MILCS: A Mutual Information based Learning Classifier System

Thesis submitted by **Kun Jiang**
in partial fulfilment of the requirements for
the degree of **Doctor in Computer Science**
of **University College London**

Department of Computer Science
University College London

December 17, 2008

I, Kun Jiang, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Recent advances have demonstrated the potential for success in developing ever more general, intelligent and reliable classification and predictive technologies. We aim to explore and investigate the feasibility of designing and implementing robust classification and predictive systems that can deliver high quality predictions at the same time as human-understandable explanations for those predictions. A robust machine learning technique, known as a Learning Classifier System (LCS), can yield robust, accurate predictions, with the explanatory power (in essence, potential for a human to understand machine-learned concepts) not available in other methods.

The focus of this thesis is to develop novel methodologies to design representations and operators for the LCS that will draw on Cascade Correlation Neural Networks (CCNNs) and Information Theory. The integration of these techniques become a new variety of Learning Classifier System, called MILCS (Mutual Information based Learning Classifier System), which utilises mutual information as fitness feedback. Unlike most LCSs, MILCS is specifically designed for supervised learning as traditional LCSs originate from reinforcement learning.

In this thesis, classic data mining/classification problem as well as sophisticated protein structure prediction problems make up the experimentation process. Experimental results of the system along with results from other machine learning systems called XCS, UCS, GAssist, BioHEL, C4.5 and Naïve Bayes, are studied and analysed. The explanatory power of the resulting rule sets is discussed, and a new technique for visualising explanatory power is introduced since explanatory power is an abstract concept. MILCS is also shown to encourage Default Hierarchies (DHs), an important advantage of LCS. Final comments include future directions for this research, including investigations in neural networks and other systems.

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor, Dr Robert Smith, for accepting me as his PhD student, for his remarkable knowledge, guidance, support and patience.

I owe my loving thanks to my parents, Jiang Chunyou and Li Jingbo, for their endless encouragement and confidence. Without them, it would be impossible to finish this work. My loving thanks are due to my English parents, Bill and Ruth, for their unflagging love and care, and a cosy English home.

Last but not least, I gratefully acknowledge financial support provided by the Engineering and Physical Sciences Research Council (EPSRC) under grant number GR/T07541.

Table of Contents

1 Introduction 12
1.1 Test problems in protein structure prediction 13
1.2 Objectives and contributions 13
1.3 Road map 13
2 Background 14
2.1 Machine learning 14
2.1.1 Machine learning by algorithmic type 14
2.1.2 Machine learning by problem type 15
2.1.3 The classification problem 16
2.1.4 Knowledge representations 17
2.1.5 Rule induction algorithms 19
2.2 Genetics-based machine learning (GBML) 20
2.2.1 Evolutionary computation (EC) 20
2.2.2 Evolutionary algorithms (EA) 20
2.2.3 Paradigms of evolutionary algorithms 21
2.2.4 Genetic algorithms (GAs) 22
2.3 GBML models (Learning Classifier Systems) 24
2.3.1 Michigan approach 25
2.3.2 Pittsburgh approach 35
2.3.3 Iterative rule learning approach 38
2.3.4 LCS and data mining 39
2.3.5 Default hierarchies (DHs) 40
2.4 Artificial neural network (ANN) and cascade correlation (CC) 41
2.4.1 Biological neural network 41
2.4.2 Artificial neural network (ANN) 42
2.4.3 Back-propagation 45
2.4.4 Cascade correlation (CC) 46
2.4.5 Receptive fields 49
2.5 Information theory, entropy and mutual information 49
2.5.1 Information theory 49
2.5.2 Entropy and mutual information 50
2.5.3 Channel capacity 52
2.5.4 Applications of information theory 54
2.6 Summary of chapter 54
3 Mutual information-based learning classifier system (MILCS) 55

3.1 Comparing XCS and CCN 55
3.1.1 XCS and CCN: an analogy 55
3.1.2 Supervised versus reinforcement learning 56
3.2 Mutual information rather than correlation 57
3.2.1 Data compression and noisy channel coding: an analogy 57
3.2.2 Sensor placement and channel communication: another analogy 58
3.2.3 The role of mutual information 58
3.2.4 The need for two actions per rule 59
3.3 Description of MILCS 61
3.3.1 Conventional learning classifier aspects 63
3.3.2 Mutual information update 64
3.3.3 Supervised learning component 64
3.3.4 Fitness update 65
3.3.5 Discovery component 66
3.4 The MILCS Process 68
3.5 MILCS vs. XCS 73
3.6 Summary of chapter 73
4 Explanatory Power 75
4.1 Definition of explanatory power 75
4.2 Rule compaction and condensation 75
4.3 Visualising explanatory power 76
4.4 Quantitative measurement 81
4.5 Brief manual for Explanatory Power Visualisation Tool v1.0 82
4.5.1 Prepare the input file 82
4.5.2 Parameters pane 82
4.5.3 Plot pane and information pane 83
4.6 Summary of chapter 83
5 Experimental Results 84
5.1 Multiplexer problem experiments 84
5.1.1 Problem description 84
5.1.2 Parameter settings 86
5.1.3 Results and analysis 88
5.1.4 Scalability 95
5.1.5 Visualisation of explanatory power 96
5.1.6 Effect of deletion parameters 107
5.2 Even parity problem 110

5.2.1 Problem description 110
5.2.2 Parameter settings 111
5.2.3 Results and analysis 112
5.3 Summary of chapter 118
6 Knowledge Discovery in Protein Structure Prediction 119
6.1 Knowledge discovery and machine learning 119
6.2 Protein structure prediction (PSP) 121
6.2.1 Coordination number prediction (CN) 122
6.2.2 Relative solvent accessibility prediction (RSA) 129
6.3 Summary of chapter 136
7 Encouraging DHs in MILCS 138
7.1 New subsumption and deletion algorithms 138
7.1.1 Subsumption by acting 138
7.1.2 Deletion by prediction 139
7.2 Default hierarchy test problems 139
7.2.1 Problem description 140
7.2.2 Parameter settings 140
7.2.3 Results and analysis 143
7.3 Default hierarchies in the PSP problems 143
7.3.1 Default hierarchies in the CN problem 144
7.3.2 Default hierarchies in the RSA problem 145
7.4 Summary of chapter 147
8 Conclusions and Further Work 148
8.1 Global conclusions of the thesis 148
8.2 Further work 150
8.2.1 Multi-class problems 150
8.2.2 Beyond rule learning 150
8.2.3 Reinforcement learning 150
8.2.4 “Cascade” of CCN 151
8.2.5 Improving the visualisation tool for explanatory power 151
9 Appendix 152
9.1 Algorithmic description of MILCS 152
9.1.1 Initialisation and main loop 152
9.1.2 Sub-procedures 153
10 References 163

List of Tables

Table 1: Learning stage of different approaches 19
Table 2: Comparison between non-default hierarchical and default hierarchical rule set 40
Table 3: Comparison of XCS and CCN 56
Table 4: Properties of MILCS classifier 68
Table 5: MILCS parameters 69
Table 6: MILCS rule sets 70
Table 7: MILCS main methods 71
Table 8: Truth table of 3 multiplexer 85
Table 9: MILCS parameters for the multiplexer problems 87
Table 10: XCS and UCS parameters for the multiplexer problems 88
Table 11: QT clustering results of XCS, UCS and MILCS 107
Table 12: MILCS parameters for the even parity problems 111
Table 13: XCS and UCS parameters for the even parity problems 112
Table 14: MILCS parameters for the CN problem 125
Table 15: XCS and UCS parameters for the CN problem 126
Table 16: HP problem rule translation 127
Table 17: Results of XCS, MILCS, UCS, GAssist, C4.5 and Naïve Bayes on the two-state CN problem using HP representation of various window sizes. A • means that MILCS is statistically better than the algorithm to the left. Student T-test with 95% confidence level have been applied 128
Table 18: Redundancy and inconsistency rate of the test set of Real-HP dataset 129
Table 19: MILCS parameters for the RSA problem 132
Table 20: XCS and UCS parameters for the RSA problem 133
Table 21: Results of XCS, UCS, MILCS, BioHEL, C4.5 and Naïve Bayes on a two-state RSA problem 134
Table 22: Default hierarchies of MILCS rules for the RSA problem 136
Table 23: MILCS parameters for the DHs problems 141
Table 24: XCS and UCS parameters for the DHs problems 142
Table 25: Results of XCS, UCS and MILCS on the DHs problems 143
Table 26: MILCS DHs parameters for the CN problem 144
Table 27: DHs results of MILCS on the two-state CN problem comparing to previous results 145
Table 28: MILCS DHs parameters for the RSA problem 146
Table 29: DHs results of MILCS on the RSA problem comparing to previous results 147

List of Figures

Figure 1: Representation of the learning process for classification tasks 17
Figure 2: The general scheme of an EA in pseudo-code 21
Figure 3: One-point crossover 23
Figure 4: Two-point crossover 23
Figure 5: GAs process 24
Figure 6: work cycle of ZCS 25
Figure 7: Work cycle of XCS (for single-step problems) 29
Figure 8: Evolutionary pressure of XCS 33
Figure 9: Structure of a neuron 41
Figure 10: Action potential 42
Figure 11: A perceptron 43
Figure 12: A fully connected multi-layer perceptron 44
Figure 13: Cascade correlation 48
Figure 14: Transmitting message through noisy channel with encoder and decoder 50
Figure 15: Relationships between entropies 52
Figure 16: Communicating over a noisy channel, the big picture 53
Figure 17: Work cycle of MILCS 61
Figure 18: Pseudo-code of MILCS process 72
Figure 19: Distance between two circles 77
Figure 20: Mass-spring-damper system 78
Figure 21: Circle-circle intersection 79
Figure 22: Explanatory power visualisation tool 81
Figure 23: Semantics of a 3 multiplexer 85
Figure 24: An 11 multiplexer rule explained 86
Figure 25: Results from XCS applied to the 6 multiplexer 89
Figure 26: Results from UCS applied to the 6 multiplexer 90
Figure 27: Results from MILCS applied to the 6 multiplexer 90
Figure 28: Results from XCS applied to the 11 multiplexer 91
Figure 29: Results from UCS applied to the 11 multiplexer 92
Figure 30: Results from MILCS applied to the 11 multiplexer 92
Figure 31: Results from XCS applied to the 20 multiplexer 93
Figure 32: Results from UCS applied to the 20 multiplexer 94
Figure 33: Results from MILCS applied to the 20 multiplexer 94

Figure 34: Log-log plot of polynomial scaling of XCS, UCS and MILCS on the multiplexer problems 96
Figure 35: Visualisation of the final rule set developed by XCS on the 6 multiplexer 98
Figure 36: Visualisation of the final rule set developed by UCS on the 6 multiplexer 99
Figure 37: Visualisation of the final rule set developed by MILCS on the 6 multiplexer 100
Figure 38: Visualisation of the final rule set developed by XCS on the 11 multiplexer 101
Figure 39: Visualisation of the final rule set developed by UCS on the 11 multiplexer 102
Figure 40: Visualisation of the final rule set developed by MILCS on the 11 multiplexer 103
Figure 41: Visualisation of the final rule set developed by XCS on the 20 multiplexer 104
Figure 42: Visualisation of the final rule set developed by UCS on the 20 multiplexer 105
Figure 43: Visualisation of the final rule set developed by MILCS on the 20 multiplexer 106
Figure 44: del_{range} effect on the 11 multiplexer problem 108
Figure 45: $freq_{inc}$ effect on the 11 multiplexer problem 109
Figure 46: $freq_{del}$ effect on the 11 multiplexer problem 110
Figure 47: Results from XCS applied to the 5 bit even parity 113
Figure 48: Results from UCS applied to the 5 bit even parity 113
Figure 49: Results from MILCS applied to the 5 bit even parity 114
Figure 50: Results from XCS applied to the 6 bit parity 115
Figure 51: Results from UCS applied to the 6 bit parity 115
Figure 52: Results from MILCS applied to the 6 bit even parity 116
Figure 53: Results from XCS applied to the 7 bit parity 117
Figure 54: Results from UCS applied to the 7 bit parity 117
Figure 55: Results from MILCS applied to the 7 bit parity 118
Figure 56: Knowledge discovery process 120
Figure 57: Sample data of a two-state three-window Real-HP dataset 123
Figure 58: Distribution of hydrophobic/polar target residues classes in the Real-HP dataset: h=hydrophobic, p=polar 124
Figure 59: Sample data of binary attributes window size four AlphaRed_SA dataset 130
Figure 60: Distribution of two-letter alphabets target residues classes in the AlphaRed_SA dataset 131
Figure 61: Default hierarchies of MILCS rules for the RSA problem 135
Figure 62: Default hierarchy test problem 140

List of Acronyms

AA	Amino acid
AI	Artificial intelligence
ANN	Artificial neural network
CA	Credit allocation/assignation
CC	Cascade-correlation
CCN	Cascade-correlation network
CN	Coordination number
CR	Conflict resolution
DH	Default hierarchy
EA	Evolutionary algorithm
EC	Evolutionary computation
ECGA	Extended compact genetic algorithm
EP	Evolution programming
ES	Evolution strategies
GA	Genetic algorithm
GBML	Genetic-based machine learning
GP	Genetic programming
KDD	Knowledge discovery in databases
LCS	Learning classifier system
NN	Neural network (usually refer to ANN)
MI	Mutual information
ML	Machine learning
PSP	Protein structure prediction
RSA	Relative solvent accessibility

Chapter 1

Introduction

As a broad subfield of *artificial intelligence* (AI), *machine learning* (ML) is concerned with the development of algorithms and techniques which allow computers to "learn" and extract information from data automatically. In recent years more and more problems have been handled with techniques belonging to this discipline. Examples of these tasks are prediction, decision-support system, scheduling, automatic classification, etc. This thesis concentrates on one of the sub-categories of ML, supervised learning, which is defined as the learning process where there is a "teacher" that gives "the learner" direct feedback about its performance. Moreover, this thesis is focused on a paradigm of supervised ML called evolutionary learning, or *genetics-based machine learning* (GBML). This paradigm involves any learning technique which uses *evolutionary computation* (EC). EC uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search to achieve a desired goal. Such processes are inspired by biological mechanisms of evolution. To be more precise, the focus of this thesis is on an *evolutionary algorithm* (EA) technique, called a *learning classifier system* (LCS), which is a subset of EC. First described by John Holland [50], a LCS consists of a population of general rules[1] with a *genetic algorithm* (GA [47] [53]) searching for better rules in response to environmental feedback.

However, LCSs were originally constructed for reinforcement learning [107], which feedback acts more subtly than that in the supervised learning. Unlike supervised learning, a direct feedback of "correct" or "incorrect" is not given by the "teacher", instead, a feedback on how well the system has performed is given. It is particularly useful when the knowledge of problem domain is unknown or limited. However, when applying LCSs to supervised learning problems, this additional knowledge is typically not exploited.

This thesis also draws close connections to *cascade correlation network* (CCN [34] an *artificial neural network* (ANN or simply NN) architecture for supervised learning, and information theory. The literature shows that artificial neurons from CCNs have similarities with rules from LCSs [101] (For a detailed description, see Chapter 3).

[1] Rules are also often called classifiers in the LCS literature

1.1 Test problems in protein structure prediction

As this thesis is part of a larger *Engineering and Physical Sciences Research Council* (EPSRC) funded project, it is necessary to motivate the related use of *protein structure prediction* (PSP) problems as tests for the system that is developed. PSP is one of the most significant technologies pursued by computational biology. Its aim is to determine 3D structure of proteins from their *amino acid* (AA) sequences. One approach to this problem is to predict some attributes of a protein, such as secondary structure prediction, the *relative solvent accessibility* (RSA), *coordination number* (CN) and the disulfide bonding states of cysteines. This attributes have been studied with various methods including neural networks [91], Bayesian methods [108], approaches based on information theory [81], protein descriptors based on logistic functions [80], multiple sequence alignments [25], decision trees [94] and combination of more than one method [23]. However, these methods, in the particular problems the author is concerned with, produce results with only moderate accuracy, and very limited explanatory power. The advanced machine learning methodology developed in this thesis is used to produce better explanatory power and accuracy, and the PSP problems serve as a powerful real-world test set.

1.2 Objectives and contributions

This thesis yields a new form of learning classifier system that uses *mutual information* (MI [97] [98]) as its primary fitness feedback, in supervised learning settings. This system is called the *mutual information learning classifier system* (MILCS [103], pronounced "my LCS"). In addition to drawing on current LCS research and information theoretic concerns, the system draws on an analogy to CCNs in its design.

1.3 Road map

This thesis is split into 7 chapters. Chapter 2 describes the background knowledge on ML, GBML, LCS, ANNs, and information theory. Chapters 3 to 7 contain the main contributions of this thesis. Chapter 3 is driven by the motivations behind the methodology being developed, and reviews appropriate literature along the way. The analogy between artificial neurons from CCN and rules from LCS are analysed, and the innovative idea of their integration, along with how MI is used as the fitness function in the proposed system, is explained in detail. Chapter 4 introduces a visualisation method developed for explanatory power analysis. The experimentation process is shown in Chapter 5 to 7, and results are studied with comparison to other machine learning systems. Final conclusions and future work are discussed in Chapter 8 followed by the Appendix.

Chapter 2

Background

2.1 Machine learning

The aim of this section is to provide a sufficient overview of ML to place the contribution of the thesis in context. For this reason, it emphasises supervised learning problems, classification problems and rule-based knowledge representations.

The section is structured as follows. Subsection 2.1.1 and 2.1.2 seeks to categorise the machine learning paradigms in two different ways, each emphasising on a different aspect. Subsection 2.1.3 shows what a classification problem is and describes the protein structure prediction problems. Subsection 2.1.4 focuses on the knowledge representation of different machine learning approaches and draws attention on the learning classifier system for the underlying structural learning. Subsection 2.1.5 provides two main rule induction algorithms for learning rules.

2.1.1 Machine learning by algorithmic type

ML involves developing computer systems that automatically improve their performance measure *P* over task *T* through experience *E*. Here is an example,

Example: Learn to play checkers

T: Play checkers

P: Percentage of games won in the world tournament

E: Opportunity to play against itself

There are many ways to classify the machine learning paradigms. Langley [66] classifies the learning paradigms depending on how they learn, defining five categories:

- Inductive learning

This paradigm employs condition-action rules, decision trees or similar logical knowledge structures. Information about classes or predictions is stored in the rule actions sides of the rules or the leaves of the tree. Learning algorithms in the rule-induction framework usually carry out a

greedy search through the space of decision trees or rule sets, using statistical evaluation functions to select attributes to incorporate into the knowledge structure.

- Instance-based or case-based learning

This paradigm represents knowledge in terms of specific cases or experiences and relies on flexible matching methods to retrieve and these cases to new situations. One common approach simply finds the stored case nearest (according to some distance metric) to the current situation, and then uses it for classification or prediction.

- Analytic learning

This paradigm also represents knowledge as rules in logical form but typically employs a performance system that uses search to solve multi-step problems. A common technique is to represent knowledge as inference rules, then to phrase problems as theorems and to search for proofs. Learning mechanisms in this framework use background knowledge to construct proofs or explanations of experience, then compile the proofs into more complex rules that can solve similar problems either with less search or in a single step.

- Connectionist learning

This paradigm, also called neural networks, represents knowledge as a multi-layer network of threshold units that spreads activation from input nodes through internal units to output nodes. Weights on the links determine how much activation is passed on in each case. The activations of output nodes can be translated into numeric predictions or discrete decision about the class of the input.

- Evolutionary learning

This paradigm, as stated in Chapter 1 , is defined as any kind of learning task which employs as its search engine a technique belonging to the evolutionary computation field [76]. Evolutionary computation techniques are optimisation tools inspired loosely by natural evolution. Typically, a population of candidate solutions (individuals) are transformed (evolved) through a certain number of iterations of a cycle, containing an almost blind recombination of the information contained in the individuals and a selection stage that directs the search towards the individuals considered good by a given evaluation function. A broader description of evolutionary computation and evolutionary learning (or GBML) can be found later in Section 2.2.

2.1.2 Machine learning by problem type

The above distinct classifications of these paradigms are more historical than scientific. However, recent experimental comparisons between different learning methods have helped break these boundaries. Hybrid methods that cross paradigm boundaries are increasingly common. These include algorithms for inducting decision trees that contain linear threshold units and techniques for transforming rules into neural networks and back again. These convergences are the signs of a balanced and maturing field. Thus a better classification, using a more general, problem-based point of view suggests three main categories:

- Supervised learning

A learning process where there is some kind of tutor (automatic or human) that gives the learner direct feedback about the appropriateness of its performance. Relating this definition to the example given in Subsection 2.1.1, one must have the performance measure P to perform supervised learning.

- Unsupervised learning

This kind of learning is characterised by having no performance feedback, in the previous example, that is, no *P*. In this case, the task of the learning system is to construct some kind of knowledge, based only on the flow of experience *E*, typically trying to identify the regularities existing on *E*.

- Reinforcement learning [107]

This paradigm could be considered a middle point of the two previous ones. In this case, the feedback acts in a subtle way, indicating the performance of the system as a kind of reward, good or bad, instead of informing in a specific way what is being done correctly or incorrectly.

However, in the Bayesian network [48] perspective, machine learning approaches can be broken into two components,

- Parameter learning

The learning of continuous-valued parameters in a solution representation. In Bayesian networks, this is the learning of probabilities of events, conditioned on one another (the probabilities that reside within Bayesian network nodes).

- Structural learning

The learning of discrete connections between elements of the solution representation. In Bayesian networks, this is the learning of which events are conditioned on which (the links in the Bayesian network).

It is generally acknowledged that the latter is more difficult than the former. Also, the latter has a profound effect on the computational complexity of the former: the number of probabilities one must learn goes up exponentially with the number of links. Moreover, structural learning is also associated with generalisation, parsimony, and explanatory power: fewer discrete connections make for a more understandable representation of a solution.

As the thesis is focused on supervised learning tasks (and explanatory power is one of the targets), aspects other than supervised learning and structural learning are shown in minimum detail.

2.1.3 The classification problem

This thesis focuses on one of the supervised learning tasks called *classification*, which is the most studied data mining task (See Subsection 2.3.4). Classification is defined as the construction of a theory that models the concept or concepts represented by a set of examples.

Given a set of instances *I*, each of them is associated with a finite set of class *C*, and the task of classification is to create a theory *T* based on *I* and *C* that, given an un-associated new instance, gives a prediction of the class of this instance (see Figure 1).

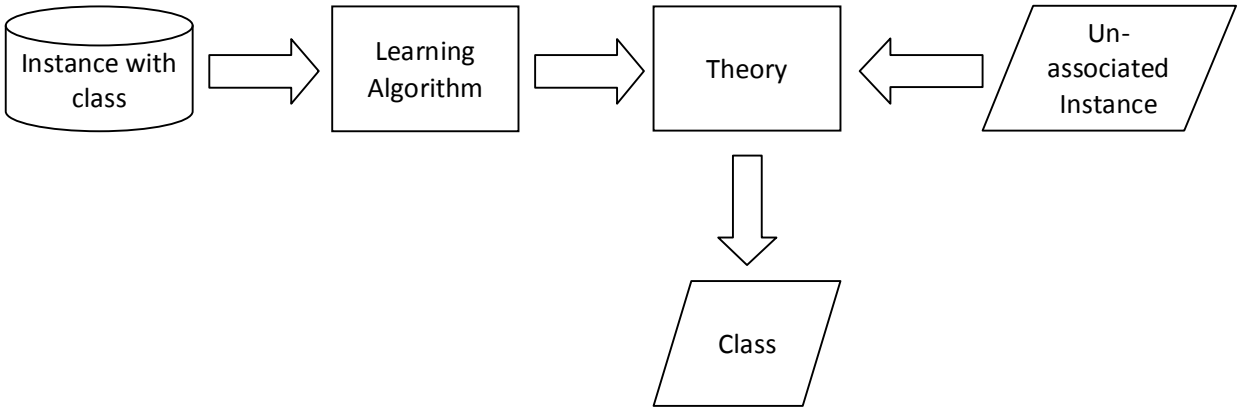

Figure 1: Representation of the learning process for classification tasks

Figure 1 shows the two stages of a classification process: *exploration* (also called *training*) with associated instances and *exploitation* (also called *testing*) with un-associated instances. When developing a classifying system, the process of exploitation is typically simulated. Therefore the associated instances are divided into two non-overlapping sets which are called the *training set* and the *test set*.

The training set allows the system to generate a theory, and the test set validates the accuracy of the theory (that is, how correctly the system classifies the un-associated instances). The capacity for generating a theory that models correctly the concept or concepts represented by the training set is known as *generalisation capacity*. Good performance on the test set is a sign of good generalisation.

One should note that most research on machine learning, with the exception of work in the analytic paradigm, focuses on simple classification or prediction tasks. The most robust learning methods are designed for such problems. The restriction to classification is not really very severe, since one can usually decompose a complex process such as design, control, or planning into a sequence of individual steps, each of which involves simple classification or prediction.

2.1.4 Knowledge representations

In order to present a theory that is discovered through learning, a knowledge representation is required to model the theory. The objective of knowledge representations is to express knowledge in computer-tractable form, such that it can be used to help agents perform well [93]. The following are the different knowledge representations being widely used for the classification tasks.

- Rule-based

Rules are the simplest and most comprehensible way of expressing knowledge. Although the form of rules may differ, rules are commonly in the following syntax,

if *condition* then *action*

Usually the condition is a predicate in a certain logical system and the action is an associated class, meaning that if one detects the input satisfying the condition, then action is fired. However, chances

are that many rules match, and one has to decide on which rule makes the best prediction. There are two main types of procedures used in literature:

- Decision lists

  Rules are sorted based on pre-defined performance metrics. The first rule in the list that matches the condition is used to predict its class. Typical performance metrics include those based on *accuracy*[2], and those based on *experience*[3].

- Voting processes

  A decision list chooses a single rule to classify an input instance. Alternatively, one can combine the outcome of all the rules that match an instance. The simplest approach is to choose the majority class from the matched rules. Another alternative is to sum, for each class, the number of instances of the class matches previously by these rules, and then choose the class most covered.

- Artificial neural networks (ANNs)

ANN is an information processing paradigm that is inspired by the way biological nervous system, such as the brain, processes information. A biological neural network consists a large number of interconnected neurons. As a rough analogy, typical ANNs are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output (which may become the input to many other units). Although this knowledge representation is not used in the proposed system, it draws close analogy to rule sets (see Section 2.4).

- Bayesian networks [48]

Bayesian networks are directed acyclic graphs whose nodes represent variables, and whose links encode conditional probabilistic independencies between the variables.

- Decision trees

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values or range of values for this attribute [78].

- Case-based

This knowledge representation is used when a system solves new problems based on the solution of similar past problems. Thus, it consists of storing sets of instances.

Table 1 is a summary of the learning stages of various machine learning approaches [86]:

[2] The degree to which the rule predicts the correct class on training examples

[3] The number of times the rule fires, for instance

Approach	Structural Elements	Parameters	Parameter Update	Structural Update
Neural Networks	Nodes and Connections	Weights	Back-propagation	Pruning
Bayesian Networks	Nodes and Connections	Conditional Probabilities	Conditional Probability Updates	Typically None
Rule-based Systems	Rules	Conflict Resolution Parameters	Conflict Resolution Parameters	Typically None
Decision Trees	Nodes	Thresholds for Splitting on Data Attributes	Threshold Update	Greedy Addition of Nodes
Function Approximation (kernel methods, etc.)	Basis Set	Coefficients	Weight Update	Arbitrary Truncation
Learning Classifier Systems	Generalized Rules	Performance Parameters	Performance Parameter Updates	Genetic Learning

Table 1: Learning stage of different approaches

One will notice that decision trees, function approximation and learning classifier systems (see Subsection 2.3) employ sophisticated techniques for structural learning, whereas the others are basic or typically none. However, there are many notable exceptions, amongst them the technique employed in CCN, a neural network architecture (see Subsection 2.4.4).

Each of the above knowledge representations also categorises the different paradigms of data mining algorithms, such as rule induction, decision-tree building, evolutionary algorithms, neural networks, instance-based learning, Bayesian learning and so on.

2.1.5 Rule induction algorithms

Among the paradigms of data mining algorithms, the combination of rule induction and evolutionary algorithms (see Subsection 2.2.2) are the focus of this thesis. Many rule induction algorithms have been studied in the literature. This section describes two of the most widely used algorithms:

- Separate-and-conquer

Separate-and-conquer has its origins in the AQ family of algorithms [77] under the name “covering strategy” and it is the most well-known rule induction algorithm. The algorithm basically searches for a rule that covers a part of its training instances, separates these examples, and recursively conquers the remaining examples by learning more rules until no examples remain. This ensures that each instance of the original training set is covered by at least one rule.

When tackling a binary classification task (positive/negative examples), the order in which the rules are learned or used for classification does not matter, because the rules only describe one class, the positive class. Negative examples, i.e. when no rule fires for a given example, will be classified as negative. This is equivalent to assuming a default rule for the negative class at the end of an ordered rule list.

However, many real world problems are concerned with multi-valued or even continuous class variables. In these cases the order of the rules are very important because different predictions will be made. To overcome this problem a decision list [89] is used. A good example uses this technique is CN2 [24].

- Learning all rules at the same time

Although separate-and-conquer is popular for rule induction systems such as CN2, it has been pointed out that the ever decreasing number of examples being available for learning successive rules will adversely affect the system performance. A suggested alternative is to evolve all rules simultaneously, using the entire training instances for each, i.e. if the induction algorithm "conquers without separating." An example of this strategy is the RISE system [30]. One can also see evolutionary computation (the subject of Section 2.2) as this kind of rule induction.

2.2 **Genetics-based machine learning (GBML)**

This section gives a detailed overview of a specific paradigm of machine learning, evolutionary learning, also known as GBML. The section focuses on LCSs, the most common form of GBML, and the one used in this thesis.

The section is structured as follows. Subsection 2.2.1 gives an introduction on EC and its biological inspiration. Subsection 2.2.2 provides a type of EC, EA, and its general scheme. Subsection 2.2.3 focuses on the different types of EA paradigms. Subsection 2.2.4 shows details of one particular EA paradigm, GAs.

2.2.1 Evolutionary computation (EC)

EC techniques are optimisation tools that solve problems using procedures inspired by biological mechanisms of evolution. They use an iterative progress, such as growth or development in a population [4] to transform them in a manner inspired by natural evolution. Selection and recombination produce an implicit, directed exploration of the search space, robustly converging to the regions of the space where the best solutions are placed.

2.2.2 Evolutionary algorithms (EA)

EA is the collective name for a range of problem-solving techniques based on principles of biological evolution. There are many different EAs. However, the common underlying idea behind all these techniques is the same: given a population of individuals, some environmental pressure

[4] Candidate solutions to a problem

causes selection (survival of the fittest) and some set of operators cause genetic variation. The combination of these two effects causes a rise in the fitness of the population.

Given a quality (fitness) function to be maximised one can randomly create a set of candidate solutions, i.e., elements of the function's domain, and apply the quality function as selection. Higher-fitness candidates are chosen to seed the next generation. They do so through the application of *recombination* and *mutation* operators.

Recombination is an operator applied to two or more selected candidates (parents) and results in one or more new candidates (offspring). Mutation behaves like a mistake in copying genes, thus it is applied to candidates with a certain probability to create new candidates. Executing recombination and mutation leads to a set of new candidates that compete, based on their fitness (and possibly age or other metrics), with the old candidates for a place in the next generation. This process can be iterated until candidates with sufficient quality are found or a previously set computational limit is reached (See Figure 2).

```
BEGIN
      INITIALISE population with random candidate
solutions;
      EVALUTATE each candidate;
      REPEAT UNTIL (TERMINATION CONDITION is satisfied)
DO
            1. SELECT parents;
            2. RECOMBINE parents;
            3. MUTATE the resulting offspring;
            4. EVALUATE new candidates;
            5. SELECT individuals for the next generation;
      OD
END
```

Figure 2: The general scheme of an EA in pseudo-code

The various types of EA all follow this general outline, and differ only in technical details. For instance, the representation of a candidate solution is often used to characterise different EA types. Typically, the candidates are represented by strings over a finite alphabet in GAs, real-valued vectors in *evolution strategies* (ES), finite state machines in classical *evolutionary programming* (EP) and trees in *genetic programming* (GP). These differences have mainly historical origins [32].

2.2.3 Paradigms of evolutionary algorithms

Using a classical classification, four main EA paradigms [39] can be categorised,

- Evolution strategies (ES) [87]

These techniques typically use an individual representation consisting of a real-valued vector. Early ES emphasised mutation as the main exploratory search operator, but currently both mutation and

recombination are used. An individual often represents not only real-valued variables of the problem but also parameters controlling the mutation distribution, characterising a self-adaptation of mutation parameters. The mutation operator usually modifies individuals according to a multivariate normal distribution, where small mutations are more likely than large mutations.

- Evolutionary programming (EP) [37]

Originally developed to evolve finite-state machines, but it is now often used to evolve individuals consisting of a real-valued vector. Unlike ES, in general it does not use recombination. Similar to ES, it also uses normally distributed mutations and self-adaptation of mutation parameters.

- Genetic algorithms (GAs) [47] [53]

This is the most popular paradigm of EA. GAs emphasise recombination as the main exploratory search operator, and consider mutation as a minor (but necessary) operator. Mutation is typically applied with a very low probability. In early ("classic") GAs individuals were represented by binary strings, but nowadays more elaborate representations, such as real-valued strings, are also used.

- Genetic Programming (GP) [65]

This paradigm is often described as a variation of GA rather than a mainstream EA paradigm itself. Individuals being evolved in this paradigm are various kinds of computer programs, consisting not only of data structures but also of functions (or operations) applied to those data structures. These programs are usually represented using trees.

- Estimation of distribution algorithms (EDAs) [71]

In the previous EA techniques, individuals are generated by combining and modifying existing ones in a stochastic way and the underlying probability distribution of new individual over the space of possible individuals is not explicitly specified. In EDAs, the distribution is explicit, and is used to generate new candidate solutions directly. The distribution is directly modified as a part of the evolutionary process.

As the thesis mainly focuses on GAs (which is the underlining search function for LCS), the rest of this section focuses on this particular EA type.

2.2.4 Genetic algorithms (GAs)

GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimisation problems. Although randomised, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space.

GAs simulate the survival of the fittest among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to biological chromosome in DNA. Each individual represents a point in a search space and a possible solution. The individuals in the population are then made to go through a process of evolution:

- Individuals in a population compete for resources and mates.
- Those individuals that are most successful in each competition will produce more offspring than those individuals that perform poorly.
- Genes from "good" individuals propagate throughout the population.

In order to understand GAs, it is necessary to understand all three GAs operators,

- Selection

Fitness is used to determine the selection of individuals that are used to produce new solutions.

- Recombination

Once the parents are selected, they are recombined to generate offspring. There are a large number of recombination methods in the GAs literature. A typical one is *one-point crossover*, which operates as shown in Figure 3. By extension, *two-point crossover* is shown in Figure 4.

