunities afforded the other operators to improve the plan would be few and far between because of the lack of environmental information. However, the system can build a winning strategy by finding appropriate rule sequences, and it is here that the COUPLING operator is particularly useful. Moreover, because there was insufficient information for it to form useful relationships, the addition of the CONSTANT_TO

_VARIABLE operator had no impact upon system performance. When several operators were added to the system simultaneously, the results became less predictable. Here, the generalisation operators tended to perform poorly, but the best performance was observed when the repair mechanism, COUPLING, and the triggered operators were all used simultaneously.

Table 6.2: Results using 'sighted' system

Operators	Times Successful (out of 40)	Learning Rate
Base Operators	22	87.6
COUPLING	28	107.6
Repair Mechanism	30	74.3
Triggered Operators	31	92.5
Generalisation Operators	15	79.3
CONSTANT_TO_VARIABLE	38	48.1
COUPLING and Repair Mechanism	22	87.6
COUPLING, Triggered Ops and Repair Mechanism	29	89.1
COUPLING, Generalisation Ops and Repair Mechanism	15	81.0

Table 6.2 contains the analogous results derived from the 'sighted' system. Clearly, knowledge of the distance to the target should improve the ability of the system to learn. This was the case and in fact, the system was able to find a successful strategy on 22 out of the 40 trials using only the base operators. Again the repair mechanism, the COUPLING and triggered operators all improved system performance. However the generalisation operators degraded the performance quite significantly - this being largely due to the fact that the other operators were able to develop a useful plan by themselves and generalising a successful plan will only likely make matters worse. As was expected, the CONSTANT_TO_VARIABLE operator performed best of all due to its ability to link together '*distance_to_target*' and '*range*' so that firing occurred whenever the target was in range. Significantly though, employing

more than one operator did not improve performance further and in most cases, system performance was degraded. This is likely to be due to the fact that, as with the SCS-EA system, when employing a number of genetic operators, too many operations will be performed and the constant state of rules being deleted from and added to the rule base means that there is little opportunity for the system to develop a coherent strategy.

6.4 An investigation into the effectiveness of the SAGA system

6.4.1 Experimental details

The aim of this investigation was to determine the SAGA system's overall effectiveness at solving simple military problems. By discovering this, it can be determined whether or not the central aim of this thesis, namely the construction of a system capable of learning military tactics, has been achieved.

The major part of this investigation concentrated on the TTP but two other problems, the Underwater Warfare Hide and Seek Problem [Koopman, 79] and a simple Collision Avoidance task were also investigated to determine if the system could be successfully applied to other military problems. The results recorded from these two problems are reported in section 6.5.

Unless otherwise stated, each experiment consisted of running 1,000 episodes (games) on each of the 10 CCSs, giving a total of 10,000 episodes. Each experiment started with the SAGA system generating a random rule base, Σ, of 20 rules (as described in section 5.4.1) and assigning Σ to all 10 CCSs. In the course of an experiment, the only information received by T1 from the world model was the linear variable 'distance to the target' - no other information was provided by the environmental message.

6.4.2 Experiment 1: CLP1 with a simple critic

In this first experiment, the SAGA system was applied to CLP1 (section 4.4.4) - a slightly more difficult TTP in which T1 has to score two hits on T2 in order to win the game. As with the version in Chapter 4, T2 employs a fixed strategy of doing nothing until it is hit, at which point it either moves forward one 'square', or it moves backward and then forward on the following move. The critic employed for this experiment was fairly simple, namely awarding 20 points for a victory, 10 for a draw, and zero for a defeat.

When the results from applying the SAGA system to 10,000 episodes of CLP1 were analysed, it was found that T1 (the learning agent) failed to win a single game. Instead, it was found that, without exception, each CCS adopted a strategy of 'DO NOTHING' on every move, thus managing to draw every game. The reason for this is straightforward - a CCS, α say, can with relative ease, discover a 'DO NOTHING' strategy and thus, achieve a reward of 10 every game. However, a different CCS, β say, which doesn't adopt this strategy is, in the short term, likely to fail in its attempts to discover a better strategy and will accordingly receive less than 10 points per game. Consequently, α will be favoured by the plan-level EA and hence, over a relatively short time, CCSs adopting the 'DO NOTHING' strategy will come to dominate the population.

In an attempt to rectify this situation, the critic was altered such that no reward was given for a draw. However, the results from employing this critic showed little difference from the previous set with, once again, each CCS adopting a 'DO NOTHING' policy.

6.4.3 Experiment 2: CLP1 with a sub-goal reward critic

As noted in Chapter 4, such a simple critic as that employed in Experiment 1 cannot be used to solve relatively difficult problems such as CLP1. Consequently, the more complex sub-goal reward critic that was described in section 4.4.4 was adopted for use in the SAGA system, the only difference being that the version implemented for this experiment awarded an extra 50

points to T1 if it succeeded in winning the game.

The number of victories achieved by T1 over the 10,000 episodes is shown in Figure 6.1. As the results show, the system gradually improves its performance with time but it still performs relatively poorly after the 10,000 episodes, having achieved a total of only 27 victories.

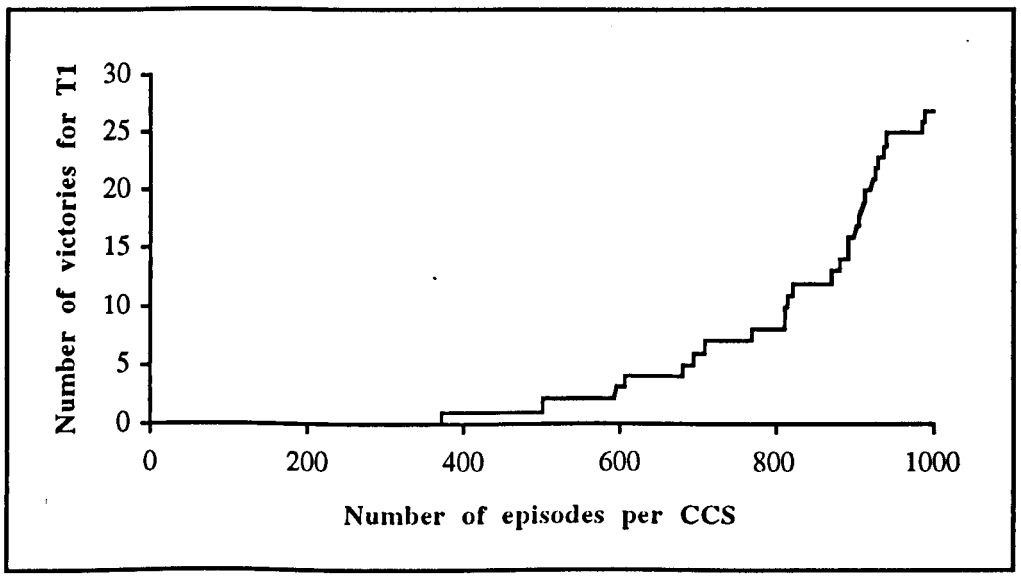


Figure 6.1: Performance of the SAGA system when employing a sub-goal reward critic

Moreover, although it appears from Figure 6.1 that the performance is improving more rapidly with time, this was shown not to be the case by running the SAGA system for a further 10,000 episodes on the same problem. The results from this extended trial are shown in Figure 6.2 and demonstrate that although the system still manages to achieve the occasional victory, the rate at which it achieves them has actually decreased after the first 10,000 episodes. Thus, it would appear that the SAGA system is unable to learn effective tactics for this problem.

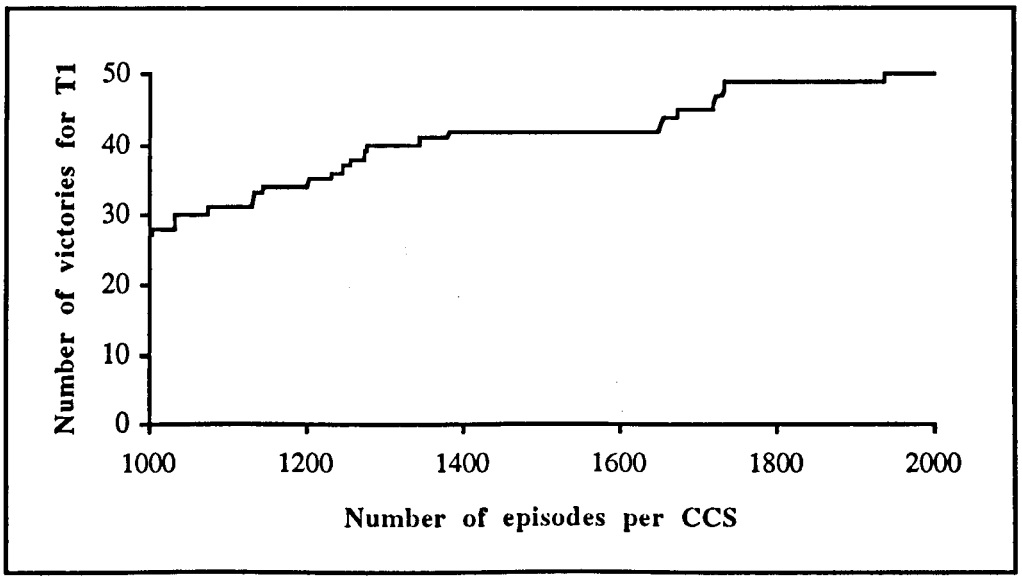


Figure 6.2: Performance of the SAGA system over an extended trial

As with the SCS-EA system, the main reason for the poor performance of the SAGA system when applied to CLP1 is that the system often ends up achieving one of the sub-goals rather than the desired goal. This was demonstrated by two further experiments. In these, the amount awarded to T1 upon winning the game (that is, for hitting the opponent a second time) was reduced in the first case from 59 to 9 and then, in the second case to 1. In the first case when 9 was awarded, the number of victories achieved by T1 was reduced from 27 to 3, and then, when only 1 point was awarded for a second hit, no victories at all were recorded. Upon examining the behaviour developed by the system, it was found that T1 was exploiting weaknesses in the critic and was achieving sub-goals. For example, in the experiment where only 1 point was awarded for a second hit, it was found that for much of the time, the tactic adopted by the system over the 9 moves was to move forward twice (gaining +1 each time) and then alternate between increasing and decreasing its gun elevation (4 increases and 3 decreases giving a net total of +1), thereby achieving an overall score of +3 points. Since T1 only receives, at most, +7 for a perfect game (involving a strategy which is relatively hard to discover) when only 1 point is awarded for each of its seven moves but can achieve +3 from a relatively easily found strategy, it is easy to see how the simple strategy can rapidly come to

dominate the population. Therefore, this experiment suggests that it is vital that there is a smooth transition from one sub-goal state to the next all the way to the ultimate goal; should this not be the case and one sequence of sub-goals lead to a dead end, then the system may develop a set of rules to take this route instead and will never find the ultimate goal.

6.4.4 Experiment 3: An investigation into the importance of the plan-level EA

In Experiments 1 and 2, both ELITE_PLAN and PLAN_CROSSOVER were employed by the SAGA system to facilitate the learning process. The aim of this experiment was to determine their relative utility to the system - an aim which was achieved by setting, in turn, the probability of using ELITE_PLAN,P_{EP}, and the probability of using PLAN_CROSSOVER, P_{PC}, to zero.

The number of wins recorded by the SAGA system when applied to CLP1 with P_{EP}=0.0 is given in Figure 6.3. This clearly demonstrates that although the system can solve the problem without the presence of ELITE_PLAN, it is not as successful as when ELITE_PLAN was employed (only 10 victories as opposed to the 27 achieved with ELITE_PLAN).

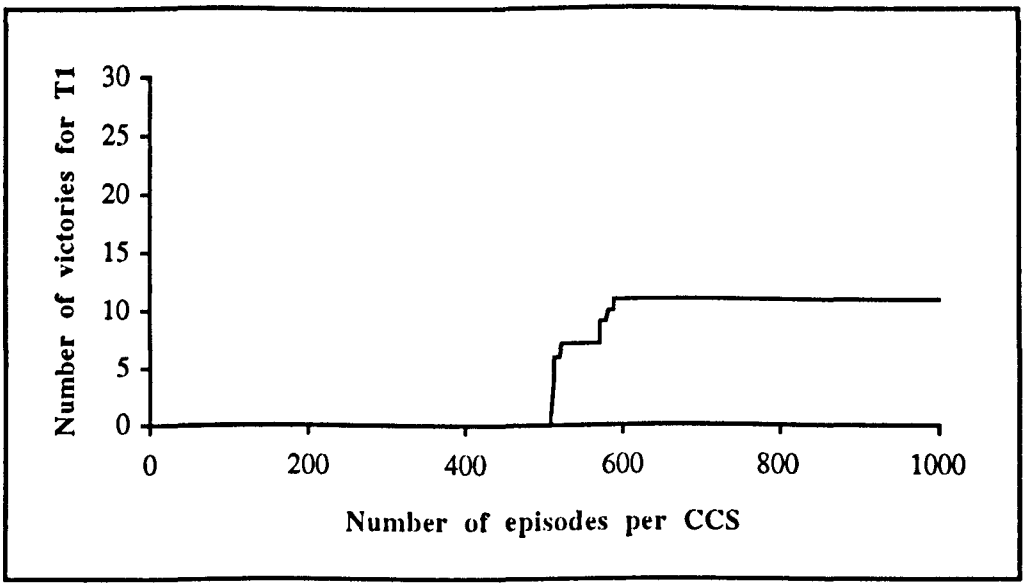


Figure 6.3: Performance of the SAGA system with P_{EP}=0.0

When ELITE_PLAN was restored to the system but PLAN_CROSSOVER removed, it was found that the performance of the system fell very dramatically and no victories at all were recorded. This was almost certainly due to the fact that as a result of design constraints, the plan-level selection process in the SAGA system only occurs if PLAN_CROSSOVER is employed. Consequently, without any selection at the plan-level, the system is reduced to a set of independent CCSs with no communication capability, the effect of which is a dramatic fall in system performance.

6.4.5 Experiment 4: An investigation into the amount of data available to the SAGA system

In the previous three experiments, the only environmental data which the system could base its rules upon was the linear variable 'distance_to_target'. To determine if the addition of more information improves system performance, a second linear variable, 'range', was added to the environmental message and experiment 2 (that is, with a critic awarding +59 for the second hit on the opponent) was repeated. With this extra variable, rather surprisingly, it was found that the number of victories fell from 27 to 2. When the Predator A population was analysed, it was found that almost all of the CCSs adopted a strategy of increasing gun elevation twice and then moving forward three times, thus giving a net overall score of +1.

This, at first, appeared to be very surprising because it would be expected that the more information a CCS has at its disposal, the easier it would be for it to discover useful solutions. However, when the trial was analysed in greater detail, it was found that this poor performance was due to the extra condition significantly reducing the number of rules matching on any cycle of the CCS. For example, the addition of a single variable will mean that, on average, only half the rules that previously matched will match with the new variable in place. This is important because it has the same effect as a reduction in the size of the rule base (cf. size of population) with the consequence that the system performance is significantly reduced. Therefore, this implies that there is a link between the size of the environmental message (and

consequently the length of a rule) and the size of the rule base. This is not surprising since more standard EA applications (see, for example, [Goldberg,et al, 91]) have also demonstrated a link between population size and string length. However, as demonstrated by the number of attempts at quantifying this relationship (see, for example, [Smith, 93]), it is a difficult problem even for simple bit-string representations, let alone more complex representations such as that employed by the SAGA system. Hence, although it has been identified that a link exists between the number of rules in a rule base and the size of the environmental message, the subject requires a much more in-depth analysis than can be undertaken here, suffice it to say that for complex problems which involve many problem variables, very large rule bases are likely to be required.

6.4.6 Experiment 5: An investigation into co-adaptive behaviour

In experiments 1 to 4, the Predator B population was disabled and the role of the opponent was played by a fixed strategy. The aim of this experiment was to investigate what effects occur when the opponent is played by a co-evolving predator B CCS.

The run-time parameters for this experiment were exactly the same as in previous experiments but the problem scenario had to be changed so that T2 was not at an advantage. This was achieved by placing a second river at position 13 (3 'squares' in front of T2) and a second mine at position 17 (directly behind T2).

However, one further problem with the system was that the sub-goal reward critic employed in the previous three experiments could not be employed in this experiment because of the fact that the opponent was no longer deterministic. Consequently, it was necessary to resort to employing a simple critic of the form adopted in experiment 1 within the SAGA system. This in turn meant that the problem, as it stood, would be too difficult for either the predator A or predator B population to learn an effective strategy. Consequently, the problem was made simpler for both players by:

(a) increasing the ammunition level for each tank from 2 to 5;

(b) allowing the opponent to be destroyed by a single hit.

The precise form of the critic employed by both the predator A and predator B CCSs involved awarding 200 points for a victory, 20 for a draw and 0 for a loss.

Figures 6.4 and 6.5 show the number of CCSs in the predator A and predator B populations that record victories for T1 and T2 respectively, the results being a moving average over a period of 500 episodes. The two graphs show that during the earlier iterations of the system, neither tank manages to develop a useful strategy. However, after about 2,500 episodes, T2 develops a successful strategy and starts to win about 50% of games. It manages to maintain this level until shortly after 4,000 episodes, at which point T1 develops a strategy to combat this and it, rather than T2, starts to win on a regular basis. This situation is maintained until shortly after 8,000 episodes, at which point the performance of T1 rises sharply. However, this is short lived and T2 rapidly develops a strategy to counter T1, which dramatically reduces the success rate for T1 to almost zero by the end of the 10,000 episodes.

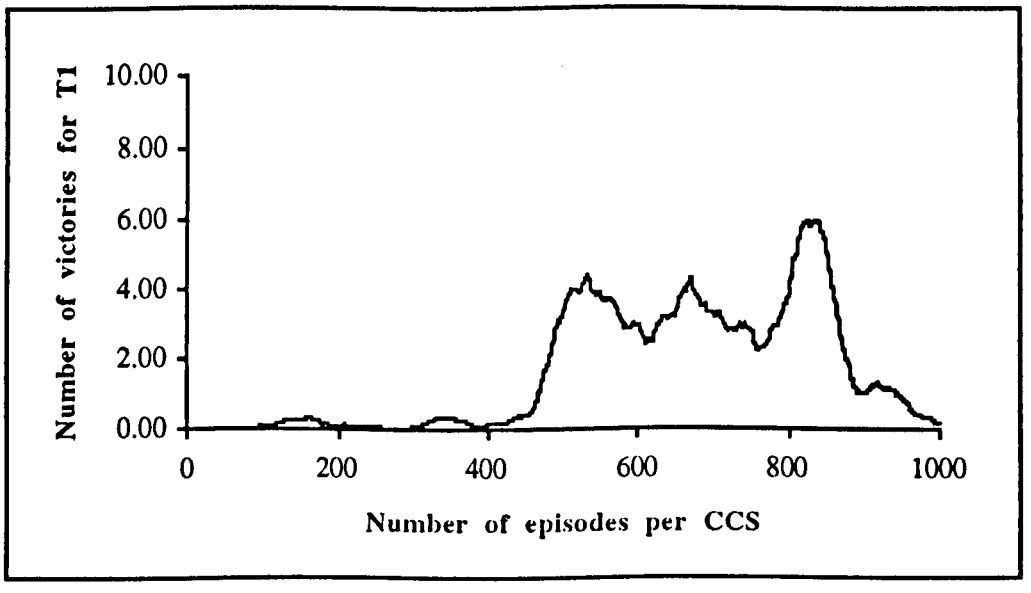


Figure 6.4: Number of CCSs in predator A population that have recorded a victory for T1

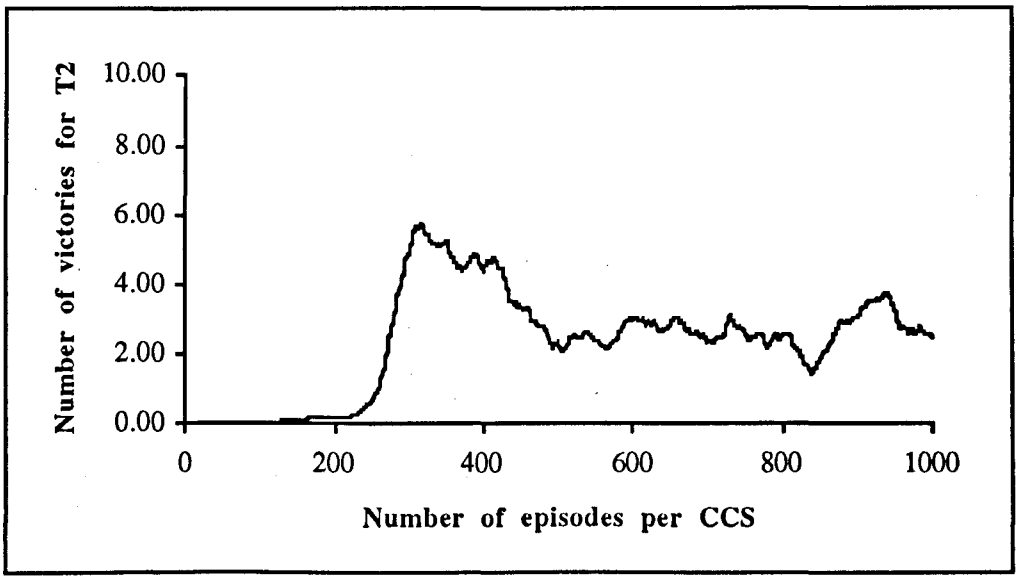


Figure 6.5: Number of CCSs in predator B population that have recorded a victory for T2

These results demonstrate that the SAGA system is capable of displaying co-adaptive behaviour. However, this is likely to be for only simple problems or those more difficult problems for which an appropriate critic exists. For example, when the experiment was repeated with the ammunition level reduced to 2 and two hits being required to destroy the opponent (as in experiments 1 to 4), the number of wins recorded by both T1 and T2 was reduced to zero (as expected when employing such a simple critic).

6.4.7 Experiment 6: An investigation into different predator A and predator B populations

In experiment 5 both the predator A and predator B CCSs (and therefore T1 and T2) were identical. The aim of this experiment was to determine what effect a difference between the two CCS populations has on the results produced. This was investigated by altering the critic for the predator B population such that 400 points were awarded to T2 for winning but leaving the amount received by T1 at only 200 points.

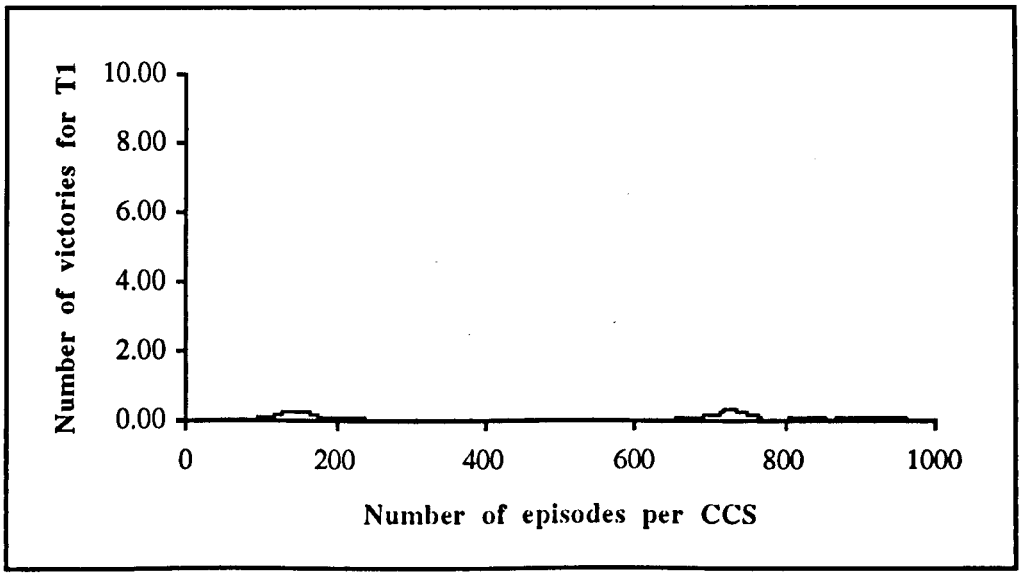


Figure 6.6: Number of CCSs in predator A population that have recorded a victory for T1 when it is awarded less than T2

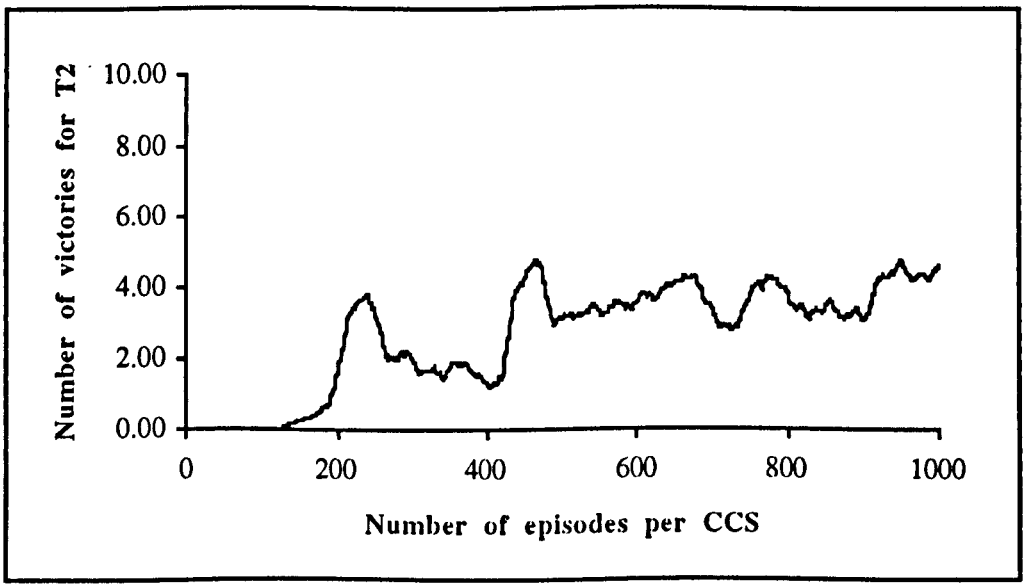


Figure 6.7: Number of CCSs in predator B population that have recorded a victory for T2 when it is awarded more than T1

The results from this experiment are shown in Figures 6.6 and 6.7. The two graphs clearly show that a difference between the critics of the two CCSs can have a dramatic effect upon the

relative performance of the tanks, with, in this case, T1 hardly ever managing to achieve a victory. One conclusion which can be drawn from this is that paying a higher reward may be a profitable strategy to employ. in a learning system. For example, the higher the reward, the quicker a significant payoff reaches the stage-setting rules under the BBA. Moreover, such behaviour is mirrored in the real-world where, for example, "big business" can often gain control over smaller, less-powerful, rival companies.

However, paying larger sums is only likely to be beneficial when rewarding the 'ultimate' goal; giving larger rewards to sub-goals is unlikely to improve matters. This is because the higher the reward paid, the more likely the system is to fix its behaviour at the state where it achieved that reward. Clearly, if the system is achieving the 'ultimate' goal, then this is fine. However, paying a larger reward for achieving a sub-goal makes it more likely that the system will get stuck in the state where it achieved that sub-goal. This was demonstrated in an extension to the above experiment in which the points awarded to T2 after an unsuccessful episode were also doubled. In this experiment, it was found that the performances of both T1 and T2 returned to similar levels to those encountered when employing identical critics.

Moreover, even paying more for achieving the 'ultimate' goal does not guarantee that the CCS employing that strategy will be more successful. For example, when the situation described above was reversed with the predator A population (T1) this time being awarded 400 points for a victory, the results depicted in Figures 6.8 and 6.9 were derived. These results show that, despite being awarded a greater amount, T1 gains no advantage over T2. Furthermore, this cannot be attributed to any unsymmetrical features of the problem since although the two tanks take alternative moves, for all intents and purposes, the problem is symmetrical with the probability of winning being identical for both T1 and T2.

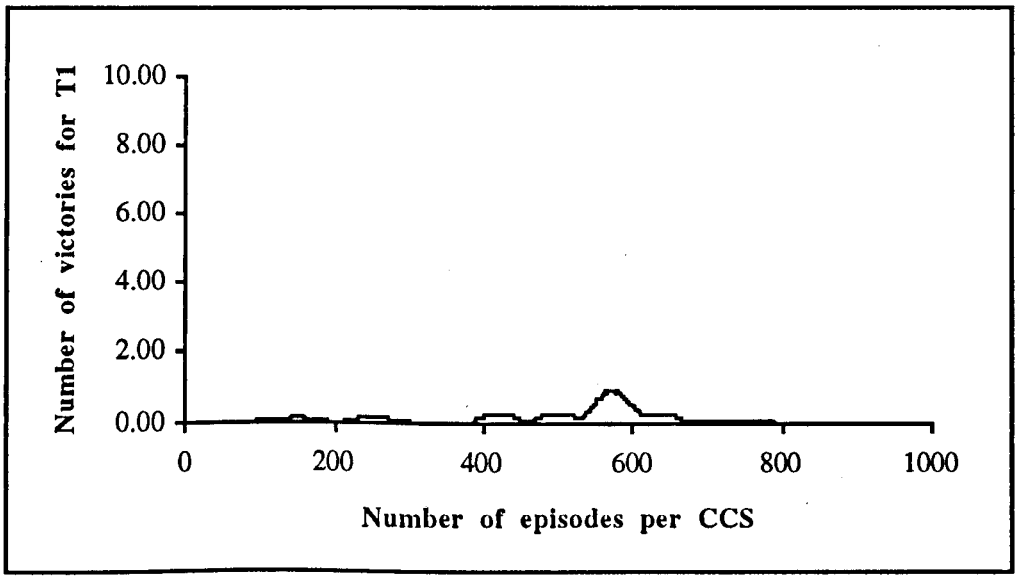


Figure 6.8: Number of CCSs in predator A population that have recorded a victory for T1 when it is awarded more than T2

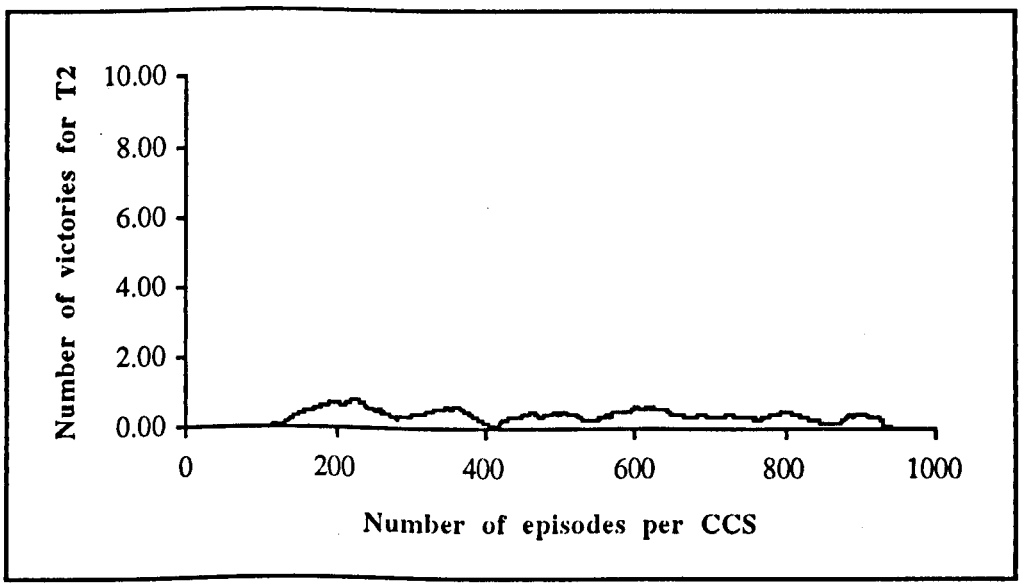


Figure 6.9: Number of CCSs in predator B population that have recorded a victory for T2 when it is awarded less than T1

6.5 Application of the SAGA system to other problems

6.5.1 Two simple military problems

The work of the previous section investigated the application of the SAGA system to several versions of the TTP. To determine if it can be used to develop strategies for other military problems, the SAGA system was applied to two further problems, namely the *Underwater Warfare Hide and Seek Problem* and a simple *Collision Avoidance* task. These problems, together with the results derived from applying the system to them, are detailed in the following sections.

6.5.2 The Underwater Warfare Hide and Seek Problem

The scenario for the Underwater Warfare Hide and Seek Problem [Athanasopoulou, 92] [Koopman, 79] is depicted in Figure 6.10. The problem is set in a confined square area of L^2 units and involves two players, a searcher, *S*, and its target, *T*. The objective of *S* is to detect *T* whilst that for *T* is to remain undetected. *T* has an advantage in that its detection range is 1/2L, whilst that of *S* is only 1/4L. However, *S*'s speed, u, is 50% greater than v, the speed of *T*, and so, *T* cannot hope to simply outrun *S*.

Although at first sight this problem may appear to be very abstract, it can be regarded as a simplification of a real-life problem, namely that of a naval force performing an area search to detect a submarine. In such a problem, the bounded area could represent a confined area of operations such as the Caribbean Sea, and the difference in detection ranges could be as a result of *S* and *T* employing active and passive sonar techniques respectively.