Before crossover:		After crossover:	
Parent A	0000|0000	Offspring A	0000|1111
Parent B	1111|1111	Offspring B	1111|0000

Figure 3: One-point crossover

Before crossover:		After crossover:	
Parent A	000|000|00	Offspring A	000|111|00
Parent B	111|111|11	Offspring B	111|000|11

Figure 4: Two-point crossover

These figures show the typical crossover types used in GAs where strings are swapped between cut-off points. Other techniques, such as *uniform crossover*, where random bits are selected from each parent individual, are also frequently used. In this case, the operator decides (with probability typically set to 0.5) which parent contributes each of the string bits.

- Mutation

Mutation takes place with some low probability. Each new individual will have some of their variables randomly altered. In a binary-encoded GA, bits are flipped[5]. This maintains diversity within the population and inhibits premature convergence of the population to a single individual. Along with selection but without crossover, mutation creates a parallel, noise-tolerant, hill-climbing algorithm as is often used in other evolutionary computation paradigms.

The following flowchart shows the general scheme of GAs,

[5] If one bit is 0 then it becomes 1 and vice versa

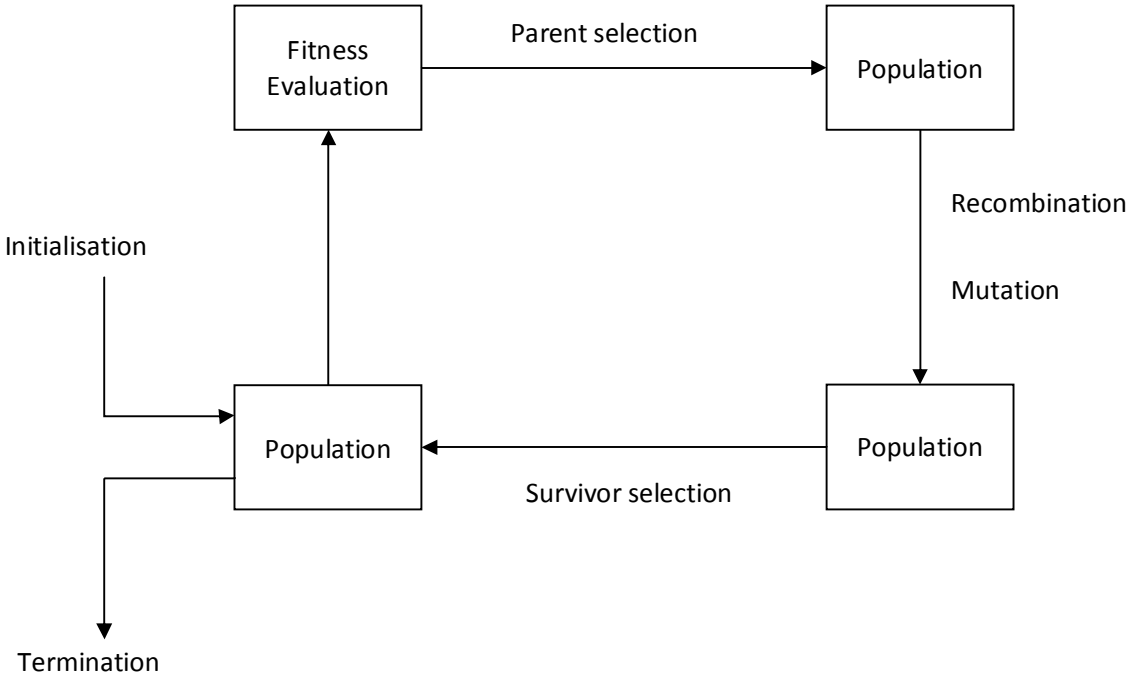

Figure 5: GAs process

2.3 GBML models (Learning Classifier Systems)

LCSs, first introduced by John Holland [50], are machine learning techniques which combine reinforcement learning [107], evolutionary computing and other heuristics to produce adaptive systems. They are rule-based systems, where the rules are usually in the traditional production system form of:

If *condition* Then *action*

Evolutionary computing techniques and heuristics are used to search the space of possible rules, whilst reinforcement learning techniques are used to assign utilities to existing rules, guiding the search for better rules. Depending on how GA acts, there are usually two types of models in the literature depending on how GA acts. Hybrid methods, such as the iterative rule learning approaches have also rise to surface. In general, the main motivation for using GAs in the discovery of high-level prediction rules is that they perform a global search and cope better with attribute interaction than the greedy rule induction algorithms often used in data mining [36].

The section is structured as follows. Subsection 2.3.1.1, 2.3.1.2 and 2.3.1.3 overviews three Michigan-style LCSs. Subsection 2.3.1.4 draws attention to the difference lying between UCS and the previous two LCSs. Subsection 2.3.2.1 and 2.3.2.2 introduces two Pitt-style LCSs. Subsection 2.3.3.1 describes a system using the iterative learning approach. Subsection 2.3.5 shows the concept of default hierarchies (DHs) and its potential importance to LCSs.

2.3.1 Michigan approach

Holland and Reitman introduced the first LCS [51] based on Holland's well-known GA in 1978 but revised it twice in the next decade. Although it was a standard system at the time, it was found that the system was complex and hard to be applied to real world applications. To overcome these disadvantages, Wilson came up with ZCS [112] (his Zeroth-level Classifier System) which, in his words, kept much of Holland's original framework but simplified it to increase understandability and performance. Since the original LCS is complicated and only its descendants are the focus of this thesis, its description is not included in the thesis.

2.3.1.1 ZCS

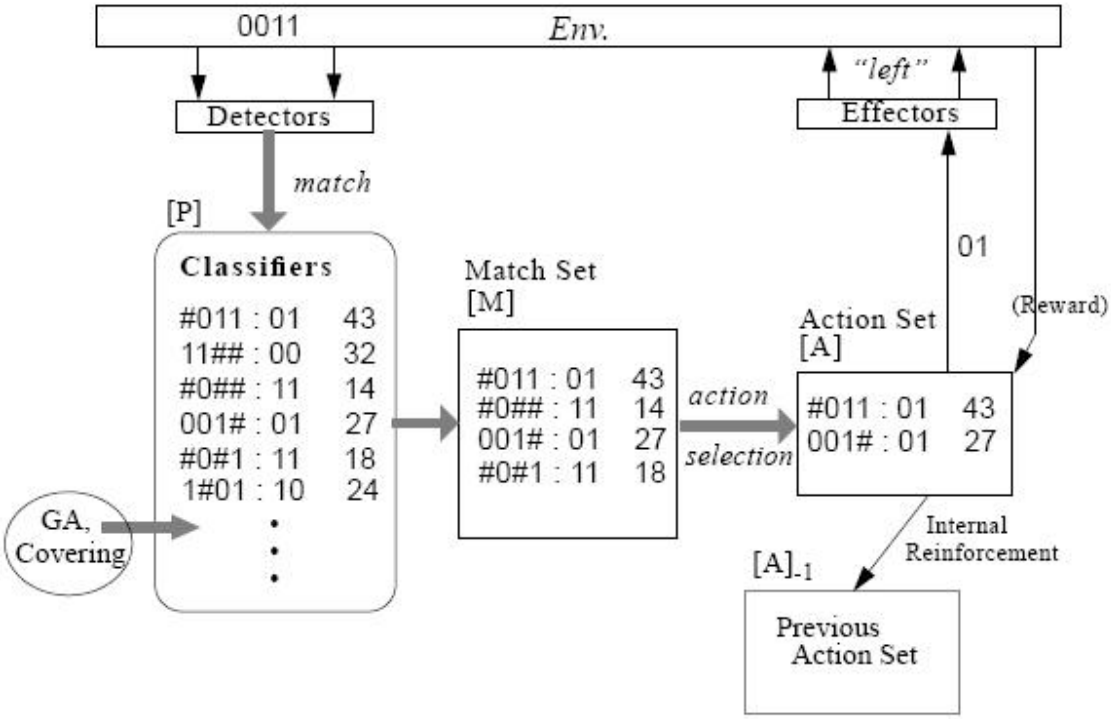

Figure 6: work cycle of ZCS

- Rule sets
  - [P] Population set, contains the classifier population
  - [M] Match set, contains the classifiers that match the input training case
  - [A] Action set, contains the classifiers from match set [M] advocating the chosen action
  - $[A]_{-1}$ Action set of the previous time step
- Classifier representation

Each rule is formed by a condition string and an action string. In the most typical LCS encoding, the condition string is formed by alphabet {0, 1, #}. The "#" symbol indicates "don't care" so it matches either 0 or 1. The action string is the associated action or class. Thus in Figure 6, a rule's condition and action is separated by a colon and followed by its strength (fitness).

- The parameters used in ZCS are:
  - N Population size
  - $P_{\#}$ Probability of a # at an allele position in the condition of a classifier created through *covering*, and in the conditions of classifiers in the initial randomly generated population
  - S_0 Strength assigned to each classifier in the initial population
  - β Learning rate for strength updates under the bucket brigade
  - γ Discount factor for the bucket brigade
  - τ Fraction of strength deducted from classifiers in [M] - [A]
  - χ Probability of crossover per invocation of the GA
  - μ Probability of mutation per allele in an offspring. Mutation takes 0, 1, # equiprobably into one of the other allowed alleles
  - ρ Average number of new classifiers generated by the GA per time-step of the performance cycle
  - φ If the total strength of [M] is less than φ times the mean strength of [P], covering occurs

2.3.1.1.1 Performance component

Each rule's condition in [P] is compared with the string provided by *detectors*[6]. If the bit at every non-# position of a rule matches the corresponding bit of the detector string, it is placed in the *match set* [M]. A roulette wheel with sectors sized according to the strengths of members of [M] is used for the action selection (*conflict resolution* or CR). Thus a particular action a is selected with probability equal to the sum of the strengths of the classifiers in [M] which advocate that action, divided by the total strength of classifiers in [M]. Next, an *action set* [A] is formed, consisting of all members of [M] which advocated a. Finally, a is sent to the *effectors*[7] interface, and the corresponding action is fired.
In the case that [M] is empty, i.e. no classifier in [P] matches the input, or when the total strength in [M], $S_{[M]}$, is less than a fraction φ of the population mean strength, the covering operation is triggered. It creates a new classifier whose condition matches the input and contains a probabilistically determined number of #s. The classifier's action is chosen randomly and the strength is set to the population average. It is believed to be a relatively crude operation resembling rote learning or imprinting; it does not build on knowledge already in the population. However, it is not unusual to have an empty [M] in most systems. Acting randomly is one solution but covering also allows the testing of a hypothesis of the created classifier at the same time.

2.3.1.1.2 Reinforcement component

[6] Used by the system to perceive the state of the environment

[7] Control the systems actions on and within the environment

Once the selected action is fired, the environment responds with a *reward*, which is assigned to members of [A]. The *credit assignment* (CA) procedure works as the following,

1) A fixed fraction β ($0 < \beta \leq 1$) of the strength of each member of [A] is deducted from the member's strength and placed in an (initially empty) common "bucket" B. If $S_{[A]}$ is the total strength of members of [A], the effect is to deduct $\beta S_{[A]}$ from $S_{[A]}$ and place it in the bucket.
2) If the system receives an immediate reward r_{imm} from the environment after taking action a, a quantity $\frac{\beta\, rimm}{|A|}$ is added to the strength of each classifier in [A] ($|A|$ is the number of classifiers in [A]). The effect is to increase $S_{[A]}$ by $\beta\, r_{imm}$.
3) Classifiers in $[A]_{-1}$ (if it is non-empty) have their strengths incremented by $\frac{\gamma B}{|A-1|}$, where γ is a discount factor ($0 < \gamma \leq 1$), B is the total amount put in the bucket in step a, and $|A_{-1}|$ is the number of classifiers in $[A]_{-1}$.
4) Finally, [A] replaces $[A]_{-1}$ and the bucket is emptied.

2.3.1.1.3 Discovery component

ZCS employs a basic panmictic (i.e., over entire population) GA. Once some performance criteria is met, the GA is triggered and selects two classifiers based on their strengths, copies them to form the offspring, recombines or mutates them according to some fixed probabilities, before inserting them back to the population. Half of each parent's strength is deducted from the parent and assigned to the offspring. If recombination occurs, the copy strengths are reset to the parent's mean. In order to maintain [P] of fixed size, two rules are selected and deleted with probability inverse to their strengths.

A *covering* procedure is also used when [M] is empty (i.e., no classifiers in [P] match the detector input) or when the total strength in [M], $S_{[M]}$, is less than a fraction φ of the population mean strength, $\frac{\varphi S_{[P]}}{|P|}$ (|P| is the number of classifiers in [P]). The procedure creates a new classifier which matches the input but with randomly determined number of #s, based on parameter $P_{\#}$. The classifier's action is chosen randomly, and the strength is set to the population average, $\frac{S_{[P]}}{|P|}$. The classifier is then inserted to [P] and one of the classifiers in [P] is deleted as in the GA.

The descendant of ZCS, XCS, shares many aspects of ZCS, such as rule representation, matching strategy, covering and many others, but differs in key details that have led to greatly increased success in the literature.

2.3.1.2 XCS

In many LCSs, such as LCS and ZCS, the classifier strength parameter serves as a predictor of future payoff, and this prediction is used as a measure of fitness in the GA. The role of strength parameter is to estimate the payoff the classifier system will receive, when the satisfied condition is met and the best action is chosen. Therefore the strength parameter is also used as the measure of fitness for the discovery component's GA. However, Wilson indicates that there are several problems when using strength-based fitness [113]:

1) In strength-based LCS, a set of classifiers match a set of environmental states and these states usually have different payoff levels. It is the nature of a LCS system to concentrate on classifiers with high payoff. The solution is to have a sharing technique [47] that each active classifier has a fraction of the available payoff rather than having a full value. It also could be

reinforced by implementing a non-panmictic (not over the entire population) GA, which means apply GA only on part of the entire population. Both techniques add generalisation pressure to the system to balance off the high accuracy in an implicit way.

2) However, implementing a sharing technique alters the original meaning of "strength-based", because a classifier's fitness no longer predicts payoff; instead, the "shared" strength does.

3) The GA cannot distinguish an accurate classifier with moderate payoff from an overly general classifier with the same payoff level on the average. If a non-panmictic GA is used as noted above, over-general classifiers are preferred in the system.

Give the above problems, it seems necessary to reconsider the basis of classifiers fitness. Since Wilson pointed out that the LCS could not distinguish the difference between accurate and over-general classifiers (as stated in problem 3), it seemed reasonable to base the fitness on classifier *accuracy*. This is the approach taken by Wilson in constructing XCS [113].

A further reason to confirm this move comes from reinforcement learning, which emphasises the formation of relatively complete mappings $X \times A \Longrightarrow P$ from the product set of situations and actions to payoffs. The conventional strength-based LCS attempts to discover best rule without knowing the payoff sequence of every possible action. With reinforcement learning, a suboptimal classifier could then be considered as an incomplete exploration and, the system is oriented towards learning relatively complete maps of the consequences of each action; then it is easy to decide the best action.

While the research focus has been shifted towards accuracy based LCSs, Bull and Hurst [18] showed that ZCS does not necessarily suffer from over-general classifiers (which is caused by fitness sharing) and is able to perform optimally with the right parameter settings.

Since the introduction of XCS in 1995, there have been several important additions and modifications in the literature that increased the robustness of the system [114] [62]. The description here is based on Butz's report [11] which summarised the basic framework of XCS and its most important updates.

The following graph shows the work cycle of XCS (for single-step problems),

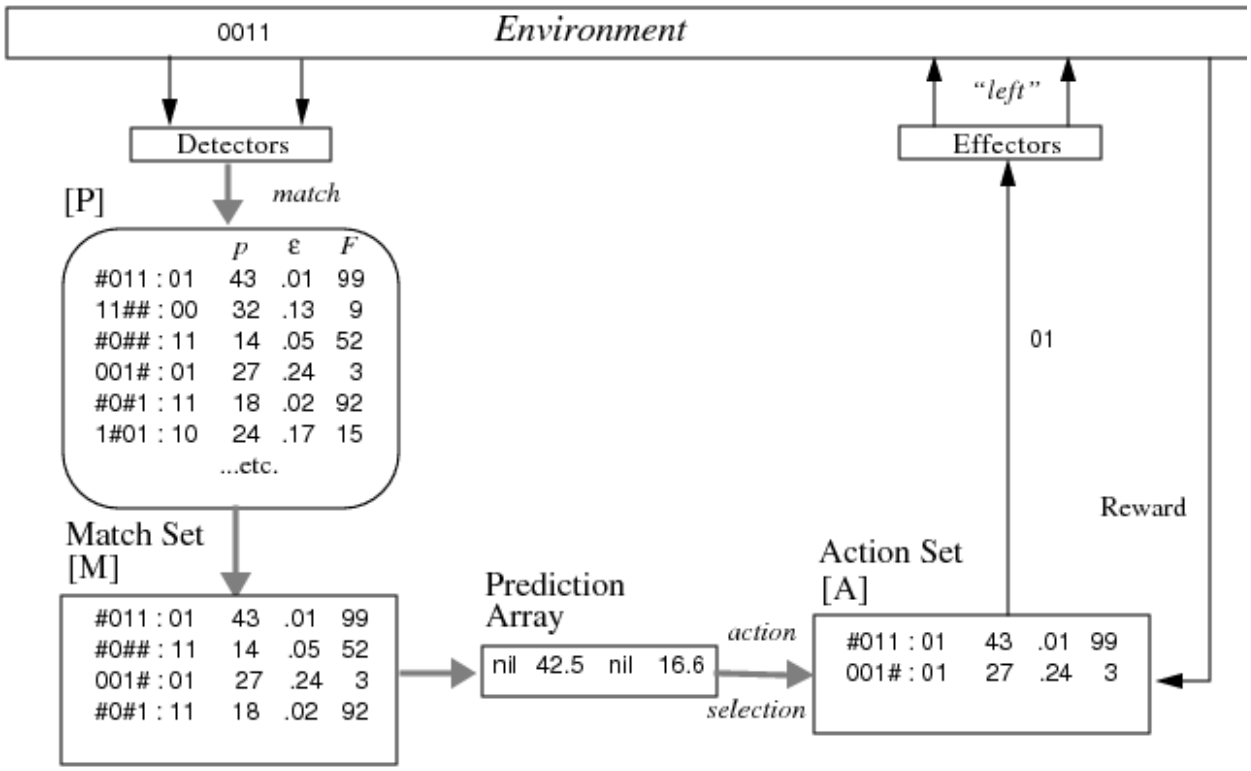

Figure 7: Work cycle of XCS (for single-step problems)

- Rule sets
  - [P] Population set, contains the classifier population
  - [M] Match set, contains the classifiers that match the input training case
  - [A] Action set, contains the classifiers from match set [M] advocating the chosen action
- Parameters for classifier
  - *p* Expected reward of the classifier if it classifies correctly an example
  - ε Estimation of the prediction error of the classifier
  - *k* Accuracy of the classifier based on ε
  - *F* Fitness of the classifier based on the inverse of accuracy *k*
  - *exp* Experience parameter of the classifier which is increased by one each time the classifier's parameters are updated
  - *num* Numerosity of the classifier, number of copies of this classifier in the population
  - *as* Average size of action sets the classifier has participated
- Parameters for XCS
  - *N* Maximum population size
  - β Learning rate for prediction, prediction error, and fitness updates

- γ Discount factor
- θ_{GA} Do a GA in this [M] if the average number of time-steps since the last GA is greater than θ
- ε_0, α, ν Parameters of the accuracy function
- χ Probability of crossover per invocation of the GA
- μ Probability of mutation per allele in an offspring. Mutation takes 0, 1, # equiprobably into one of the other allowed alleles
- θ_{del} is the deletion threshold. If the experience of a classifier is greater than this value, its fitness may be considered in its probability of deletion
- δ Specifies the fraction of the mean fitness in [P] below which the fitness of a classifier may be considered in its probability of deletion
- θ_{sub} is the subsumption threshold. The experience of a classifier must be greater than this value in order to be able to subsume another classifier
- θ_{mna} specifies the minimal number of actions that must be present in [M], or covering will occur
- $P_{\#}$ Probability of a # at an allele position in the condition of a classifier created through covering, and in the conditions of classifiers in the initial randomly generated population
- p_I, ε_I, F_I Prediction, prediction error, and fitness assigned to each classifier in the initial population
- P_{explr} specifies the probability during action selection of choosing the action uniform randomly

Similar to ZCS, XCS is also composed of three components, performance component, reinforcement component and discovery component.

2.3.1.2.1 Performance component

Given an input, [M] is formed and the covering operation is inherited from ZCS. The system then forms a *system prediction P(a)* for each action *a* represented in [M]. There are several reasonable ways to determine *P(a)*. The typical way is a fitness-weighted average of the predictions of classifiers advocating *a*. Presumably, one wants a method that yields the system's "best guess" as to the payoff—internal and/or external—to be received if *a* is chosen. The *P(a)* values are placed in a *prediction array*[8], and an action is selected.

Many action-selection methods are possible. The system may simply pick the action with the largest prediction and this is referred to as deterministic action selection. Alternatively, the action may be selected probabilistically, with the probability of selection proportional to *P(a)*; which is called *roulette-wheel action selection* (similar to what ZCS uses for its action selection). In some cases the action may be selected completely at random (from actions with non-null predictions), ignoring the *P(a)*. There are of course additional schemes. Once an action is selected, the system forms an [A] consisting of the classifiers in [M] advocating the chosen action. That action is then sent to the effectors and an immediate reward *r* may (or may not) be returned by the environment.

[8] Some of whose slots will receive no value if there is no corresponding action in [M] as indicated as "nil" in Figure 7

2.3.1.2.2 Reinforcement component

XCS's reinforcement component consists updating the p, ε, and F parameters of classifiers in [A], as shown in Figure 7. Once an action is selected, the environment returns a reward r, which is used to adjust the parameters of the classifiers in [A]. There are five steps in this calculation.

1) Prediction (deterministic action selection)

$$P = \gamma \max\big(P(a)\big) + r$$

where γ ($0 < \gamma \leq 1$) is the discount factor

$$p = \begin{cases} p + \dfrac{(P-p)}{exp}, & exp < \dfrac{1}{\beta} \\ p + \beta(P-p), & otherwise \end{cases}$$

where *exp* is the rule's experience and β ($0 < \beta \leq 1$) is the learning rate

2) Error

$$\varepsilon = \begin{cases} \varepsilon + \dfrac{|P-p| - \varepsilon}{exp}, & exp < \dfrac{1}{\beta} \\ \varepsilon + \beta(|P-p| - \varepsilon), & otherwise \end{cases}$$

where *exp* is the rule's experience, ε is the error; r is reward; p is prediction.

3) Accuracy

Accuracy of an inverse function of the classifier's error:

$$k = \begin{cases} 1, & \varepsilon < \varepsilon_0 \\ \alpha \left(\dfrac{\varepsilon}{\varepsilon_0}\right)^{-\nu}, & otherwise \end{cases}$$

The parameter ε_0 ($\varepsilon_0 > 0$) determines the threshold error under which a classifier is considered to be accurate. The parameters α ($0 < \alpha < 1$) and ν ($\nu > 0$) controls the degree of decline in accuracy if the classifier is inaccurate [12].

4) Relative accuracy

Relative accuracy is the accuracy of a classifier divided by the sum of accuracies in [A]

$$k' = \frac{kn}{\sum_i k_i n_i}, over\ [A]$$

where n is the numerosity of the classifier.

5) Fitness

$$F = F + \beta(k' - F)$$

This is fairly straight forward that fitness update is based on relative accuracy and previous fitness. Note that the basing fitness on the relative accuracies provides sharing [55] among the classifiers belonging to the same action set.

In summary, the goal of the fitness function is two-fold: maximising the number of covered examples and minimising the number of classification mistakes over the training set of the rule.

Note that for single-step problems XCS updates reinforcement parameters on [A] because there is no connection between each input instances whereas for multi-step problems the credit assignment

is delayed by one time step so reinforcement parameters updates on the previous action set $[A]_{-1}$. As the focus of this thesis is to explore single step problems only, it is not shown in Figure 7.

2.3.1.2.3 Discovery component

In contrast to ZCS, the GA in XCS is applied to the action sets, rather than panmictically (i.e., over the whole population). It is also called a "niche GA". The original XCS used a Roulette-Wheel (proportional) Selection algorithm, which chooses a classifier for reproduction proportional to the fitness of the classifiers in set [A]. In later versions, tournament selection was added and it is shown to make XCS more independent from various parameter settings [16]. Once two parents from [A] are selected, they are recombined and mutated with probabilities χ and μ respectively.

The resulting offspring are introduced into the population. First, each offspring is checked for subsumption [114] with its parents. If one of the parents is sufficiently experienced, accurate and more general than the offspring, then the offspring is not introduced and its parent's *num* is incremented. This process is called *GA subsumption*. If the offspring classifier cannot be subsumed, it is inserted in the population, deleting another classifier if the population size is at a user-specific maximum value *N* [62]. The deletion probability of a classifier is proportional to the average of the size of the action sets in which it has participated (stored in the parameter *as*). Also, if the classifier is sufficiently experienced and its fitness is low, its deletion probability is higher. XCS also has *action set subsumption* which allows rules of [A] to subsume each other at every GA iteration.

Similar to ZCS, XCS employs a covering algorithm in the case that [M] is empty, i.e. none of the classifiers in [P] matches the given input condition.

2.3.1.2.4 XCS, the big picture

The strength of XCS is to evolve accurate and maximally general classifiers, and this is achieved by applying the right *evolutionary pressure*. Butz and Pelikan [13] identified five evolutionary pressures in XCS:

1) Set pressure represents the general tendency due to more general classifiers are in the action set and the deletion algorithm takes place on the population set.
2) Mutation pressure pushes towards an equal number of 1s, 0s and don't cares.
3) Deletion pressure emphasises the evolution of equally distributed classifier subsets and high-fitness classifiers additional to the set pressure influence. Thus it becomes part of the set pressure.
4) Subsumption pressure pushes towards generality and allows a fast convergence of the population.
5) Fitness pressure focuses on an accurate tendency which is the pressure from inaccurate, over-general classifiers to accurate, maximally general ones.

These pressures are illustrated in Figure 8. It can be seen that fitness pressure and set pressure are the main pressures to allow a balanced evolution between accuracy and generality and this is why XCS is able to evolve maximally general classifiers. However, this fine balance is achieved "implicitly" via non-panmictic GA and a complex second order fitness function. Its pure mathematical origin is somehow lack of a firm theoretical background. This is one of the motivations of this thesis: evolving maximally general classifiers "explicitly".

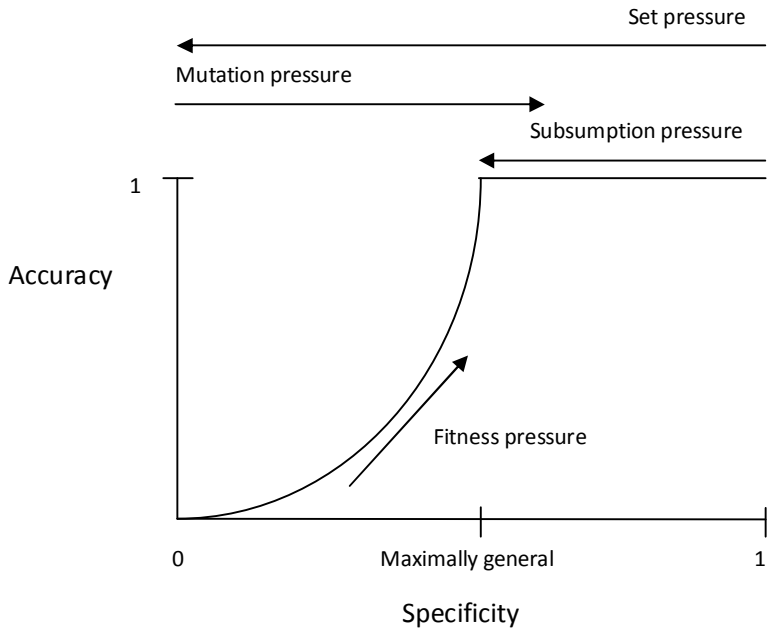

Figure 8: Evolutionary pressure of XCS

2.3.1.3 UCS

Recently, a descendant of XCS which employs a supervised learning scheme, UCS (sUpervised Classifier System) [8], came to surface. It is able to evolve a best action map rather than a complete action map of which XCS evolves. UCS keeps the principal features of XCS, a fitness based on accuracy and a niche GA, but changes the way in which accuracy is computed.

- Rule sets
  - [P] Population set, contains the classifier population
  - [M] Match set, contains the classifiers that match the input training case
  - [C] Correct set, contains the classifiers from match set [M] advocating the correct action
  - [!C] Incorrect set, contains the rest of the classifiers from match set [M]
- Parameters for classifier
  - *acc* accuracy of the classifier
  - *F* fitness of the classifier based on *acc*
  - *num* numerosity, number of copies of the classifier
  - *exp* experience parameter of the classifier which is increased by one each time the classifier's parameters are updated

Parameters for UCS are mostly inherited from XCS thus they are not repeated here. Notice that because of its supervised nature, XCS's action prediction is not inherited.

2.3.1.3.1 Performance component

In UCS, learning is performed using a supervised learning scheme, which means that the input example comes along with the associated class. This differs from XCS where, according to a reinforcement learning scheme, the input example condition is presented, the system responds with an action and the environment returns a reward.

In classification problems, UCS benefits from a supervised scheme as it is described in the following. During learning, an input example x with the associated class a is presented. From x, [M] is formed from [P] (with the same covering operation as in XCS), consisting of those classifiers whose condition matches x. Those classifiers in [M] which have the correct known class a form the *correct set* [C].The rest of the classifiers are placed in *incorrect set* [!C]. Since the correct class is given by the example, there is no need to generate reward from the environment for the parameter updates, as oppose to XCS. For the same reason, action selection is only needed during exploit/test mode. UCS uses a deterministic (best) action selection from votes (weighted by fitness) of classifiers of [M].

2.3.1.3.2 Parameter updates

The classifier's parameters in UCS are updated in the following way. The classifier's accuracy is computed as the proportion of correct classifications with respect to the number of matches (exp):

$$acc = \frac{no.of\ correct\ classifications}{exp}$$

This value is updated each time a classifier belongs to [M], and thus it is an average over all the examples that the classifier has matched. The fitness is computed as a function of accuracy:

$$F = (acc)^{\nu}$$

where ν is a constant.

Compared to that of XCS, two key points can be highlighted. First, in UCS, the accuracy parameter acc directly estimates the accuracy rate of each classifier. Second, UCS does not perform any fitness sharing.

2.3.1.3.3 Discovery component

The non-panmictic GA in UCS is also inherited from XCS and it is applied to [C]. It selects two classifiers from [C] with probability proportional to fitness and applies crossover and mutation. The resulting offspring are inserted in the population. GA Subsumption is also included in UCS in the same way as in XCS. The only difference is that a classifier is considered to be accurate when its accuracy acc is greater than a threshold acc_0. If [P] size if full when inserting new rules, other classier have to be deleted. The deletion algorithm is also borrowed from XCS. Action set subsumption is not implemented in [84].

Covering is also applied in UCS when [C] is empty. In this case, a classifier covering the current input is created, with the same class provided with the input.

2.3.1.4 Comparison between UCS and other Michigan-style LCSs

UCS maintains the same structure as XCS, but uses a different accuracy computation. It inherits the generalisation algorithms from XCS, which are mainly based on the fact that the GA is applied on niches rather than on the whole population. Niches in UCS are defined by [C], and therefore it is expected that UCS will generalise over the search space of correct rule sets, leaving the incorrect rules out of exploration. UCS also shares some features with other LCS. The way in which UCS divides the classifiers in [M] into the [C] and [!C] resembles NEWBOOLE [10]. The accuracy

computation is equivalent to Frey and Slate's classifier system [40]. Both systems were also designed for supervised problems.

2.3.1.5 Other supervised XCS systems

UCS is not the first, and certainly not the only supervised learning classifier systems in the literature. The introduction of XCS and its stability generated a lot of interested in the field and various modified versions started to appear. Wilson's XCSF [115], is a supervised learning algorithm which is based on XCS for function approximation. The prediction estimation mechanism was used to form the approximations: given an input vector x, the value y of the function to be approximated was treated as a pay-off to be learned. X-NCS [19] for function approximation, a neural learning classifier system based on XCS, is another example. In this system, each traditional condition-action rule is replaced by a fully connected multi-layer perceptron (See Section 2.4.2). More recently the back-propagation search algorithm has been included [83] which provides local search on top of the traditional GA-based search. More recently, Lanzi et al. shows also that XCSF can be applied to the learning of Boolean functions and to typical multi-step problems involving delayed rewards [70].

2.3.2 Pittsburgh approach

As suggested by Holland, a natural way to proceed is to represent an entire rule set as a string (an individual), maintain a population of candidate rule sets, and use selection and genetic operators to produce new generations of rule sets. Historically, this was the approach taken by Smith and his students while at the University of Pittsburgh [104] [105], which gave rise to the name "the Pitt approach". Because of the nature of the genetic search in this approach, an advantage of such systems is a compact solution size, often smaller than that of a Michigan-styled system. Due to the fact that the best candidate rule set is searched in the GA. However, this also becomes a weakness because a large population will increase the computational cost dramatically. There is a large number of different Pitt approach LCSs in the literature [68]. In the following Subsections, those relevant to the comparisons are reviewed.

2.3.2.1 GABIL (GA Batch-Incremental concept Learner)

GABIL was developed by DeJong and Spears in 1991 [28].

- Knowledge representation
  - Each individual is a variable-length set of rules:

$$I = (R1 \vee R2 \ldots \vee Rn)$$

  - Each classification rule has binary representation, fixed length and codifies a predicate. This system performs concept learning from positive/negative examples. Rules only cover the positive examples, thus, there is no class associated to the rule. The semantic representation of the rule is:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \wedge \ldots \wedge (A_n = V_2^n \vee A_n = V_m^b))$$

$A_i, i \in [1..n]$ is the attribute i of the dataset

$A_i^j, i \in [1..n], j \in [1..m]$ is the value j that can take the attribute i

- These predicates can be mapped to a binary string with the following procedure:
  - Imagine one has 4 attributes: *(A1, A2, A3, A4)*. The values of *A1* are *(A, B, C, D)*, the values of *A2* are *(E, F, G)*, the values of *A3* and *(H, I, J, K, L)* and finally the values of *A4* are *(M, N)*.
  - The predicate "*(A1 is B or C)* and *(A2 is E or F or G)* and *(A3 is H or K)* and *(A4 is M)*" is represented as:

A1	*A2*	*A3*	*A4*
0110	111	10010	10

- By looking at the examples one can see that all bits associated to the attribute *A2* are set to 1. This is the mechanism that the representation has to indicate that this attribute is irrelevant. (like the "don't care" mechanism in other LCSs)

- Fitness function. The fitness function is computed after classifying all instances of the training set, and consists simply of a squared accuracy function:

$$fitness(individual) = \left(\frac{no.of\ instnaces\ correctly\ classified}{total\ no.of\ instances}\right)^2$$

- Recombination operator. This operator needs a small restriction to guarantee that semantically correct offspring are created. Cut points can take place in any rule of the individual, which does not have to be the same for both parents, but it has to be placed in the same position inside the rule.
- Process
  - The system starts learning with only one training example. A rule set is generated covering it.
  - After generating the initial rule set, the system tries to classify a second example with it.
  - If the new example is classified correctly, the same test is repeated with more examples.
  - If not, it is run again, using all the instances tested so far.