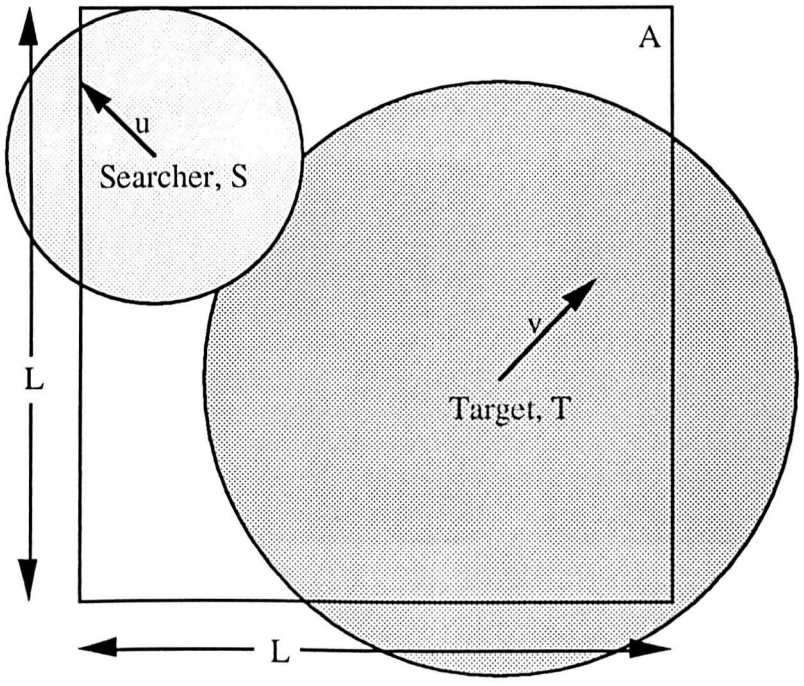


Figure 6.10: Scenario for the Underwater Warfare Hide and Seek Problem

At the start of the game, both S and T head in randomly generated directions. During an episode, they travel in straight lines until either:

(a) T detects S, at which point T may decide upon a new bearing to try to avoid S, or

(b) S detects T, at which point the game ends.

If at any stage during the game, either player comes into contact with the edge of the area, then a 'random bounce' strategy is employed in which that player rebounds off the edge and starts heading in a random direction away from the incident edge.

The aim of the problem investigated here is to determine if the SAGA system can derive a suitable rule set to select appropriate new directions for T upon detecting S. Therefore, the plan to be developed should be a set of rules, each of which has a particular state of the game represented in its conditional part, and an action part specifying a direction in which T should move.

It was decided that the state of the game would be passed to the CCSs (via the

environmental message) in the form of 5 variables:

(a) a cyclic variable recording the relative direction of *S* from *T*;

(b) 4 boolean variables recording whether *T* would, if it maintained its present course, hit the top, bottom, left or right walls of the box respectively. This would give *T* an indication of the orientation of the area of operation.

Given this information, three actions were made available to *T*:

(a) change course to an absolute direction;

(b) change course relative to own direction;

(c) change course relative to the direction of *S*.

6.5.3 Experiment 7: The SAGA system applied to the Hide and Seek Problem

Experiment 7 consisted of applying the SAGA system to 10,000 instances of the Hide and Seek problem. The parameters employed were as described in section 6.2. Figure 6.11 shows a moving average (over the last 500 episodes) of the number of occasions during an episode in which *T* successfully managed to avoid *S*. For comparison, Figure 6.11 also depicts the equivalent results from employing a random-walk strategy (that is, *T* selects a random direction upon detecting *S*) and a relatively successful strategy of moving directly away from *S* (that is, *T* adopts the same course as *S*).

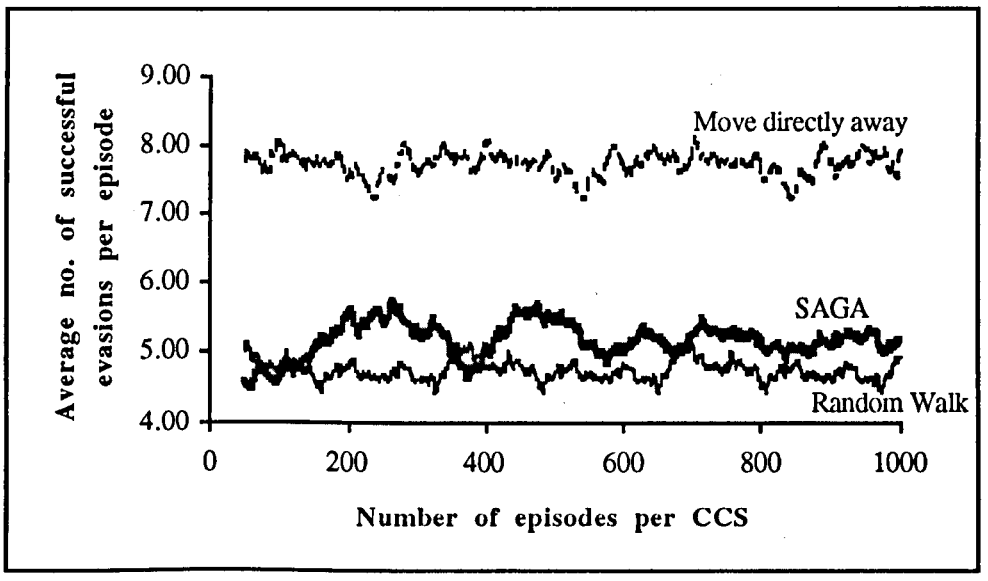


Figure 6.11: Average number of successful evasions

As the results demonstrate, the strategy adopted by the SAGA system is only slightly better than a random walk strategy and significantly worse than the 'move directly away' strategy. To discover why system performance was so poor, the predator A population was analysed and it was found that new rules were being generated too frequently, thus denying a CCS the opportunity to develop an effective strategy. For example, despite sequences of actions not being required for this problem, P_{UC} was still set to 0.9. Consequently, in the above trial, the COUPLE operator was employed to generate a high percentage of coupled rules which have little chance of being useful.

To correct this problem, the probabilities of selecting the various operators were reduced and the experiment repeated to determine if any improvement could be found. The parameters of the original experiment whose values were changed are shown in Table 6.3, and the corresponding results depicted in Figure 6.12.

Table 6.3: New parameter settings

Parameter	Value
P UC	0.0
P IR	0.1
P CV	0.0
Triggered Operators used	FALSE

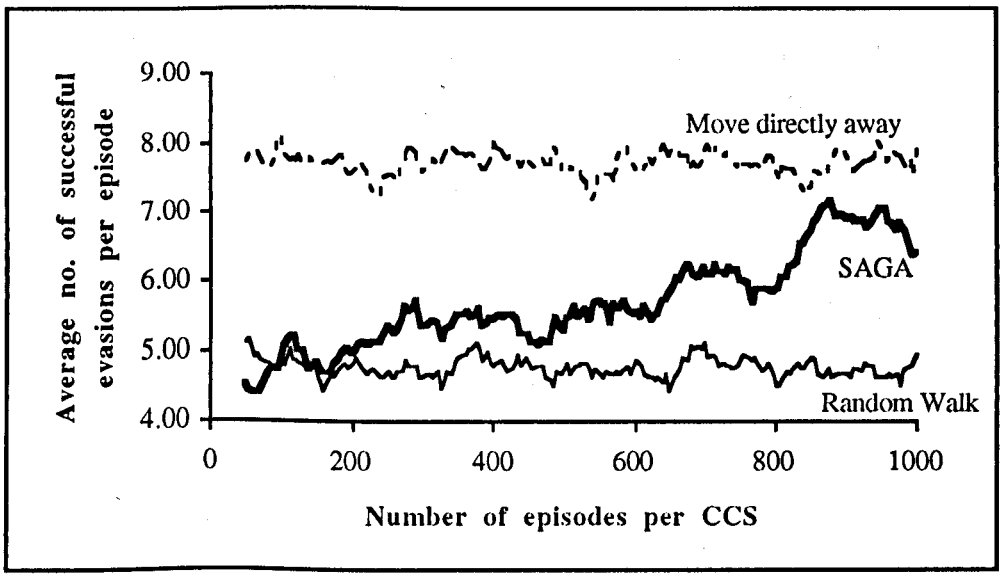


Figure 6.12: Average number of successful evasions with new parameter settings

The new results clearly show a significant improvement in performance with *T* managing to remain undetected for 40% longer with the learnt strategy than with a random strategy. Unfortunately, performance is still relatively poor with the 'move directly away' strategy still superior. However, there is an approximate on-going improvement and it would be anticipated that further optimization of the parameter settings would yield yet greater benefits but, as previously mentioned, this is both a difficult and an extremely time consuming task.

6.5.4 Collision Avoidance

Navigation problems, in which the objective is for an agent to learn a route to reach a particular goal state, are popular for studies in AI. Such problems are also of particular interest to researchers working with *Autonomous Underwater Vehicles* (AUVs), for which the objective is to navigate through a set of obstacles such as a minefield. One way for an AUV to accomplish such a task is to employ a Knowledge Base of stimulus-response rules to govern its behaviour. A preliminary investigation into the application of GBML to the development of such rules for navigation tasks (or *Collision Avoidance* as it is referred to in a military context) has already been carried out by [Schultz, 91] but it was felt that this would still be a useful problem from the viewpoint of analysing the SAGA system's problem solving ability.

The version of the problem adopted for the work reported here (and depicted in Figure 6.13) is set on a 15x15 grid and involves a single player, *A*, whose objective is to reach a goal state, *G*. The initial locations of *A* and *G* are (8, 3) and (8, 13) respectively, with *A* initially facing north (that is, towards *G*). To reach the goal state, *A* has four actions which it can take on any move of the game. These are:

(1) move forward 1 position;
(2) move backward 1 position;
(3) turn clockwise 90°;
(4) turn anticlockwise 90°.

However, standing in the way of A reaching G are three forms of obstacle:

(1) mines - these are placed around the perimeter of the grid;
(2) a forest - this is situated in the centre-most 25 grid locations;
(3) mud patches - n mud patches, $0 \leq n \leq 8$ are placed at selected locations around the grid - the number and locations of the patches being dependent upon the version of the problem used (see later).

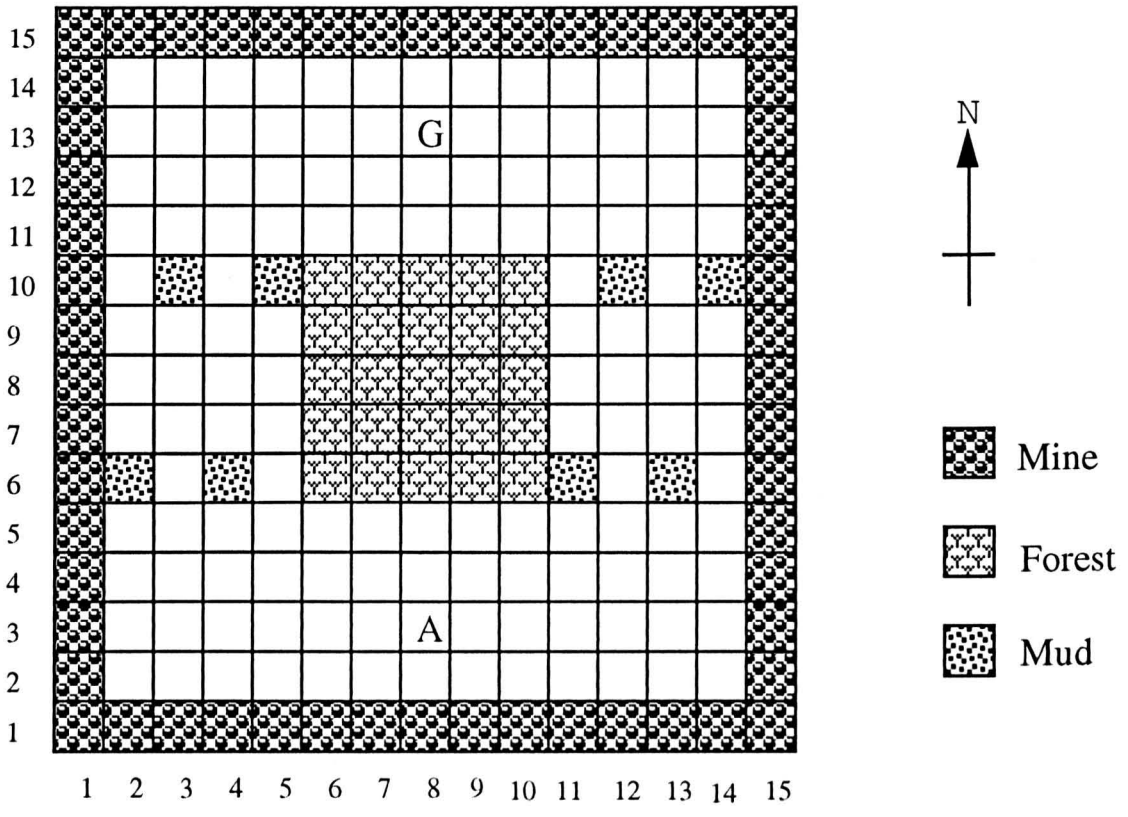


Figure 6.13: Scenario for Collision Avoidance problem

A loses the game if it moves to a location occupied by one of the obstacles, or fails to reach the goal state within 30 moves. To enable it to avoid the obstacles, *A* receives 8 items of environmental data on the state of the game. These are:

(1) direction - this gives the direction in which *A* is currently facing and is set to one of 0, 90 180 or 270 corresponding to north, east, south and west respectively;

(2) dist_to_forest - this gives the distance from *A* to the forest but only in the direction which *A* is facing. If *A* is not facing the forest, then this distance is effectively infinite and so is set to a large value (100 is sufficient for this game);

(3) dist_to_mine - this gives the distance from *A* to the minefield in the direction which *A* is facing.

(4) first_move - this is a boolean variable and is set to true on the first move of the game and false otherwise;

(5-8) mud_front, mud_behind, mud_left, mud_right - these are also boolean variables and are set to true if there is a mud patch in front of, behind, to the left of and to the right of A, respectively, relative to the direction in which it is facing.

Three versions of the problem were studied in this investigation:

(a) CA1 - no mud patches;

(b) CA2 - 4 mud patches in positions (11,6), (12,6), (13,6) and (14,6), thereby blocking a route to the east of the forest;

(c) CA3 - 8 mud patches in positions (2,6), (3,10), (4,6), (5,10), (11,6), (12,10), (13, 6) and (14,10) (as depicted in Figure 6.13) thereby forcing *A* to perform an extra manoeuvre through the mud.

6.5.5 Experiment 8: The SAGA system applied to the Collision Avoidance Problem

Experiment 8 involved the application of the SAGA system to the CA problem. The parameters employed for this investigation were those used with the Two Tanks and Hide and Seek problems. For the critic, it was deemed necessary to employ one which rewards sub-goals, which, for this problem, are to get as close to G as possible and to survive for as long as possible. Therefore, the payoff, P, awarded at the end of an episode of the CA problem is 100 if G is reached, otherwise P is

$$\frac{\left(\frac{length\_of\_game}{2} + dist\_north\right)}{10}$$

where *length_of_game* is the number of time steps that the game lasted and *dist_north* is the distance that *A* had moved north during the game - calculated by taking *A*'s original y co-ordinate (that is, 3 in this case) from its final y co-ordinate.

Firstly, the SAGA system was applied to 5000 instances (that is, 500 per CCS) of CA1. Figure 6.14 shows the number of CCSs in the population (averaged over 500 episodes) which successfully manoeuvre *A* from its starting point to *G*.

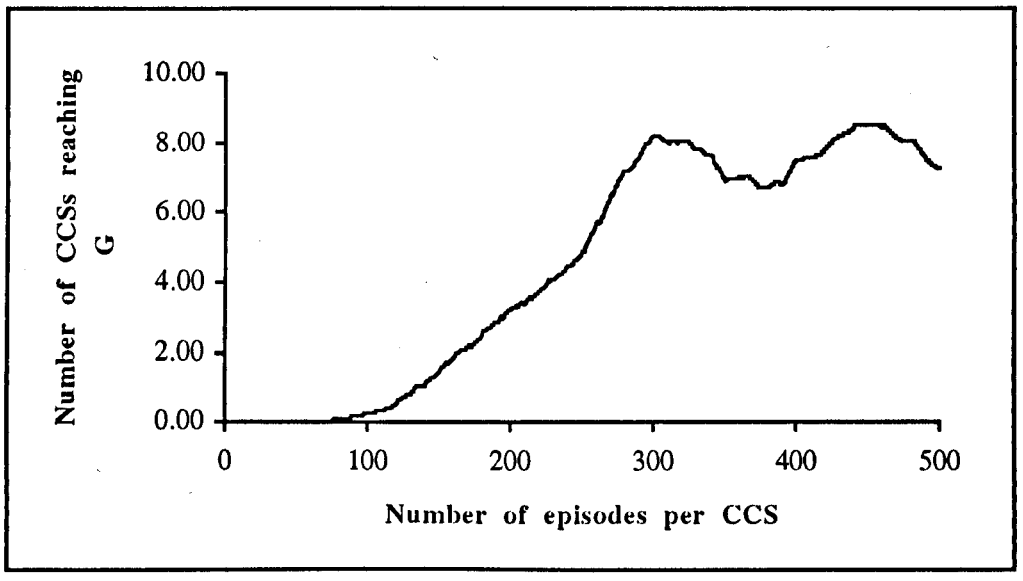


Figure 6.14: Success rate of the SAGA system when applied to CA1

The results show that the SAGA system requires fewer than 1000 episodes before it discovers a successful route. After its initial successes, the successful plan is reproduced by the plan-level EA and quickly spreads through the population. However, as the results show, the system is unlikely to achieve a success rate above 80% because it is continually generating new rules to test and many of these will suggest poor actions to take.

When the performance of the system was analysed more closely, it was discovered that the strategy adopted by all the CCSs was to take the route to the east of the forest using the following actions:

(1) turn 90° anti-clockwise on the first move;

(2) move backward 6 times;

(3) turn 90° anti-clockwise;

(4) move backward 10 times;

(5) turn 90° anti-clockwise;

(6) move backward 6 times.

Consequently, to defeat this strategy, the SAGA system was then applied to CA2 in which the route to the east of the forest was blocked. The results when the system was applied to 5000 instances of CA2 are shown in Figure 6.15.

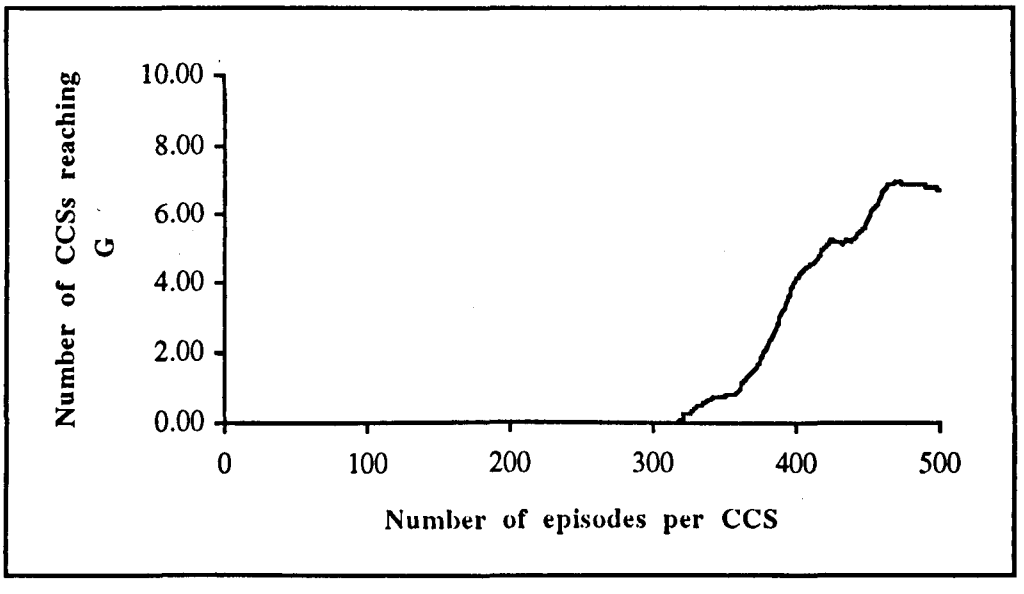


Figure 6.15: Success rate of the SAGA system when applied to CA2

As the results show, with the eastern route blocked, it takes the SAGA system over 3000 episodes before it successfully achieves its goal for the first time. However, as with CA1, once the system discovers an appropriate route, the plan detailing it spreads rapidly through the population so that after only 4500 episodes, on average, 6 out of 10 CCSs succeed in manoeuvring *A* to *G*.

The results from applying the SAGA system to CA1 and CA2 have demonstrated that the system can develop a successful routing strategy to navigate around a single object (the forest). To determine whether it could solve more difficult problems, the system was then applied to 5000 instances of CA3, a problem which requires the system to discover fairly well developed collision avoidance and navigation strategies. When the results from the trial were analysed, it was found that the system failed to win a single game. Instead, it was found that the strategy adopted by all the CCSs was to move forward on the first move and then rotate on the spot for the remaining 29 moves, thus achieving a payoff of (30/2 + 1)/10 = 1.6. Therefore, once again, it has been demonstrated that, given a relatively difficult problem to solve, the system ends up achieving a sub-goal, thereby managing to receive a small amount of credit, rather than trying to achieve the ultimate goal, in which case, because of the difficulty of the problem, it is likely to receive less credit.

6.6 Discussion of results

The results from experiments 1 to 8 show that the SAGA system can discover successful strategies for relatively simple problems but fails to do the same for more complex problems. Throughout the experiments, the common factor behind the cause of this poor performance is the critic. If a simple critic is employed in which no reward is given except for a win, then for relatively complex problems where the probability of a random strategy succeeding is extremely small, the system has almost no chance of developing a successful strategy. However, by employing a critic which rewards sub-goals as well as ultimate goals, for certain non-trivial problems (for example, CLP1) it was found that relatively complex, successful strategies could be developed. However, more often than not, this was not the case. Instead, the main difficulty with using a sub-goal reward critic was found to be that the small amount of credit a CCS receives for a sub-goal is usually sufficient for it to survive, reproduce and eventually dominate the population. The problem can be somewhat eased by better designed

critics but for the majority of problems, critics which support the gradual development of solutions do not exist. Consequently, for any candidate problem for which an appropriate critic cannot be found, it is highly unlikely that the SAGA system will be able to discover an appropriate solution and given that the vast majority of military problems do not possess such a critic, then the system is likely out be of limited use.

When the results generated in experiments 1 to 8 are compared with those generated by the system most similar to the SAGA system, namely SAMUEL [Grefenstette, et al, 90], at first sight it appears that the SAGA system performs significantly worse on a similar set of problems. For example, when applied to the Evasive Manoeuvres problems, the Cat and Mouse problem, and a dogfighting scenario, SAMUEL exhibits impressive performance with its success rate rising from between 20 to 40% for the initial population, to 80% after 100 generations of the EA. The results for a navigation problem [Schultz, 91] are even more impressive, rising from 8 to 96% success rate after 100 EA generations. However, a more in-depth analysis shows this not to be the case with the difference in results being due to the disparate nature of the test problems used by both systems in their evaluation.

The crucial difference between the two sets of test problems is the probability of a random strategy succeeding. For all the problems used to investigate SAMUEL, the results show that the performance of an initial plan is always between 8 and 40%. This means that, even at the start of the learning process, SAMUEL will have plenty of opportunities to learn from successful games and this is vital if SAMUEL is to learn successfully since the operators it employs rely on a relatively high rate of success to learn efficiently. By comparison, the probability of a random strategy succeeding for the problems used to investigate the SAGA system is extremely small. For example, a calculation for CLP1 reveals that the probability of T1 hitting its opponent twice in the limited time whilst avoiding all the pitfalls is only 3.3×10^{-5}. If such a problem was used with SAMUEL, then it is likely that SAMUEL's performance would be as poor, if not worse than that of the SAGA system because of its reliance on a relatively high success rate. An alternative to this form of comparison would be to test the SAGA system with SAMUEL's set of test problems. However, in view of the

considerable effort that would be required to generate the complex environments used to represent the world model in SAMUEL (eg. aircraft simulation models), it was felt that there is little mileage in doing this because SAMUEL's test problems appear to be unrealistic. For example, in the Dogfight scenario [Grefenstette, 91b], the chances of moving the cockpit controls completely at random and being successful against even a novice opponent can only be extremely remote. However, as SAMUEL starts off with a success rate in the region of 40% for a random plan for this problem, it would appear that the test problem must be either extremely simplistic or very unrealistic in its modelling of the environment.

6.7 Conclusions

This Chapter has presented the results from a series of experiments investigating various aspects which affect the SAGA system's performance. The first set of experiments, reported in section 6.3, investigated the system's rule-level operators in the context of a simplified version of the TTP. Although the SAGA system as a whole performed relatively poorly, a number of the mechanisms it employs were shown to make useful contributions to the learning process. Thus, the following tentative conclusions about the various rule-level operators can be suggested:

- for situations in which the environment provides insufficient data, the COUPLING operator proved particularly useful;

- the CONSTANT_TO_VARIABLE operator is necessary to exploit common environmental features;

- the addition of the triggered operators (CLOSING_INTERVAL, OPENING_INTERVAL, and CROSSOVER) improved system performance, especially when more

environmental data was available;

- the generalisation operators (CONJUNCTION_TO_DISJUNCTION, CONDITION_DROPPING, and EXTEND_BOUNDS) may degrade performance by over-generalising successful rules.

Moreover, the investigation suggested that adoption of the repair mechanism does indeed improve the learning capability of a system when used in tandem with a range of different operator combinations. This is due to the mechanisms ability to reinstate previously successful rules when a generalisation has produced a relatively poor rule.

Section 6.4 presented the results from a set of experiments investigating the SAGA system's overall effectiveness at solving the TTP. The results from experiments 1 and 2 show that, by employing suitable sub-goal reward critics, some progress can be made towards solving relatively difficult problems (eg. CLP1) but that typically the system ends up developing solutions which achieve some of the sub-goals (as defined by the critic employed). It is important to note that for the same problem (that is, CLP1), the SAGA system actually performs worse than the relatively simple SCS-EA system investigated in chapter 4.

Experiment 3 investigated the importance of the two plan-level operators employed by the SAGA system, with the results showing that ELITE_PLAN, whilst not critical to the learning process, is certainly beneficial , but that PLAN_CROSSOVER is necessary if complex solutions are to be developed. This is almost certainly due to the fact that selection and reproduction at the plan level only occur when PLAN_CROSSOVER is employed. Consequently, without PLAN_ CROSSOVER, the SAGA system operates as a set of independent CCSs, and this significantly reduces system performance.

This chapter has also presented details of a brief investigation into the possibility of developing appropriate strategies through the use of co-evolving learning agents. The investigation demonstrated that for relatively simple problems, the system does indeed exhibit co-evolutionary behaviour. However, this is unlikely to be of much use because, as with fixed

opponents, unless an appropriate critic can be found, both learning agents are likely to just solve their own sub-goals and little progress will be made by either agent towards developing a strategy to solve the desired goal.

Finally, the results from applying the SAGA system to two other military problems are presented. When applied to the first of these, the Underwater Warfare Hide and Seek Problem, the SAGA system initially demonstrates poor performance and barely improves upon a random-walk strategy. However, after altering several operator rates so that fewer new rules were generated, it was found that the system performed much better, although the performance level thus achieved was still somewhat less than that achieved with a relatively simple ad hoc strategy. Moreover, with the second problem, a collision avoidance task, the results achieved were fairly similar to those for the TTP in that effective solutions could be generated for relatively simple versions of the problem but for more complex cases, the system failed totally. Once again, the reason for this poor performance is a consequence of the system being encouraged to achieve sub-goals rather than the desired goal.

7 Summary and Conclusions

7.1 Summary

The preceding six chapters of this thesis present the current state of the author's research into the application of Evolutionary Algorithms (EAs) to the development of military tactics.

Chapter 1 identified some of the problems associated with generating the knowledge required for Command and Control (C^2) systems, and reviewed a number of potential solutions from the field of Artificial Intelligence (AI). Of these, EAs, as part of a Machine Learning (ML) system, were identified as being particularly suitable. Two of the available approaches to using them in a ML environment were considered, namely the Michigan and Pittsburgh approaches, and a hybrid of the two approaches proposed as the architecture to be adopted for further study.

Chapter 2 presented some of the results deriving from an investigation into ways of improving the performance of an EA in terms of the solution generated. The work was based on a simple Operational Research (OR) problem, the knapsack problem, and focussed on five facets of EAs including mating schemes, implementation of the crossover operator and mutation rates. Two of the key conclusions from this work were that a Lamarckian crossover operator and a mutation rate which incorporated environmental information significantly improved the performance of the EA when compared to standard schemes.

Chapter 3 addressed the credit assignment aspect of a Genetics-based Machine Learning (GBML) system and presented details of two investigations in the area. The first was of a more theoretical nature and employed ideas from a relatively new branch of mathematics, Evolutionary Game Theory (EGT), to analyse the evolution of a hypothetical rule set. The main conclusion from this work was that the standard method of assigning credit in a GBML system, the Bucket Brigade Algorithm (BBA), was shown to be a form of Evolutionary Stable

Learning Rule (ESLR) and, as such, does not tend to maximise the expected payoff to the system but rather evolve the system towards an Evolutionary Stable (ES) state. The second investigation demonstrated that the presence of three particular types of redundant rule (duplicate, subsuming and equivalent) in a plan can significantly affect the rule strength as determined by the BBA. A number of amendments to the BBA were then proposed and two of these were shown to help alleviate many of the problems identified.

Chapter 4 presented a description of a classifier system for the Two Tanks Problem (TTP). Incorporated in this system were a number of novel features including a mutation operator with an adaptive application rate and an adaptive BBA which gives preference to different types of rule depending upon the system's performance level. To facilitate the analysis of the features of this system, a meta-EA was employed to search the space of possible operator / mechanism combinations and determine which were generally beneficial to the system. This investigation showed that the adaptive BBA was usually selected by the meta-EA as being the best credit assignment scheme, but that not all the genetic operators were always of benefit to the system. For example, it was shown that employing three types of coupling operator together with two forms of mutation operator was excessive, and tended to degrade performance.

Using the results from chapters 2 to 4, a version of the system first proposed in chapter 1 was devised; its functional description being given in chapter 5. This system, called the SAGA system, incorporated an adaptive credit assignment scheme, Lamarckian operators, and a repair mechanism (used to alleviate problems of forgetfulness) in a co-evolutionary architecture which also allowed the opponent in a 2-player game to evolve.

Finally, chapter 6 presented an analysis of the results derived from applying the SAGA system to three military problems. The results showed that, in general, the system can develop solutions to relatively simple problems, but for more complex problems, it tends to generate solutions that achieve sub-goals which are necessarily defined by the critic.

This chapter completes the thesis by drawing together the most important conclusions from this work and relating them to some of the directions which future research in the field of

EAs, and GBML in particular, should take.

7.2 Conclusions

The SAGA system is relevant in the field of EA research because it combines a number of novel features into a single genetic learning system. In particular, it is one of the first systems to combine the most appropriate features of both the Michigan and Pittsburgh approaches in a single hybrid architecture - the only others being SAMUEL [Grefenstette, et al, 90] and GIL [Janikow, 91]. For credit assignment, the SAGA system incorporates an algorithm which is both adaptive and a hybrid of the BBA and an epochal scheme. Moreover, the SAGA system is one of the few GBML systems which incorporates a co-evolutionary component in the learning process. Finally, the SAGA system employs a wide variety of novel genetic operators for the rule discovery process and incorporates a repair mechanism to alleviate problems of forgetfulness which are often inherent in genetic learning systems. Thus, the results achieved with the SAGA system can be used to make some tentative conclusions about a wide range of research topics which are currently of interest to the EA community.

One of the primary conclusions derived from this work is that increasing the complexity of an evolutionary learning system does not necessarily lead to an improvement in performance. This is illustrated by the fact that the SAGA system actually performs worse with one problem (the Combined Learning problem CLP1 - section 4.4.4) than the simple classifier system SCS-EA, investigated in chapter 4. There are a number of reasons why this might be the case.

Primary amongst these is that there are too many parameters and too much randomness involved to be able to identify, with ease and certainty, what is happening in the system and what the cause(s) of poor performance might be. Specifically, the SAGA system employs in excess of 30 user-defined constants and run-time parameters, and because of the timescales involved, it was not possible to perform any significant degree of optimization of their values.

Instead, only a very limited investigation involving the trial and error analysis of a small number of parameters was possible. Consequently, it is almost inevitable that some constants / parameters will be employed with inappropriate default values, thereby causing the system to perform at a sub-optimal level. It may have been more appropriate to employ a smaller set of operators and mechanisms than that which was employed in the SAGA system but the complexity of the knowledge representation in particular, and the system architecture in general (eg. EAs at two levels, a hybrid credit assignment scheme and the ability to support co-evolution), meant that this was not possible.