2.3.2.2 GAssist

GAssist [3] is a newly developed Pitt-style system which is based on the GABIL system. Most of its components are similar of GABIL's, such as matching strategy, basic nominal presentation, fitness function, GA recombination and mutation operations, etc. However, it consists some novel elements such as integrating an explicit and static default rule in the system, the adaptive discretisation intervals rule representation, windowing techniques for generalisation and run-time reduction, bloat control and generalisation pressure methods. The system has been tested on various problems and benchmarked against other learning systems such as C4.5 [85], SVM and many others.

The GAssist system was originally inspired in GABIL, thus, several of the following details are common in both systems:

- Matching strategy:

  Individuals (rule sets) are treated as a decision list. Therefore, the first rule that is matched for an input instance is used to classify it

- Fitness function:

GABIL's squared accuracy fitness function

- Base nominal representation:

  The GABIL representation

- Recombination operator:

  Same as GABIL. This means selecting cut points in equivalent position inside the rule (but in any rule) for both parents

- Mutation operator:

  The GABIL's bit-flipping mutation is used for the nominal representation.

- Missing values policy:

  When dealing with datasets with missing values a substitution policy is used. That is, by gathering the instances belonging to the same class as the one with missing values, one substitutes the missing value by either the most frequent value or the average value, depending on the type of attribute (nominal or real-valued).

- Process:

  Same as GABIL

According to the author of GAssist, the innovative features of the system are in the following four areas,

1) An interesting feature of encoding the individuals of a Pitt-style LCS as a decision list is the emergent generation of a default rule. With a default rule one can generate more compact and accurate rule sets. However, the performance of the system is strongly tied to the learning system choosing the correct class for this default rule. For this reason, GAssist has a mechanism automatically decides the class of the default rule. This technique works by integrating in a single population individuals having all possible default classes and compete with each other. In order to prevent some premature individuals from competing, a niched tournament selection is used.

2) GAssist has an *adaptive discretisation intervals* (ADI) rule representation, which evolves rules that can use multiple discretisation algorithms, letting the evolution choose the correct discretisation for each rule and attribute. Also, the intervals defined in each discretisation can split or merge among them through the evolution process, reduction the search space where it is possible. With these two characteristics, the proposed representation gains robustness and has an efficient exploration of the search space

3) In order to reduce the total computational cost of the system, GAssist applied a windowing technique called ILAS (*incremental learning with alternating strata*). The objective of this method is to reduce the cost of fitness computations by using only a subset of the training examples to evaluate each individual, thus reducing the total computational cost of the system.

4) Bloat control and generalisation pressure are very important issues in Pitt-style LCSs, in order to achieve simple and accurate solutions in a reasonable time.

   The first problem is a common issue in evolutionary computation techniques that use variable-length representations: the bloat effect. It consists in a growth without control of the size of the individuals. GAssist controls this problem by implementing a rule deletion operator that eliminates rules that do not contribute to the fitness of the individual. This operator, properly controlled can be beneficial in two aspects: run-time reduction and introduction of diversity.

The second issue is related to the machine learning field: the capacity of the learning system to generate well generalised solutions. Usually a well generalised solution is identified as an accurate solution of low complexity. Thus, the explicit control of the generalisation issue is also closely tied to the control of the individual size. Two alternative methods have been proposed in GAssist to apply generalisation pressure in the system, which are the hierarchical selection operator, and the more complex, MDL (*Minimum Description Length*) based fitness function

2.3.3 Iterative rule learning approach

2.3.3.1 BioHEL (Bioinformatics-oriented Hierarchical Evolutionary Learning)

BioHEL [5] [6] is a GBML system following the separate-and-conquer approach (see Subsection 2.1.5) from the same author of GAssist. It is strongly influenced from GAssist therefore several of its features have been inherited.

- Knowledge representation: same as GAssist and GABIL
- Fitness function:

$$Fitness = TL \times W + EL$$

where *TL* is the theory length, underlining the complexity of the solution, and *EL* is exceptions length, indicating the accuracy of the solution. This fitness function has to be minimised.

W is a weight that adjusts the relation between *TL* and *EL*. BioHEL uses the automatic weight adjustment heuristic proposed for GAssist.

TL is defined as:

$$TL(R) = \frac{\sum_{i=1}^{i=NA} NumZeros(R_i)/Card_i}{NA}$$

where *R* is a rule, *NA* is the number of attributes of the domain, R_i is the predicate of rule *R* associated to attribute *i*, *NumZeros* counts the number of bits set to zero for a given predicate in GABIL representation and $Card_i$ is the cardinality of attribute *i*.

It is said that $0 < TL < 1$ is always true. It is designed in this way to simplify the tuning of *W*. The *NumZeros* in the GABIL predicates are a measure of specify. Therefore, promoting the minimisation of zeros means promoting general and thus less complex rules.

EL is designed to achieve the balance between accuracy and coverage (number of examples matched). Therefore it is biased towards covering a certain minimum of examples and once a given coverage threshold has been reached the bias is reduced.

It is defined as:

$$EL(R) = 2 - ACC(R) - COV(R)$$

$$ACC(R) = \frac{corr(R)}{matched(R)}$$

$$COV = \begin{cases} MCR \times \dfrac{RC}{CB} & If\ RC < CB \\ MC + (1 - MCR) \times \dfrac{RC - CB}{1 - RC} & If\ RC \geq CB \end{cases}$$

$$RC = \frac{matched(R)}{|T|}$$

where *COV* is the adjusted coverage metric that promotes the coverage of at least a certain minimum number of examples, *RC* is the raw coverage of the rule. *ACC* is the accuracy of the rule, *corr(R)* is the number of examples correctly classified by *R*, *matched(R)* is the number of examples matched by *R*, *MCR* is the weight given in the coverage formula to achieve the minimum overage, *CB* is the minimum coverage threshold and $|T|$ is the total number of training examples.

- Process:

  BioHEL has two general stages. In the main level there is a separate-and-conquer style algorithm. Once each rule is obtained, the training examples that are covered by this rule are removed from the training set, to force the GA of the next iteration to explore other areas of the search space. By using an explicit default rule covering the majority class, each evolved rule aim to cover the other classes.

  In the inner level, the GA is run repeatedly with the same set of instances and the best offspring from all the GA runs are added to the population (and covered examples are removed from the training set). The GA operators are similar to the ones of GAssist.

2.3.4 LCS and data mining

As mentioned in Subsection 2.1.3, on classification problems, one of the most studied data mining tasks, is the focus of this thesis. Since the introduction of Holland's LCS, learning classifier systems have been successfully applied to many domains, especially *data mining*. Data mining is a process of extracting hidden pattern from data. With the ever-increasing data gathered in the world, it becomes an important tool of transforming large amount of data into useful information. To be precise, data mining can be categorised into six procedures [21]:

- Data extraction – the collation of data from one or more sources.
- Data cleansing – the identification and treatment of erroneous or missing data
- Data reduction – the removal of features which are insufficiently correlated to the given task.
- Data modelling – the discovery of patterns in the data
- Model interpretation – identification of the discovered patterns.
- Model application – use of the identified patterns, e.g., for future predictions.

The strength of LCSs lies in data reduction and modelling. Early applications include the use of various versions Holland's LCS on gas pipe line control [46], two-class discrimination task [88], MONKS problem [95], letter recognition [40] and letter sequence prediction tasks [90]. Wilson's Boole LCS implementation [110] for classification problems and later adapted to Newboole [10] by Bonelli and Parodi with a modified reinforcement update. Other systems such as COGIN [45], GABIL and a hybrid system called REGAL [44], have shown competitive performances against other machine learning techniques on a number of well-known datasets.

Due to the complexness of Holland's LCS paradigm, research in data mining using learning classifier systems faced difficulties. Wilson's XCS, improved from his ZCS, the first simplified LCS model, has shown to overcome the difficulty and produced competitive or better performances than other machine learning algorithms. Since then, XCS has captured the researchers' attention and been used for mining protein structure prediction data, breast cancer and image processing data [21], etc. See Chapter 6 for an elaborated discussion of the process of data mining.

2.3.5 Default hierarchies (DHs)

The concept of DHs was introduced by Holland et al. [54]. He pointed out that the rules that constitute a category do not provide a definition of the category. Instead a set of expectations (which are taken to be true) are provided only so long as more specific information do not contradict them. These "default" expectations, in the absence of additional information, provide the best guess of the current situation. Therefore rules can be organised into default hierarchies, that is, hierarchies ordered by default expectations based on subordinate/super-ordinate relations among concepts. For instance, one has some default expectations about a vehicle but it can be overridden by more specific expectations produced by evidence that the vehicle is a car. These expectations, in turn, can be overridden by more specific expectations, such as evidence that the car is a BMW. Early on, the classifier system has reliable information only about parts in very simple contexts. It can exploit this information, but more complex contents will provide frequent surprises, departures, and exceptions. The system gathers more complicated contexts as the system gaining experiences so that is can bias its choices accordingly. As a result, the system builds a hierarchical structure that grow from early "defaults", bases on simple contexts, to layers of exceptions based on more detailed contexts.

Both Goldberg [46] and Smith [99] demonstrated the existence of DHs. Smith described it as the following:

"Default hierarchies are sets of rules where the utilities of partially correct, but broadly applicable rules (defaults) are augmented by additional rules (exceptions). By forming default hierarchies, an LCS can store knowledge in parsimonious rule sets that can be incrementally refined. To do this, an LCS must have conflict resolution mechanisms that cause exceptions to consistently override defaults."

Here is an example,

Non-default hierarchical rule set		**Default hierarchical rule set**		
100000	0	100#0#	0	(default rule)
100101	0	100001	1	(exception rule)
100001	1			
100100	0			

Table 2: Comparison between non-default hierarchical and default hierarchical rule set

DHs are an important potential advantage of LCSs. However, encouraging the development of DHs in a designing LCS is difficult [99]. What is more, XCS does not support DHs because they involve inherently inaccurate classifiers [63]. For the same reason, UCS does not encourage them either. However, the proposed system may solve this problem. The DHs results are shown in Subsection 6.2.2.4 and Chapter 7.

2.4 Artificial neural network (ANN) and cascade correlation (CC)

The last section is about training rule-based systems and this section is focused on training neural networks. A brief description of ANNs is outlined in Subsection 2.1.4. The theory this thesis presents draws a close analogy between neurons and classifiers and puts that theory into practise. Thus it is necessary to understand the fundamentals behind ANN. As supervised classification problem is the main interest of this thesis, two of the most popular supervised learning algorithms for neural networks are studied in detail.

The section is structured as follows. Subsection 2.4.1 gives an introduction on biological neural network, the inspiration of ANN. Subsection 2.4.2 shows the three basic components when constructing an ANN. Subsection 2.4.3 and 2.4.4 focuses on two of the most widely used supervised learning algorithms for ANN.

2.4.1 Biological neural network

The biological neuron, in a nut shell, is the cell that living organisms use to detect both the internal and external environment of their bodies, to formulate behavioural responses to those signals and to control their bodies based on these responses. A vast network of neurons are either physically connected or functionally related in the central nervous system and the peripheral nervous system. Following graph shows the simplified structure of a neuron:

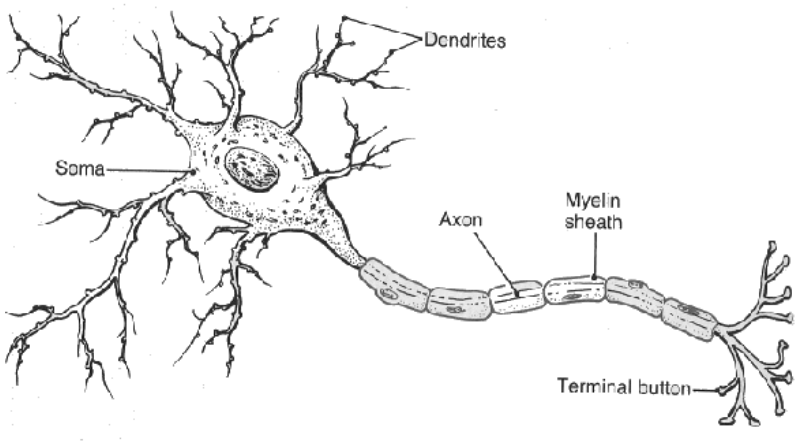

Figure 9: Structure of a neuron

A neuron consists of a body (the *soma*), a set of input lines (the *dendrites*) and a single output line (the *axon*). The dendrites are branching structures which connect with the axons of other neurons through terminals (the *synapses*). The great number of connections between dendrites and axons form a gigantic neural network.

The state of a neuron is described as either of *inhibited* or *excited*. When it is in an inhibited state, or resting potential, the inside of the neuron is negatively charged relative to the outside. This is caused by the semi-permeable membrane which lets the potassium ions (K+) cross in freely (in other words, potassium channels) but a "pump" pumps out every three sodium ions (Na+) for every

two K+ ions it puts in. Along with the rest of the negatively charged ions in the neuron, it reaches -70mV at equilibrium.

Signals between neurons are sent electrochemically. Electrochemical signals (stimulus) are sent from synapses of one neuron to another it is connected to, causing the latter's resting potential to move towards 0mV. The receiving neuron adds it to the rest of its input signals and only if it exceeds an internal-defined threshold (usually -55mV) it will fire an action potential. First the sodium channels will open to cause Na+ ions rushing into the neuron (because of the neuron's negative charge). Thus the neuron becomes more and more positive until the potassium channels start to open and K+ ions rush out. Meanwhile the sodium channels start to close. This causes the action potential to go back towards -70mV. The potassium channels stay open until it goes past -70mV and eventually it will once again reach the balance of -70mV. The following graph illustrates the action potential of a neuron,

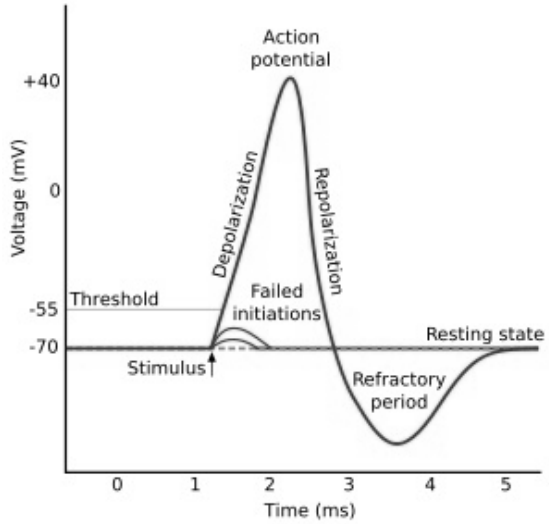

Figure 10: Action potential

The human brain contains roughly 100 billion neurons, linked with up to 10,000 connections each. It works very differently from conventional digital computer. The brain is a highly complex, non-linear, and parallel computer. It can perform certain computations (such as pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. This is the motivation of ANNs.

2.4.2 Artificial neural network (ANN)

ANNs attempt to use machines to solve tasks that human brains do well through abstract low-level brain function, based on biological neural network. However, modern software implementations using ANNs do not strictly follow biological inspiration due to practical reasons.

There are typically three things that need to be defined when constructing an ANN.

2.4.2.1 Architecture

This concerns the topology of the network. Let us start with simplest form of a neural network, a *single-layer feed-forward* neural network (*single-layer perceptron* or simply *perceptron*), used for the classification of patterns that are linearly separable, i.e. patterns that lie on opposite sides of a hyperplane. It consists of a single neuron with adjustable synaptic weights (parameters) and bias.

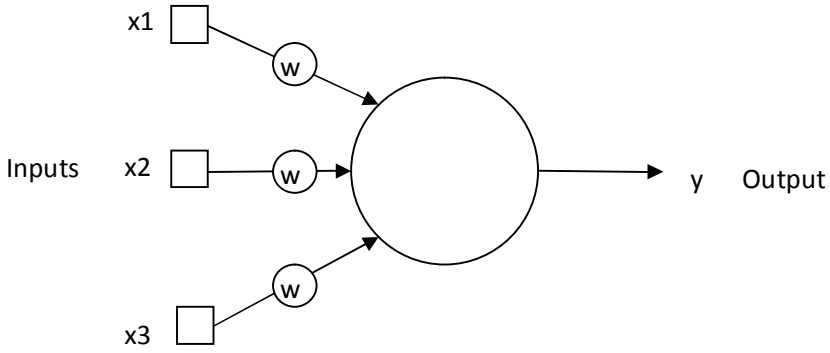

Figure 11: A perceptron

The inputs of an artificial neuron act as dendrites whereas the output as axon. In ANN terminology, neurons are normally referred to as "nodes" or "units". The reason it is called "feed-forward" is because the input layer of source nodes project onto an output layer of neurons, but not vice versa. There are no cycles or loops in the network.

When many nodes connect to each other they are structured in "layers". Outputs of nodes in one layer are the inputs to another layer. When the perceptron has one or more hidden layer that is not part of the input layer or output layer, it becomes a *multi-layer perceptron*,

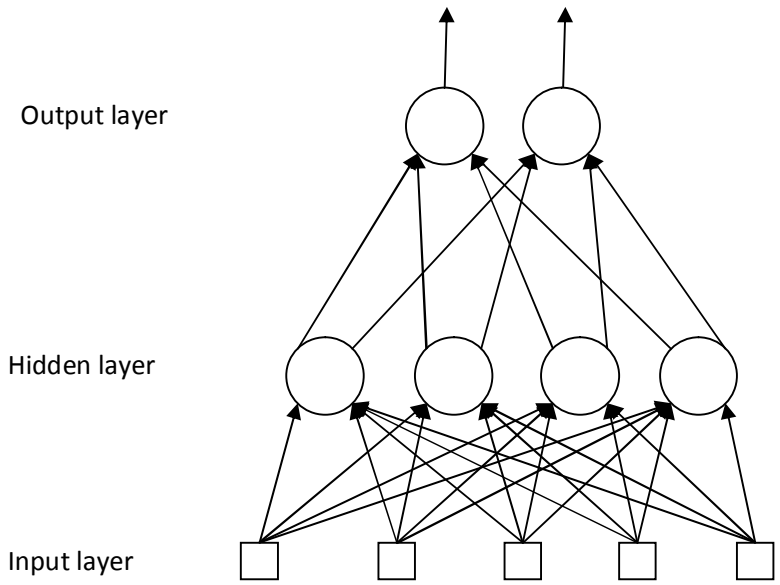

Figure 12: A fully connected multi-layer perceptron

One other distinctive feature of this network is that each neuron should have a non-linear activation function (or cost function) which will be explained in detail in the upcoming Subsections. This network in Figure 12 is said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. If some connections are missing, it is then said to be partially connected.

There are many other topologies of network architecture but it is worth to mention a fundamentally different architecture from the previous two: *recurrent networks*. This type of network distinguishes itself from the feed-forward network by having at least one feedback loop. For example, the output signals could be sent back to the input layer of neurons. However, this type of network is outside the scope of this thesis.

2.4.2.2 Activation function

As a biological neuron, the activities of each artificial neuron change in response to each other. Typically the activities are based on the weights (parameters, w1, w2, w3, as shown in Figure 11) in the network. There are two steps to define an activation rule.

First, activation potential of a neuron is calculated as the following,

$$v = \sum_{i=0}^{I} w_i x_i$$

where $i = 1, \ldots, I$.

Second, the output y in Figure 11 is set as function $\varphi(v)$ of the activation. There are many possible functions here, but the followings are the most popular,

- Linear: $y = v$

- Sigmoid: $y = \varphi(v) = \frac{1}{1+e^{-a}}$
- Threshold function: $y = \varphi(v) \equiv \begin{cases} 1 & v > 0 \\ -1 & v \leq 0 \end{cases}$

The activation function usually depends on the tasks one wishes to perform with the neural network. Thus it is closely related to the learning algorithm.

2.4.2.3 Learning algorithm

So how does neural network solve a problem? The typical answer is that the network is "trained" to some existing input-output behaviour of one wishes to model (supervised learning). The weights are adjusted by some learning algorithms so that the network models the generalised behaviour suggested by the examples given. ANNs can use many learning paradigms, but given the intent of this thesis, the most typical supervised parameter learning algorithms are the focus of the next Subsections.

2.4.3 Back-propagation

Back-propagation [92], developed by Rumelhart and McLennan, it is a computational implementation of the *generalised Delta rule*[9]. The algorithm is based on gradient descent, and is certainly the most popular algorithm for the supervised parameter learning in multi-layer perceptrons. Here is a summary of this algorithm:

To start, weights of all neurons are initialised to some random number between 0 and 1, and then the following steps are repeated:

1) Pass the training data $(\boldsymbol{x}(n), \boldsymbol{d}(n))$ to the network. The input vector ***x(n)*** applied to the input layer of sensory nodes and the desired response vector ***d(n)*** are presented to be the output layer of computation nodes at iteration *n*. (from this step onwards, the notation *(n)* indicates "at iteration n", *(n-1)* indicated "at previous iteration before n" and *(n+1)* indicates "at next iteration after n")
2) Compute the output as follows,
   a) The action potential for neuron *j* in layer *l* is computed as,

$$v_j^{(l)}(n) = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n) y_i^{(l-1)}(n)$$

   where $y_i^{(l-1)}(n)$ is the output (function) signal of neuron *i* in the previous layer *l-1* and $w_{ji}^{(l)}(n)$ is the synaptic weight of neuron *j* in layer *l* that is fed from neuron *i* in layer *l-1*.

   b) Assuming the use of a sigmoid function, the output signal of neuron *j* in layer *l* is,

$$y_j^{(l)} = \varphi(v_j(n))$$

   If neuron is in the first hidden layer (i.e. *l=1*), set

[9] It is named for the Delta-rule weight updating method used in early singe-layer perceptrons

$$y_j^{(0)}(n) = x_j(n)$$

where $x_j(n)$ is the *j*th element of the input vector ***x(n)***.

If neuron is in the output layer (i.e. l=L), where L is referred to the depth of the network), set

$$y_j^{(L)}(n) = o_j(n)$$

3) Compare the resulting output with the desired output for the given input to find the "error". The error signal is then computed as,

$$e_j(n) = d_j(n) - o_j(n)$$

where $d_j(n)$ is the *j*th element of the desired response vector ***d(n)***.

4) Compute the "local error" of all neurons according to this "error" found in step 3 in a back-propagating manner from the output layer,

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'\left(v_j^{(L)}(n)\right) & for\ neuron\ j\ in\ output\ layer\ L \\ \varphi_j'\left(v_j^{(l)}(n)\right)\sum_k \delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n) & for\ neuron\ j\ in\ hidden\ layer\ l \end{cases}$$

where the prime in $\varphi_j'(\cdot)$ denotes differentiation with respect to the argument.

5) Adjust the weights for all neurons to lower the "local error",

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n)$$

where η is the learning rate parameter and α is the momentum constant

6) Repeat the process until the "error" reaches a threshold.

Although widely used, back-propagation has several problems. It can often be slow [34]. What is more, as noted earlier in this thesis, back-propagation is a type of parameter learning algorithm. It lacks a technique for structural learning, that is, there is no structure update in the network. Since structure directly affects computational efficiency, this is a definite difficulty.

It is also interesting to note that, in the field of protein structure prediction, the back-propagation neural network algorithm is a commonly used method for predicting the secondary structure of proteins. Whilst popular, this method can be slow to learn so Wood and Hirst compared it with an alternative: the cascade correlation (CC). Using a constructive algorithm, CC achieves predictive accuracies comparable to those obtained by back-propagation, in shorter time [117].

For all of these reasons, CC is the focus of the next Subsection.

2.4.4 Cascade correlation (CC)

CC is a supervised, structural learning algorithm for artificial ANNs proposed by Fahlman [34]. Instead of just adjusting the weights in a fixed network, the algorithm starts with a minimum network and gradually trains and adds hidden nodes one by one, creating a multiple-layer network.

There are two main concepts in this algorithm, "Cascade" and "Correlation". "Cascade" refers to the architecture, which means the hidden nodes are added to the network one at a time in a cascaded

connection pattern: all new nodes take all old nodes as inputs. "Correlation" refers to the learning algorithm, which creates and installs the new hidden units. For each new hidden unit, the algorithm tries to maximise the magnitude of the correlation between the new unit's output and the residual error signal of the network.

Here is the CC learning algorithm,

Begin with a single layer neural network (i.e., only input layer and output layer, there is no hidden layer). Repeat until a measure of convergence is achieved:

1) Train existing output layer connection weights (parameter learning), typically a variation of back-prop, called "quick-prop [33]", to reduce error on the training patterns (supervised learning). It has the following steps,

   a) Calculate the sum squared error of all output nodes for all training patterns using,

$$E = \sum_{p} \frac{1}{2} \sum_{o} (d_{p,o} - y_{p,o})^2$$

   where o ranges over output nodes and p ranges over the training patterns, $d_{p,o}$ is the desired output and $y_{p,o}$ is the observed output of the output node o for a training pattern p.

   b) Residual error of output node o for pattern p is,

$$e_{p,o} = (d_{p,o} - y_{p,o}) \varphi'_{p,o}$$

   where $\varphi'_{p,o}$ is the derivative of an activation function of an output node o for a training pattern p.

   Partial derivative of E with respect to each of the weight between input node or hidden node i and output node o, $w_{i,o}$, is computed in order to minimise E

$$\frac{\partial E}{\partial w_{i,o}} = \sum_{p} e_{p,o} I_{i,p}$$

   where $I_{i,p}$ is the activation potential of an input node or hidden node i for a pattern p.

   c) Perform gradient decent using $\frac{\partial E}{\partial w_{i,o}}$

2) Insert a new hidden layer node, with inputs from all existing inputs and hidden layer nodes in the network. Note that the output of this node is not yet connected to the output layer nodes of the network.

3) Train the input weights of this new node to maximise the absolute correlation of the node's output to the error of the existing network's output on the training pattern (supervised learning) using quick-prop. It is done in the following steps,

   a) Calculate the sum over all output nodes o of the magnitude of the correlation (strictly speaking, covariance) between y, the new node's output, and e_o, the residual output error observed at node o,

$$S = \sum_{o} \left| \sum_{p} (y_p - \bar{y})(e_{p,o} - \bar{e_0}) \right|$$

   where o ranges over output nodes and p ranges over the training patterns, the quantities $\bar{y}$ and $\bar{e_0}$ are averaged over all training patterns.

b) Partial derivative of S with respect to each of the new node's incoming weights, w_i, is computed in order to maximise S,

$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o \left(e_{p,o} - \bar{e_0}\right) \varphi_p^{'} \, I_{i,p}$$

where σ_o is the sign of the correlation between the new node's output and the residue error at output o, $\varphi_p^{'}$ is the derivative of the new node's activation function with respect to its activation potential for training pattern p, and $I_{i,p}$ is the input the new node receives from node i for pattern p.

c) Perform gradient ascent using $\frac{\partial S}{\partial w_i}$ to maximise S.

4) Connect the output of the new node to all output layer nodes and freeze its input weights.

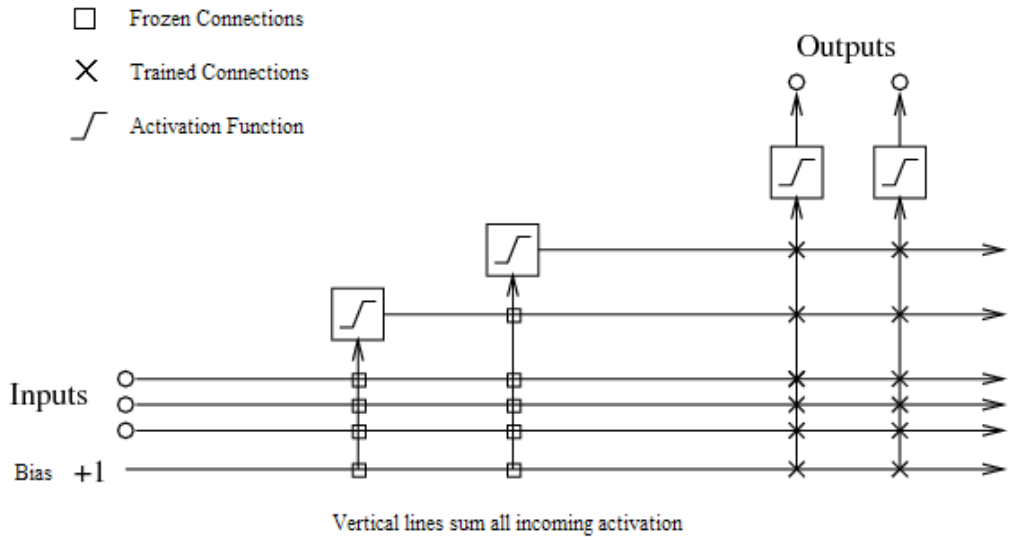

Figure 13: Cascade correlation

Because of the absolute value in the formula for S, a new node only cares about the magnitude of its correlation with the error at a given output, and not about the sign of the correlation. In this case, if a node correlates positively with the error at an output node, it will develop a negative weight to that node, and vice versa. Since a node's weights to different outputs can have mixed sign, it can serve two purposes by developing a positive correlation with the error at one output and negative correlation with the error at another output.

From Figure 13, one can see that CC is a relatively straightforward algorithm, with the exception of the somewhat mysterious element of step 3). One must consider why *maximise* the absolute correlation between a node's output and existing error signal. Upon consideration, it becomes clear that this step will allow one to cancel out that existing error with the new node, through the weight adjustments in step 1). This is a key concept: it suggests that *new neurons should have input weights (and, in effect, connectivity) that maximises the relationship between their output (over the input space) and the error of the existing network they are to be added to, so that this output can be used to cancel out that error*.

As one might also notice, each hidden node's input weights are frozen at the time it is added to the net and only its output weights are trained. Thus leads to the creation of very powerful "feature detectors" and available for producing outputs and more complex features. One could think that each hidden node solves its own part of the network error and will never solve the rest. This makes the network very sensitive to errors: whenever there's an error, a new hidden node is created trying to "cover" the error.

At first, CC may not seem to meet the discrete-optimisation criteria for structural learning discussed before in Subsection 2.1.4. However, note that input layer node weights are often adjusted to values near zero in step 3) of the algorithm. In effect, this "turns off" the connection between a hidden layer node and a particular input (or another hidden layer node). Moreover, rather than using a single candidate node it is possible for CC to employ a population of randomly-initialised nodes in step 2) and 3), culling this population to one node in step 4). If these nodes are initialised with less-than-full input connectivity, the discrete optimisation of structural learning is clear.

2.4.5 Receptive fields

At this point of the exposition, it is useful to introduce the concept of "receptive fields". As a general concept, a node's receptive field is the area of some abstract input space to which that node responds. In this abstract space, nodes with centres near the current inputs are involved in output, and the area around the nodes centre which gives non-zero output is that node's "receptive field". In terms of CC, the goal is to create a map of receptive fields that allow adequate responses to the range of inputs. By correlating new nodes to existing error, their receptive fields are aligned with the area that is causing the most error. Adjustment of this particular node will correct that error.

2.5 **Information theory, entropy and mutual information**

This thesis is mainly inspired by CC's powerful "feature detectors" and structural learning ability. However, the core component of CC, "encoding" network error to a hidden node, leads one to think about Shannon's work. Could that provide more theoretically well-founded inspiration? This section gives a brief insight of Shannon's information theory and especially, mutual information.

The section is structured as follows. Subsection 2.5.1 gives an introduction on information theory and its motivation. Subsection 2.5.2 shows the measurement of information content, entropy, and most importantly, mutual information. Subsection 2.5.3 focuses on the measurement of noisy channel capacity. Subsection 2.5.4 provides recent applications of mutual information.

2.5.1 Information theory

When communicating over an imperfect channel, the message sent is often not identical to the message received at the other end. It is desirable to minimise the probability of these errors. The typical method is to employ an *encoder* and a *decoder*, to compress the messages, and possibly include error correcting information.

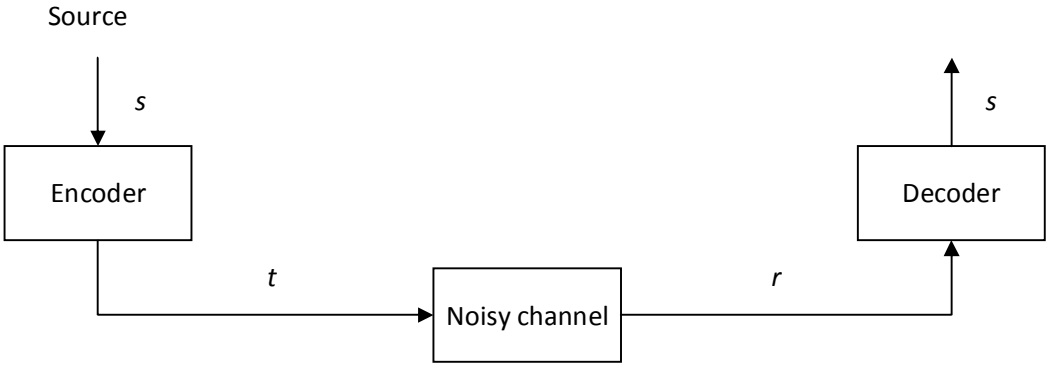

Figure 14: Transmitting message through noisy channel with encoder and decoder

An encoder is a device to change a signal or data into a code. There are two kinds of encoding. One approach is to compress data for storage, transmission or encryption. So after encoding ideally the data should be of a smaller size. The other approach is completely the opposite. It is to add redundancy code to the data for encryption and error-checking after transmission purposes. Thus the data size is larger after encoding. A decoder, on the other hand, is a device that coverts encoded data to its original form.

As shown in Figure 14, a message *s* is first passed through an encoder and the encoder encodes *s* to transmitted message *t* with different size and added redundancy data. Noise is then added to *t* and it becomes received message *r*. The decoder uses the known redundancy previously added by encoder to decode *r* to its original form, separating from the added noise. If well designed, this approach can turn a noisy channel into a reliable channel. The designer of such systems is concerned with limitations and potentials of the computations involved. Information theory, introduced by Claude Shannon [97] [98], is a discipline in applied mathematics involving the quantification of data with the goal of enabling as much data as possible to be reliably stored on a medium and/or communicated over a channel.

2.5.2 Entropy and mutual information

In order to study information theory, it is necessary to understand how information content is measured. Shannon gave the following,

$$h(x) = \log_2 \frac{1}{P(x)}$$

It is a measurement of the information content of a symbol *x*. *P(x)* is the probability of *x*. Typically, the base of logarithm is 2, so the information content is measured in bits.

Based on information content, Shannon introduced the concept of information *entropy*. The entropy of a discrete symbol space *X* is a measure of the amount of uncertainty one has about which symbol will be chosen. It is defined as the average self-information of a symbol *x* from that symbol space *X*, in bits (base 2 of the logarithm is omitted throughout the rest of the section),

$$H(X) = \sum_{x \in X} p(x) \log\left(\frac{1}{p(x)}\right) = -\sum_{x \in X} p(x) \log\big(p(x)\big)$$

In other words, information content of symbol x times its probability, and summed over all of the symbols from X, is the entropy of symbol space X. It is a measure of average amount of information per symbol has, in bits. $H(X)$ is also called the marginal entropy of X.