A further reason why highly complex systems such as the SAGA system may produce poor results is the presence of software errors. Throughout the design, implementation and testing of the SAGA system, standard software engineering practices were adopted. For example, the system was designed in a top-down fashion with all the system mechanisms broken down into simple units with testing taking place not only at the system level but also at the module and sub-system levels. However, validating programs as large as the SAGA system (in excess of 10,000 lines) is a highly non-trivial task even when extensive timescales, manpower and other facilities are available. Given the large degree of randomness that is inherent in the selection process, the genetic operators, etc, and the reduced timescales, only a limited amount of testing of the SAGA system could be performed, and it is highly probable that the final system does contain some software errors. Moreover, the effects of these errors cannot be predicted. Clearly then, software validation schemes for systems such as the SAGA system must play a vital role in system development, and this point will be discussed further in section 7.3.

Another conclusion which can be drawn from this work is that, due to the poor results generally achieved with classifier systems, hybrid approaches, which rely upon much of the learning taking place at the rule-level, are unlikely to be successful. (The inability of the classifier system part of a hybrid architecture to learn effectively means that the knowledge necessary for the higher level to be successful will be absent.) This is probably the main reason behind the lack of success achieved with the SAGA system.

The Pitt approach, however, appears to be a much more suitable approach to GBML, not least because it avoids the very difficult issues associated with credit assignment. This view is reinforced by the recent successes that have been reported with Genetic Programming (GP) which is essentially a Pitt approach with a different knowledge representation (that is, Lisp programs are used rather than production rules). The only circumstances wherein a Pitt-based approach will generally perform less ably than one that is Michigan-based are those for which there is a continuous stream of payoff (that is, payoff after each action in a problem solving sequence rather than only at the end). However, because of their success in such circumstances, the use of neural networks would appear to be more suitable.

Another important issue which has arisen from this work is that, although some authors have urged caution and suggested that it is too early to judge the success or failure of a particular approach, it is the authors opinion that classifier systems face too many problems to be of any great use with complex, real-world problems. Primary amongst these difficulties is the credit assignment problem, and this has been a recurrent theme throughout the thesis. Classifier systems may provide an excellent vehicle for studying complex systems and their interactions but for performing a specific task such as discovering tactics, they are unlikely to be the best learning mechanism. Whilst this point of view is apparently at odds with the recent work of [Dike and Smith, 93] wherein a classifier system (GLS) successfully discovered novel tactics for an agile fighter aircraft, the author has reservations about the complexity of the applications to which their system was applied. The reason for the authors scepticism here is two-fold. The first is that most other researchers have achieved little success in applying classifier systems to complex, real-world problems - a view which is reinforced by the findings from this thesis. Therefore, for the GLS system to give such an impressive performance is quite surprising. Secondly, scenarios exist wherein developing an effective solution is not too difficult [Fairley and Yates, 94a]. For example, over 250 million possible condition/action states are possible in the GLS system. However, with regard to its particular test problem (a dogfight scenario), it appears that a particularly adept manoeuvre involving spiralling downwards is possible with only a single action. Furthermore, as this manoeuvre is effective

at any stage during a problem run, the conditional part of the rules becomes almost irrelevant and this effectively reduces the search space to that of only 256 possible actions. Thus, the success of GLS is likely to be a result of the particular problem to which it is applied. However, an in-depth analysis of both the problem and the solutions generated by GLS would be required before a more definitive statement could be made.

7.3 Future research

From the work presented in this thesis, the following are areas which the author feels may prove fruitful with further investigation.

(a) *Repair mechanisms for GBML systems*

The results reported in chapter 6 demonstrate that a repair mechanism shows great initial promise as a means for alleviating the important problem of the deletion of useful rules in a GBML system. This is probably the most significant of the novel operators / mechanisms employed by the SAGA system and warrants further analysis to determine its effect in other GBML systems.

(b) *Operators for ESSs*

The investigation of evolutionary stability and the BBA in chapter 2 illustrates that, given my old assumptions, the rule base in a classifier system will develop into an Evolutionary Stable Strategy (ESS). However, upon application of genetic operators, this strategy is likely to be disrupted, and will take a number of iterations before the rule base again defines an ESS. This suggests that if a genetic operator could be developed which, when applied to a rule base, left the rule base in an ESS (whether it be the same ESS or an alternate one), then this could be of immense benefit to the learning process by significantly reducing the number of iterations required to evaluate a plan.

(c) *Validation of software / analysis of results*

As identified in section 7.2, validating the code for EAs in general, and GBML systems in particular, is an extremely difficult and time consuming task. The benefits of some form of tool to facilitate this process would, therefore, be of immense benefit. The tool could take the form of either a library of standard EA / GBML routines, or a statistical package for analysing the processes occurring in an EA / GBML system.

Some progress is being made towards the first of these goals in that many researchers are making their code available to others via the Internet, and as such, a library of validated routines is beginning to form. However, it is the authors opinion that there are many potential benefits to be achieved by investigating the second option, that of a statistical analysis package for EAs. This opinion is based on the fact that such a package would not only facilitate the validation process, but would also help analyse some of the more complex effects that are produced by the operators / mechanisms that are currently being developed in the EA community.

(d) *An additional possible level of EA application*

Throughout this thesis and in the GBML community in general, it is assumed that there are only two levels at which the EA can be applied in a GBML system, namely the rule-level and the plan-level (corresponding to the Michigan and Pitt approaches respectively). However, yet another level becomes apparent if a plan is viewed as being partitioned into q subsets, where $1<q<r$ (the number of rules in the plan), in such a way that the rules in each subset are related in some way. The rules constituting a subset can be viewed as a population, and since every rule has an associated strength, the population may be subjected to an EA.

The immediate difficulty presented by utilising an EA at this 'sub-plan' level is the necessity of applying it q times. However, this may be answered by using parallel processing facilities, thereby attracting only a minimum overhead. The benefits, on the

other hand, are likely to depend heavily upon the partitioning scheme, that is to say, upon the size of q and the nature of the relationship between the rules in the individual parts of the partition (sub-plans). For example, when discussing classifier systems in chapter 1, the tendency of the EA to search for a 'super-rule' was noted. Analogously, when applied to a sub-plan, it would be anticipated that the EA would tend to seek a super-rule commensurate with the conditions represented in the antecedents of the rules in that sub-plan. Now, whereas a single efficacious super-rule that will cover all contingencies is unlikely to exist, a set of super-rules (one for each sub-plan) may be viable, depending, of course, on the partitioning scheme. Moreover, as well as improving the search of the EA, since each sub-plan would have a different sub-goal, such a scheme may help alleviate the problems identified in this research of a system generating a solution to reach a sub-goal rather than a desired goal.

7.4 The development of military tactics: A research proposal

It is the author's hope that the experience gleaned from performing this research can be applied to the design of a system that is capable of developing the sorts of complex tactics required by the military. The following is a proposal which the author feels has a realistic chance of successfully developing tactics.

The majority of simulation models used by the military, whether they be concerned with a low level task such as controlling a guided weapon, or a high level task such as commanding a fleet, represent their tactics in a pseudo-code form. This pseudo-code typically takes the form of a simple programming language and incorporates many of the structures that are found in such languages including procedures, conditional statements and loops. The command statements for such languages also tend to be represented in a high level form and, for obvious reasons, are problem specific (eg. *new bearing*, *speed*, etc, for a torpedo).

Within the simulation models, this pseudo-code is represented in an intermediate string

form wherein each command has a particular character code associated with it. For example, the high level code

```
:
:
begin phase 3
        set direction to 45 degrees
        wait 5 minutes
        :
```

could be represented by the string

.... * 3 D 45 W 5 ...

where *, D and W represent the start of a phase and the 'set direction' and wait' commands respectively. Therefore, an entire tactical plan can be represented by a single, albeit long, character string. Clearly, optimizing such strings would be an ideal application for an EA or GP but there a number of difficulties which arise from this approach. These include:

(a) generating an initial random set of valid tactics is extremely difficult given the complex structures (that is, loops, etc) that are involved;

(b) the length of time taken by the evaluation process may significantly reduce the number of solutions which can be investigated;

(c) the length of the strings involved and the high cardinality of the alphabet (due to each command corresponding to a unique identifier) mean that the solution space is extremely large;

(d) generating valid offspring from mating two solutions is likely to be a difficult process.

With regard to the first of these problems, that of generating an initial set of feasible tactics, in the author's opinion, the most appropriate solution is to employ the tactics which are currently

used in the models to serve as the initial population. Although this may significantly bias the search of the EA towards a particular area of the solution space, it is felt that this is the only way that the system can attain the necessary 'grist for the mill'. Moreover, by starting the search in a favourable area, it would be hoped that some of the not insignificant difficulties that are likely to be caused by the third problem, that of extremely large search spaces, would be somewhat reduced.

The second problem, that of the length of time taken to evaluate a particular tactic, can be resolved by using one of a number of very fast simulation models that are currently available which, despite not giving detailed results for a particular engagement, give sufficient information for the approximate evaluations necessary to employ an EA.

However, in the authors opinion, it will be success in resolving the fourth difficulty, that of designing appropriate mating schemes for the pseudo-code / intermediate representation, that will decide the excellence of the approach. A small number of basic mating schemes can be imagined (for example, crossing over at phase boundaries) but more complex ones which involve crossover taking place in the middle of phases would also need to be investigated.

7.5 A final word

The SAGA system, has its place in the field of machine learning as it helps highlight a number of the difficulties faced by researchers investigating such systems. Primary amongst these problems is that of credit assignment where the absence of feasible algorithms means that systems adopting the Michigan approach to GBML are unlikely to be successful. As a result of this, it is envisaged by the author that in the long term, systems adopting the Pitt approach to GBML will become the standard and indeed, this is already being shown to be the case with the proliferation of systems employing Genetic Programming as their learning paradigm.

The results produced with the SAGA system also demonstrate that it is not always beneficial to increase the complexity of the system. For example, it was found that by adding

more and more operators and mechanisms in an attempt to improve system performance, it became increasingly more difficult to analyse system performance and determine where the system was performing poorly. Notwithstanding, a number of the SAGA system's novel features were shown to be beneficial to system performance, especially the repair mechanism which, it is suggested, is a viable approach to alleviating the problems of forgetfulness that are typically encountered by GBML systems.

8. Bibliography

Albrecht, R. F, Reeves, C. R. and Steele, N. C. (eds) (1993) *Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag, 1993.

Antonisse, H. J. and Keller, K. S. (1987) 'Genetic Operators for High-Level Knowledge Representations' in [Grefenstette, 87a], 69-76.

Antonisse, J. (1989) 'A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint' in [Schaffer, 89], 86-91.

Athanasopoulou, M. (1992) 'The Underwater Warfare Hide and Seek Problem' MSc Dissertation, London School of Economics.

Axelrod, R. (1987) 'The Evolution of Strategies in the Iterated Prisoner's Dilemma' in [Davis, 87], 32-41.

Bäck, T. (1992) 'Self-Adaptation in Genetic Algorithms' Proceedings of the First European Conference on Artificial Life', Paris, MIT Press, Cambridge, MA, 263-271.

Bäck, T, Hoffmeister, F, Kursawe, F, Rudolph, G. and Schwefel, H-P. (1990) 'Four Contributions to the Development of Evolution Strategies and Genetic Algorithms', Technical Report, Department of Computer Science, University of Dortmund, December, 1990.

Bäck, T, Hoffmeister, F. and Schwefel, H.-P. (1991) 'A Survey of Evolution Strategies' in [Belew and Booker, 91], 2-9.

Bagchi, S, Uckun, S, Miyabe, Y. and Kawamura, K. (1991) 'Exploring Problem Specific Operators for Job Shop Scheduling' in [Belew and Booker, 91], 10-19.

Baker, J. E. (1985) 'Adaptive Selection Methods for Genetic Algorithms' in [Grefenstette, 85], 101-111.

Baker, J. E. (1987) 'Reducing Bias and Inefficiency in the Selection Algorithm' in [Grefenstette, 87a], 14-21.

Ball, N. (1991) 'Cognitive Maps in Learning Classifier Systems' Unpublished PhD Dissertation, University of Reading.

Ball, N. (1993) 'Towards the Development of Cognitive Maps in Classifier Systems' in [Albrecht, et al, 93], 712-717.

Beauchamp, K. G. (1975)*Walsh Functions and Their Applications,* Academic Press, London.

Belew, R. K. and Booker, L. B. (eds) (1991) *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.

Belew, R. K. and Forrest, S. (1988) 'Learning and Programming in Classifier Systems' in *Machine Learning*, 3(2/3), 193-224.

Belew, R. K. and Gherrity, M. (1989) 'Back Propagation for the Classifier System' in [Schaffer, 89], 275-281.

Belew, R. K, McInerney, J. and Schraudolph, N. N. (1991) 'Evolving Networks: Using the Genetic Algorithm with Connectionist Learning' *Artificial Life II, SFI Studies in the Sciences of Complexity*, Vol X, 511-547, Addison Wesley.

Bellman, R. (1957) *Dynamic Programming* Princeton University Press, Princeton, NJ.

Bench-Capon, T. J. M. (1990) *Knowledge Representation: An Approach to Artificial Intelligence*, Academic Press, London.

Bergman, A. (1992) 'An Evolutionary Approach to Designing Neural Nets' *Sigbio newsletter*, 12(2).

Bersini, H. and Varela, F. J. (1990). 'Hints for Adaptive Problem Solving Gleaned from Immune Networks' in Proceedings of the First Workshop on Parallel Problem Solving from Nature', Springer-Verlag, Berlin, 343-354.

Bersini, H. and Varela, F. J. (1991) 'The Immune Recruitment Mechanism: A Selective Evolutionary Strategy' in [Belew and Booker, 91], 520-526.

Bickel, A. S. and Bickel, R. W. (1987) 'Tree Structured Rules in Genetic Algorithms' in

[Grefenstette, 87a], 77-81.

Boffey, T. B. (1984) *Graph Theory in Operations Research* Macmillan Press, London.

Bolc, L. (1980) *Natural Language Question Answering Systems*, Hanser, Munchen.

Bomze, I. M. and Potscher, B. M. (1989) 'Game Theoretical Foundations of Evolutionary Stability' in *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag, Berlin.

Booker, L. (1982) 'Intelligent Behaviour as an Adaptation to the Task Environment' PhD Dissertation, Ann Arbor, University of Michigan.

Booker, L. B. (1985) 'Improving the Performance of Genetic Algorithms in Classifier Systems' in [Grefenstette, 85], 80-92.

Booker, L. B. (1987) 'Improving Search in Genetic Algorithms' in [Davis, 87], 61-73.

Booker, L. B. (1988) 'Classifier Systems that Learn Internal World Models' in *Machine Learning*, 3(2/3), Kluwer Academic Press, Netherlands, 161-192.

Booker, L. B. (1989) 'Triggered Rule Discovery in Classifier Systems' in [Schaffer, 89], 265-274.

Booker, L. B, Goldberg, D. E. and Holland, J. H. (1989) 'Classifier Systems and Genetic Algorithms' *Artificial Intelligence*, 40(1-3), Elsevier Science Publishers, BV (North-Holland), 235-282.

Bose, R. C. and Nelson, R. J. (1962) 'A Sorting Problem' *JACM*, 9, 282-296.

Bruderer, E. and Shevoroshkin, A. (1993) 'Hierarchical Search and the Discovery of Strategies' Unpublished Report, March 23.

Byrne, C. D, Miles, J. A. H. and Lakin, W. L. (1989) 'Towards Knowledge-Based Naval Command Systems' *Third International Conference on Command, Control, Communications and Management Information Systems*, Bournemouth, UK,33-42.

Carbonell, J. G. (1989) 'Introduction: Paradigms for Machine Learning' *Artificial*

Intelligence, 40(1-3), Elsevier Science Publishers, BV (North-Holland), 1-9.

Clancey, W. J. and Soloway, E. (1990) 'Artificial Intelligence and Learning Environments: Preface' *Artificial Intelligence*, 42(1), Elsevier Science Publishers, BV (North-Holland), 1-6.

Cleveland, G. A. and Smith, S. F. (1987) 'Using Genetic Algorithms to Schedule Flow Shop Releases' in [Grefenstette, 87a], 160-169.

Cobb, H. G. and Grefenstette, J. J. (1991) 'Learning the Persistence of Actions in Reactive Control Rules' in *Proceedings of the 8th International Machine Learning Workshop*, San Mateo, CA, 293-297.

Cohoon, J. P, Hedge, S. U, Martin, W. N. and Richards, D. (1987) 'Punctuated Equilibria: A Parallel Genetic Algorithm [Grefenstette, 87a], 148-154.