It is also worth to know some other information measures, such as,

Joint entropy of two random variables,

$$H(X,Y) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log\big(p(x,y)\big)$$

Conditional entropy of two random variables,

$$H(Y|X) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log\big(p(y|x)\big)$$

This measures the average uncertainty that remains about Y when X is known.

Chain rule and a summary of the above formulae,

$$H(X,Y) = H(Y|X) + H(X) = H(X|Y) + H(Y)$$

In words, this says that the uncertainty of X and Y is the uncertainty of X plus the uncertainty of Y given X, the uncertainty of Y plus the uncertainty of X given Y.

The mutual information of two random variables,

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = \sum_{y\in Y}\sum_{x\in X} p(x|y)p(y)\log\frac{p(x|y)}{p(x)}$$

This is the mutual information between X and Y. It shows the average amount of information that X conveys about Y, and vice versa. In other words, this is a measure of how much, on the average, the probability distribution on X will change if the value of Y is given. Note that the mutual information between X and Y is symmetric, that is,

$$I(Y;X) = I(X;Y)$$

And always non-negative,

$$I(X;Y) \geq 0$$

There is also an interesting interpretation of mutual information $I(X;Y)$ in terms of the Kullback-Leibler divergence.

The Kullback-Leibler divergence, or relative entropy between two probability distributions, is,

$$D_{KL}\big(p(X) \parallel q(X)\big) = \sum_{x\in X} p(x)\log\frac{p(x)}{q(x)}$$

It measures the "distance" from $q(X)$ to $p(X)$ but it is not a true metric due to its not being symmetric. Note that the relative entropy plays an important role in pattern recognition, neural networks and information theory.

In probability, if X and Y are independent then,

$$P(X,Y) = p(X)p(Y)$$

In other words, actual joint distribution is the same as the product of the marginal distributions. So one could take the Kullback-Leibler distance between these two as a measure of the dependence of X and Y,

$$D_{KL}\big(p(X,Y) \parallel p(X)p(Y)\big) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

Thus, the dependence between *X* and *Y* can be computed by calculating the relative entropy of *P(X, Y)*, *P(X)* and *P(Y)*.

One also knows that a joint probability of *X* and *Y*, *p(X,Y),* is the product of marginal probability of *Y, p(Y),* and conditional probability *p(X|Y)*, thus,

$$p(X|Y) = \frac{p(X,Y)}{p(Y)}$$

One could also immediately deduct the following,

$$I(X;Y) = D_{KL}\big(p(X,Y) \parallel p(X)p(Y)\big)$$

In words, the mutual information *I(X;Y)* between *X* and *Y* is equal to the Kullback-Leibler divergence between the joint probability *p(X,Y)*, and the product of marginal distributions *p(X)* and *p(Y)*.

The following graph summarises the relationships,

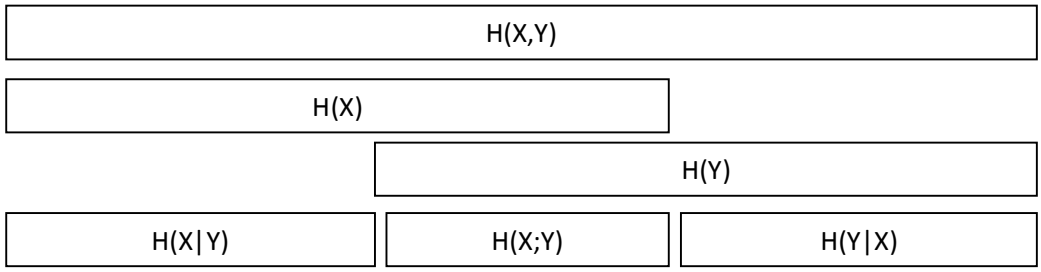

Figure 15: Relationships between entropies

2.5.3 Channel capacity

Shannon's fundamental work in information theory addresses the following concern: given an input signal to a communication channel, how does one maximise the rate of communication, while minimising error in that communication?

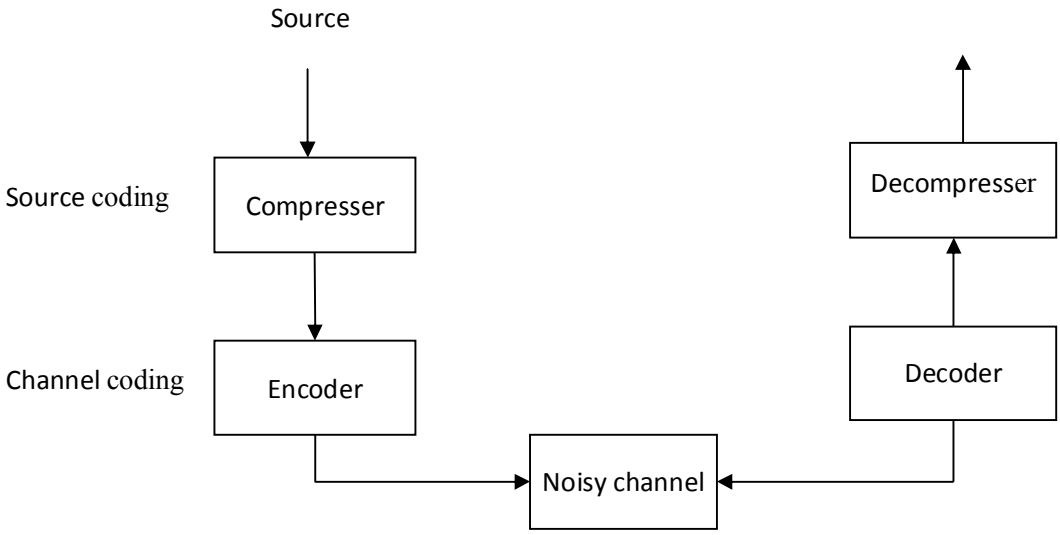

Figure 16: Communicating over a noisy channel, the big picture

Figure 16 shows the big picture when communicating over a noisy channel. Previously it was noted that an encoder is a device to either remove redundancy from source signal or add error-correcting data to the source signal. In this figure, two different devices are assigned to do those tasks. The reason for this is because that Shannon's source coding theorem deals with data compression, a single probability distribution, whereas Shannon's noisy channel coding theorem is to do with two joint dependent probability distributions, input x and output y. However, calculating information entropies for both cases are outside the scope of this thesis, details of these theorems are not discussed.

In order to maximise the rate of communication, one must first find out such noisy channel's maximum possible transfer rate. How is this measured?

The capacity of a noisy channel is defined by,

$$C(Q) = \max_{p_x} I(X;Y)$$

where Q is a noisy channel and the distribution Px that achieves the maximum mutual information is called the optimal input distribution. This result, Shannon's Noisy Channel Coding Theorem, is a central contribution in information theory.

The formula, in other words, is saying that the maximum mutual information between input X and output Y is the maximum amount of error-free information that can be transmitted over the channel per unit time. It is the amount of discrete information that can be reliably transmitted over a channel.

2.5.4 Applications of information theory

The first prominent application in machine learning using information theory was ID3, a decision tree induction algorithm, Later the same author implemented C4.5 [85] and it became one of the most commonly used algorithms in the machine learning and data mining communities.

The last years have seen a surge in highly successful application of information theory to complex systems, including artificial life systems [1] [60] [96] [109]. In the field of machine learning, Friedman et al, also talked about using mutual information for learning the structure of Bayesian network [41]. In the domain of protein structure prediction, an *Extended Compact Genetic Algorithm* (ECGA) has been used to find an alphabet reduction policy and it is guided by fitness function based on mutual information metric [6].

2.6 **Summary of chapter**

The aim of this chapter is to provide necessary background for the primary subject of this thesis.

Section 2.1 provides an introductory understanding of machine learning, by categorising its paradigms in two different ways. The concentration of this thesis is supervised learning, especially classification. After reviewing the most widely-used knowledge representations in the literature, a comparison is drawn, showing that the rule-based learning classifier system has a significant asset in its advanced structural learning via evolutionary computation. Such learning is associated with generalisation, parsimony and explanatory power, which is another contribution the thesis seeks to provide.

Section 2.2 gives an overview of the area in which this thesis is placed, GBML. GA, paradigm of EA and a search technique from EC, is described in detail.

Section 2.3 draws close attention to three types of GBML learning systems, Michigan-style, Pitt-style and iterative rule learning LCSs, based on how GAs behave in each system. Six LCSs have been reviewed along with applications of LCSs in data mining. An important feature of LCS, named default hierarchies, has also been discussed.

Section 2.4 briefly outlines biological neural network, which inspired ANNs. The different types of architecture, activation functions and learning algorithms of ANN are studied. Most of the section draws close attention to the supervised learning algorithms: back-propagation and CC. It is shown that CC is a very sensitive algorithm to the network error due to its powerful, focused "feature detectors". What is more, unlike back-propagation, CC features structural learning as previously mentioned in Subsection 2.1.4.

Section 2.5 gives a brief introduction to Shannon's information theory. "Entropy", the average information content of random variable, and its joint and conditional properties are described. Most importantly, mutual information between two random variables shows the amount of information one coveys about the other. What is more, maximising mutual information between inputs and outputs of a noisy channel (provably) maximises that channel's information carrying capacity. This is a central contribution of information theory and it plays a vital part in the learning system this thesis presents. Some machine learning systems based entropy have been discussed.

With all necessary materials covered, the background chapter ends with this section. The central contribution of this thesis proceeds in the next chapter.

Chapter 3

Mutual information-based learning classifier system (MILCS)

In this chapter, a novel learning classifier system called MILCS [103], is presented. It is a Michigan-style learning classifier system which means each GA individual is a single rule. MILCS was originally inspired by XCS, thus many of its basic components (match strategy, GA operators, etc.) share similar properties. These components along with the novel contributions of MILCS, are described in this Chapter.

The chapter is structured as follows. Section 3.1 compares XCS to CCN and draws analogies in several different ways. Section 3.2 shows the motivation to use mutual information rather correlation as fitness in the proposed system. Section 3.3 provides a detailed overview of the proposed system. Section 3.4 outlines the pseudo-code for re-creating MILCS.

3.1 Comparing XCS and CCN

3.1.1 XCS and CCN: an analogy

Comparing learning classifier systems and neural networks has been a recurring theme in the literature. Mapping a LCS to a back-propagation network has been studied both conceptually [35] and functionally [27]. A concise analogy between the two is also offered by Smith and Cribbs [101] and a LCS/NN mapping was presented and tested. This analogy is related to work presented by Wilson [111] where a perceptron-building GA was shown. More recently neural paradigms have been incorporated into LCSs, with either each individual rule replaced by a neural network [19] [83], or a rule's prediction calculated using neural network [82] [74], or a rule's action is represented using neural network [26]. One might notice that the back-propagation network has often been used when developing these models, either as a substitute or as a supplement for GA.

As described in Subsection 2.4.4, the CCN architecture simultaneously evolves both neural network topology and connection weights. The quick-propagation algorithm for weights update makes it a very fast learning algorithm compared to back-propagation. Perhaps more important is the structural learning aspect of CCN. By starting off with a single layer (or minimum multi-layer) network, and adding one hidden node at a time, it determines its own size and topology. Once a hidden unit is

added, its input weights do not change for the remainder of the training; i.e., the unit becomes a permanent feature detector. This makes CCN very sensitive to any new unit added.

XCS, perhaps the most popular LCS paradigm, has a notable similarity to CCN. These common features are summarised in Table 3.

XCS	CCN
• Rules define a set of generalisations (rules' conditions) over (usually binary) input variables, and map them to outputs.	• Hidden layer neurons parameterise receptive fields over input variables, usually with a threshold function, which effectively define a range of activation
• The XCS conflict resolution scheme uses parameters to mediate between all the rules whose conditions are matched	• Output nodes are parameterised reactions to nodes whose receptive fields are stimulated
• The use of (inverse of) of accuracy (which is related to the inverse of variance of prediction), a second order statistics, as fitness, for the creation of new rules in the GA	• The use of maximisation of absolute correlation, a second order statistics, in its creation of hidden layer nodes

Table 3: Comparison of XCS and CCN

If one imagines an analogy where XCS rules are like hidden layer nodes and condition of a rule is like the receptive field of a node, there is a clear correspondence.

However, it is not an exact correspondence, leading one to ask why a difference exists. Recall that in step 3) of the CCN algorithm (Subsection 2.4.4), correlation to existing error is justified by supervised learning training of the output layer weights, to cancel out that error. However, no such "cancellation" exists in XCS, since a rule's output (action) is not tuned via supervised learning. XCS, like many other LCSs, is a reinforcement learning system.

3.1.2 Supervised versus reinforcement learning

In XCS, rules maintain a reward prediction which is used for conflict resolution. Rules also maintain an accuracy value based on variance of its prediction value. This value is then used in the GA for rule condition determination. However, it also determines the rule's action. The inverse of variance is used for fitness, and reward prediction cannot mediate the value of these outputs. Instead, actions are searched for via the GA, or various covering operations. XCS grows out of the LCS tradition of reinforcement learning. Reinforcement learning is defined by the lack of supervisory feedback that indicates the correct action the system should take. Instead, only "reward" or "punishment" type feedback is available. Thus LCSs generally do not employ supervised update of actions. Some newly developed systems, named XCSF [115] and UCS [8] (which has been reviewed in Subsection 2.3.1.3), both based on XCS, do use supervised learning to train their outputs to reduce errors.

However, in CCN, one can update towards the correct output directly. CCN exploits this supervision in its update of output layer weights, which justifies its use of correlation to existing error in hidden layer weights.

At the beginning of this thesis, it is pointed out that PSP problems, are supervised learning problems. Given that CCN has such similarities with XCS and having many advantages over back-propagation, why not integrate the concepts from CCN into XCS for supervised learning?

3.2 **Mutual information rather than correlation**

In CCN, absolute correlation between the output of the hidden node and the active network error is calculated. The reason for that is the weight adjusted receptive field of the node is aligned with area of input space which is causing the most error in the network, so that the error can be corrected later. In other words, the error is "encoded" in the hidden node in the best possible way. This is clearly related to information theory. Smith and Behzadan [100] state that:

"Correlation tends to find the co-movement of two random variables based on their co-placement above or under their expected values. Hence, it is quite likely that the calculated correlation tends to be zero while in reality a tense interrelationship between the variables exists. This is stated more mathematically by (Freund & Walpole, 1986, p. 452): "if two random variables are uncorrelated they are not necessarily independent".

MI has significant advantages over conventional correlation. MI, in contrast to correlation, tends to calculate the interdependence of two variables by using the conditional probability. MI is able to measure the general dependence of two variables, while correlation can only measure their existing linear relations (Li, 1990; Dionísio, Mendes, & Menezes, 2003). Since MI deals with probabilities, it is applicable on both symbolic and numerical sequences (Li, 1990), in contrast to correlation, which is based on simple algebraic operations, restricting its application to numerical random events. MI also shows more sensitivity, making it desirable when high precision is required. Moreover, since it is defined by a logarithmic term, it is convenient for many mathematical manipulations."

Therefore, correlation does not have a firm theoretical foundation whereas information theory has wide array of ramifications into different fields, and is backed by a more powerful mathematical arsenal. It is preferred to have such theoretical foundation in MILCS.

3.2.1 Data compression and noisy channel coding: an analogy

Shannon's coding theorem on lossless data compression is to maximise compression ratio for lossless representation of data so that the compressed data has the same information content as the source data. However, his work on noisy channel coding theorem is also to do with maximising the rate of communication so that the signal received is identical (i.e. same information content) to the signal sent. The analogy between lossless data compression and noisy channel coding is clear.

Shannon showed that the zero-error maximum communication rate for a channel is given by maximising the mutual information between the channel's input and output. Maximisation of mutual information is accomplished by manipulation of the probabilities of various inputs to the channel, or through the manipulation of the coding of input signals. Since coding is similar to compression, the channel capacity can be seen as the maximisation of compression ratio for lossless representation of data.

Imagine that the existing error in step 3) of the CCN procedure is an input signal to be encoded. In this case, the hidden layer node plays the role of an encoder for signal. The maximisation of the

mutual information between network error and the node's receptive field enables the best encoding (closest to lossless) of the network error to the node's receptive field, so that the error is optimally "covered" (encoded) in the system.

3.2.2 Sensor placement and channel communication: another analogy

Another useful analogy is sensor placement. Imagine that the task of placing temperature sensors in a large space with a temperature distribution. The sensors have known receptive fields, based on their positioning. Ultimately the task is to place the sensors so that the maximum amount of information about temperature distribution is "communicated" through the sensors. This is analogical to channel communication in some sense, if one considers the temperature distribution as the input signal (the existing network error) and the sensors as communication channel. The similarity to the placement of conditions of classifiers or the receptive fields of nodes can be seen. Therefore, in accordance with Shannon's noisy channel coding theorem, it is theoretically sound to place the available sensors such that they maximise mutual information between the temperature distribution and the receptive fields distributions.

In summary, a firm theoretical foundation for using the mutual information has been found to position the receptive fields of hidden nodes (rules' conditions) so that each of them covers the existing error of the rest of the system. By adjusting the output layer weights of the hidden node (or rules' actions) through supervised learning the existing error can be cancelled. Essentially, this process builds up a hierarchical structure of rules set by creating exception rules that cover the error (See Subsection 2.3.5).

3.2.3 The role of mutual information

In MILCS, for the reasons motivated above, mutual information between a rule's "output" and the existing system's error is used as the fitness for each rule so that higher fitness encourages rules to be selected by the GA. Thus their offspring (which captures a generalisation of their parents) "cover" the existing system error. As this fitness measure is the core of the system, it is necessary to explore this first before going any further.

The fitness is the mutual information between the system's error (over the input messages presented to the LCS) and a rule's response (over the input messages). Thus let X be the distribution of the error in the existing system (a function of the inputs), and Y be the distribution of a rule's responses (a function of the inputs that gives a binary value, depending on whether a rule matches the given input or not),

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x|y)p(y) \log \frac{p(x|y)}{p(x)}$$

It is useful to examine the elements of this expression, in terms that are common to LCSs:

- Accuracy term: $p(x|y)$. This term associates the relevance of the rule's responses to existing error.
- Generalisation term: $p(y)$. It is the distribution of the rule's responses. This term associates with how often the rule matches, or how often the rule does not match.

- Specificity term: $\frac{p(x|y)}{p(x)}$. This term compares the rule's existing error to the overall system error.

Thus, the mutual information expression offers a balance of accuracy, generalisation, and specificity, in an optimal fashion indicated by Shannon's theorems.

Let one assume that the distribution of *X* has two values, "there is an error" and "there is no error". Similarly, the distribution of *Y* also has two values, "matched" and "not matched". So there exist four combinations, which are,

- there is no error and the rule does not match the input
- there is no error and the rule matches the input
- there is error and the rule does not match the input
- there is error and the rule matches the input

The mutual information of these two distributions is the sum of all four terms. However, if there exists more than two payoff levels (thus, more than two actions), the distribution of *X* would have a corresponding number of values. This is because the extra payoff levels refer to different levels of "error". Therefore, there will be more than four terms in the MI expression. Note that the distribution of Y always contains two values because a classifier either "matches" or "does not match".

However, if one recalls in CCN that when a hidden node is added to a network it is only connected to the input nodes and existing hidden nodes. This hidden node is not fully connected to the network because its output has not connected to the output nodes yet. This prevents the network error being affected by the newly added hidden node. When this applies to LCSs, however, it means the GA selected rules, which are to generate the offspring rules to fix the error, should not be connected to the system. How does one achieve such with MILCS? The answer is simple; each rule has to be temporally removed from the system one-by-one. Once it is removed, its mutual information between system error and its generalisation can then be calculated before it is added back into the system. The fitness of the rule is the MI between its matching, and the error of the rest of the system with that rule removed. Because such evaluation between a rule and all other rules takes place in the fitness calculation, the DHs (See Subsection 2.3.5.) are encouraged.

3.2.4 The need for two actions per rule

In a conventional LCS, each classifier has a condition and an action. Once the condition is matched then its action can fire.

If one recalls the CCN algorithm in Subsection 2.4.4, it is the *absolute* correlation being calculated. A new node only cares about the *magnitude* of its correlation with the error at a given output, and not about the sign of the correlation. In this case, if a node correlates positively with the error at an output node, it will develop a negative weight to that node, and vice versa. As mutual information is known to be non-negative, rules, like nodes, also need to have a "positive" and a "negative" output. Therefore, to conform to the CCN analogy, and articulate all the terms in the sums of the mentioned four combinations, two actions are given to each MILCS rule: one for when the rule matches, and one for when the rule does not match. Both are updated via simple supervised learning. Similarly, there also exist two predictions for each corresponding action.

Covering operation is an important mechanism in XCS and UCS although it is needed at the beginning of a run when alternative rules are not yet available in the population. It allows the test of

a hypothesis of a condition-action mapping and a way of escaping if the system is stuff in a loop [113]. However, MILCS does not require a coving operation because the entire input space is covered by the matched-not-matched feature of MILCS. Because of this, initialling population at the beginning of a run is a must for MILCS. The initialisation size will be highly depended on the test problem.

3.3 **Description of MILCS**

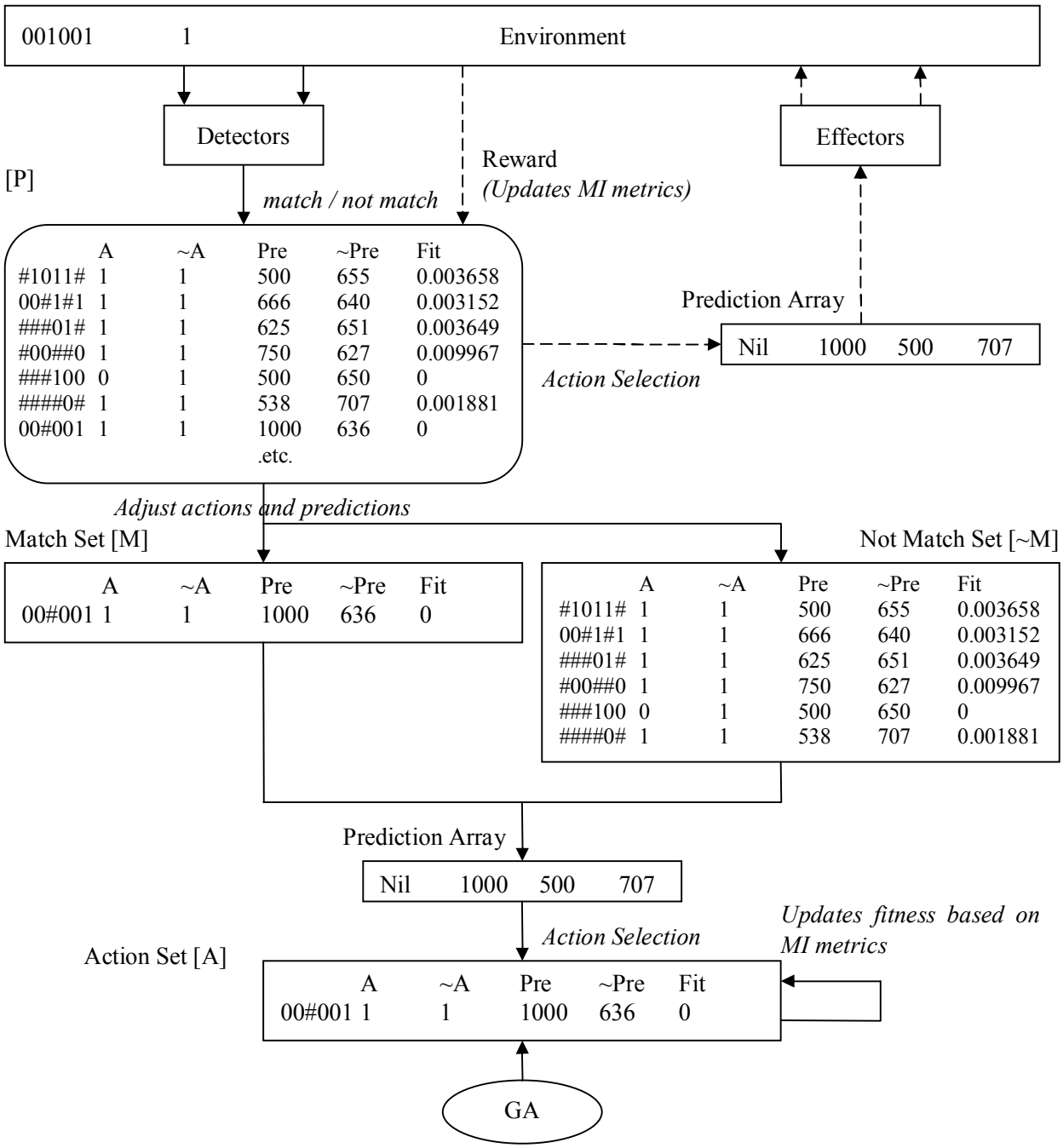

Figure 17: Work cycle of MILCS

Note: dotted arrows indicate iterative process of mutual information metrics update

- Rule sets
  - [P] Population set, contains the classifier population
  - [M] Match set, contains the classifiers that match the input training case
  - [~M] Not-matched set, contains the classifiers that do not match the input training case
  - [A] Action set, contains the classifiers from match set [M] or not-matched set [~M] advocating the chosen action
- Parameters for classifier
  - *Pre* Expected reward of the classifier if it classifies correctly an example with matched action
  - *~Pre* Expected reward of the classifier if it classifies correctly an example with not matched action
  - *MI* Metrics for calculating the fitness
  - *F* Mutual information fitness
  - *exp* Experience parameter of the classifier, which is increased by one each time the classifier's fitness is updated
  - *num* Numerosity of the classifier, number of copies of this classifier in the population
  - *mat* Maturity of the classifier, number of examples the rule has exposed to
  - *freq* Frequency of the classifier fires in the last del_{range} number of iterations
  - *Ts* Last iteration time when it participated in the tournament selection GA
  - *noOfWins* Number of times its action has successfully fired
- Parameters for MILCS
  - N Population size
  - N_I Initial number of randomly generated classifiers in [P]
  - θ_{GA} Do a non-panmictic GA in this [A] if the average number of time-steps since the last GA is greater than θ
  - θ_{Sub} is the subsumption threshold. The experience of a classifier must be greater than this value in order to be able to subsume another classifier
  - χ Probability of crossover per invocation of the GA
  - μ Probability of mutation per allele in an offspring. Mutation takes 0, 1, # equiprobably into one of the other allowed alleles
  - $P_{\#}$ Probability of a # at an allele position in the condition of a classifier created through covering, and in the conditions of classifiers in the initial randomly generated population
  - P_I, F_I Prediction and fitness assigned to each classifier in the initial population
  - α_P, α_F Average smoothing factors for prediction and fitness update
  - mat_{act} Maturity required for a classifier to participate in the action selection process
  - mat_{sub} Maturity required for a classifier to participate in the subsumption process

- mat_{del} Maturity required before a classifier can be deleted
- del_{range} Deletion window range
- $freq_{del}$ Lowest *freq* of a classifier before it is deleted if batch rule deletion is triggered
- $freq_{inc}$ $freq_{del}$ is incremented by this value if batch rule deletion is triggered and $freq_{max}$ is not reached
- $freq_{max}$ Maximum value of $freq_{del}$
- θ_{perf} Performance threshold for triggering batch deletion
- *tournamentSize*, *selectTolerance* Parameters for tournament selection GA

More parameters are added to MILCS in Section 7.1.

3.3.1 Conventional learning classifier aspects

MILCS inherits most of the basic aspects from conventional LCSs such as XCS. Features like rule representations, population sets, matching algorithm, conflict resolution on action selection, subsumption and GA operators, have a similar resemblance of XCS.

Each rule is formed by a condition string and an action string. The condition string is formed by alphabet {0, 1, #}. The "#" symbol indicates "don't know" so it could be either 0 or 1. The action string is the associated class.

The typical rule representation is,

Condition	Action	Not_Matched_Action
0#01011##10	1	0

This particular has 3 bits of # so the generalisation of this rule (input instances this rule will match) is,

00010110010
00010110110
00010111010
00010111110
01010110010
01010110110
01010111010
01010111110

Any other input instances are the ones it will not match. Note that other representations are possible in MILCS (the representation above is not a vital aspect of the system), but this is the representation used throughout this thesis.

3.3.2 Mutual information update

This is the first step of the MILCS process. As is mentioned earlier, each rule has to be temporarily removed from the system for fitness update before it is added back in, in order to conform to the CCN analogy. However, it is not exactly the case, mutual information fitness is only updated for rules in [A] because this system employs a non-panmictic GA (not over entire population). Therefore only the counters (for probability distribution calculation) which are necessary for mutual information fitness computation are updated in this step.

The following counters are kept for each rule, to mirror the four terms in the MI expression discussed in Subsection 3.2.3.

- e0m0: is incremented if system has no error and the rule does not match the input
- e0m1: is incremented if system has no error and the rule matches the input
- e1m0: is incremented if system has an error and the rule does not match the input
- e1m1: is incremented if system has an error and the rule matches the input

The process starts by initialising the population [P] and removing one classifier from it to form [P']. Each rule's condition in [P'][10] is compared with the detector string. If the bit at every non-# position of a rule matches the corresponding bit of the detector string, it is placed in *match set* [M']. Otherwise, it is placed in *not match set* [~M']. To calculate the system output, the system has to form a *system prediction P(a)* for each action *a* represented in [M'] and [~M']. Recall that each MILCS rule has two actions therefore there exist two predictions, one for the matched action and the other for the not matched action. The *P(a)* values are placed in a *prediction array*. Some of the slots in this array will receive no values, if there is no corresponding action in [M'] or [~M']. The *P(a)*, best "guess" of a payoff to be received if *a* is chosen, however, it is calculated differently from XCS. Whereas XCS and UCS use a fitness-weighted average of the predictions of classifiers advocating *a*, MILCS simply selects the highest prediction of a classifier advocating *a*. The maximum-reward-prediction action is always selected in MILCS. Note that this has significant implications for DHs, discussed in Subsection 6.2.2.4.

However, in MILCS, rules have *mat*, which is maturity, i.e., the number of input instances the rule has exposed to. Only rules with their maturity higher than mat_{act} can participate in the prediction array.

An action is selected deterministically, which means *ai* with the largest prediction is selected. It is then sent to the effectors and an immediate reward *r* may (or may not) be returned by the environment. The *MI* counters of the removed rule are updated based on this output and its match status to the given input before it is added back to [P']. The entire process is repeated on the next rule of [P] until all rules in [P] have been updated.

3.3.3 Supervised learning component

Because the correct $X \Rightarrow Y$ mapping is available in supervised problems, MILCS uses this knowledge to adjust all rules' actions to be consistent with the pattern. However, each rule has

[10] The prime of [P'] indicates the population set with one particular rule removed. Same applies to [M'] and [~M']. However these sets are not shown in Figure 17.

generalisation which could "cover" more than one input instance. Therefore, its actions are adjusted to the majority class. Each rule keeps track of class of the input instances it has exposed to so far and keeps its actions updated to the majority class.

Similarly, each rule's predictions are adjusted to the averaged reward sum of its corresponding actions. A pre-defined average smoothing factor α_P is added to the prediction calculation to eliminate input noise for certain problems, therefore,

$$Pre_a = (1 - \alpha_P)Pre_a + \alpha_P \overline{\sum r_a}$$

where $\overline{\sum r_a}$ is the averaged reward sum of its corresponding action *a*, and $0 \leq \alpha_P \leq 1$

freq, i.e. frequency of firings, of each rule in the set is also updated. Each rule keeps track of the iteration time when it is fired and the system sums up the successful firings in a pre-defined deletion window range del_{range} and computes the *freq*. It is later used in the deletion algorithm.

3.3.4 Fitness update

After the supervised learning actions and predictions update, given the same input signal, [M] and [~M] are formed from [P] (the entire population, with no rules removed). Once again, *P(a)* is formed from "mature" classifiers for each action *a* represented in [M] and [~M]. The action *a* with the highest prediction value is selected and the system forms an *action set* [A] consisting of the classifiers in [M] or [~M] advocating the chosen action.

The fitness of rules in [A] are calculated and updated based on the *MI* metrics. From the four *MI* counters specified earlier, one could find,

No. of input instances = e0m0 + e0m1 + e1m0 + e1m1

No. of matches = e0m1 + e1m1

No. of no-matches = e0m0 + e1m0

No. of errors = e1m0 + e1m1

No. of no-errors = e0m0 + e0m1

Probability of no error in the system is,

$$p(x) = \frac{no.of\ times\ of\ no\ error}{no.of\ inputs} = \frac{e0m0 + e0m1}{e0m0 + e0m1 + e1m0 + e1m1}$$

Probability of no match in the system is,

$$p(y) = \frac{no.of\ times\ \ rule\ does\ not\ match}{no.of\ inputs} = \frac{e0m0 + e1m0}{e0m0 + e0m1 + e1m0 + e1m1}$$

Probability of no error in the system given that the rule does not match is,

$$P(x|y) = \frac{no.of\ times\ of\ no\ error\ and\ rule\ does\ not\ match}{no.of\ times\ rule\ does\ not\ match} = \frac{e0m0}{e0m0 + e1m0}$$

Thus put these terms in the following mutual informal formula,

$$p(x|y)P(y)\log\frac{p(x|y)}{p(x)}$$

It gives the partial fitness value for the term e0m0. Repeat the above with the rest three terms and sum up to give the fitness of this particular rule. Similar to the prediction update, an average smoothing factor α_f is introduced when updating rule's fitness,

$$F = (1 - \alpha_f)F' + \alpha_f F$$

where *F'* is the rule's previous fitness, and $0 \leq \alpha_f \leq 1$.

As previously mention in Subsection 3.2.3, when there is more than two payoff levels, e.g., three payoff levels (thus, three actions), there will be six terms in the *MI* expression.

3.3.5 Discovery component

Recall a process called "subsumption" from XCS and UCS. This process is also used in MILCS. The goal of subsumption is to remove rules whose receptive fields are "covered" by other rules' receptive fields, but with certain criteria such as rule experience and so on. Here is a simple example,

C_1	0#01011##10
C_2	0#010110110

Because C_1 covers all input instances that rule C_2 covers, plus others, C_1 is said to be more "general" than C_2. If C_1 also meets both experience and accuracy requirement (it is at least as accurate as C_2), and both C_1 and $C2$ have the same action. C_2 is deleted and the numerosity of C_1 is incremented. There are two types of subsumption in XCS and UCS. *GA subsumption* is to subsume newly generated offspring rules to its parents and *action set subsumption* is to subsume each rule in [A] to the rest of the rules in [A] one-by-one.

In MILCS, only action set subsumption is applied but it works differently from XCS. In XCS, the most general rule of [A] is searched for and then this particular rule is used to subsume the others in [A] one by one. However, the subsumption in MILCS simply let each rule subsume each other. It is even more powerful than the action set subsumption in XCS in turns of population size reduction. The criteria for action set subsumption in MILCS are from XCS and UCS. It is stricter in some sense. C_1 not only has to be experienced enough and more "general" than C_2 but both rules have to meet the following requirements as well.