Compiani, M, Montanari, D, Serra, R. and Simonini, P. (1989) 'Asymptotic Dynamics of Classifier Systems' in [Schaffer, 89], 298-303.

Coyne, J. A. (1992) 'Genetic Algorithms and the Lamarckian Operator', Unpublished MSc Dissertation, University of Liverpool.

Crawford, V. P. (1989) 'Learning and Mixed-Strategy Equilibria in Evolutionary Games' *Journal of Theoretical Biology*, 140, Academic Press Inc (London) Ltd, 537-550.

Darwin, C. (1859)*The Origin of the Species by Means of Natural Selection* Murray, London.

Davidor, Y. (1991) *Genetic Algorithms and Robotics*, World Scientific, London.

Davis, L. (ed) (1987) *Genetic Algorithms and Simulated Annealing* Pitman, London.

Davis, L. (1988) 'Mapping Classifier Systems into Neural Networks' in *Proceedings of the 1988 Neural Information Processing Systems Conference*, Denver, Colorado.

Davis, L. (1989a) 'Adapting Operator Probabilities in Genetic Algorithms' in [Schaffer, 89], 61-69.

Davis, L. (1989b) 'Mapping Neural Networks into Classifier Systems' in [Schaffer, 89], 375-

378.

Davis, L. (1991) *Handbook of Genetic Algorithms*, Von Nostrand Reinhold, New York.

Dawkins, R. (1976)*The Selfish Gene*, Oxford University Press, New York.

Dean, T. and Boddy, M. (1988) 'An Analysis of Time-Dependent Planning' *Proceedings of the Seventh National Conference on AI (AAAI-88)*, St Paul, MN, Morgan-Kaufmann, 49-54.

De Garis, H. (1990) 'Genetic Programming: Modular Evolution for Darwin Machines' *Proceedings of an International Joint Conference on Neural Networks*, Washington DC.

De Jong, K. A. (1975) An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, PhD Dissertation, University of Michigan, Ann Arbor.

De Jong, K. A. (1985) 'Genetic Algorithms: A 10 Year Perspective' in [Grefenstette, 85], 169-177.

De Jong, K. A. (1988) 'Learning with Genetic Algorithms: An Overview' in *Machine Learning*, 3(2/3), Kluwer Academic Press, Netherlands, 121-138.

De Jong, K. A. and Spears, W. M. (1989) 'Using Genetic Algorithms to Solve NP-Complete Problems' in [Schaffer, 89], 124-132.

De la Maza, M. 'A SEAGUL visits the race track' in [Schaffer, 89], 208-212.

Deb, K. and Goldberg, D. E. (1989) 'An Investigation of Niche and Species Formation in Genetic Function Optimization' in [Schaffer, 89], 42-50.

Dietterich, T. G and Buchanan, B. G. (1981) 'The Role of the Critic in Learning Systems' Technical Report No. HPP-81-19, Department of Computer Science, Stanford University, CA.

Dike, B. A. and Smith, R. E. (1993) 'Application of Genetic Algorithms to Air Combat Maneuvering' *TCGA Report No. 93002*, Department of Engineering Science ad Mechanics, University of Alabama.

Dobzhansky, T, Ayala, F. J, Stebbins, G. L. and Valentine, J. W. (1977) *Evolution* W H Freeman and Company, San Francisco, CA.

Erickson, M. D. and Zytkow, J. M. (1988) 'Utilizing Experience for Improving the Tactical Manager' in *Proceedings of the Fifth International Conference on Machine Learning*, University of Michigan, Ann Arbor, MI, 444-450.

Eshelman, L. J, Caruana, R. A. and Schaffer, J. D. (1989) 'Biases in the Crossover Landscape' in [Schaffer, 89], 10-19.

Fairley, A. and Yates, D. F. (1992) 'A Comparison of Methods of Choosing the Crossover Point in the Genetic Crossover Operation' Internal Working Paper, Dept. of Computer Science, University of Liverpool.

Fairley, A. and Yates, D. F. (1993) 'Improving Simple Classifier Systems to Alleviate the Problems of Duplication, Subsumption and Equivalence of Rules' in [Albrecht, et al, 93], 408-418.

Fairley, A. and Yates, D. F. (1994a) 'Applications of Genetic Algorithms for Developing Military Tactics: A Review', Internal Report No. DRA/OS/N/TR94100/1.0, Naval Studies Department, Defence Research Agency.

Fairley, A. and Yates, D. F. (1994b) 'Inductive Operators and Rule Repair in a Hybrid Genetic Learning System: Some Initial Results' in [Fogarty, 94], 166-179.

Farmer, J. D, Packard, N. H. and Perelson, A. S. (1986) 'The Immune System, Adaptation, and Machine Learning' *Physica 22D*, North-Holland, Amsterdam, 187-204.

Feigenbaum, E. A. (1963) 'The Simulation of Verbal Learning Behaviour' in *Computers and Thought*, E. A. Feigenbaum and J. Feldman (eds), McGraw-Hill, NY.

Fitzpatrick, J. M. and Grefenstette, J. J. (1988) 'Genetic Algorithms in Noisy Environments' in *Machine Learning*, 3(2/3), Kluwer Academic Press, Netherlands, 101-120.

Fogarty, T. C. (1989a) 'The Machine Learning of Rules for Cobustion Control in Multiple Burner Installations' in *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications*, 215-221.

Fogarty, T. C. (1989b) 'Varying the Probability of Mutation in the Genetic Algorithm' in [Schaffer, 89], 104-109.

Fogarty, T. C. (1992) 'Evolving Controllers' *IEE Digest*, IEE London, 106.

Fogarty, T. C. (1993) 'Reproduction, Ranking, Replacement and Noisy Environments: Experimental Results' in [Forrest, 93], 634.

Fogarty, T. C. (ed) (1994) *Proceedings of a Workshop on Evolutionary Computing*, Lecture Notes in Computer Science Series, 865, Springer-Verlag, Berlin.

Fogel, I. J, Owens, A. J. and Walsh, M. J. (1966) *Artificial Intelligence through simulated evolution*, New York, John Wiley.

Forrest, S. (1985) 'A Study of Parallelism in the Classifier System and its Application to Classification in KL-ONE Semantic Networks' PhD Dissertation, University of Michigan, Ann Arbor.

Forrest, S. (1991) *Parallelism and Programming in Classifier Systems*, Pitman, London.

Forrest, S. (1993) *Proceedings of the 5th International Conference on Genetic Algorithms*, Morgan Kaufmann, CA.

Forrest, S. and Perelson, A. S. (1992) 'Computation and the Immune System' *Sigbio newsletter*, 12(2), 52-56.

Freisleben, B. and Härtfelder, M. (1993) 'Optimization of Genetic Algorithms by Genetic Algorithms' in [Albrecht, et al, 93], 392-399.

Frey, P. W. and Slate, D. J. (1991) 'Letter Recognition Using Holland-Style Adaptive Classifiers' in *Machine Learning*, 6, Kluwer Academic Press, Netherlands, 161-182.

Fujiki, C. and Dickinson, J. (1987) 'Using the Genetic Algorithm to generate Lisp Source Code to Solve the Prisoner's Dilemma' in [Grefenstette, 87a], 236-240.

Gervasio, M. and DeJong, G. (1989) 'Explanation-based Learning of Reactive Operators'

Proceedings of the Sixth International Workshop on Machine Learning, Morgan Kaufman, Ithaca, NY.

Glover, F. (1990) 'Tabu Search: A Tutorial' *Interface*, 20, 74-94.

Goldberg, D. E. (1983) 'Computer-aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning' PhD Dissertation, University of Michigan, Ann Arbor.

Goldberg, D. E. (1987) 'Simple Genetic Algorithms and the Minimal Deceptive Problem' in [Davis, 87], 74-88.

Goldberg, D. E. (1989a) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Goldberg, D. E. (1989b) 'Sizing Populations for Serial and Parallel Genetic Algorithms' in [Schaffer, 89], 70-79.

Goldberg, D. E, Deb, K, and Clark, J. H. (1991) 'Genetic Algorithms, Noise, and the Sizing of Populations' IlliGAL Technical Report No. 91010, University of Illinois at Urbana-Champaign.

Goldberg, D. E. and Holland, J. H. (1988) 'Genetic Algorithms and Machine Learning' in *Machine Learning*, 3(2/3), Kluwer Academic Press, Netherlands, 95-100.

Goldberg, D. E. and Rudnick, M. (1991) 'Genetic Algorithms and the Variance of Fitness', Technical Report 91001, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Goldberg, D. E. and Smith, R. E. (1987) 'Nonstationary Function Optimization using Genetic Algorithms with Dominance and Diploidy' in [Grefenstette, 87a], 59-68.

Gorczynski, R. M. and Steele, E. J. (1981) 'Simultaneous Yet Independent Inheritance of Somatically Acquired Tolerance to Two Distinct H-2 Antigenic Haplotype Determinants in Mice' *Nature*, 289.

Gordon, D. F. (1991) 'An Enhancer for Reactive Plans' in *Proceedings of the 8th International Machine Learning Workshop*, San Mateo, CA, 505-508.

Gordon, D. F. and Grefenstette, J. J. (1990) 'Explanations of Empirically Derived Plans' in *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, Morgan Kaufmann198-203.

Gould, J. L. (1982) 'Ethology: The Mechanisms and Evolution of Behaviour', W. W. Norton and Company, New York.

Grefenstette, J. J. (ed) (1985) *Proceedings of the First International Conference on Genetic Algorithms* Lawrence Erlbaum Associates, Hillsdale, N. J.

Grefenstette, J. J. (1986) 'Optimization of Control Parameters for Genetic Algorithms' *IEEE Transactions of Systems, Man, and Cybernetics*, Vol 16, No 1, 122-128.

Grefenstette, J. J. (ed) (1987a) *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications* Lawrence Erlbaum Associates, Hillsdale, N.J.

Grefenstette, J. J. (1987b) 'Incorporating Problem Specific Knowledge into Genetic Algorithms' in [Davis, 87], 42-60.

Grefenstette, J. J. (1987c) 'Multilevel Credit Assignment in a Genetic Learning System' in [Grefenstette, 87a], 202-209.

Grefenstette, J. J. (1988) 'Credit Assignment in Rule Discovery Systems based on Genetic Algorithms' in *Machine Learning*, 3(2/3), Kluwer Academic Press, Netherlands, 225-246.

Grefenstette, J. J. (1989) 'A System for Learning Control Plans with Genetic Algorithms' in [Schaffer, 89], 183-190.

Grefenstette, J. J. (1991a) 'Strategy Acquisition with Genetic Algorithms' in [Davis, 91], 186-201.

Grefenstette, J. J. (1991b) 'Lamarckian Learning in Multi-agent Environments' in [Belew and Booker, 91], 303-310.

Grefenstette, J. J. (1993) 'The Evolution of Strategies for Multiagent Environments' *Adaptive Behaviour*, Vol. 1, No. 1, 65-90.

Grefenstette, J. J. and Baker, J. E. (1989) 'How Genetic Algorithms work: A Critical look at Implicit Parallelism' in [Schaffer, 89], 20-27.

Grefenstette, J. J. and Ramsey, C. L. (1992) 'An Approach to Anytime Learning' in *Machine Learning: Proceedings of the 9th International Conference*, Morgan Kaufmann, San Mateo, CA, 189-195.

Grefenstette, J. J, Ramsey, C. L. and Schultz, A. C. (1990) 'Learning Sequential Decision Rules Using Simulation Models and Competition' in *Machine Learning (Special Issue on Genetic Algorithms*, Vol 5, Kluwer Academic Press, Netherlands.

Hancock, P. (1994) 'An Empirical Comparison of Selection Methods in Evolutionary Algorithms' in [Fogarty, 94], 80-94.

Harley, C. B. (1981) 'Learning the Evolutionary Stable Strategy' *Journal of Theoretical Biology*, 89, Academic Press Inc (London) Ltd, 611-633.

Harley, C. B. (1983) 'Letter to the editor: When do Animals Learn the Evolutionary Stable Strategy?' *Journal of Theoretical Biology*, Academic Press Inc (London) Ltd, 179-181.

Harley, C. B. (1987) 'Letter to the editor: Learning Rules, Optimal Behaviour, and Evolutionary Stability' *Journal of Theoretical Biology*, Academic Press Inc (London) Ltd, 377-379.

Harp, S. A. Samad, T. and Guha, A. (1989) 'Towards the Genetic Synthesis of Neural Networks' in [Schaffer, 89], 360-369.

Harris, C. J. (ed) (1988) *Artificial Intelligence in Command and Control*, Peter Pergrinus, London.

Harvey, I. (1992) 'Evolutionary Robotics and SAGA: The Case for Hill Crawling and Tournament Selection', Internal Report of School of Cognitive and Computing Sciences, University of Sussex.

Hayes-Roth, B. (1985) 'A Blackboard Architecture for Control' *Artificial Intelligence*, 26, Elsevier Science Publishers, BV (North-Holland), 251-321.

Hillis, W. D. (1991) 'Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure' *Artificial Life II*, SFI Studies in the Sciences of Complexity, Vol X, Addison-Wesley, 313-324.

Hines, W. G. S. (1982) 'Mutations, Perturbations and Evolutionary Stable Strategies' *Journal of Applied Probability*, 19, Applied Probability Trust, Israel.

Hinton, G. E. and Nowlan, S. J. (1987) 'How Learning can Guide Evolution' *Complex Systems*,1, Complex Systems Publications Inc.

Hinton, G. E. and Sejnowski, T. J. (1986) 'Learning and Relearning in Boltmann Machines' in *Parallel Distributed Processing*, Vol 1 & 2, MIT Press, Cambridge, MA.

Hirst, R. A. (1989) 'Allegiance Assessment Using Explanation-Based Reasoning' *Third International Conference on Command, Control, Communications and Management Information Systems*, Bournemouth, UK, 56-63.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Holland, J. H. (1985) 'Properties of the Bucket Brigade Algorithm' in [Grefenstette, 85], 1-7.

Holland, J. H. (1986a) 'A Mathematical Framework for Studying Learning in Classifier Systems' *Physica 22D*, North-Holland, Amsterdam, 307-317.

Holland, J. H. (1986b) 'Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-based Systems' in R. S. Michalski, J. G. Carbonell, and Mitchell, T. M. (eds) *Machine Learning: An Artificial Intelligence Approach*, Vol 2, Los Altos, CA: Morgan Kaufmann.

Holland, J. H. (1987) 'Genetic Algorithms and Classifier Systems: Foundations and Future Directions' in [Grefenstette, 87a], 82-89.

Holland, J. H. (1990) 'Concerning the Emergence of Tag-Mediated Lookahead in Classifier Systems' *Physica D 42*, North-Holland, Amsterdam, 188-201.

Holland, J. H, Holyoak, K. J, Nisbett, R. E. and Thagard, P R. (1986) 'Induction: Processes of Inference, Learning and Discovery' Cambridge: MIT Press.

Holland, J. H. and Reitman, J. (1978) 'Cognitive Systems Based on Adaptive Algorithms' in Waterman, D. and Hayes-Roth, F. (eds) *Pattern Directed Inference Systems*, Academic Press, New York, 313-329.

Hopfield, J. J. (1982) 'Neural Networks and Physical Systems with Emergent Collective Computational Abilities' in *Proceedings of the National Academy of Sciences*, 79, 2554-2558.

Houston, A. (1983) 'Letter to the editor: Comments on "Learning the Evolutionary Stable Strategy" *Journal of Theoretical Biology*, Academic Press Inc (London) Ltd, 175-178.

Husbands, P. and Mill, F. (1991) 'Simulated Co-evolution as the Mechanism for Emergent Planning and Scheduling' in [Belew and Booker, 91], 264-270.

Ikegami, T. and Kaneko, K. (1991) 'Computer Symbiosis - Emergence of Symbiotic Behaviour Throough Evolution' pp 235 - 243 in *Emergent Computation: Self-organizing Collective, and Co-operative Phenomena in Natural and Artificial Computing Networks*, Special Issue of Physica D, MIT Press, Cambridge, MA.

Janikow, C. (1991) 'Inductive Learning of Decision Rules in Attribute-Based Examples: A Knowledge-Intensive Genetic Algorithm Approach' Unpublished PhD Dissertation, University of North Carolina at Chapel Hill.

Janikow, C. and Michalewicz, Z. (1991) 'An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms' in [Belew and Booker, 91], 31-36.

Jog, P. Suh, J. Y. Van Gucht, D. (1989) 'The Effects on Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Travelling Salesman Problem' in [Schaffer, 89], 110-115.

Klahr, D, Langley, P. and Neches, R. (eds) (1987) *Production System Models of Learning and Development*, Cambridge, MA, MIT Press.

Klincewicz, J. G. and Luss, H. (1986) 'A Lagrangian Relaxation Heuristic for Capacitated Facility Location with Single-source Constraints' *Journal of the Operational Research Society*,

37(5), 495-500.

Kodratoff, Y. and Michalski, R. (1990) *Machine Learning: An Artificial Intelligence Approach*, Vol 3, Morgan Kaufmann, Los Altos, CA.

Kohonen, T. (1988) *Self-Organization and Associative Memory*, 2nd Edition, New York, Springer-Verlag.

Koopman, B. O. (1979) *Search and Screening*, Pergamon Press, New York.

Koza, J. R. (1991) *Genetic Programming,* MIT Press, Cambridge, MA.

Laird, J. E, Rosenbloom, P. S. and Newell, A. (1986) 'Chunking in SOAR: The Anatomy of a General Learning Mechanism' *Machine Learning*, 1, Kluwer Academic Press, Netherlands.

Lenat, D. B. (1983) 'The Role of Heuristics in Learning by Discovery: Three Case Studies' in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonall and T. M. Mitchell (eds).

Liepins, G. E, Hilliard, M. R, Palmer, M. and Rangarajan, G. (1989) 'Alternatives for Classifier System Credit Assignment' in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 756-761.

Liepins, G. E, Hilliard, M. R, Palmer, M. and Rangarajan, G. (1991) 'Credit Assignment and Discovery in Classifier Systems' *International Journal of Intelligent Systems*, Vol 6, John Wiley and Sons, Inc, 55-69.

Manderick, B. (1991) 'Selectionist Systems as Cognitive Systems' in *Proceedings of the First European Conference on Artificial Life*, MIT Press, 441-447.

Maynard-Smith, J. (1982) *Evolution and the Theory of Games*, Cambridge University Press, Cambridge, UK.

Maynard-Smith, J. (1986) 'Evolutionary Game Theory' *Physica D22*, North-Holland, Amsterdam, 43-49.

Maynard-Smith, J. (1988) 'Can a Mixed Strategy be Stable in a Finite Population' *Journal of*

Theoretical Biology, Academic Press Inc (london) Ltd, 247-251.

Maynard-Smith, J. (1989) *Evolutionary Genetics* Oxford University Press, Oxford, UK.

Mercer, R. E. and Sampson, J. R. (1987) 'Adaptive Search Using a Reproductive Meta-Plan' in *Kybernetics*, 7.

Michalewicz, Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin.

Michalewicz, Z. and Janikow, C. (1991) 'Handling Constraints in Genetic Algorithms' in [Belew and Booker, 91], 151-157.

Michalski, R. S. (1983) 'A Theory and Methodology of Inductive Learning', *Artificial Intelligence*, 20, Elsevier Science Publishers, BV (North-Holland).

Miles, J. A. H. (1988) 'Artificial Intelligence and Command and Control' in [Harris, 88], 1-17.

Miles, J. A. H. (1989) 'Architectures for C^2 Knowledge-Based Systems' *Third International Conference on Command, Control, Communications and Management Information Systems*, Bournemouth, UK, 64-69.

Miller, G. F, Todd, P. M. and Hegde, S. U. (1989) 'Designing Neural Networks using Genetic Algorithms in [Schaffer, 89], 379-384.

Newell, A. and Simon, H. A. (1972) *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ.

Nix, A. E. and Vose, M. D. (1992) 'Modelling Genetic Algorithms with Markov Chains' *Annals of Mathematics and Artificial Intelligence*, 5, J C Baltzer AG, Scientific Publishing Company, Basel, Switzerland, 79-88.

Oliver, I. M, Smith, D. J. and Holland, J. R. C. (1987) 'A Study of Permutation Crossover Operators on the Travelling Salesman Problem' in [Grefenstette, 87a], 224-230.

Paton, R. C. (1992a) 'Introduction to the Special Edition on Biologically Motivated

Computing' *Sigbio Newsletter*, 12(2).

Paton, R. C. (1992b) 'Some Perspectives on Adaptation and Environment' Internal Working Paper, Department of Computer Science, University of Liverpool.

Perelson, A. S. (1989) 'Immune Network Theory' *Immunol Rev*, 100.

Quinlan, J. R. (1988) 'An Empirical Comparison of Genetic and Decision-tree Classifiers' in *Proceedings of the Fifth International Conference on Machine Learning*, San Mateo, CA, Morgan Kaufmann.

Rackman, P. (ed) (1990) *Jane's C3I Systems, 1990-91.*

Ramsey, C. L. and Grefenstette, J. J. (1993) 'Case-Based Initialization of Genetic Algorithms' in [Forrest, 93], 84-91.

Rawlins, G. (1991) 'Foundations of Genetic Algorithms' *First Workshop on the Foundations of Genetic Algorithms and Classifier Systems*, Morgan Kaufmann Publishers, Los Altos, CA.

Reeves, C. R. (1991) Private Communication.

Reeves, C. R. (1992) 'A Genetic Algorithm for Flowshop Sequencing' *Computers and Operations Research.*

Reeves, C. R. (ed) (1993) *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford.

Rennie, J. (1992) 'Living Together: Trends in parasitology' in January edition of *Scientific American*, 105-113.

Richardson, J. T, Palmer, M. R, Liepins, G. E. and Hilliard, M. (1989) 'Some Guidelines for Genetic Algorithms with Penalty Functions' in [Schaffer, 89], 191-197.

Riolo, R. L. (1987a) 'Bucket Brigade Performance: I. Long Sequences of Classifiers' in [Grefenstette, 87a], 184-195.

Riolo, R. L. (1987b) 'Bucket Brigade Performance: II. Default Hierarchies' in [Grefenstette,

87a], 196-201.

Riolo, R. L. (1989a) 'The Emergence of Coupled Sequences of Classifiers' in [Schaffer, 89], 256-264.

Riolo, R. L. (1989b) 'The Emergence of Default Hierarchies in Learning Classifier Systems' in [Schaffer, 89], 322-327.

Roberts, G. (1989) 'A Rational Reconstruction of Wilson's ANIMAT and Holland's CS-1' in [Schaffer, 89], 317-321.

Roberts, G. (1993) 'Dynamic Planning for Classifier Systems' in [Forrest, 93], 231-237.

Robertson, G. G. (1987) 'Parallel Implementation of Genetic Algorithms in a Classifier System' in [Grefenstette, 87a], 140-147.

Robertson, G. G. and Riolo, R. L. (1988) 'A Tale of Two Classifier Systems' in *Machine Learning*, 3(2/3), Kluwer Academic Press, Netherlands, 139-160.

Rosenbloom, P. and Newell, A. (1987) 'Learning by Chunking: A Production System Model of Practice' in D. Klahr, P. Langley and R. Neches (eds) *Production System Models of Learning and Development*, Cambridge, MA, MIT Press.

Schaffer, J. D. (1989) *Proceedings of the Third International Conference on Genetic Algorithms* , Morgan Kaufmann, San Mateo, CA.

Schaffer, J. D, Caruana, R. A, Eshelman, L. J. and Das, R. (1989) 'A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization' in [Schaffer, 89], 51-60.

Schaffer, J. D. and Morishima, A. (1987) 'An Adaptive Crossover Distribution Mechanism for Genetic Algorithms' in [Grefenstette, 87a], 36-40.

Schultz, A. C. (1991) 'Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation' in *Proceedings of the 7th International Symposium on Unmanned, Untethered Submersible Technology*, Durham, NH.

Schultz, A. C. and Grefenstette, J. J. (1990) 'Improving Tactical Plans with Genetic Algorithms' in *Proceedings of IEEE Conference on Tools for AI 90*, Washington DC, 328-334.

Schuurmans, D. and Schaeffer, J. (1989) 'Representational Difficulties with Classifier Systems' in [Schaffer, 89], 328-333.

Schwefel, H-P. (1975) 'Evolutionstrategie und numerische Optimierung' Unpublished Dissertation, Technische Universität Berlin, Berlin.

Schwefel, H.-P. and Männer, R. (eds) (1990) *Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN)*, Dortmund, Germany.

Shaefer, C. G. (1987) 'The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique' in [Grefenstette, 87a], 50-56.

Shaw, M. J. (1987) 'Applying Inductive Learning to Enhance Knowledge-Based Expert Systems' *Decision Support Systems*, 3.

Shu, L. and Schaeffer, J. (1989) 'VCS: Variable Classifier Systems' in [Schaffer, 89], 334-339.

Shu, L. and Schaeffer, J. (1991) 'HCS: Adding Hierarchies to Classifier Systems' in [Belew and Booker, 91], 339-345.

Sirag, D. J. and Weisser, P. T. (1987) 'Toward a Unified Thermodynamic Genetic Operator' in [Grefenstette, 87a], 116-122.

Smith, R. E. (1993) 'Adaptively Resizing Populations: An Algorithm and Analysis' in [Forrest, 93], 653.

Smith, R. E, Forrest, S. and Perelson, A. S. (1992) 'Population Diversity in an Immune System Model: Implications for Genetic Search' in [Whitley, 92], 153-166.

Smith, R. E. and Valenzuela-Rendon, M. (1989) ' A Study of Rule Set Development in a Learning Classifier System' in [Schaffer, 89], 340-346.

Smith, S. F. (1980) 'A Learning System Based on Genetic Adaptive Algorithms' Unpublished PhD Dissertation, University of Pittsburgh.

Spears, W. M. (1992) 'Crossover or Mutation?' in [Whitley, 92], 221-238.

Spiessens, P. (1988) 'Genetic Algorithms: Introduction, Applications and Extensions' *AI Memo*, Virje Universiteit Brussel.

Sumida, B. H. (1992) 'Genetics for Genetic Algorithms' *Sigbio Newsletter*, 12(2).

Sumida, B. H. and Hamilton, W. D. (1994) 'Both Wrightian and "Parasite" Peak Shifts Enhance Genetic Algorithm Performance' in R. Paton (ed) *Computing with Biological Metaphors*, Chapman & Hall, London. Chapter 16.

Sumida, B. H, Houston, A. I, McNamara, J. M. and Hamilton, W. D. (1990) 'Genetic Algorithms and Evolution' *Journal of Theoretical Biology*, 147, Academic Press Inc (London) Ltd, 59-84.

Sutton, R. S. (1988) 'Learning to Predict by the Methods of Temporal Differences' *Machine Learning*, 3, Kluwer Academic Press, Netherlands, 9-44.

Sutton, R. S. (1990) 'Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming' in *Proceedings of the Seventh International Conference on Machine Learning*, 216-224.

Syswerda, G. (1989) 'Uniform Crossover in Genetic Algorithms' in [Schaffer, 89], 2-9.

Valenzuela-Rendon, M. (1989) 'Boolean Analysis of Classifier Sets' in [Schaffer, 89], 351-358.

Vickers, G. T. and Cannings, C. (1987) 'On the Definition of an Evolutionary Stable Strategy' *Journal of Theoretical Biology*, 129, Academic Press Inc (London) Ltd, 349-353.

Virr, L. E, Fairley, A. and Yates, D. F. (1993) 'Tactical Application of Genetic Algorithms', *Journal of Naval Science*, Vol 19, No 3.

Vose, M. (1992) 'Modelling Simple Genetic Algorithms' in [Whitley, 92].

Vose, M. and Liepins, G. (1991) 'Punctuated Equilibria in Genetic Search' *Complex Systems*, 5, Complex Systems Publications Inc.

Walker, T. C. and Miller, R. K. (1986) *Expert Systems 1986: An Assessment of Technology and Applications*, SEAI Technical, Madison.

Watkins, C. J. C. H. (1989) 'Learning from Delayed Rewards', PhD Dissertation, Cambridge University.

Westerdale, T. H. (1987) Altruism in the Bucket Brigade' in [Grefenstette, 87a], 22-26.

Westerdale, T. H. (1989) 'A Defense of the Bucket Brigade' in [Schaffer, 89], 282-290.

Westerdale, T. H. (1991) 'Redundant Classifiers and Prokaryote Genomes' in [Belew and Booker, 91], 354-361.

Whitley, D. (1987) 'Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery' in [Grefenstette, 87a], 108-115.

Whitley, D. (1989) 'The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best' in [Schaffer, 89], 116-122.

Whitley, D. (ed) (1992) *Foundations of Genetic Algorithms 2*, Morgan Kaufmann, San Mateo, CA.

Whitley, D. (1993) 'A Genetic Algorithm Tutorial' Technical Report No. CS-93-103, Department of Computer Science, Colorado State University.

Whitley, D. and Hanson, T. (1989) 'Optimizing Neural Networks Using Faster, More Accurate Genetic Search' in [Schaffer, 89], 391-396.

Whitley, D, Mathias, K. and Fitzhorn, P. (1991) 'Delta Coding: An Iterative Search Strategy for Genetic Algorithms' in [Belew and Booker, 91], 77-84.

Wilson, D. S. and Sober, E. (1989) 'Reviving the Superorganism' *Journal of Theoretical Biology*, 136, Academic Press Inc (London) Ltd, 337-356.

Wilson, S. W. (1987a) 'Classifier Systems and the Animat Problem' *Machine Learning*, 2, Kluwer Academic Press, Netherlands, 199-228.

Wilson, S. W. (1987b) 'The Genetic Algorithm and Biological Development' in [Grefenstette, 87a], 247-251.

Wilson, S. W. (1987c) 'Hierarchical Credit Allocation in a Classifier System' in *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 217-220.

Wilson, S. W. (1987d) 'Quasi-Darwinian Learning in a Classifier System' in *Proceedings of the Fourth International Workshop on Machine Learning*, 59-65.

Wilson, S. W. (1989) 'Bid Competition and Specificity Reconsidered' *Complex Systems*, 2, Complex Systems Publications Inc, 705-723.

Wilson, S. W. and Goldberg, D. E. (1989) 'A Critical Review of Classifier Systems' in [Schaffer, 89], 244-255.

Winston, P. H. (1977) *Artificial Intelligence*, Addison-Wesley, Reading, MA.

Woodcock, A. E. R. (1989) 'Toward a Theory of Combat with Embedded Command and Control' *Third International Conference on Command, Control, Communications and Management Information Systems*, Bournemouth, UK, 127-137.

Yates, D. F. and Fairley, A. (1993) 'An Investigation Into Possible Causes of, and Solutions to, Rule Strength Distortion Due to the Bucket Brigade Algorithm' in [Forrest, 93], 246-253.

Yates, D. F. and Fairley, A. (1994) 'Evolutionary Stability in Simple Classifier Systems' in [Fogarty, 94], 28-37.

Zhou, H. H. (1990) 'CSM: A Computational Model of Cumulative Learning' in *Machine Learning (Special Issue on Genetic Algorithms*, Vol 5, Kluwer Academic Publishers, Netherlands, 383-406.

Zhou, H. H. and Grefenstette, J. J. (1989) Learning by Analogy in Genetic Classifier

Systems' in [Schaffer, 89], 291-297.