- Both C_1 and C_2 have the same *act* and *~act*.
- Both C_1 and C_2 have to be mature enough. In other words, both rules have to be exposed to at least mat_{sub} number of input instances.
- Either C_1 has an equal or higher *Pre* than that of C_2, and C_1 has a higher *Pre* than C_2's *~Pre*, or C_1 has an equal or higher *~Pre* than that of C_2, and C_1 has a higher *~Pre* than C_2's *Pre*

If both rules meet the above requirements then C_2 is said to be "subsumed" by C_1. It is removed from the population and C_1's numerosity is incremented.

Like in XCS, once the action set subsumption is finished, MILCS applies the GA non-panmictically over [A]. To be more precise, GA is not applied over the entire [A]. Instead, only rules of [A] that mature mat_{sub} enough can participate in the GA. This is to ensure GA candidates having stable fitness and unfit rules to be subsumed away. Tournament selection is used and a pair of rules is selected from [A]. Then, the parents are recombined and mutated with probabilities χ and μ

respectively, as borrowed from XCS. Note that a second subsumption algorithm has been added to MILCS and it can found in Section 7.1.

The offspring rules are introduced to the system and no GA subsumption is applied here (new rules are not mature enough). As in XCS, the population size is the sum of numerosity of all rules in the system. If, however, the population size exceeds *N*, a deletion algorithm will start operating. Similar to the prediction array calculation and subsumption, a rule has to be "matured" before it is entitled to deletion. The deletion method has two modes: single deletion and batch deletion.

- Single rule deletion mode

The classifier with the lowest freq below $freq_{del}$ is selected to be deleted. $freq_{del}$ is calculated by $noOfWins / del_{range}$

- Batch deletion

In most cases when the population is reaching its limit, single rule deletion is triggered. However, if the system performance (MILCS monitors its performance in the exploit mode) is greater than $\theta_{perf} \times maximum\ reward$, all rules having their *freq* below $freq_{del}$, and *mat* greater than mat_{del}, are deleted. What is more, $freq_{del}$ is incremented by $freq_{inc}$, each time the batch deletion is triggered, up to $freq_{max}$.

If it happens to be that none of the "mature" rules meet the above criteria then a rule's (randomly chosen) numerosity, *num,* is decremented if it is greater than one. One should consider increasing the maximum population size, *N*, or increasing the $freq_{del}$ (if deletion by acting is used) if all rules' *num* are one and all "mature" rules have high *freq* at some point of an experiment, to avoid population overgrowth.

The offspring rules inherit certain properties from its parents, such as (averaged) fitness, (averaged) *freq*, etc. Once they are inserted to the population, all *MI* metric counters of rules from [A] are reset to 0.

The batch deletion mode is a crude way of compacting the population. If $freq_{del}$ is not carefully set before a run, certain fit classifiers still face the chance of being deleted. It is more likely to happen to newly discovered classifier in the early stage of a run at which point their *freq* can be relatively low. On the other hand, the overall classifiers' *freq* in the population are relatively high towards the end of the run and $freq_{del}$ is raised little by little in order to keep only the fit members in the final population.

However, due to the nature of frequency-based deletion algorithm, the system is likely to suffer from *class imbalance*. Class imbalance problems have been seen as a challenge to learning systems because they tend to be biased towards the majority class and leave a poor generalisation for the minority class. To overcome this problem, a second deletion algorithm is added to MILCS and it is covered in Section 7.1.

3.4 The MILCS Process

Given the above considerations, the details of MILCS are summarised in the following four tables and pseudo-code process:

Properties	Explanation	Initial value
act	Action when matched	Randomly selected
actNotMatched	Action when not matched	Randomly selected
pre	Prediction value when matched	0.0
preNotMatched	Prediction value when it is not matched	0.0
fit	Fitness value	0.01
noOfTrainingCases	Maturity, number of training cases exposed to	0
noOfMatches	Number of times the classifier matches to the input	0
noOfWins	Number of times the classifier successfully fires its action	0
actionCounter	Action counters used for adjusting actions	0 (2x2 array)
rewardSum	Reward sum, used for adjusting predictions	0 (2x2 array)
exp	Number of times the fitness is updated	0
gaIterationTime	The iteration time when last participated in the GA	0
num	Numerosity (number of the same classifier)	1
MI	Mutual Information counters for modelling an empirical probability distribution over the matched/not-matched condition of the removed rule, and the error of the remaining rules. Fitness is calculated from this bivariate probability distribution	0 (2x2 array)
winningFrequency	Frequency of the classifier fires	1

Table 4: Properties of MILCS classifier

Parameters	Explanation
MAX_POP_SIZE	Maximum pop size
THETA_GA	Threshold for running GA to generate new classifiers. Once the average *gaIterationTime* of all classifiers in the action set is >= THETA_GA, GA is triggered
INIT_POP_SIZE	Initial population size
INIT_FIT	Initial fitness
INIT_PRE	Initial prediction
ALPHA_FIT	Average smoothing factor for fitness calculation
ALPHA_PRE	Average smoothing factor for prediction calculation
MATURITY_ACT_SELECT	*noOfTrainingCases* required to participate in action selection
MATURITY_SUB	*noOfTrainingCases* required to participate in subsumption
THETA_SUB	If a classifier's *exp* is > THETA_SUB, it can be a "subsumer"
MATURITY_DEL	*noOfTrainingCases* required for a classifier to be deleted
DEL_RANGE	Iteration range for deletion. Number of iterations the system should look back in the history
THETA_PERFORMANCE	If the system performance accuracy reaches this threshold, and the population size is larger than the maximum size, batch rule deletion is triggered next time
DONT_CARE_PROB	The probability for having #s in the newly generated classifiers
WIN_FREQ	A deletion threshold: if a classifier's *freq* falls below this value, it can be deleted. If the system performance is above THETA_PERFORMANCE, a group of classifiers which fall below this value are deleted
WIN_FREQ_INC	If batch rule deletion is triggered, WIN_FREQ is incremented by this value
MAX_WIN_FREQ	Maximum value of the WIN_FREQ
CROSSOVER_TYPE	One point, two point or uniform crossover
CHI_GA	The probability to do recombination/crossover
MU_GA	Probability of mutating one bit
TOURNAMENT_SIZE	Percentage of action set that takes part in tournament
SELECT_TOLERANCE	The tolerance with which classifier fitness is considered similar

Table 5: MILCS parameters

Rule sets	Explanation
pop	Population set. Population of classifiers
mset	Match set. A sub-set of *pop* which contains classifiers that match the given training case
nmset	Not matched set. A sub-set of *pop* which contains classifiers that do not match the given training case
aset	Action set. A sub-set of *mset* or *nmset* which contains classifiers that has the corresponding actions

Table 6: MILCS rule sets

Syntax	Explanation
initialise (classifierSet *set*, integer *i*)	Initialise the classifierSet *set* with *i* number of randomly generated rules
removeClassifier (classifierSet *set*)	Remove the least active classifier from the classifierSet *set*
addClassifier (classifier *i*, classifierSet *set*)	Add a classifier *i* to the classifierSet *set*
action *act* = actionSelection (classifierSet *set*)	Select action *act* based on *pre* and *preNotMatched* of the classifiers of *set*. Only classifiers with *noOfTrainingCases* > MATURITY_ACT_SELECT are taken into account
reward *r* = doAction (action *act*, trainingCase *t*)	Calculate the reward *r* based on the selected action *act* and the given training case *t*
classifierSet *newSet* = createSet (classifierSet *set*, trainingCase *t*)	Create match set or not matched *newSet* based on the given trainingCase *t* and classifierSet *set*
classifierSet *newSet* = createActionSet (classifierSet *set*, action *act*)	Create action set *newest* based on the selected action *act* and chosen classifierSet *set*
updateMI (classifier *i*, reward *r*, trainingCase *t*)	Update the MI counters of classifier *i* based on its reward *r* and if it matches the trainingCase *t* or not
updateWinningFequency(classifier *i*)	Update the *winningFrequency* of classifier *i* if its *noOfTrainingCases* > MATURITY_DEL
adjustActions (classifier *i*, trainingCase *t*)	Adjust the actions of classifier *i* using supervised learning
updatePredictions (classifier *i*, trainingCase *t*)	Adjust the predictions of classifier *i* using supervised learning
updateFitness (classifier *i*)	Update the *fit* of classifier *i* based on its *MI* metrics
moreGeneral (classifier *i*, classifier *j*)	Returns true if each bit of the condition of classifier *i* is either the same as that bit of the condition of classifier *j*, or a "don't care" symbol #
subsumes (classifier *i*, classifier *j*)	If classifier *i* subsumes classifier *j* then *j* is removed and *num* of classifier *i* is incremented
doGA (classifierSet *set1*, THETA_GA, classifierSet *set2*)	If the average of *gaIterationTime* of classifier *i* of classifier *set1* >= THETA_GA, run a non-panmictic GA on *set1* and two offspring classifiers are added to classifierSet *set2*
resetMI (classifierSet *set*)	Reset the MI array to 0s for all classifiers of *set*

Table 7: MILCS main methods

```
initialise(pop, INIT_POP_SIZE)
for each random trainingCase t:
  for each classifier i from pop
    removeClassifier(i,pop)
    action act = actionSelection(pop)
    reward r = doAction(act,t)
    updateMI(i, r, t)                    //update MI counters
   addClassifier(i,pop)
  for each classifier i from pop
    updateWinningFrequency(i)
    adjustActions(i,t)                   //supervised learning
    updatePredictions(i,t)
    i.noOfTrainingCases++
  classifierSet mset = createSet(pop,t)
  classifierSet nmset = createSet(pop,t)
  action act = actionSelection(pop)
  if act is from mset
    classifierSet aset = createActionSet(mset, act)
  else
    classifierSet aset = createActionSet(nmset, act)
  for each classifier i from aset
    updateFitness(i)                     //update fitness
    i.exp ++
    for each rule j from the aset
      subsumes(i,j) if criterion are met
  doGA(aset, THETA_GA, pop)              //GA
  if pop.size > MAX_POP_SIZE             //batch rule deletion
    if systemPerformance > THETA_PERFORMANCE
      while(removeClassifier(pop)){}
      if WIN_FREQ < MAX_WIN_FREQ
        WIN_FREQ += WIN_FREQ_INC
    else                                 //single rule deletion
        removeClassifier(pop)
  resetMI(aset)
```

Figure 18: Pseudo-code of MILCS process

For a more detailed algorithmic description of MILCS, please refer to the Appendix. The algorithmic description is written in a way that MILCS with multiple actions (multiple payoff levels) can be implemented.

3.5 MILCS vs. XCS

Compared to XCS, MILCS has the following advantages,

1) MILCS deals directly with supervised learning rather than reinforcement learning. It has the ability to adjust the output directly. When applied to supervised learning problems, XCS is not using existing knowledge to improve its output.
2) MILCS does not require the complete "model building" approach of XCS. "Model building" is the way XCS maintains its rule set. It contains both "good" and "bad" rules (as long as their predictions, low or high, are accurate). MILCS is aimed at maintaining only "good" rules.
3) MILCS promises to naturally exploits *default hierarchical* structure [53] [54]. It is encouraged due to evaluation focusing on the comparison of classifiers to all other classifiers in the population takes place in the fitness calculation and deterministic action selection, and the lack of prediction "voting".
4) MILCS is explicit about accuracy and generalisation. MI is used to achieve this, while XCS is achieving it only *implicitly,* through a complex balance of fitness-based accuracy and the non-panmictic GA.
5) MILCS is based on firm information theory, which is believed to show improved performance and avenues for future development.

However, the main drawback of MILCS is its computational cost of mutual information update and subsumption. The cost is proportional to the population size.

3.6 Summary of chapter

This chapter presents the main contribution of this thesis.

Section 3.1 starts by drawing analogy between XCS and CCN. It is known that such comparison between LCSs and ANNs has appeared frequently in the literature. However, main difference lies in the origin of LCS: reinforcement learning. Since CCN also has structural learning ability, the motivation for an integration of the two is clear.

Section 3.2 reveals the weakness of correlation in CCN and uncovers a theoretically well-founded metrics: mutual information. Shannon's theorem shows that maximum mutual information between a system error and a rule's receptive field "encodes" the error to the rule the best thus adjusting the output of the rule through supervised learning can "cancel" the error. Therefore, mutual information is used as the fitness function for MILCS. Additionally, in order to conform to the CCN analogy, each rule is assigned a second action which is fired when it is "negatively stimulated".

Section 3.3 gives a detailed description of MILCS and its four stages of learning processes. Firstly, each rule updates its mutual information metrics. Secondly, each rule adjusts its actions using supervised learning. Thirdly, rules of the action set update their fitness and finally a non-panmictic GA is carried out to generate more offspring classifiers.

Section 3.4 aims to provide all the necessary components in tables and pseudo-code to allow the recreation of MILCS.

Section 3.5 lists the possible advantages of MILCS over XCS and its weakness.

Now this innovative system is built. However, before starting the experiments, it is necessary to introduce a tool developed for visualising classifiers. It will help analysing the explanatory power of the results yielded from different learning systems.

Chapter 4

Explanatory Power

This chapter is structures as follows. Section 4.1 defines what explanatory power is. Section 4.2 reviews the rule compaction and condensation algorithms used in LCSs. Section 4.3 explains the method for visualising classifiers. Section 4.4 introduces a quantitative measure for the visualisation results. Section 4.5 gives a brief but helpful manual of the visualisation tool.

4.1 Definition of explanatory power

While accuracy and computation time are important metrics for machine learning systems, it is also important to consider their *explanatory power* [102]. While this term has no formal definition, it is considered to be the subjective human understandability of the representation of the knowledge learned by the system. As a subjective quality, it is somewhat difficult to present, particularly when the knowledge representation exists in a highly multi-dimensional space. It is also important to avoid using pre-existing knowledge of a problem's structure in evaluating the explanatory power of resulting knowledge representations.

This Chapter presents a novel and reproducible method of visualising explanatory power in two-dimensional figures. The author believes this method is adaptable to many other knowledge representations, and is therefore a unique contribution of this thesis (though not its primary one). Therefore, its details are discussed in this chapter.

4.2 Rule compaction and condensation

Because XCS is dealing with a complete map of input/action space, it suffers from a large population, which is considered to be a cause of poor explanatory power. Wilson was aware of this, and discussed the use of condensation in [113]. It consists of running the system with the mutation and crossover rates set to zero. This suspends the genetic search as no new classifiers conditions can be generated, but allows the classifier selection/deletion dynamics in the GA to continue to operate. The result is a gradual shift in numerosity from less fit/less general classifiers to more fit/more general classifiers. Once a classifier reaches zero numerosity it is removed from the system. The end result is a condensation of the population to its fittest members. Kovacs [61] discovered this method needed a pre-defined delay for the trigger and there is no precise estimate of this delay. This

approach also scales poorly. His second method, subset extraction algorithm, showed a better performance.

Wilson introduced the *Compact Ruleset Algorithm* (CRA) in [115]. This method involves three stages. After first ordering the classifiers based on a selected property, say numerosity or experience. Stage 1 involves finding the smallest subset of classifiers which achieve 100% performance. Stage 2 is the elimination of classifiers which, are added to subset, but do not advance performance. Finally, in Stage 3 classifiers are ordered by the number of inputs matched and processed until all inputs have been matched, at which point the remaining unmatched classifiers are discarded.

On the Wisconsin Breast Cancer dataset problem [115], these procedures produced compact rulesets substantially smaller than the evolved populations, yet performance on new data was nearly unchanged. However, Fu & Davis pointed out this approach requires highly trained general and accurate classifiers, and it cannot be applied to less well-trained classifier systems [42]. This algorithm is also said to have very high time complexity and it must run off-line. Dixon, et al. [29] presented a new ruleset reduction algorithm which was reported to be similarly effective to Wilson's CRA but with considerably more favourable time complexity. Gao et al. [43] also implemented a modified CRA (to reduce time complexity) for their Ensemble Learning Classifier System. A new compaction mechanism based on *closest classifier matching* (CCM) plus condensation was implemented to XCSF [15], where "closest" is defined by the distance measure of each classifier itself. It prevents the generation of holes in the function approximation surface during compaction while condensing the population. An additional greedy compaction algorithm is used to iteratively delete classifiers that overlap with low-error classifiers.

Apart from condensation, the methods discussed above are not feasible for on-line learning and (including condensation) a size/performance tradeoff cannot be avoided. MILCS does not require explicit off-line rule compaction. The lack of "voting" in action selection and the promotion of DHs naturally compacts the population evolved.

4.3 Visualising explanatory power

In an attempt to visualise the relative explanatory power of knowledge representations, the following procedure has been developed. Consider a structural element of the knowledge representation (in particular, a rule). This rule is represented as a circle, where the diameter reflects the element's generalisation. The overlap of the receptive fields (conditions) of these elements are characterised by the overlap of the circles. The colour of the circles will represent their output (actions). If rules do not overlap, the distance between them represents some measures of their difference.

The area size of a circle is defined by:

$$A_i = \begin{cases} numberOf\#s \times Am + 1 & logarithm\ sizing \\ 2^{numberOf\#s} \times Am & actual\ sizing \end{cases}$$

where *numberOf#s* is the number of the "don't care" symbol *#s* in the classifier, Am is a multiplier factor used to adjust the size. There are two formulae to suit different knowledge representations.

If two rules' receptive fields overlap, the overlap area of circles *i* and *j* is defined by:

$$DO_{i,j} = \begin{cases} \frac{1}{Diff1_{i,j}} \times A_i & logarithm\ sizing \\ \frac{1}{2^{Diff1_{i,j}}} \times A_i & actual\ sizing \end{cases}$$

where $Diff1_{i,j}$ is the number of positions of the "don't care" symbol *#s* that exists in *i* but not in *j*. Note that the formula used here must correspond to the circle size calculation.

When two rules' receptive fields do not overlap, the distance of the two circles *i* and *j* is defined by:

$$DD_{i,j} = Diff2_{i,j} \times Dm$$

where $Diff2_{i,j}$ is the number of different bits (which does not include the "don't care" symbol #) between rule *i* and *j*, *Dm* is a multiplier factor used to adjust the distance.

Note that DD_{ij} is the distance shown in Figure 19. The distance between two circles' origins is:

$$DD_{i,j} + r_i + r_j$$

$$r_i = \sqrt{\frac{A_i}{\pi}}$$

where r_i, r_j is the radius of A_i, A_j, and A_i, A_j is the area of circle *i* and *j* respectively.

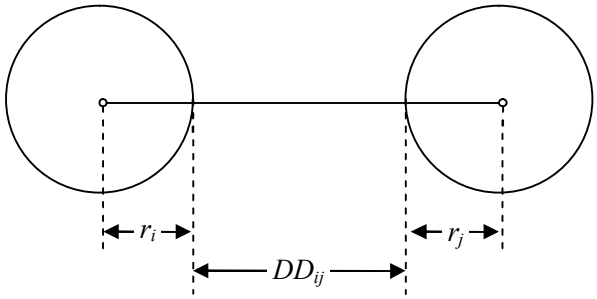

Figure 19: Distance between two circles

Also note that two circles are either *overlapped* or *distant*, but cannot be both. That is:

$$\begin{cases} if\ DO_{i,j} > 0 & then\ DD_{i,j} = 0 \\ if\ DD_{i,j} > 0 & then\ DO_{i,j} = 0 \\ if\ DO_{i,j} = 0 & then\ DD_{i,j} > 0 \\ if\ DD_{i,j} = 0 & then\ DO_{i,j} > 0 \end{cases}$$

The above is always true.

Clearly, presenting a set of rules as overlapping circles cannot, in general, conform to all the implicit constraints of overlap and distance implied by the description above. Any projection of a high-dimensional knowledge representation onto a two-dimensional figure will suffer from similar limitations. Therefore, the following procedure has been used, to evolve a compromise visualisation of knowledge representations.

Imagine that each circle acts as a simulated mass, connected to any other circles via a spring, whose spring force is zero when the desired overlap or distance is obtained. Dampers between the circles are added (to ensure convergence). Figure 20 shows a simple mass-spring-damper system with two circles.

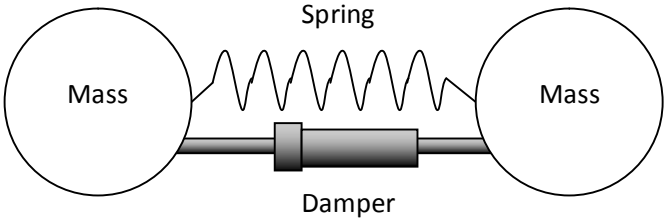

Figure 20: Mass-spring-damper system

Given this, one can use a simple dynamic systems simulation that iterates to an equilibrium, which will represent the nature of the knowledge in the system in a simple, two-dimensional space. However, similar techniques have appeared in the literature. In the field of information visualisation, there is a class of methods called multidimensional scaling to reduce a high-dimensional space to 2-3 dimensions for visualisation purposes. Within, there is a family of graph visualisation technique called spring-embedder algorithms [31].

To model such simulation, the following differential equation is solved using Runge-Kutta method:

$$y' = f(t, y), with\ initial\ condition\ y(t_0) = y_0$$

Suppose y_n is a vector of all circles' coordinates and their velocity at time t_n. The Runge-Kutta formula takes y_n and t_n and calculates an approximation for y_{n+1} at a brief time later, t_n+h. It uses a weighted average of approximated values of $f(t,y)$ at several times within the interval (t_n, t_n+h). The formula is given by:

$$y_{n+1} = y_n + \frac{h}{6}(a + 2b + 2c + d)$$

$$t_{n+1} = t_n + h$$

$$a = f(t_n, y_n)$$

$$b = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}a\right)$$

$$c = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}b\right)$$

$$d = f(t_n + h, y_n + hc)$$

To run the simulation, it starts with y_0 and finds y_1 using the above formula, then y_1 to find y_2 and so on. In a mass-spring-damper system, the spring force is expressed as:

$$F_s = -kx$$

where k is the spring constant and x is displacement (position in a coordinate system). However, in this particular dynamic system, the spring force is a bit more complicated:

$$F_{i,j} = F_{overlap(i,j)} + F_{distance(i,j)} = k\left(DO_{i,j} - AO_{i,j}\right) + \left(-k\left(DD_{i,j} + r_i + r_j - AD_{i,j}\right)\right)$$

where $DO_{i,j}$ is the desired overlap, $AO_{i,j}$ is the actual overlap, $DD_{i,j}$ is the desired distance, and $AD_{i,j}$ is the actual distance between circle i and j, r_i, r_j is the radius of circle i and j respectively.

Figure 21 shows the overlap circle i and j:

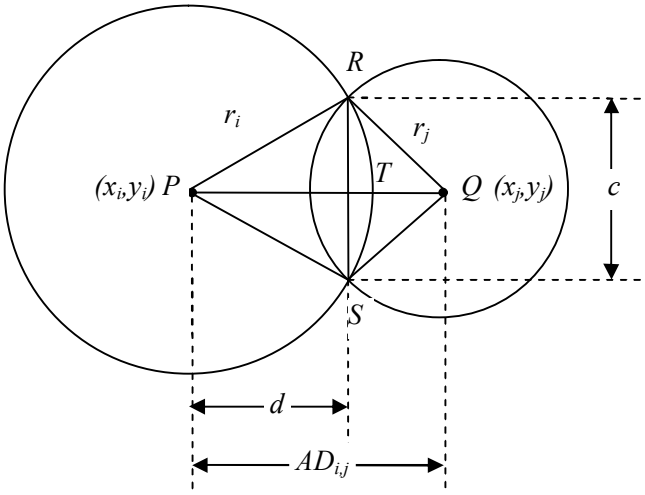

Figure 21: Circle-circle intersection

The actual distance between circle i and j is given by:

$$AD_{i,j} = \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2}$$

where x_i, y_i, x_j, y_j are the coordinates of circle i and j respectively.

From Figure 21, one knows:

$$d^2 + \left(\frac{c}{2}\right)^2 = {r_i}^2$$

$$\left(AD_{i,j} - d\right)^2 + \left(\frac{c}{2}\right)^2 = {r_j}^2$$

Combining the two gives:

$$\left(AD_{i,j} - d\right)^2 + ({r_i}^2 - d^2) = {r_j}^2$$

Multiplying through and rearranging the above equation gives:

$$d = \frac{{AD_{i,j}}^2 - {r_j}^2 + {r_i}^2}{2AD_{i,j}}$$

One knows the area of the sector *(RPS)* can be calculated using:

$$A_{sector(RPS)} = {r_i}^2 \cos^{-1}\left(\frac{d}{r_i}\right)$$

And the area of the segment *(RTS)* is:

$$A_{segment(RTS)} = A_{sector(RPS)} - A_{triangle(RPS)} = {r_i}^2 \cos^{-1}\left(\frac{d}{r_i}\right) - d\sqrt{r_i - d^2}$$

By using the above formula twice, one for each half of the overlapped segment, the actual overlapped area of circle *i* and *j* is:

$$AO_{i,j}$$
$$= {r_i}^2 \cos^{-1}\left(\frac{{AD_{i,j}}^2 + {r_i}^2 - {r_j}^2}{2AD_{i,j}r_i}\right) + {r_j}^2 \cos^{-1}\left(\frac{{AD_{i,j}}^2 + {r_j}^2 - {r_i}^2}{2AD_{i,j}r_j}\right)$$
$$-\frac{1}{2}\sqrt{(-AD_{i,j} + r_i + r_j)(AD_{i,j} + r_i - r_j)(AD_{i,j} - r_i + r_j)\left(AD_{i,j} + r_i + (AD_{i,j} - r_i + r_j)\right)}$$

With all the above calculated, $F_{i,j}$ can be found. However, in the case that when two circles are *desirably distanced* but *currently overlapped*, the following is used to make the repelling force stronger:

$$F_{i,j} = F_{overlap(i,j)} + F_{distance(i,j)} \times Om$$

where *Om* is the overlap multiplier.

To find the *x* component force of $F_{i,j}$:

$$F_{i,j} = \frac{F_{i,j}(x_j - x_i)}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}}$$

And the *y* component force of $F_{i,j}$:

$$Fy_{i,j} = \frac{F_{i,j}(y_j - y_i)}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}}$$

Repeat the above steps to find spring force to circle *i* from all the other circles and sum up its *x* component and *y* component forces. Same procedure is used to find component spring forces for circle *j*, and so on.

The damping force is expressed as:

$$F_d = -cv = -c\dot{x} = -c\frac{dx}{dt}$$

where *c* is the damping coefficient and *v* is the velocity, which is the derivative of *x* with respect to *t*.

Newton's second law gives:

$$\sum F = ma = m\ddot{x} = m\frac{d^2x}{dt^2}$$

where *m* is the mass, *a* is the acceleration of the *m*, which is the derivative of *v* with respect to *t*.

The above spring force, damping force equations and Newton's second law combine to form a second-order differential equation for displacement *x* as a function of time *t*:

$$m\ddot{x} + c\dot{x} + kx = 0$$

Rearrange to get:

$$\ddot{x} = -\frac{c}{m}\dot{x} - \frac{k}{m}x$$

Recall the previously mentioned differential equation $y' = f(t, y)$, if $y_n = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$, then $y_n' = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix}$. With the above equations and an initialised y_0, a dynamic system can be constructed with the Runge-Kutta method.

The visualisation tool uses MATLAB'S ode45 function for the dynamic system simulation. An example of this tool's interface is shown in Figure 22.

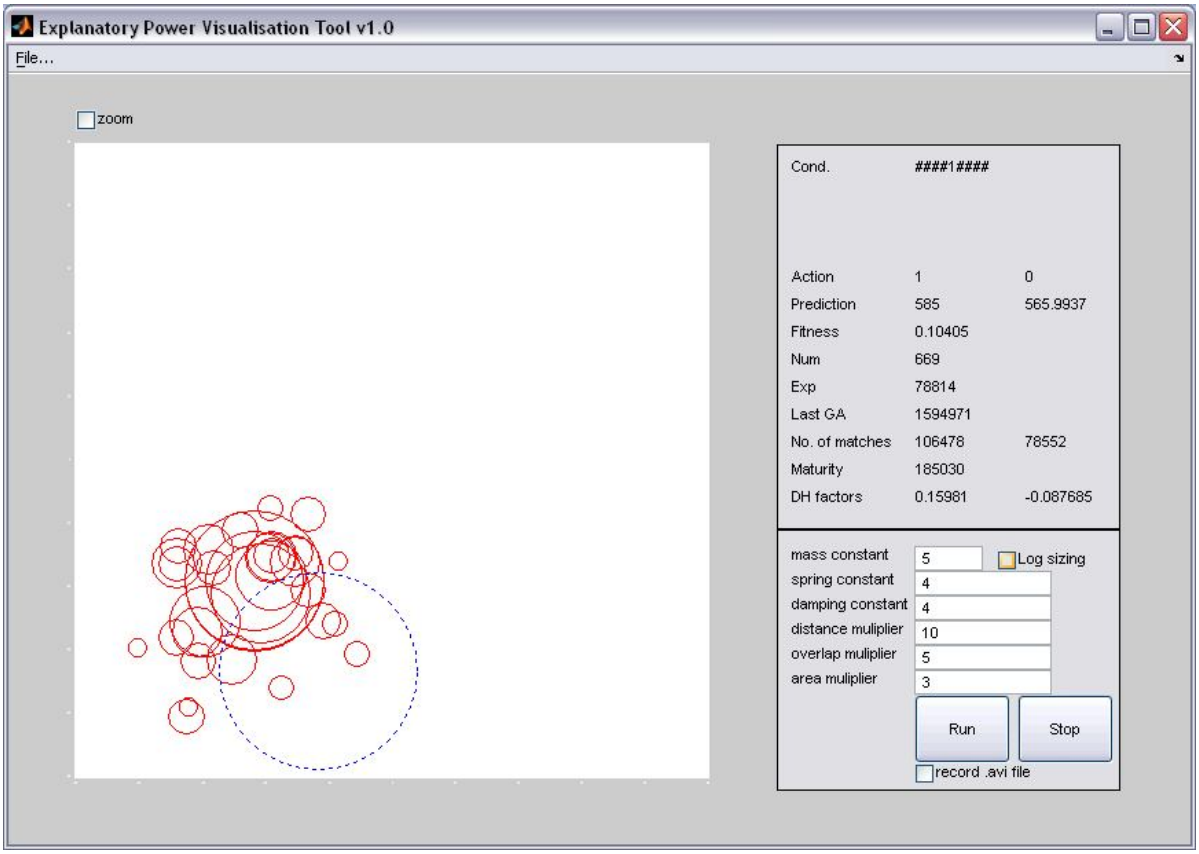

Figure 22: Explanatory power visualisation tool

4.4 Quantitative measurement

When visualising the explanatory power, a better diagram (i.e., one that indicates greater explanatory power) has in order of priority:

1) Fewer circles (regardless of whether they are red/solid-lined or blue/dotted-lined), and
2) Only if a default hierarchical structure exists, fewer clusters, and
3) Preferably an obvious decision surface that separates the red/solid-lined circles from the blue/dotted-lined circles, and
4) Only if the problem has a balanced class, greater symmetry, and

5) Fewer overlapping between circles of the same colour, and
6) If a default hierarchical structure does not exist, fewer clusters

The above list can be quantified using the following methods:

1) Counting the number of circles
2) Using the Quality Threshold (QT) clustering algorithm [49]. Clustering algorithms are used to group similar data into clusters. Unlike the traditional K-means clustering algorithm [75], QT clustering does not require specifying the number of clusters and always returns the same result when run several times.
3) Using a common algorithm that determines linear separability
4) Using various algorithms that measure degrees of symmetry (e.g. reflectional symmetry)
5) Counting the percentage of circles with no overlaps
6) Using the QT clustering algorithm

It is beyond the scope of this thesis to implement all of these different quantifications; however, the counting of circles and the counting of clusters has been implemented and is demonstrated in Section 5.1.5. The author believes these are sufficient measurement other than observers' subjective view.

4.5 **Brief manual for Explanatory Power Visualisation Tool v1.0**

4.5.1 Prepare the input file

In order to visualise explanatory power, the used knowledge representation (rules, for example) has to be interfaced to the visualisation tool in the following way:

- First row of the input file should contain the names of each attribute of the rule, such as condition, action, fitness, etc., and separated by tabs.
- The corresponding attributes of the first rule should be printed to the second row, and separated by tabs.
- The second rule starts from the third row and same applies to rest of the rules with each rule starting from a new row.

4.5.2 Parameters pane

The following settings must be specified,

- Mass constant *m* (the tool uses 5 by default). Greater value gives greater node inertia. The "Log sizing" box should be checked if one wishes to use logarithm sizing for the nodes.
- Spring constant *k* (4 by default). A greater value gives greater spring force between nodes.
- Damping constant *c* (4 by default). The greater the value the sooner all nodes reach equilibrium, but may also trap the system in local minima configurations.
- Distance multiplier *Dm* (10 by default). It is a factor used in the desired distance calculation. It adjusts the distance between two nodes when they are desirably distant. One should increase this value if nodes are clustered together and vice versa.

- Overlap multiplier *Om* (5 by default). It is a factor used in the spring force calculation. Increase this value to increase the repelling force when two nodes are currently overlapped but desirably distant.
- Area multiplier *Am* (10 by default). It is a factor used in the node's receptive field area calculation. One should increase this value if nodes appear too small on the plotting area and vice versa.

4.5.3 Plot pane and information pane

Figure 22 is the visualisation of classifiers with matched and not-matched actions. Classifiers with matched action 0 are red solid nodes and rules with matched action 1 are blue dotted nodes. The areas of these circles are the generality of rules whereas everywhere outside a circle should be considered as its not-matched receptive field.

One can click on a node (on the edge) to view its characteristics displayed on the top right information pane. Note that because each classifier has two actions, there are two values for entries such as actions, predictions, no. of matches and DH factors for each corresponding action. The default hierarchy factors indicate possible hierarchical relationships, which is previously described in Subsection 2.3.5 and some DHs results are discussed in Subsection 6.2.2.4.

Extensive results from the primary experiments using this tool are given throughout the remaining chapters of this thesis.

4.6 **Summary of chapter**

This chapter introduces a new method to visualising classifier for exploring their explanatory power.

Section 4.1 states the importance of explanatory power for machine learning systems.

Section 4.2 gives a brief review of rule compaction and condensation algorithms in the GBML literature.

Section 4.3 shows the dynamic spring-damper system used to visualising rules. The difference between rules indicates their "overlapping" and "distance".

Section 4.4 suggests several quantitative measuring techniques for the graphs generated using the visualisation tool.

Section 4.5 provides a brief manual when using the visualising tool.

With the end of this chapter, it is time to experiment with MILCS with different test problems and benchmark the results against other systems. Additionally, with the help of this visualising tool; one is able to see how they perform in terms of explanatory power.

Chapter 5

Experimental Results

This chapter presents the basic experimental results of the thesis.

This chapter is structured as follows. In Section 5.1, a basic test problem called multiplexer problem is used to compare the scale-up of MILCS to the other LCSs. In Section 5.2, another benchmark problem named even parity is tested on MILCS and other LCSs to further study their performances and explanatory power.

5.1 **Multiplexer problem experiments**

5.1.1 Problem description

The multiplexer problems have been subject of study for long time in learning classifier system research. They are particularly useful in the examining the scale-up of a system, since multiplexer problems of increasing size can be easily specified.

Multiplexing is a term used to refer to a process where multiple signals or digital data streams are combined into one signal over a shared medium. The aim is to share an expensive bandwidth resource. A digital multiplexer is a device to select one of many digital input signals and outputs that into a single line. For example, a 3 multiplexer is shown in Figure 23.