9 Appendices

9.1 Appendix 1: Knapsack problems

Problem 1

Max $100x_1+220x_2+90x_3+400x_4+300x_5+400x_6+205x_7+120x_8+160x_9+580x_{10}+400x_{11}+140x_{12}+100x_{13}+1300x_{14}+650x_{15}$

such that

$8x_1+24x_2+13x_3+80x_4+70x_5+80x_6+45x_7+15x_8+28x_9+90x_{10}+130x_{11}+32x_{12}+20x_{13}+120x_{14}+40x_{15}<=550$

$8x_1+44x_2+13x_3+100x_4+100x_5+90x_6+75x_7+25x_8+28x_9+120x_{10}+130x_{11}+32x_{12}+40x_{13}+160x_{14}+40x_{15}<=700$

$3x_1+6x_2+4x_3+20x_4+20x_5+30x_6+8x_7+3x_8+12x_9+14x_{10}+40x_{11}+6x_{12}+3x_{13}+20x_{14}+5x_{15}<=130$

$5x_1+9x_2+6x_3+40x_4+30x_5+40x_6+16x_7+5x_8+18x_9+24x_{10}+60x_{11}+16x_{12}+11x_{13}+30x_{14}+25x_{15}<=240$

$5x_1+11x_2+7x_3+50x_4+40x_5+40x_6+19x_7+7x_8+18x_9+29x_{10}+70x_{11}+21x_{12}+17x_{13}+30x_{14}+25x_{15}<=280$

$5x_1+11x_2+7x_3+55x_4+40x_5+40x_6+21x_7+9x_8+18x_9+29x_{10}+70x_{11}+21x_{12}+17x_{13}+35x_{14}+25x_{15}<=310$

$1x_3+10x_4+4x_5+10x_6+6x_8+6x_{10}+32x_{11}+3x_{12}+70x_{14}+10x_{15}<=110$

$3x_1+4x_2+5x_3+20x_4+14x_5+20x_6+6x_7+12x_8+10x_9+18x_{10}+42x_{11}+9x_{12}+12x_{13}+100x_{14}+20x_{15}<=205$

$3x_1+6x_2+9x_3+30x_4+29x_5+20x_6+12x_7+12x_8+10x_9+30x_{10}+42x_{11}+18x_{12}+18x_{13}+110x_{14}+20x_{15}<=260$

$3x_1+8x_2+9x_3+35x_4+29x_5+20x_6+16x_7+15x_8+10x_9+30x_{10}+42x_{11}+20x_{12}+18x_{13}+120x_{14}+20x_{15}<=275$

Problem 2

Max $100x_1+220x_2+90x_3+400x_4+300x_5+400x_6+205x_7+120x_8+160x_9+580x_{10}+400x_{11}+140x_{12}+100x_{13}+1300x_{14}+650x_{15}+320x_{16}+480x_{17}+80x_{18}+60x_{19}+2550x_{20}$

such that

$8x_1+24x_2+13x_3+80x_4+70x_5+80x_6+45x_7+15x_8+28x_9+90x_{10}+130x_{11}+32x_{12}+20x_{13}+120x_{14}+40x_{15}+30x_{16}+20x_{17}+6x_{18}+3x_{19}+180x_{20}<=550$

$8x_1+44x_2+13x_3+100x_4+100x_5+90x_6+75x_7+25x_8+28x_9+120x_{10}+130x_{11}+32x_{12}+40x_{13}+160x_{14}+40x_{15}+60x_{16}+55x_{17}+10x_{18}+6x_{19}+240x_{20}<=700$

$3x_1+6x_2+4x_3+20x_4+20x_5+30x_6+8x_7+3x_8+12x_9+14x_{10}+40x_{11}+6x_{12}+3x_{13}+20x_{14}+5x_{15}+5x_{17}+3x_{18}+20x_{20}<=130$

$5x_1+9x_2+6x_3+40x_4+30x_5+40x_6+16x_7+5x_8+18x_9+24x_{10}+60x_{11}+16x_{12}+11x_{13}+30x_{14}+25x_{15}+10x_{16}+13x_{17}+5x_{18}+1x_{19}+80x_{20}<=240$

$5x_1+11x_2+7x_3+50x_4+40x_5+40x_6+19x_7+7x_8+18x_9+29x_{10}+70x_{11}+21x_{12}+17x_{13}+30x_{14}+25x_{15}+15x_{16}+25x_{17}+5x_{18}+1x_{19}+100x_{20}<=280$

$5x_1+11x_2+7x_3+55x_4+40x_5+40x_6+21x_7+9x_8+18x_9+29x_{10}+70x_{11}+21x_{12}+17x_{13}+35x_{14}+25x_{15}+20x_{16}+25x_{17}+5x_{18}+2x_{19}+110x_{20}<=310$

$1x_3+10x_4+4x_5+10x_6+6x_8+6x_{10}+32x_{11}+3x_{12}+70x_{14}+10x_{15}<=110$

$3x_1+4x_2+5x_3+20x_4+14x_5+20x_6+6x_7+12x_8+10x_9+18x_{10}+42x_{11}+9x_{12}+12x_{13}+100x_{14}+20x_{15}+5x_{16}+6x_{17}+4x_{18}+1x_{19}+20x_{20}<=205$

$3x_1+6x_2+9x_3+30x_4+29x_5+20x_6+12x_7+12x_8+10x_9+30x_{10}+42x_{11}+18x_{12}+18x_{13}+110x_{14}+20x_{15}+15x_{16}+18x_{17}+7x_{18}+2x_{19}+40x_{20}<=260$

$3x_1+8x_2+9x_3+35x_4+29x_5+20x_6+16x_7+15x_8+10x_9+30x_{10}+42x_{11}+20x_{12}+18x_{13}+120x_{14}+20x_{15}+20x_{16}+22x_{17}+7x_{18}+3x_{19}+50x_{20}<=275$

Problem 3

Max $100x_1+220x_2+90x_3+400x_4+300x_5+400x_6+205x_7+120x_8+160x_9+580x_{10}+400x_{11}+140x_{12}+100x_{13}+1300x_{14}+650x_{15}+320x_{16}+480x_{17}+80x_{18}+60x_{19}+2550x_{20}+3100x_{21}+1100x_{22}+950x_{23}+450x_{24}+300x_{25}+220x_{26}+200x_{27}+520x_{28}$

such that

$8x_1+24x_2+13x_3+80x_4+70x_5+80x_6+45x_7+15x_8+28x_9+90x_{10}+130x_{11}+32x_{12}+20x_{13}+120x_{14}+40x_{15}+30x_{16}+20x_{17}+6x_{18}+3x_{19}+180x_{20}+220x_{21}+50x_{22}+30x_{23}+50x_{24}+12x_{25}+5x_{26}+8x_{27}+18x_{28}<=930$

$8x_1+44x_2+13x_3+100x_4+100x_5+90x_6+75x_7+25x_8+28x_9+120x_{10}+130x_{11}+32x_{12}+40x_{13}+160x_{14}+40x_{15}+60x_{16}+55x_{17}+10x_{18}+6x_{19}+240x_{20}+290x_{21}+80x_{22}+90x_{23}+70x_{24}+27x_{25}+17x_{26}+8x_{27}+28x_{28}<=1210$

$3x_1+6x_2+4x_3+20x_4+20x_5+30x_6+8x_7+3x_8+12x_9+14x_{10}+40x_{11}+6x_{12}+3x_{13}+20x_{14}+5x_{15}+5x_{17}+3x_{18}+20x_{20}+30x_{21}+40x_{22}+10x_{23}+5x_{25}+10x_{28}<=272$

$5x_1+9x_2+6x_3+40x_4+30x_5+40x_6+16x_7+5x_8+18x_9+24x_{10}+60x_{11}+16x_{12}+11x_{13}+30x_{14}+25x_{15}+10x_{16}+13x_{17}+5x_{18}+1x_{19}+80x_{20}+60x_{21}+50x_{22}+20x_{23}+30x_{24}+10x_{25}+5x_{26}+3x_{27}+20x_{28}<=462$

$5x_1+11x_2+7x_3+50x_4+40x_5+40x_6+19x_7+7x_8+18x_9+29x_{10}+70x_{11}+21x_{12}+17x_{13}+30x_{14}+25x_{15}+15x_{16}+25x_{17}+5x_{18}+1x_{19}+100x_{20}+70x_{21}+55x_{22}+20x_{23}+50x_{24}+15x_{25}+15x_{26}+6x_{27}+20x_{28}<=532$

$5x_1+11x_2+7x_3+55x_4+40x_5+40x_6+21x_7+9x_8+18x_9+29x_{10}+70x_{11}+21x_{12}+17x_{13}+35x_{14}+25x_{15}+20x_{16}+25x_{17}+5x_{18}+2x_{19}+110x_{20}+70x_{21}+55x_{22}+20x_{23}+50x_{24}+20x_{25}+15x_{26}+6x_{27}+20x_{28}<=572$

$1x_3+10x_4+4x_5+10x_6+6x_8+6x_{10}+32x_{11}+3x_{12}+70x_{14}+10x_{15}+30x_{21}+10x_{22}+10x_{24}+10x_{25}+5x_{26}+10x_{28}<=240$

$3x_1+4x_2+5x_3+20x_4+14x_5+20x_6+6x_7+12x_8+10x_9+18x_{10}+42x_{11}+9x_{12}+12x_{13}+100x_{14}+20x_{15}+5x_{16}+6x_{17}+4x_{18}+1x_{19}+20x_{20}+50x_{21}+30x_{22}+5x_{23}+20x_{24}+20x_{25}+10x_{26}+10x_{27}+20x_{28}<=400$

Problem 3 (Cont)

$$3x_1+6x_2+9x_3+30x_4+29x_5+20x_6+12x_7+12x_8$$
$$+10x_9+30x_{10}+42x_{11}+18x_{12}+18x_{13}+110x_{14}+20x_{15}+15x_{16}$$
$$+18x_{17}+7x_{18}+2x_{19}+40x_{20}+60x_{21}+50x_{22}+25x_{23}+25x_{24}$$
$$+25x_{25}+15x_{26}+10x_{27}+28x_{28}<=470$$
$$3x_1+8x_2+9x_3+35x_4+29x_5+20x_6+16x_7+15x_8$$
$$+10x_9+30x_{10}+42x_{11}+20x_{12}+18x_{13}+120x_{14}+20x_{15}+20x_{16}$$
$$+22x_{17}+7x_{18}+3x_{19}+50x_{20}+60x_{21}+55x_{22}+25x_{23}+30x_{24}$$
$$+25x_{25}+15x_{26}+10x_{27}+28x_{28}<=490$$

Problem 4

Max $560x_1+1125x_2+300x_3+620x_4+2100x_5+431x_6+68x_7+328x_8+47x_9+122x_{10}+322x_{11}+196x_{12}+41x_{13}+25x_{14}+425x_{15}+4260x_{16}+416x_{17}+115x_{18}+82x_{19}+22x_{20}+631x_{21}+132x_{22}+420x_{23}+86x_{24}+42x_{25}+103x_{26}+215x_{27}+81x_{28}+91x_{29}+26x_{30}+49x_{31}+420x_{32}+316x_{33}+72x_{34}+71x_{35}+49x_{36}+108x_{37}+116x_{38}+90x_{39}$

such that

$40x_1+91x_2+10x_3+30x_4+160x_5+20x_6+3x_7+12x_8+3x_9+18x_{10}+9x_{11}+25x_{12}+1x_{13}+1x_{14}+10x_{15}+280x_{16}+10x_{17}+8x_{18}+1x_{19}+1x_{20}+49x_{21}+8x_{22}+21x_{23}+6x_{24}+1x_{25}+5x_{26}+10x_{27}+8x_{28}+2x_{29}+1x_{30}+10x_{32}+42x_{33}+6x_{34}+4x_{35}+8x_{36}+10x_{38}+1x_{39}<=600$

$16x_1+92x_2+41x_3+16x_4+150x_5+23x_6+4x_7+18x_8+6x_9+12x_{11}+8x_{12}+2x_{13}+1x_{14}+200x_{16}+20x_{17}+6x_{18}+2x_{19}+1x_{20}+70x_{21}+9x_{22}+22x_{23}+4x_{24}+1x_{25}+5x_{26}+10x_{27}+6x_{28}+4x_{29}+4x_{31}+12x_{32}+8x_{33}+4x_{34}+3x_{35}+10x_{37}+6x_{39}<=500$

$38x_1+39x_2+32x_3+71x_4+80x_5+26x_6+5x_7+40x_8+8x_9+12x_{10}+30x_{11}+15x_{12}+1x_{14}+23x_{15}+100x_{16}+20x_{18}+3x_{19}+40x_{21}+6x_{22}+8x_{23}+6x_{25}+4x_{26}+22x_{27}+4x_{28}+6x_{29}+1x_{30}+5x_{31}+14x_{32}+8x_{33}+2x_{34}+8x_{35}+20x_{37}<=500$

$8x_1+71x_2+30x_3+60x_4+200x_5+18x_6+6x_7+30x_8+4x_9+8x_{10}+31x_{11}+6x_{12}+3x_{13}+18x_{15}+60x_{16}+21x_{17}+4x_{18}+2x_{20}+32x_{21}+15x_{22}+31x_{23}+2x_{24}+2x_{25}+7x_{26}+8x_{27}+2x_{28}+8x_{29}+2x_{31}+8x_{32}+6x_{33}+7x_{34}+1x_{35}+20x_{38}+8x_{39}<=500$

$38x_1+52x_2+30x_3+42x_4+170x_5+9x_6+7x_7+20x_8+3x_{10}+21x_{11}+4x_{12}+1x_{13}+2x_{14}+14x_{15}+310x_{16}+8x_{17}+4x_{18}+6x_{19}+1x_{20}+18x_{21}+15x_{22}+38x_{23}+10x_{24}+4x_{25}+8x_{26}+6x_{27}+3x_{30}+10x_{32}+6x_{33}+1x_{34}+3x_{35}+3x_{37}+5x_{38}+4x_{39}<=600$

Problem 5

Max $560x_1+1125x_2+300x_3+620x_4+2100x_5+431x_6+68x_7+328x_8+47x_9+122x_{10}+322x_{11}+196x_{12}+41x_{13}+25x_{14}+425x_{15}+4260x_{16}+416x_{17}+115x_{18}+82x_{19}+22x_{20}+631x_{21}+132x_{22}+420x_{23}+86x_{24}+42x_{25}+103x_{26}+215x_{27}+81x_{28}+91x_{29}+26x_{30}+49x_{31}+420x_{32}+316x_{33}+72x_{34}+71x_{35}+49x_{36}+108x_{37}+116x_{38}+90x_{39}+738x_{40}+1811x_{41}+430x_{42}+3060x_{43}+215x_{44}+58x_{45}+296x_{46}+620x_{47}+418x_{48}+47x_{49}+81x_{50}$

such that

$40x_1+91x_2+10x_3+30x_4+160x_5+20x_6+3x_7+12x_8+3x_9+18x_{10}+9x_{11}+25x_{12}+1x_{13}+1x_{14}+10x_{15}+280x_{16}+10x_{17}+8x_{18}+1x_{19}+1x_{20}+49x_{21}+8x_{22}+21x_{23}+6x_{24}+1x_{25}+5x_{26}+10x_{27}+8x_{28}+2x_{29}+1x_{30}+10x_{32}+42x_{33}+6x_{34}+4x_{35}+8x_{36}+10x_{38}+1x_{39}+40x_{40}+86x_{41}+11x_{42}+120x_{43}+8x_{44}+3x_{45}+32x_{46}+28x_{47}+13x_{48}+2x_{49}+4x_{50}<=800$

$16x_1+92x_2+41x_3+16x_4+150x_5+23x_6+4x_7+18x_8+6x_9+12x_{11}+8x_{12}+2x_{13}+1x_{14}+200x_{16}+20x_{17}+6x_{18}+2x_{19}+1x_{20}+70x_{21}+9x_{22}+22x_{23}+4x_{24}+1x_{25}+5x_{26}+10x_{27}+6x_{28}+4x_{29}+4x_{31}+12x_{32}+8x_{33}+4x_{34}+3x_{35}+10x_{37}+6x_{39}+28x_{40}+93x_{41}+9x_{42}+30x_{43}+22x_{44}+36x_{46}+45x_{47}+13x_{48}+2x_{49}+2x_{50}<=650$

$38x_1+39x_2+32x_3+71x_4+80x_5+26x_6+5x_7+40x_8+8x_9+12x_{10}+30x_{11}+15x_{12}+1x_{14}+23x_{15}+100x_{16}+20x_{18}+3x_{19}+40x_{21}+6x_{22}+8x_{23}+6x_{25}+4x_{26}+22x_{27}+4x_{28}+6x_{29}+1x_{30}+5x_{31}+14x_{32}+8x_{33}+2x_{34}+8x_{35}+20x_{37}+6x_{40}+12x_{41}+6x_{42}+80x_{43}+13x_{44}+6x_{45}+22x_{46}+14x_{47}+1x_{49}+2x_{50}<=550$

$8x_1+71x_2+30x_3+60x_4+200x_5+18x_6+6x_7+30x_8+4x_9+8x_{10}+31x_{11}+6x_{12}+3x_{13}+18x_{15}+60x_{16}+21x_{17}+4x_{18}+2x_{20}+32x_{21}+15x_{22}+31x_{23}+2x_{24}+2x_{25}+7x_{26}+8x_{27}+2x_{28}+8x_{29}+2x_{31}+8x_{32}+6x_{33}+7x_{34}+1x_{35}+20x_{38}+8x_{39}+14x_{40}+20x_{41}+2x_{42}+40x_{43}+6x_{44}+1x_{45}+14x_{46}+20x_{47}+12x_{48}+1x_{50}<=550$

$38x_1+52x_2+30x_3+42x_4+170x_5+9x_6+7x_7+20x_8+3x_{10}+21x_{11}+4x_{12}+1x_{13}+2x_{14}+14x_{15}+310x_{16}+8x_{17}+4x_{18}+6x_{19}+1x_{20}+18x_{21}+15x_{22}+38x_{23}+10x_{24}+4x_{25}+8x_{26}+6x_{27}+3x_{30}+10x_{32}+6x_{33}+1x_{34}+3x_{35}+3x_{37}+5x_{38}+4x_{39}+30x_{41}+12x_{42}+16x_{43}+18x_{44}+3x_{45}+16x_{46}+22x_{47}+30x_{48}+4x_{49}<=650$