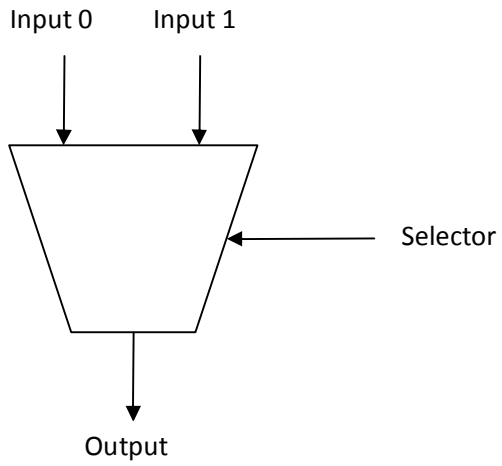

Figure 23: Semantics of a 3 multiplexer

If selector has a value of 0 would connect Input 0 to the output and value of 1 would connect Input 1 to the output. Table 7 contains the truth table for this multiplexer.

Selector	Input 0	Input 1	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 8: Truth table of 3 multiplexer

One can tell from the table that the output value is the selected input value. Since the number of bits in a multiplexer is the number of inputs plus the base 2 log of that number, there are also 6 multiplexers, 11 multiplexers, 20 multiplexers and so on. For any number *N* greater than one there is a $(\log_2 N + N)$ multiplexer. When used in a learning classifier system, selectors are referred as address bits whereas inputs are referred as data bits. For instance, an 11 multiplexer rule,

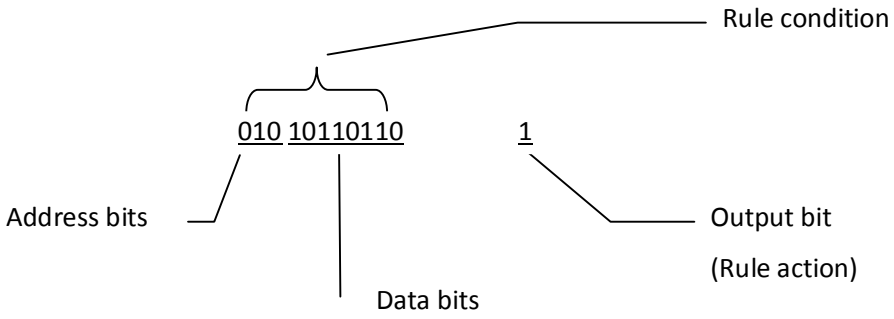

Figure 24: An 11 multiplexer rule explained

As one can see that 3 bits are required to address 8 bits of data, the address bits 010 indicate the third input (data bits) which is 1, thus the output is 1.

The multiplexer task for a learning classifier system is: given a number of training instances with the correct output (such as the one above), the system should eventually evolve a set of rules that gives the correct output by given any input. Side goals of this main goal are to do so quickly, scale up well for larger multiplexers, and yield rule sets that are compact, and have good explanatory power. In order to show the scalability of MILCS with respect to the problem complexity, a test set of 6-bit, 11-bit and 20-bit multiplexer problems are used. The performance of MILCS is compared to other state-of-the-art LCSs. XCS, as described in Subsection 2.3.1.2, is the most widely used LCS and regarded as the main standard. Mentioned in Subsection 2.3.1.3, UCS is an adaption of the standard to supervised learning. Therefore, such comparison can show a good performance overview of MILCS.

5.1.2 Parameter settings

The version of XCS used throughout the rest of the thesis is Martin's C implementation [14] and UCS Java source code was kindly provided by Orriols-Puig [84]. MILCS parameter settings are shown in Table 9 whereas XCS and UCS parameters are shown in Table 10.

Problem size	6	11	20
N	600	1500	2000
N_I	500	500	500
$P_{\#}$	0.33	0.33	0.33
θ_{GA}	25	25	25
θ_{sub}	70	70	70
mat_{act}	32	128	1024
mat_{del}	64	256	2048
mat_{sub}	64	256	2048
del_{range}	128	2048	5012
$freq_{del}$	0.0156	0.0005	0.00039
$freq_{inc}$	0.00001	0.00001	0.00003
$freq_{max}$	0.02	0.02	0.02
α_F	1.0	1.0	1.0
α_P	1.0	1.0	1.0
P_I	0.0	0.0	0.0
F_I	0.01	0.01	0.01
χ	1.0	1.0	1.0
μ	0.01	0.01	0.01
crossoverType	uniform	uniform	uniform
θ_{perf}	1.0	1.0	1.0
tournamentSize	0.4	0.4	0.4
selectTolerance	0.001	0.001	0.001

Table 9: MILCS parameters for the multiplexer problems

System	XCS	UCS
N	400/800/1500	400/800/1500
β	0.2	0.2
α	0.1	0.1
ε_0	0.01	N/A
ν	5	10
γ	0.71	N/A
θ_{GA}	25	25
χ	0.8	0.8
μ	0.04	0.04
θ_{del}	20	20
δ	0.1	0.1
θ_{sub}	20	20
$P_{\#}$	0.33	0.33
P_I	10.0	N/A
ε_I	0.0	N/A
F_I	10.0	0.01
P_{explr}	0	N/A
θ_{mna}	2	2
crossoverType	uniform	uniform
tournamentSize	0.4	0.4
selectTolerance	0.001	N/A
doGASubsumption	Yes	Yes
doActionSetSubsumption	Yes	N/A
acc_0	N/A	0.99
initializePopulation	no	N/A

Table 10: XCS and UCS parameters for the multiplexer problems

5.1.3 Results and analysis

Figure 25, Figure 26 and Figure 27 show results from XCS, UCS and MILCS applied to the 6 multiplexer. In these (and subsequent figures) three lines appear on each plot: the percentage correct over the past 50 training cases (square markers), the difference between reward and predicated reward over the past 50 training cases (triangle markers), and the number of macro-classifiers

(unique classifiers) in the population (circle markers) divided by 1000. Graphs reflect the average of 10 runs.

Note that MILCS converges more rapidly than both XCS and UCS. Both MILCS and UCS produce a smaller final population of unique classifiers than XCS.

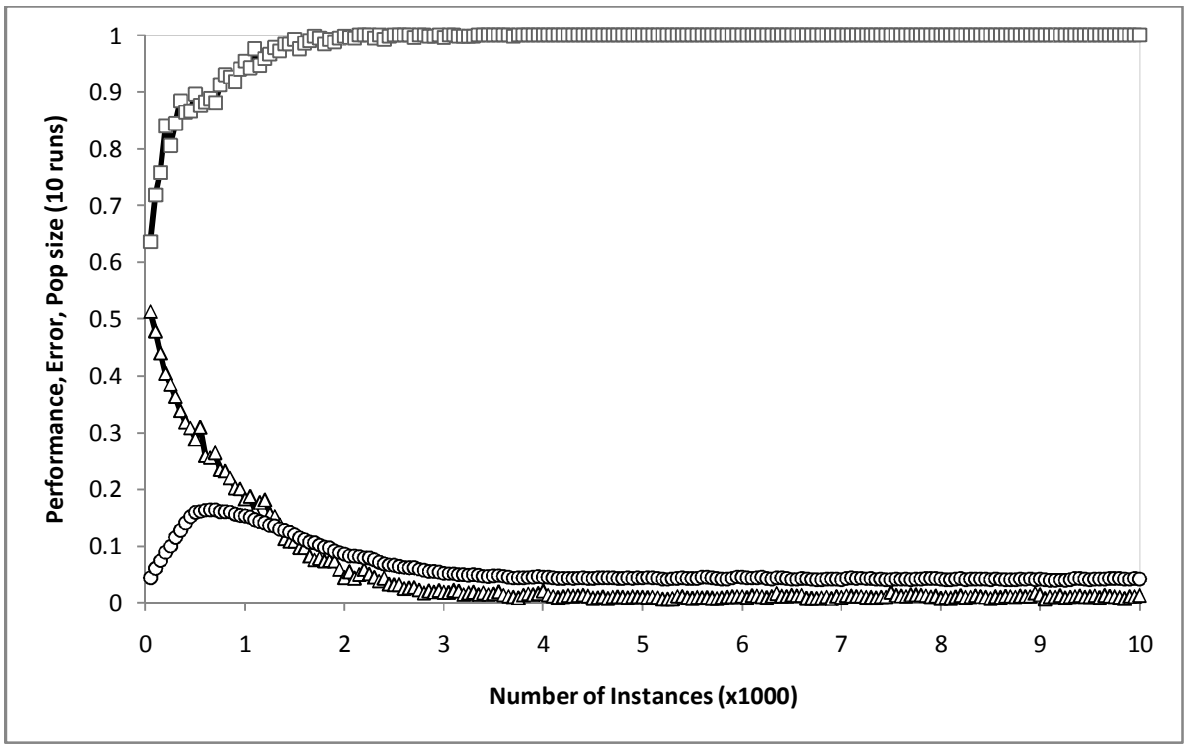

Figure 25: Results from XCS applied to the 6 multiplexer

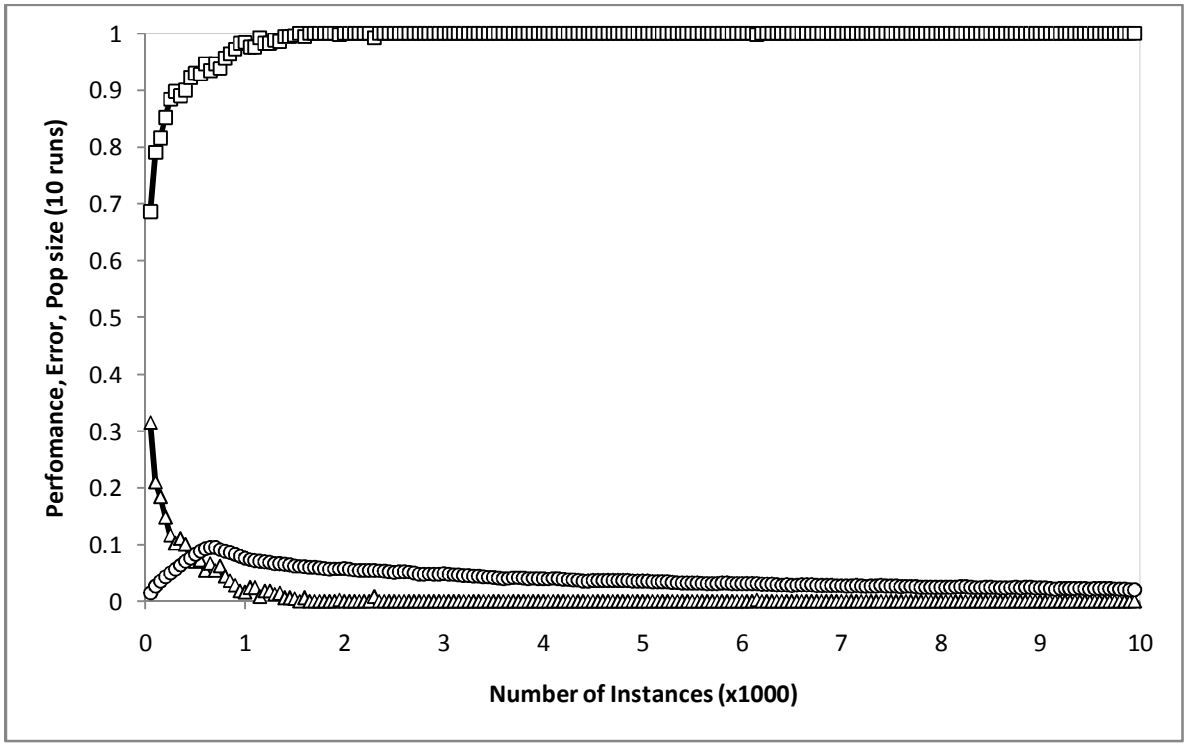

Figure 26: Results from UCS applied to the 6 multiplexer

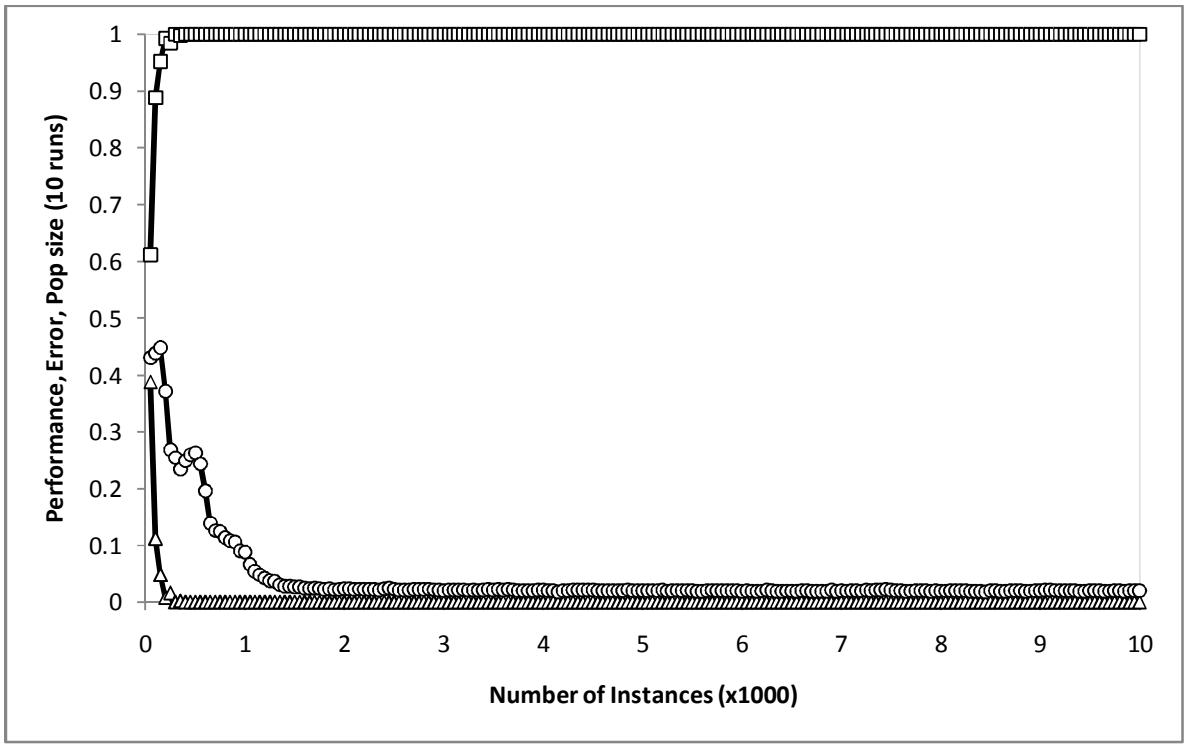

Figure 27: Results from MILCS applied to the 6 multiplexer

Figure 28, Figure 29 and Figure 30 show results from XCS, UCS and MILCS applied to the 11 multiplexer. UCS converges the fastest and to the smallest final population of unique classifiers among all three. While convergence times for both XCS and MILCS are similar, MILCS still manages to converge to a smaller final population.

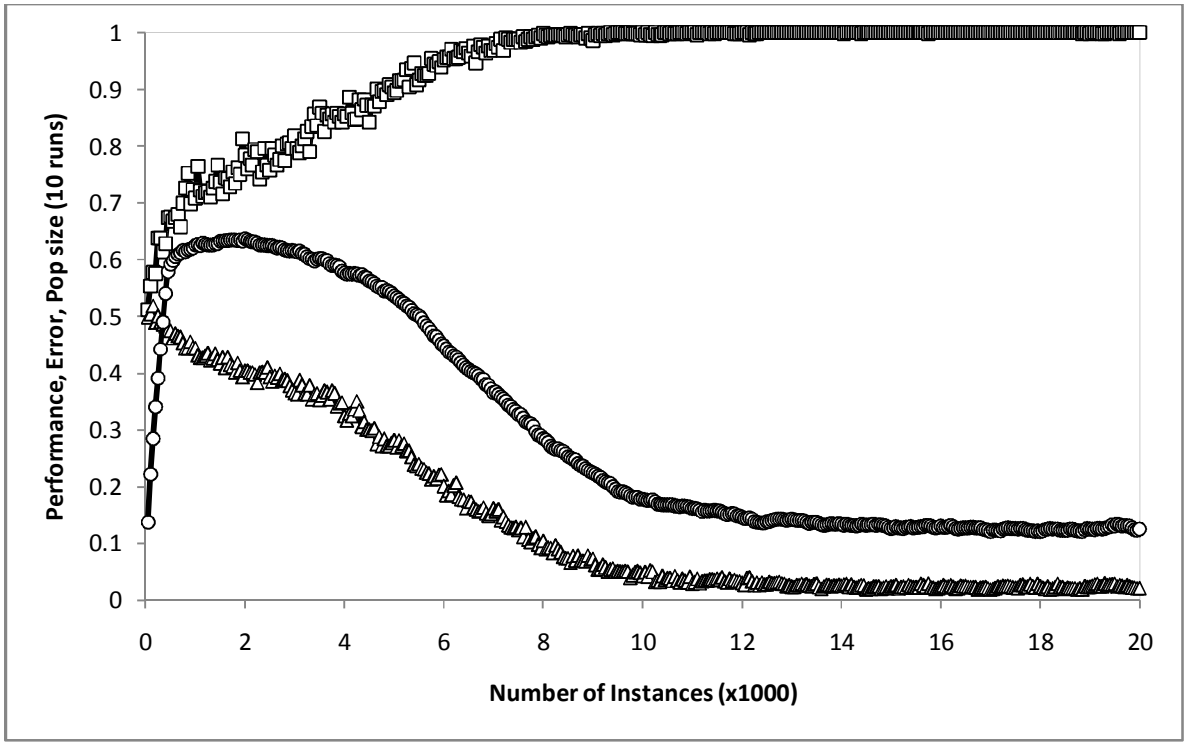

Figure 28: Results from XCS applied to the 11 multiplexer

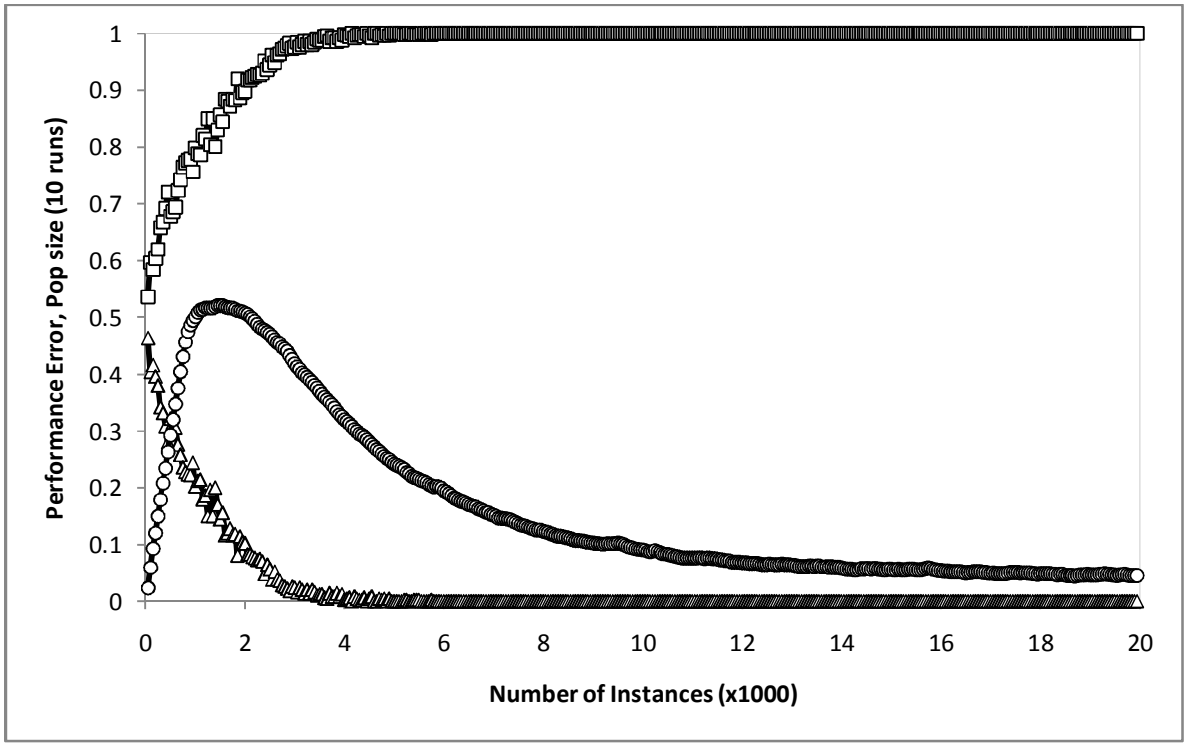

Figure 29: Results from UCS applied to the 11 multiplexer

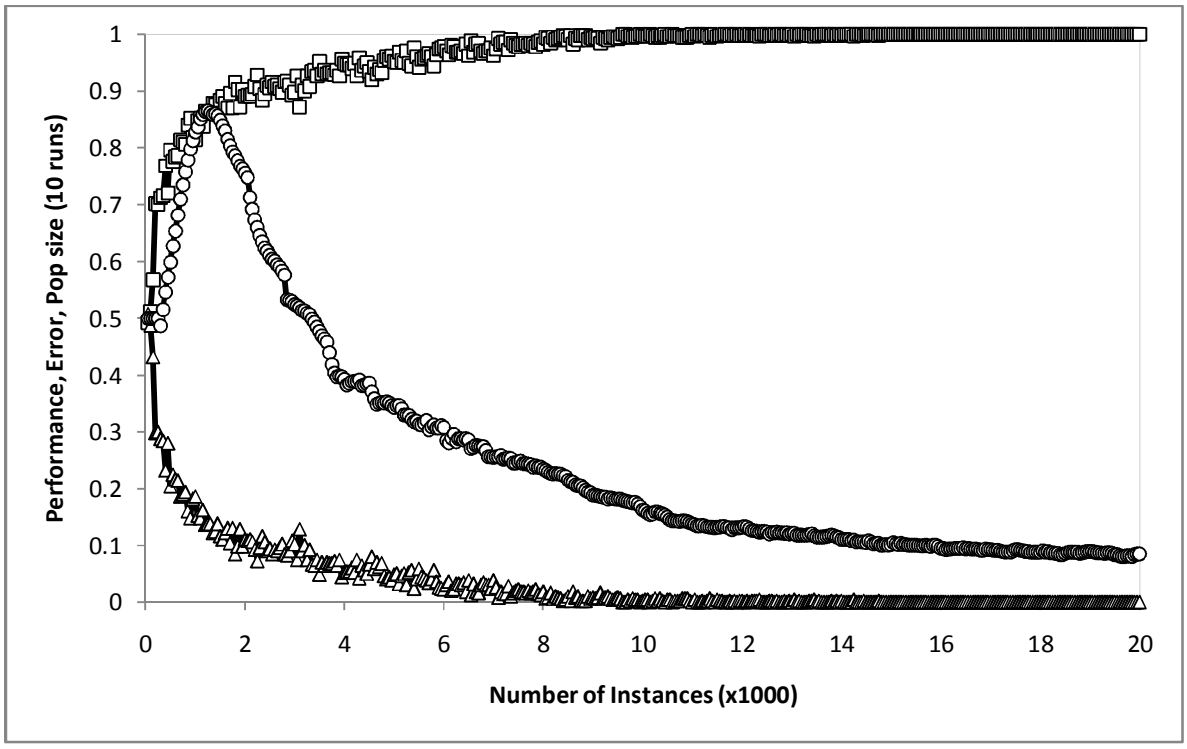

Figure 30: Results from MILCS applied to the 11 multiplexer

Figure 31, Figure 32 and Figure 33 show results from XCS, UCS and MILCS applied to the 20 multiplexer. In this case, both convergence times and final population of unique classifiers of XCS and MILCS are similar. UCS converges the fastest and achieves the smallest final population. The "unstable" population size for MILCS in the first 30,000 instances is mainly caused by its rule deletion algorithm. Each decline corresponds to the mat_{del} and del_{range} parameters.

However, the plots do not reveal a complete story. Complete tests on all possible inputs for each of the multiplexer problems were carried out on all three algorithms. Each system passed this "full test" in each situation, with the exception of XCS applied to the 20 multiplexer, which failed on a small number of cases at the end of some runs portrayed in the average of 10 shown in Figure 31. While this difficulty could not be overcome with the code provided in [14], it was able to reproduce perfect behaviour in approximately 75,000 iterations using [69]. This is consistent with the results on XCS scaling for the multiplexer problems provided in [114].

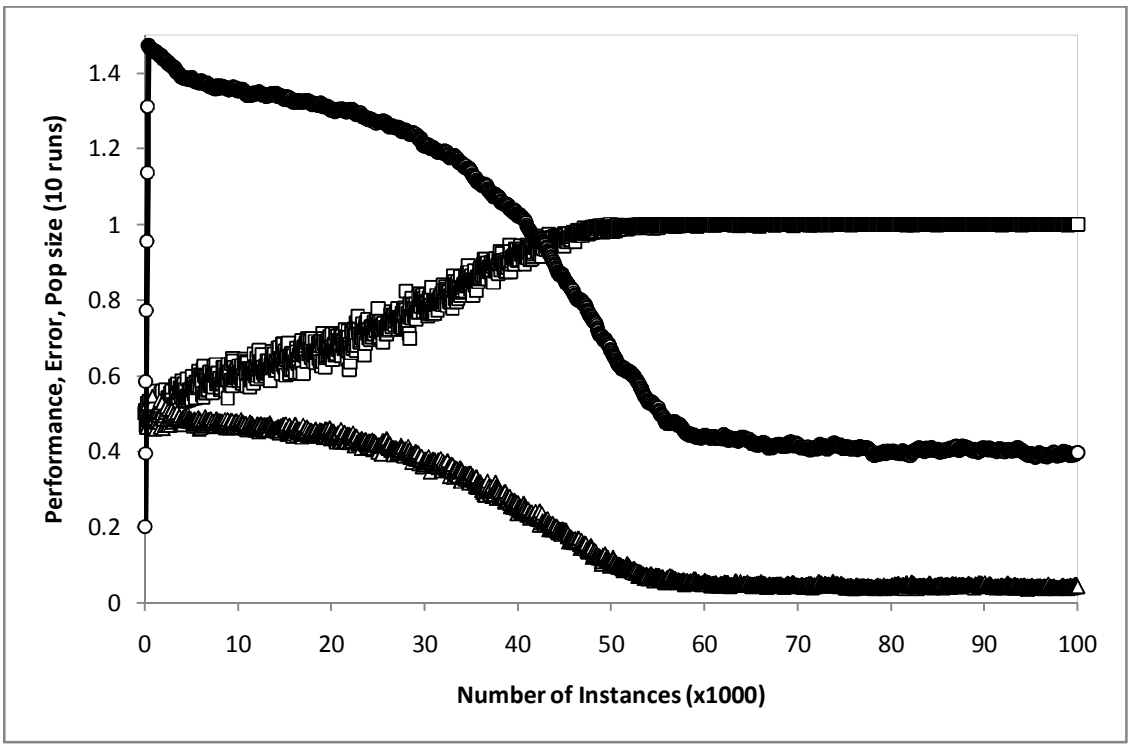

Figure 31: Results from XCS applied to the 20 multiplexer

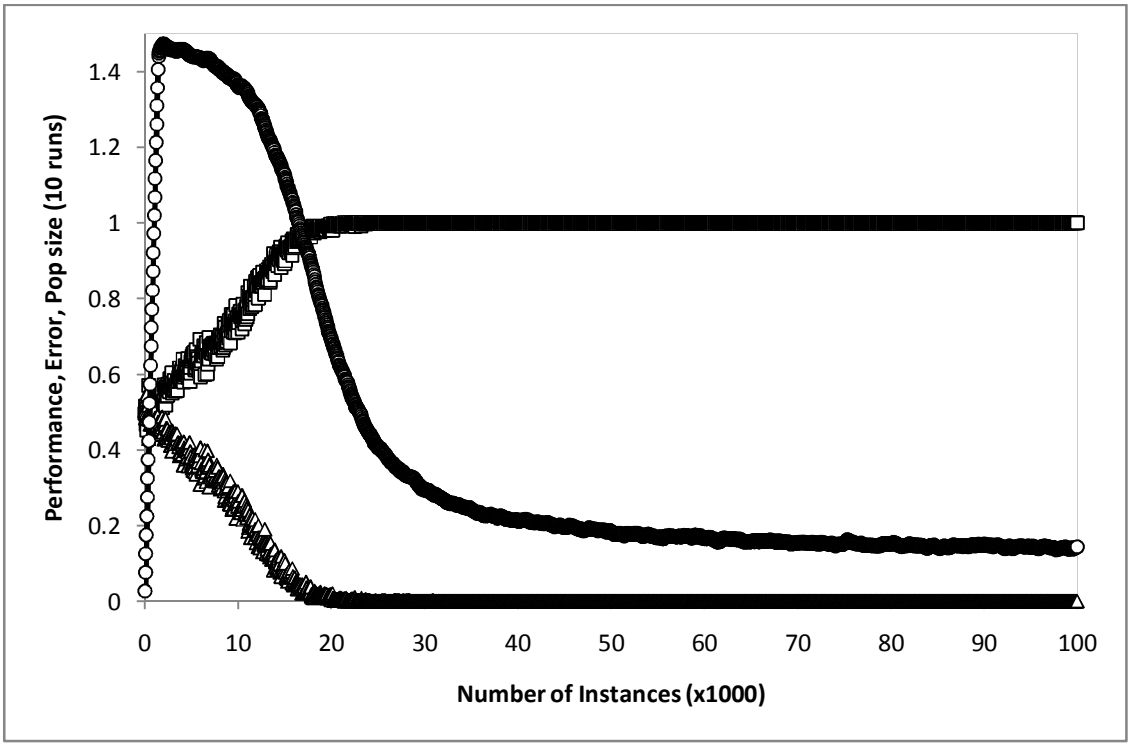

Figure 32: Results from UCS applied to the 20 multiplexer

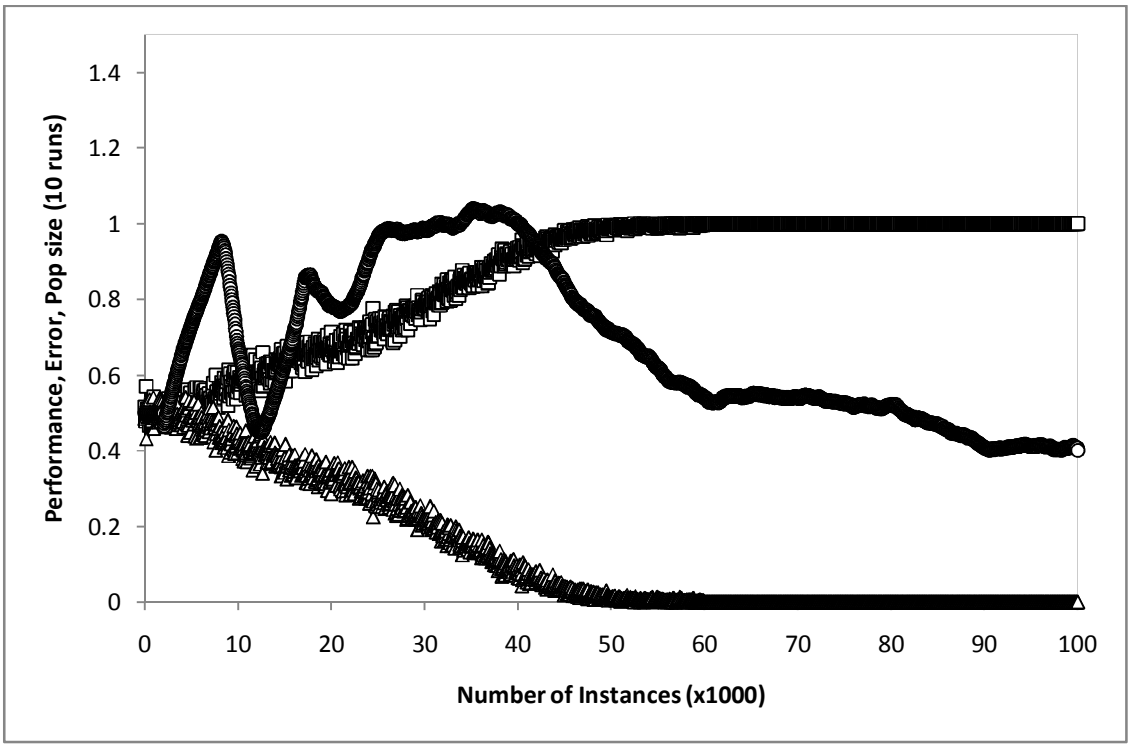

Figure 33: Results from MILCS applied to the 20 multiplexer

While an exhaustive evaluation of the MILCS parameter space has not been done, the scale-up trends for the parameters are shown as follows: N is approximately 30 × *number of optimal rules*[11] (which is calculated as $2^{position\ bits+2}$), mat_{act} is roughly 10 × *previous* mat_{act}, mat_{sub} is 2 × mat_{act}, mat_{del} is normally the same as mat_{sub}, del_{range} is 2 × *previous* del_{range}, $freq_{del}$ is usually a number between 1 and 10 (the desired number of firings at the beginning of an experiment) divided by del_{range}, $freq_{max}$ is the maximum $freq_{del}$ can reach towards the end of an experiment, $freq_{inc}$ is the difference between $freq_{del}$ and $freq_{max}$ divided by the average *number of GA runs per experiment*, and the rest of the parameters could stay the same. Because MILCS uses "maturity" to decide if a classifier can act, subsume or be removed, the settings for these maturity-related parameters require changing if the *number of optimal rules* are growing exponential with the increasing problem sizes (i.e., the multiplexer problem). In other cases, these parameters could most often remain the same. Usually the $freq_{del}$ alone w give MILCS sufficient deletion pressure for a compact final population. However, due to a relatively large amount of "buffer" rules in exist in the final population; it is possible to further reduce its size by experimenting the use of $freq_{inc}$ and $freq_{max}$.

5.1.4 Scalability

Log-log graph is used to show the scalability of all three algorithms. Log-log graphs are widely used to represent data that are expected to be scale-invariant. Such graphs are described by two variables as a scatter graph. The two axes display the logarithm of values of the variables, not the values themselves. In Figure 34, x-axis represents the logarithm of final rule set size and y-axis represents the logarithm of number of evolutions before convergence. The straight lines on these graphs imply polynomial scale up and the slope indicates the order of the polynomial.

The results reveal MILCS scaling up slightly worse than XCS and UCS (see Figure 34),

[11] A solution rule set that is complete, accurate, non-overlapping and minimal [64].

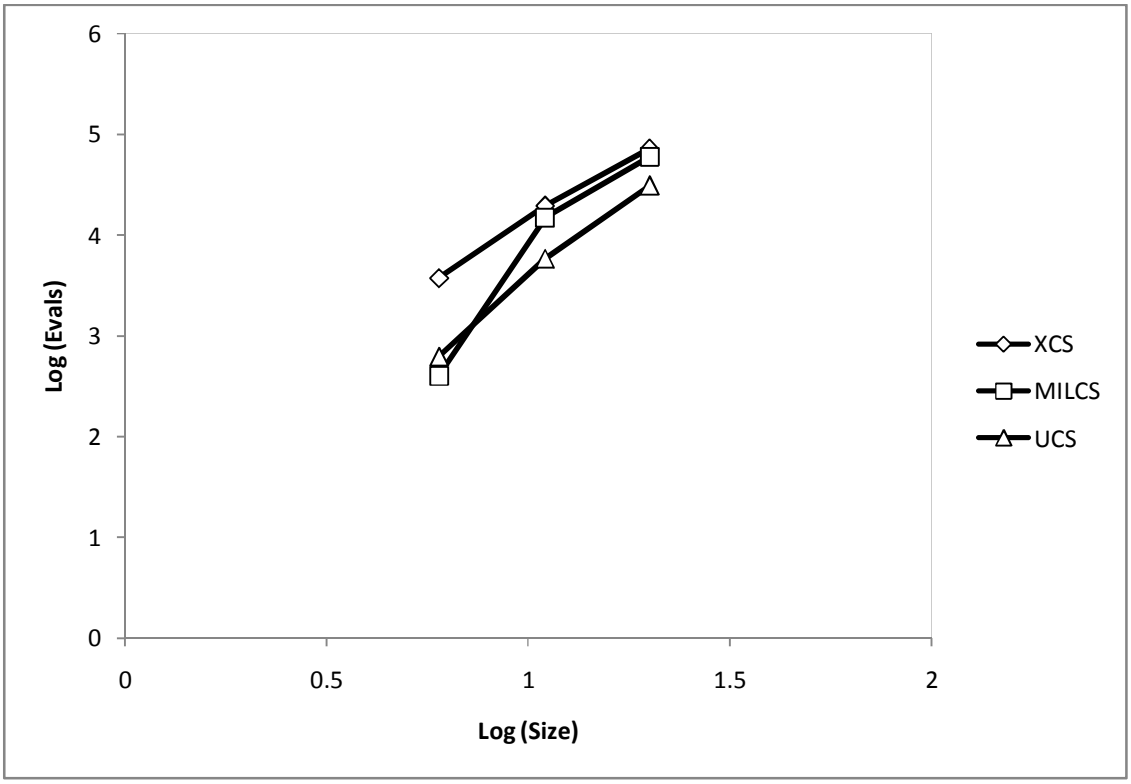

Figure 34: Log-log plot of polynomial scaling of XCS, UCS and MILCS on the multiplexer problems

However, all three are scaling as low-order polynomials. Both XCS and UCS have almost the same scalability whereas there is an apparent difference between them and MILCS. However, it is not of significant order. Moreover, the previous results indicate that MILCS is ending with a substantially smaller set of unique classifiers. Hand examination of rule sets has revealed that almost all of the mature rules at the end of MILCS runs are best-possible-generalised rules for the multiplexers. However, this examination relies on the knowledge of the underlying problem. In the following section, the visualisation is used to show the comparative explanatory power of the final rule sets.

5.1.5 Visualisation of explanatory power

To make a valid comparison that is not built on pre-existing knowledge of the problem at hand, the visualisation will include all elements (rules) that play a role in the output determination of the final system. In XCS exploitation mode (the mode in which final results are evaluated), the following factor is computed:

$$\frac{\sum(prediction \times fitness)}{\sum fitness}$$

Over all matching rules, for all actions, and the action with the highest factor is selected. Therefore, since all the rules participate in action selection, all these rules are included in the visualisation. Note that one could employ additional steps to "prune" out rules that make negligible contributions

to this factor in particular input cases. However, this generally non-trivial step is not included in the XCS or UCS algorithms, therefore it is not included in the comparisons presented below.

However, in MILCS on an "exploitation" trial, only the rules with action-selection maturity above a threshold are employed. The rule, either matched or not matched, with the maximum predicted reward is always selected as the rule that acts. Therefore, only these rules are used in the visualisation.

Figure 35, Figure 36 and Figure 37 show visualisations of the final rule sets from XCS, UCS and MILCS (respectively) applied to the 6 multiplexer. The smaller, final MILCS rule set, and its inclusion of only perfect generalisations is clear. Based on the indicators outlined in Section 4.4, it is possible for a human viewer to determine (either qualitatively or quantitatively) the relative explanatory powers of XCS, UCS and MILCS without requiring a detailed understanding of the rule-form of a correct final solution for this particular problem.

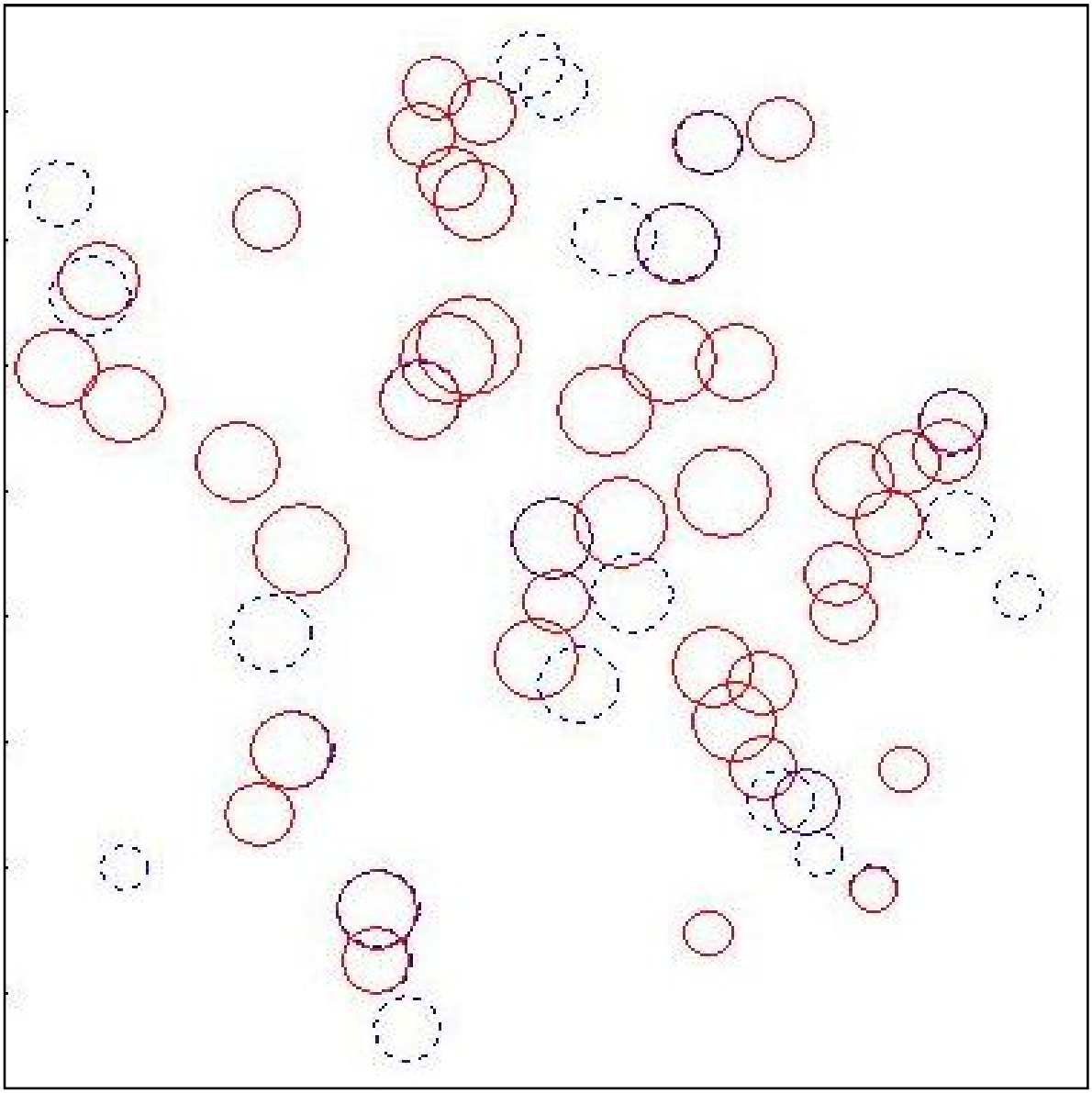

Figure 35: Visualisation of the final rule set developed by XCS on the 6 multiplexer

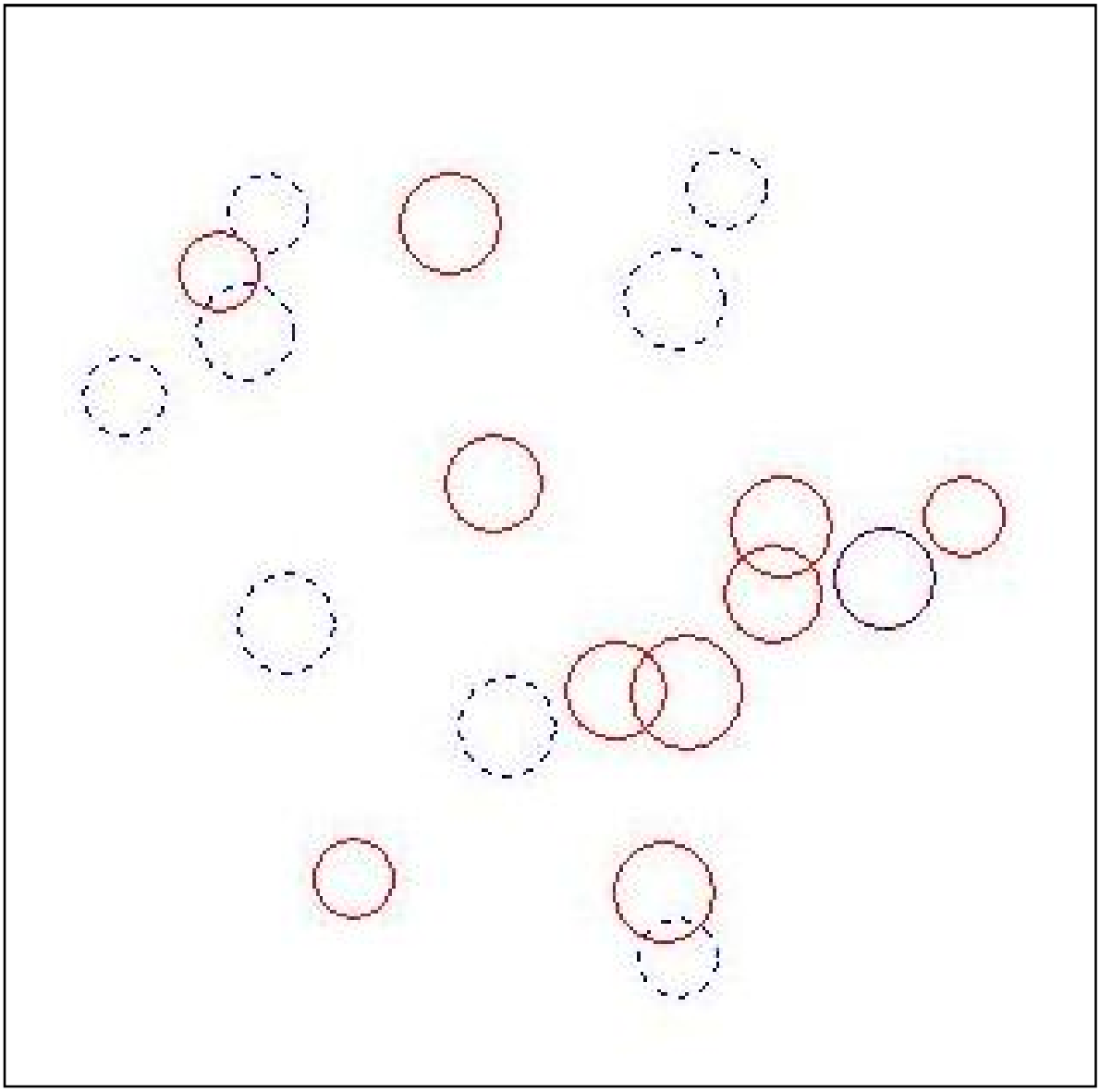

Figure 36: Visualisation of the final rule set developed by UCS on the 6 multiplexer

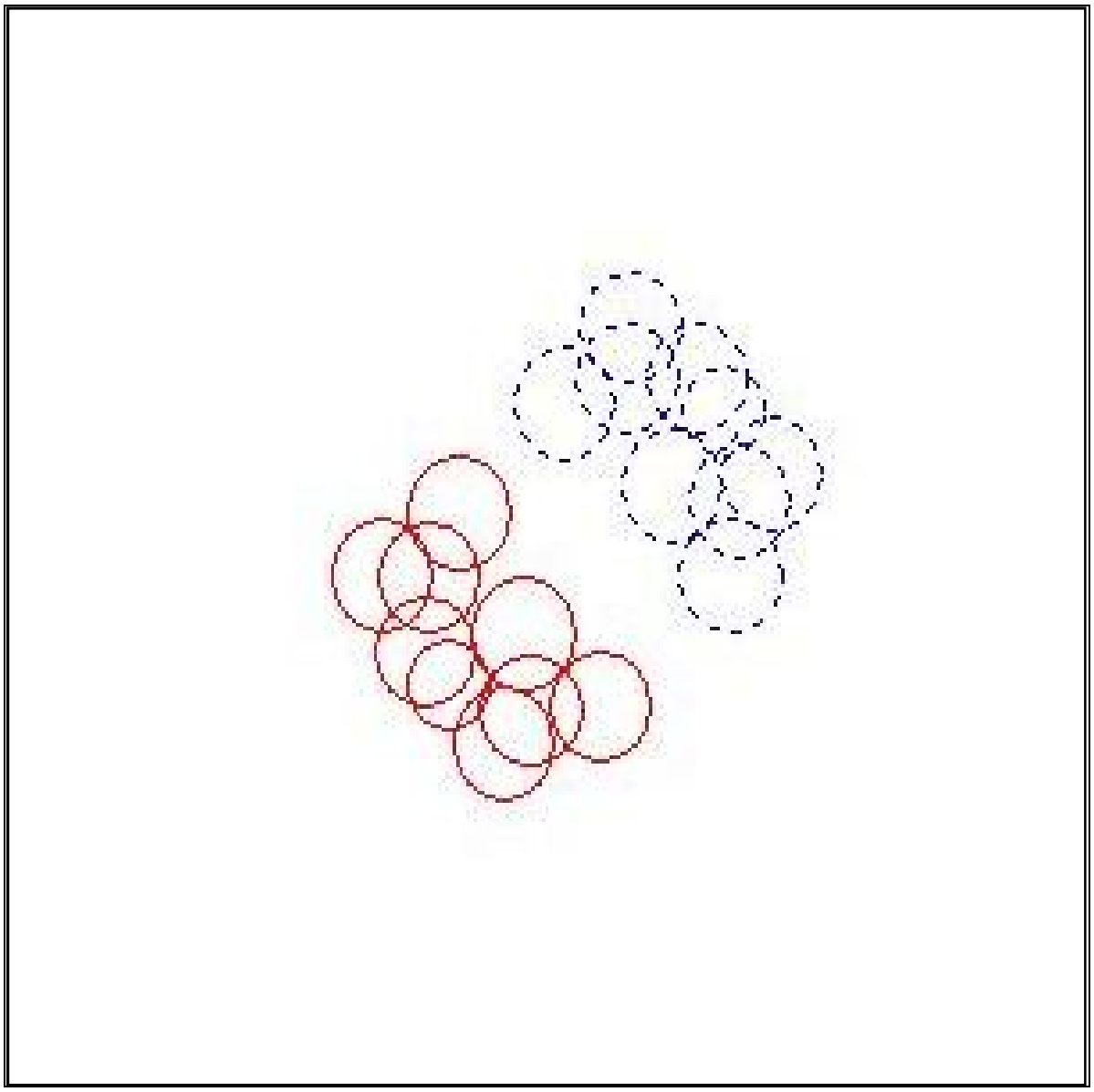

Figure 37: Visualisation of the final rule set developed by MILCS on the 6 multiplexer

Figure 38, Figure 39 and Figure 40 show visualisations of the final rule sets from XCS, UCS and MILCS (respectively) applied to the 11 multiplexer. Once again, the superior explanatory power of the MILCS rule set is apparent. As in the 6 multiplexer, a decision surface between the two actions is apparent, even after the projection of the rule set to a two-dimensional space.

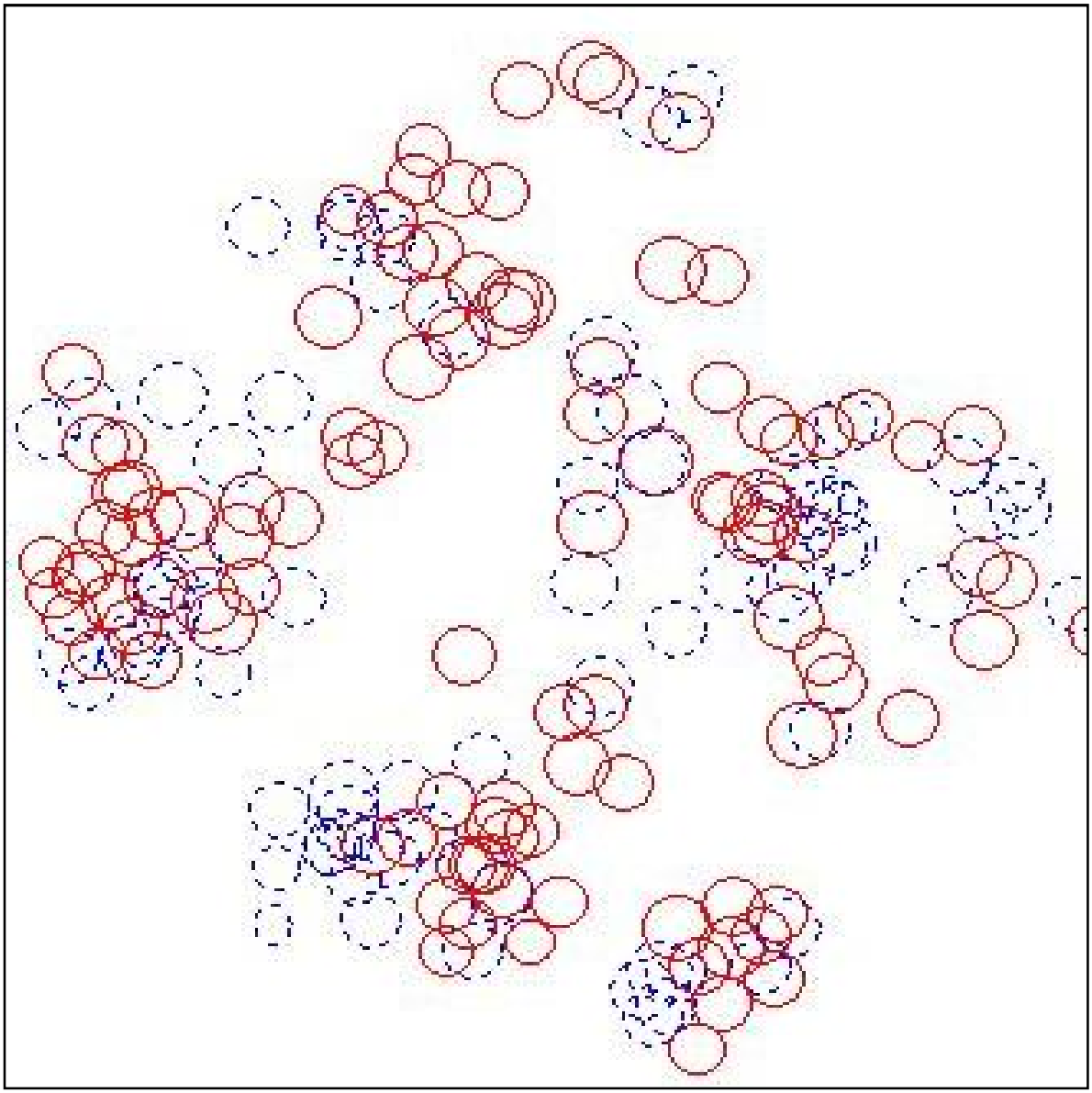

Figure 38: Visualisation of the final rule set developed by XCS on the 11 multiplexer

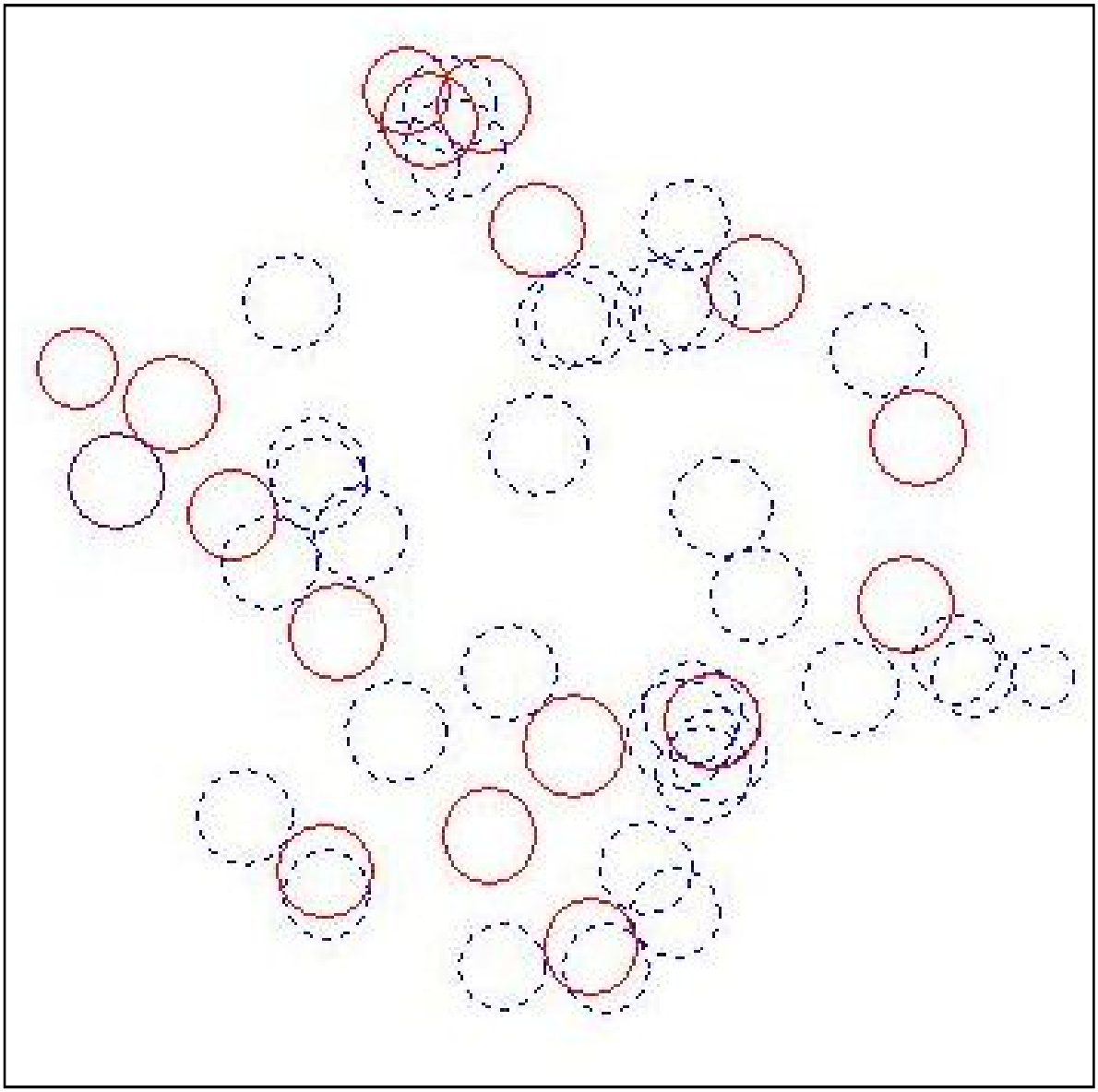

Figure 39: Visualisation of the final rule set developed by UCS on the 11 multiplexer

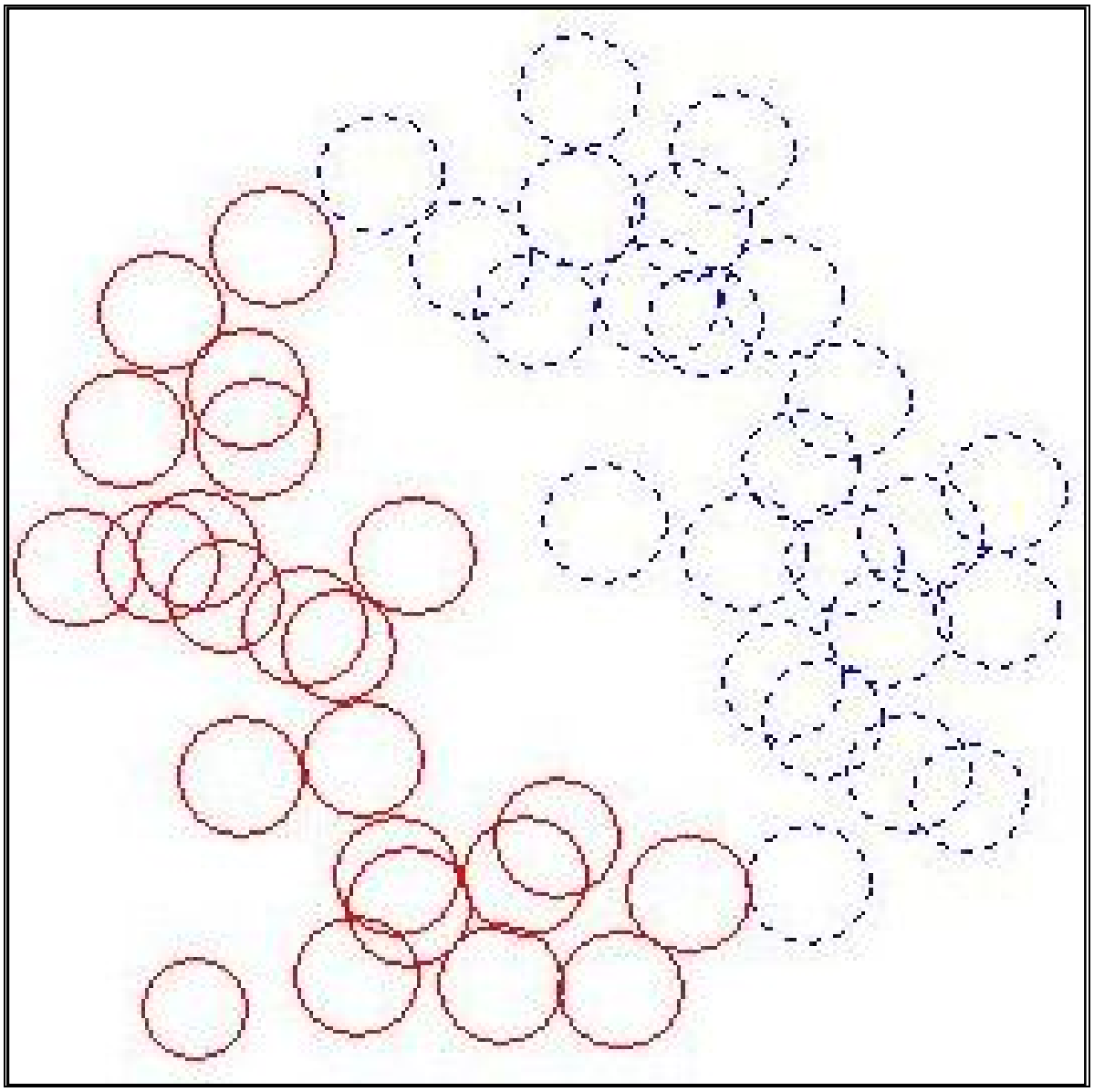

Figure 40: Visualisation of the final rule set developed by MILCS on the 11 multiplexer

Figure 41, Figure 42 and Figure 43 show visualisations of the final rule sets from XCS, UCS and MILCS (respectively) applied to the 20 multiplexer. While a linear decision surface is no longer apparent in the MILCS result, the less complex structure of the MILCS rule set when compared to the XCS and UCS rule sets is apparent.

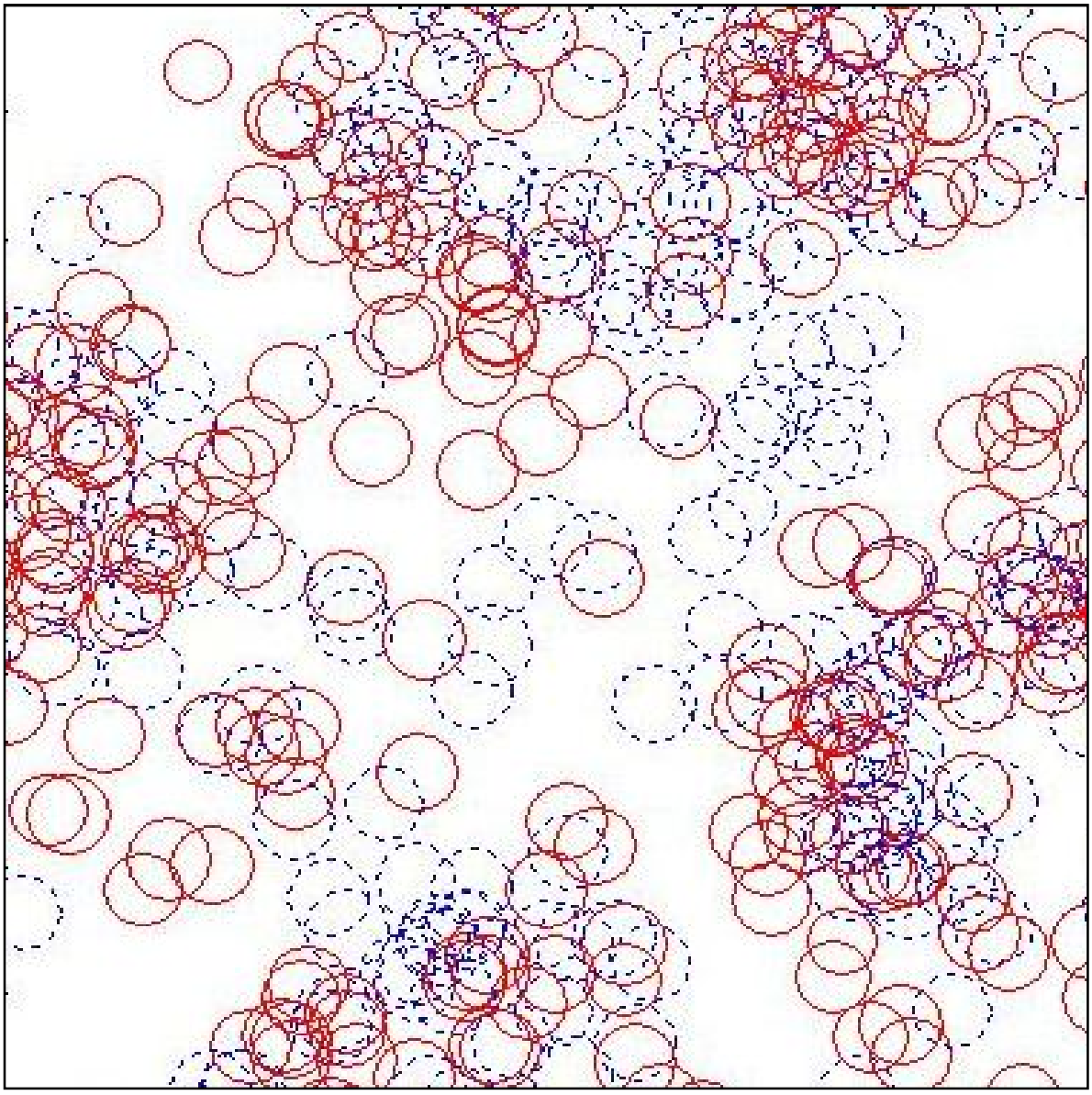

Figure 41: Visualisation of the final rule set developed by XCS on the 20 multiplexer

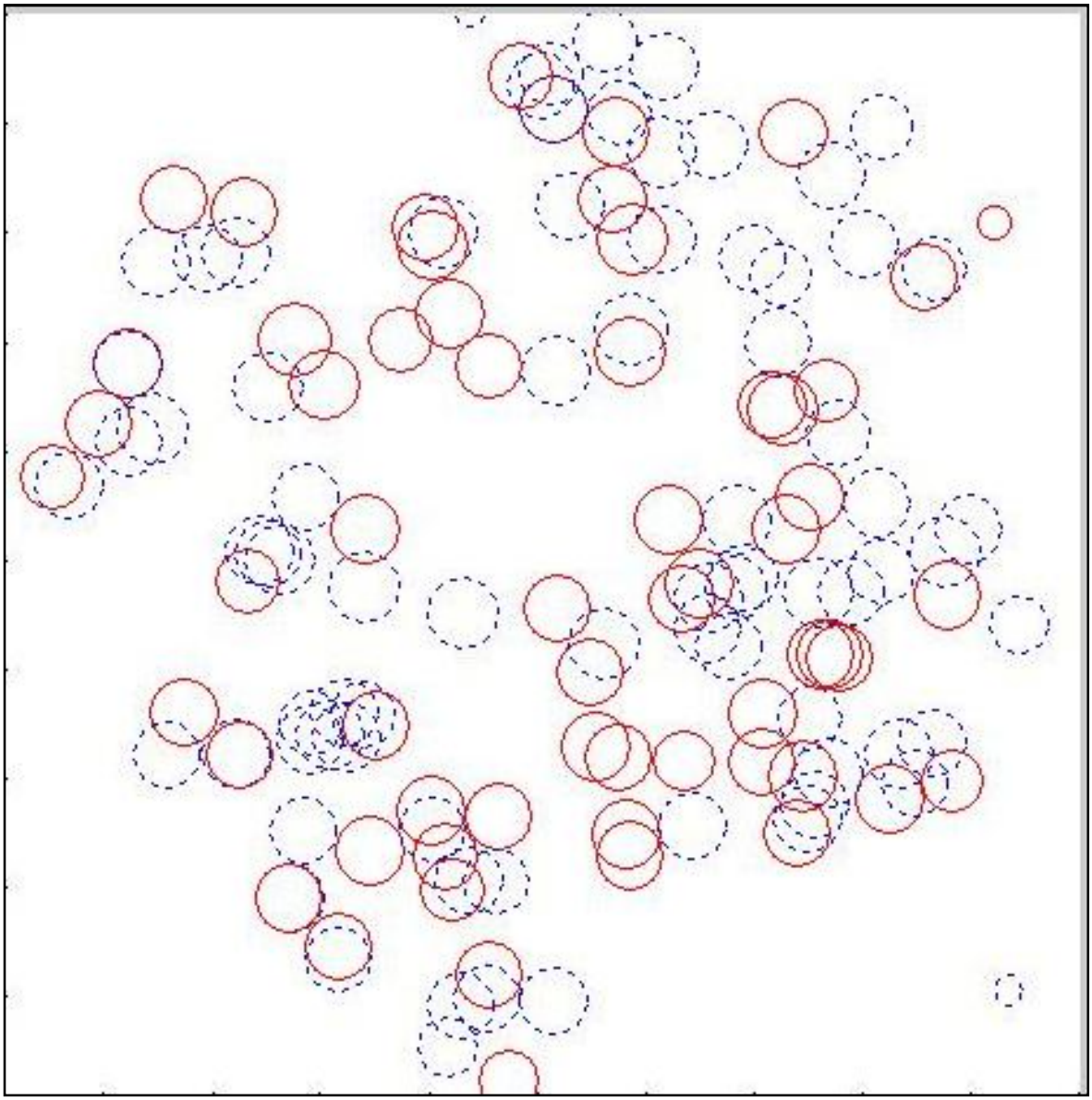

Figure 42: Visualisation of the final rule set developed by UCS on the 20 multiplexer

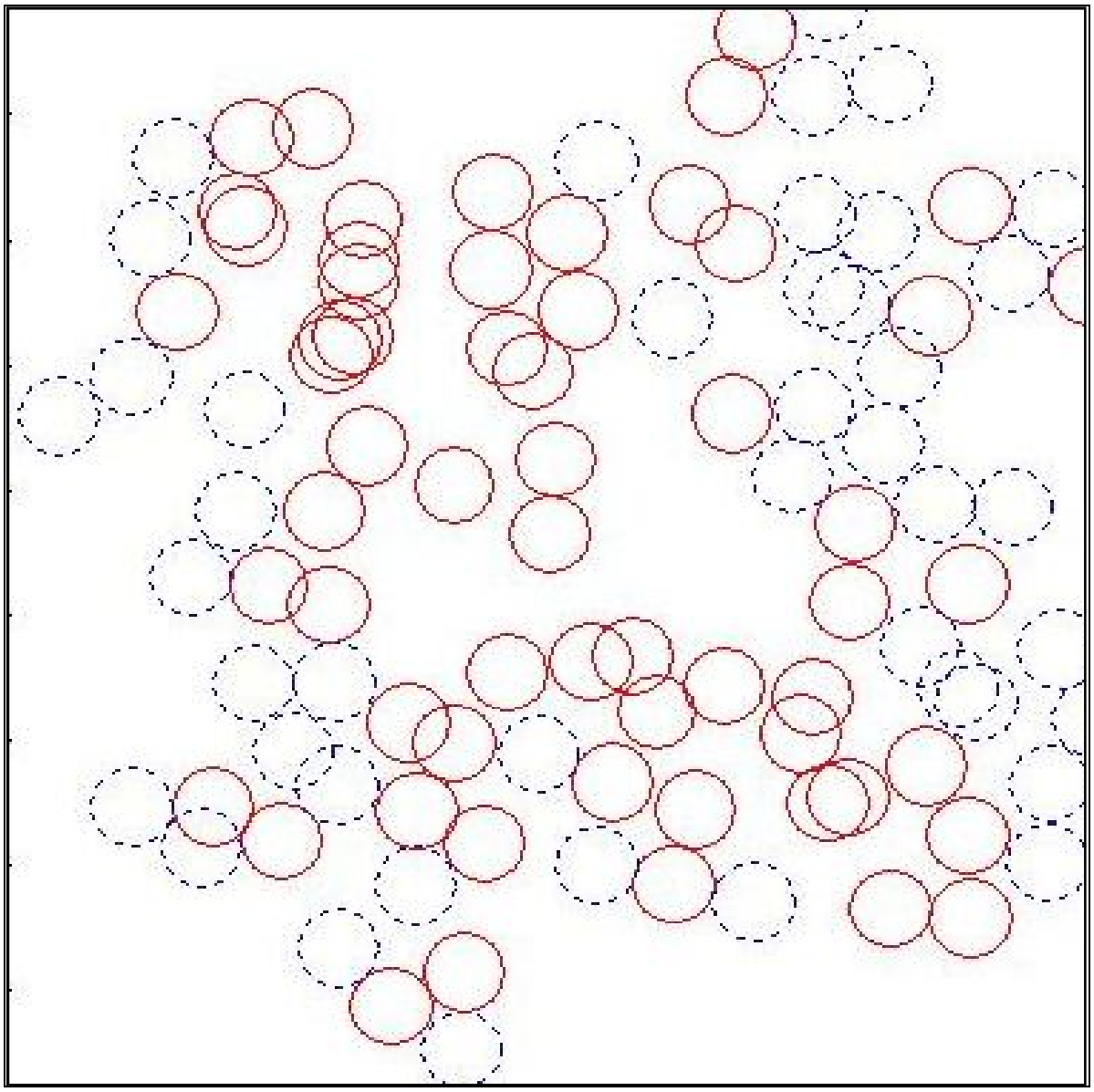

Figure 43: Visualisation of the final rule set developed by MILCS on the 20 multiplexer

The Java implementation of QT clustering algorithm [49] is used on the above graphs. Final positions of all nodes are imported to the algorithm and the threshold parameter is set to 20.

Problems	System	Number of circles	Number of clusters
6 multiplexer	XCS	46	6
	UCS	21	4
	MILCS	13	3
11 multiplexer	XCS	180	27
	UCS	53	13
	MILCS	47	10
20 multiplexer	XCS	484	85
	UCS	142	34
	MILCS	102	25

Table 11: QT clustering results of XCS, UCS and MILCS

It can be seen from Table 11 that MILCS has the smallest number of circles and clusters of all while XCS has the largest. Note that while only some small portion of the final XCS and UCS rule sets are perfect generalisations, and a human could detect these with knowledge of the multiplexer, that would not be the case in a problem of unknown structure. Since the goal is to relate an uninformed comparison of explanatory power, the author believes these figures and numbers are sufficient.

5.1.6 Effect of deletion parameters

Effective settings for the deletion parameters $freq_{del}$, $freq_{inc}$, $freq_{max}$, and del_{range} are highly problem dependent. First of all, del_{range} should be decided. It is dependent on the problem length and input space and should be large enough for a classifier to fire at least once. The $freq_{del}$ should then be set to something between 0 and 1 over del_{range} so that any classifiers do not fire at least once within the specified deletion range gets deleted. A few runs and manual examination of the results are recommended to adjust the best setting for del_{range}. $Freq_{del}$ can also be raised higher if the problem is not suffered from severe class imbalance. Usually these two parameters allow a compact solution for most problems. However, when the final solution size requires further reduction, $freq_{del}$ can be lowered for batch deletion. Also note that towards the end of a run, the optimal solution might have already been discovered but new rules are still being generated, which might result a final population consisting of both optimal solution and others. This is especially noticeable when the difference between mat_{del} and mat_{act} is big. To overcome this problem, $freq_{inc}$ and $freq_{max}$ can be used to gradually raise the deletion threshold so that unfit rules get deleted more and more quickly once the system performance is high enough (which possibly indicates the existence of optimal solution).

Figure 44 shows the effect the del_{range} on the 11 multiplexer problem. Only performance and population size is shown in this graph for a better observation. The square markers indicate the original performance (white filled square marker) and population size (black filled square marker) in Subsection 5.1.3. The round markers show the performance and population size of a decreased studying parameter and triangle markers show an increase. This applies to the rest of this subsection unless otherwise stated.

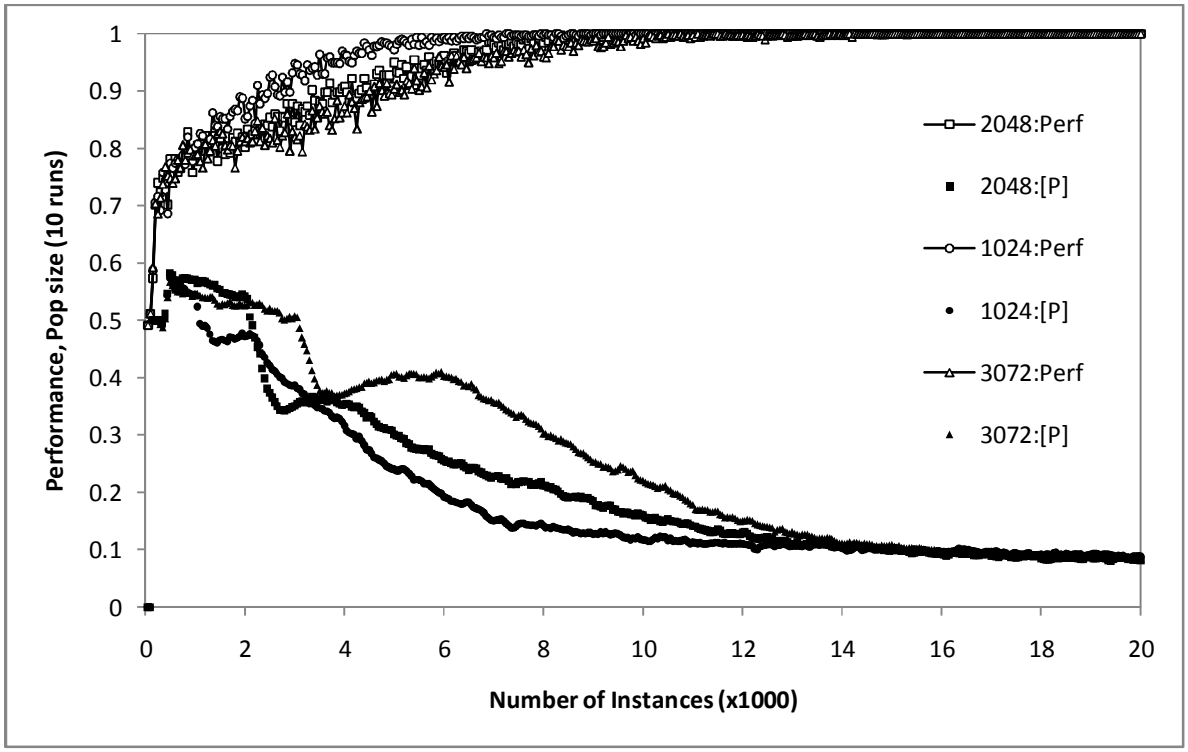

Figure 44: del_{range} effect on the 11 multiplexer problem

This parameter has some small impact on the final population size (up to 8.5%). However, it does influence the learning curve and performance convergence time. There is a noticeable dip in the population at the beginning of the experiment which corresponds to the its del_{range}. The dip also indicates where the deletion starts to act (before which point subsumption is the only cause for population decrease). In this case, del_{range} = 1024 has the steepest learning curve but with the largest final population size. However, it is not always the case that a small del_{range} will speed up performance convergence. If it is set to too small, classifiers will fail to act within the given range and the system will fall apart. On the other hand, larger del_{range} slows down the performance convergence but results a relatively smaller final population size.

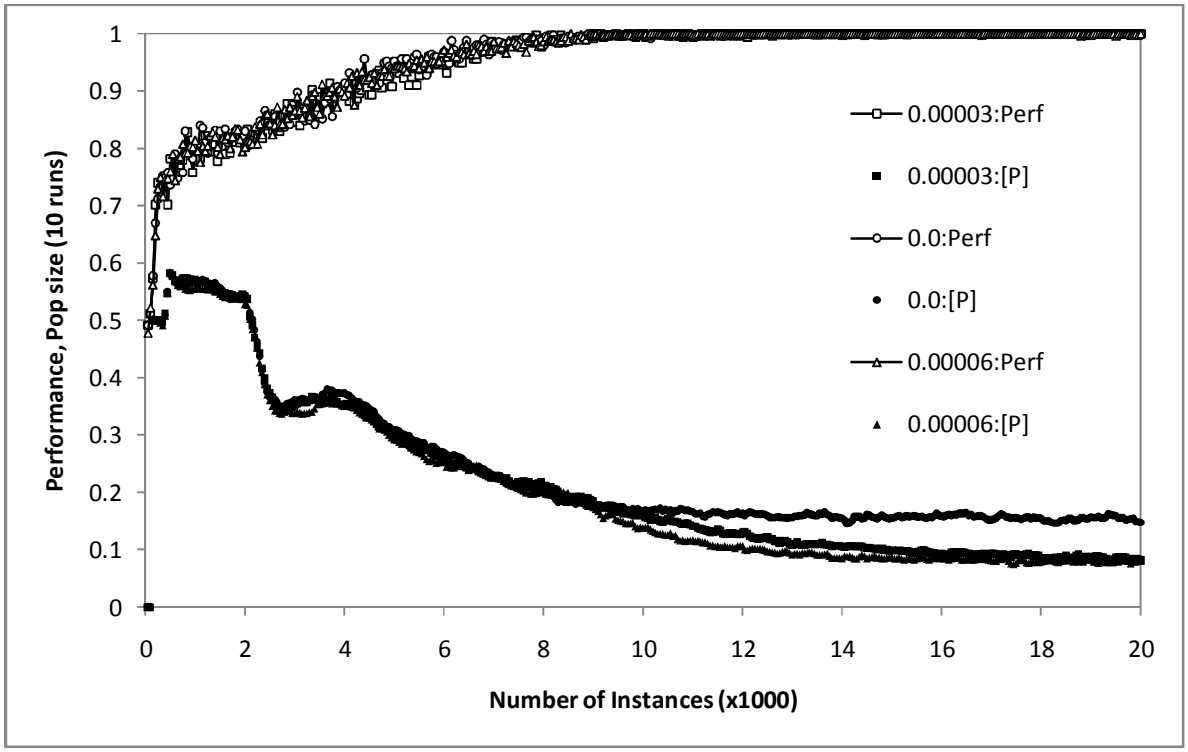

Figure 45: $freq_{inc}$ effect on the 11 multiplexer problem

It can be seen from Figure 45 that $freq_{inc}$ only starts to take effect once the performance has reached 1 which corresponds to θ_{perf}. Therefore, it does not influence performance learning curve. A larger $freq_{inc}$ results a more compact final solution size. In this case, $freq_{inc}$ = 0.00003 results a final population of only half the size of $freq_{inc}$ = 0. Larger $freq_{inc}$ can be dangerous because fit rules, especially with low matching occurrence, might get deleted by mistake. In the case of $freq_{inc}$ = 0.00006, the final size is only slightly reduced but it fails to reach 100% performance ($\theta_{perf} \times maximum\, reward$)

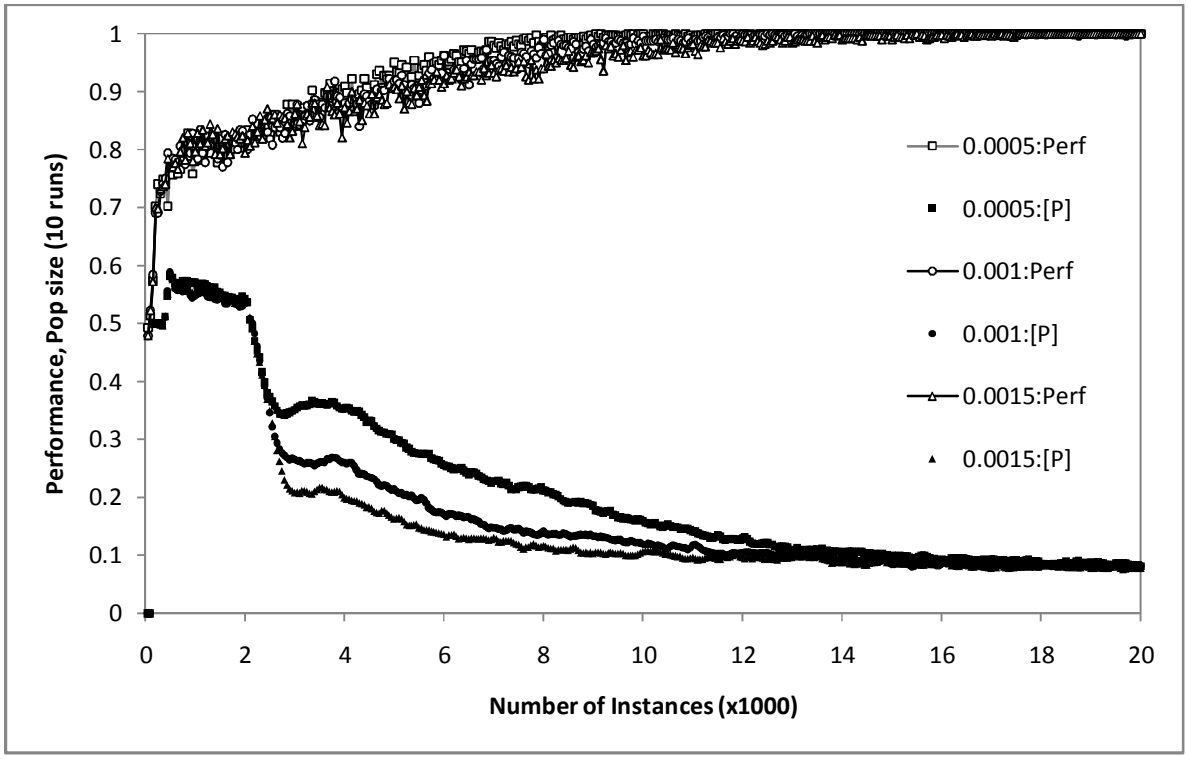

Figure 46: $freq_{del}$ effect on the 11 multiplexer problem

Since setting $freq_{del} = 0$ turns off the deletion completely, the two settings studied here are both larger than the original. The sharp decrease right after 2000 iterations corresponds to del_{range} (2048) and $freq_{del}$ immediately has an effect on the population size. A smaller $freq_{del}$ usually makes the performance converge faster but with the compensation of a larger final population size. However, in this case $freq_{del} = 0.0005$ has a steeper population convergence curve which also allows a similar final population size to the others. This is caused by the early act of $freq_{inc}$ once its performance reaches 100% and $freq_{inc}$ does not take effect until much later for the others two.

5.2 Even parity problem

5.2.1 Problem description

To further investigate the performance of MILCS, the parity problem is next to test. It has been widely used as another benchmark for learning classifier systems [64] [8]. The parity problem is defined as follows: given a binary string of fixed length, the output is one if the number of ones in the string is odd, otherwise the output is zero. The problem does not allow any generalisation at all, unless some irrelevant bits[12] are added in the input example. This problem is an extreme example of

[12] An irrelevant bit has no influence on the output class. Therefore, in the ternary alphabet, the rules can generalise it with the "don't care" symbol "#".

specificity. A single "don't care" symbol "#" in the condition makes the rule over-general. If the length of a parity string is l then the optimal final rule set should contain 2^l number of non-# rules. A test set of parity problems ranging from 5 to 7 bits (with no irrelevant bits) are used to test the performance of MILCS against XCS and UCS.

5.2.2 Parameter settings

The parameter settings for MILCS are shown in Table 12 whereas XCS and UCS parameters are shown in Table 13. $freq_{inc}$ and $freq_{max}$ is set to 0.0 because it is believed $freq_{del}$ alone is sufficient for this problem.

Problem size	**5**	**6**	**7**
N	800	1500	2000
N_I	600	1000	1500
$P_{\#}$	0.33	0.33	0.33
θ_{GA}	5	5	5
θ_{sub}	70	70	70
mat_{act}	128	256	512
mat_{del}	256	512	1024
mat_{sub}	256	512	1024
del_{range}	512	1024	2048
$freq_{del}$	0.00195	0.00097	0.00097
$freq_{inc}$	0.0	0.0	0.0
$freq_{max}$	0.0	0.0	0.0
α_F	1.0	1.0	1.0
α_P	1.0	1.0	1.0
P_I	0.0	0.0	0.0
F_I	0.01	0.01	0.01
χ	0.8	0.8	0.8
μ	0.04	0.04	0.04
crossoverType	uniform	uniform	uniform
θ_{perf}	1.0	1.0	1.0
tournamentSize	0.4	0.4	0.4
selectTolerance	0.001	0.001	0.001

Table 12: MILCS parameters for the even parity problems

System	XCS	UCS
N	800/1600/3200	800/1600/3200
β	0.2	0.2
α	0.1	0.1
ε_0	0.01	N/A
ν	5	10
γ	0.71	N/A
θ_{GA}	25	25
χ	0.8	0.8
μ	0.04	0.04
θ_{del}	20	20
δ	0.1	0.1
θ_{sub}	20	20
$P_{\#}$	0.33	0.33
P_I	10.0	N/A
ε_I	0.0	N/A
F_I	0.01	0.01
P_{explr}	0	N/A
θ_{mna}	2	2
crossoverType	uniform	uniform
tournamentSize	0.4	0.4
selectTolerance	0.001	N/A
doGASubsumption	Yes	Yes
doActionSetSubsumption	Yes	N/A
acc_0	N/A	0.99
initializePopulation	no	N/A

Table 13: XCS and UCS parameters for the even parity problems

5.2.3 Results and analysis

Figure 47, Figure 48 and Figure 49 show results from XCS, UCS and MILCS applied to the 5 bit even parity. Graphs reflect the average of 10 runs.

Note that MILCS converges more rapidly than both XCS and UCS and it also converges to a smaller final rule set. XCS is able to converge slightly fast than UCS but the latter manages to achieve a compacter final rule set.

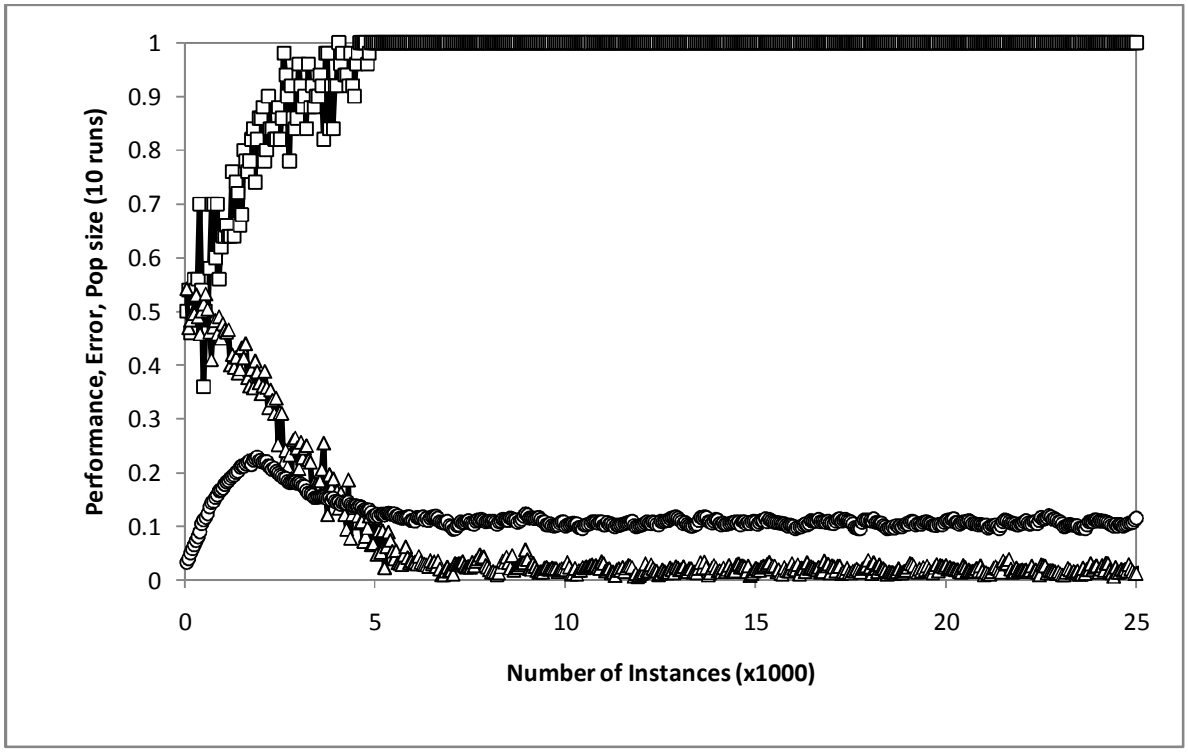

Figure 47: Results from XCS applied to the 5 bit even parity

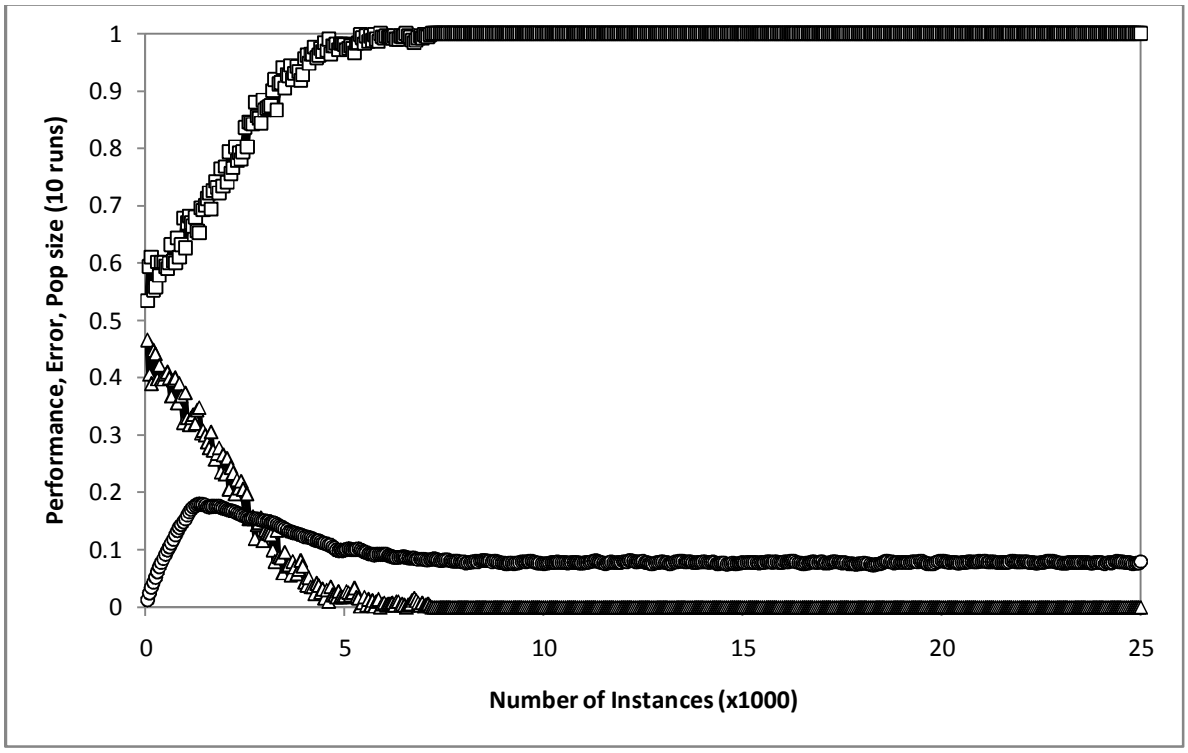

Figure 48: Results from UCS applied to the 5 bit even parity

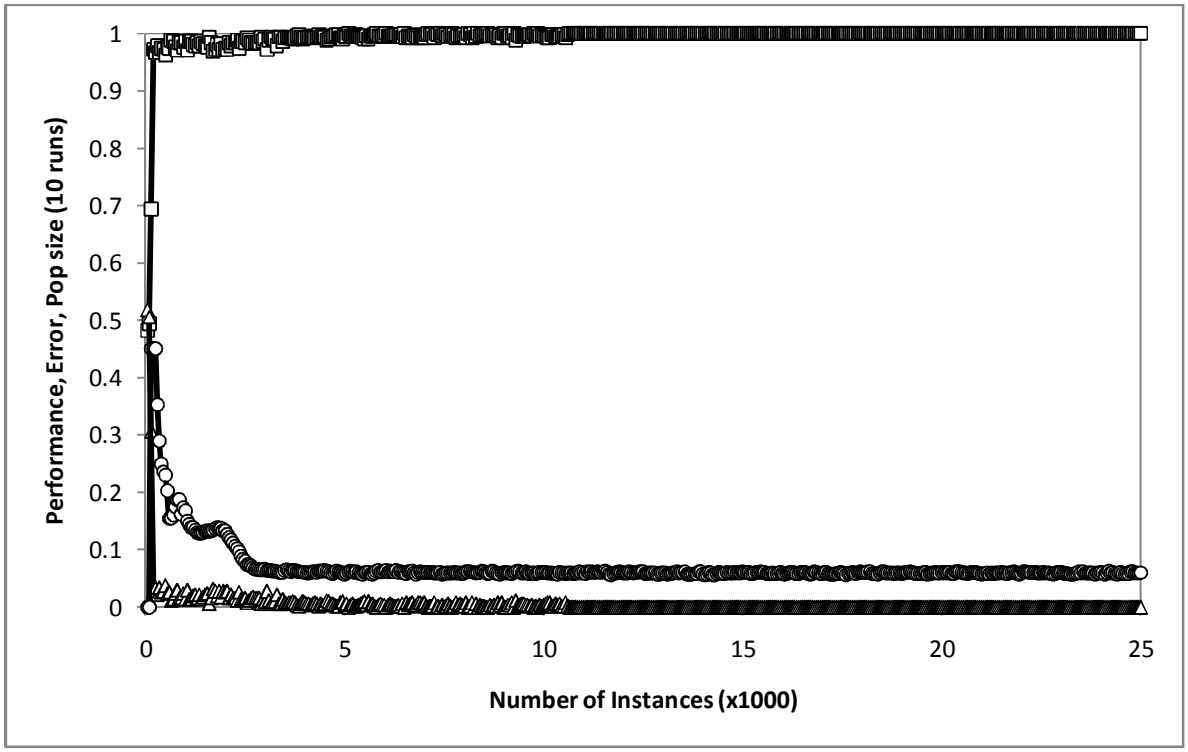

Figure 49: Results from MILCS applied to the 5 bit even parity

Figure 50, Figure 51 and Figure 52 show results from XCS, UCS and MILCS applied to the 6 bit even parity. In this case, UCS has the fastest convergence speed but MILCS still manages to converge to the smallest final population of all. XCS converges faster than MILCS but has the largest final rule set.

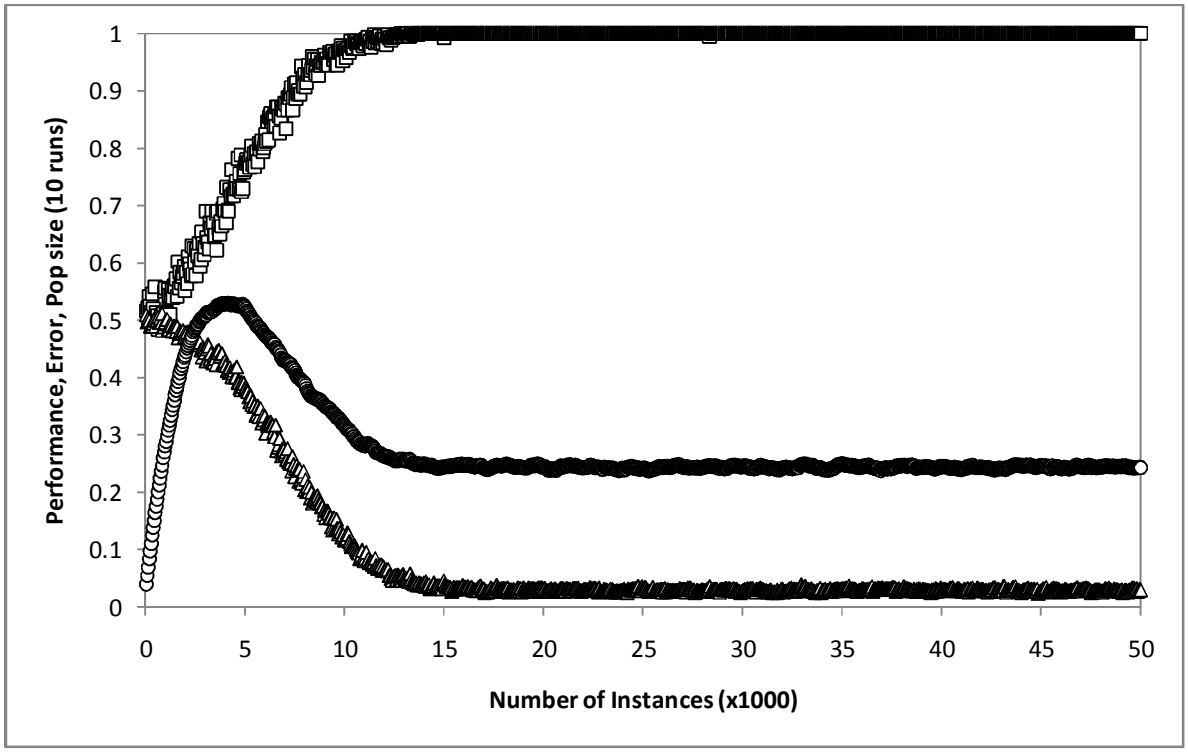

Figure 50: Results from XCS applied to the 6 bit parity

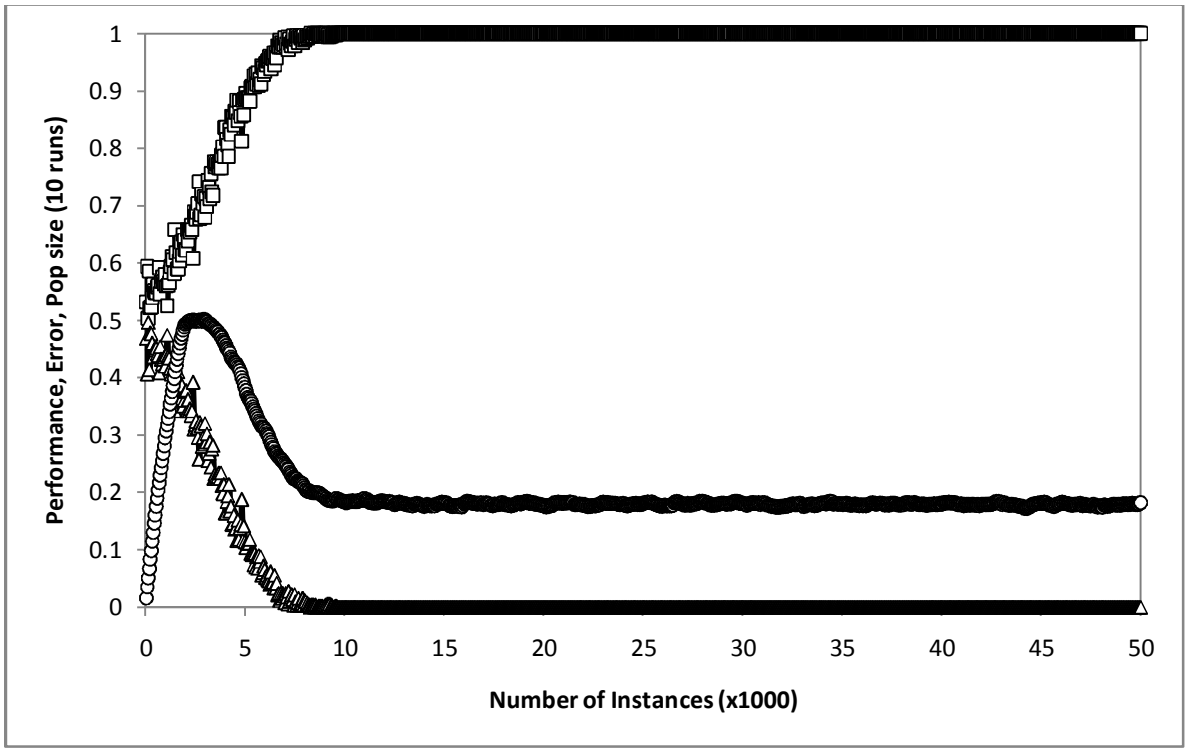

Figure 51: Results from UCS applied to the 6 bit parity

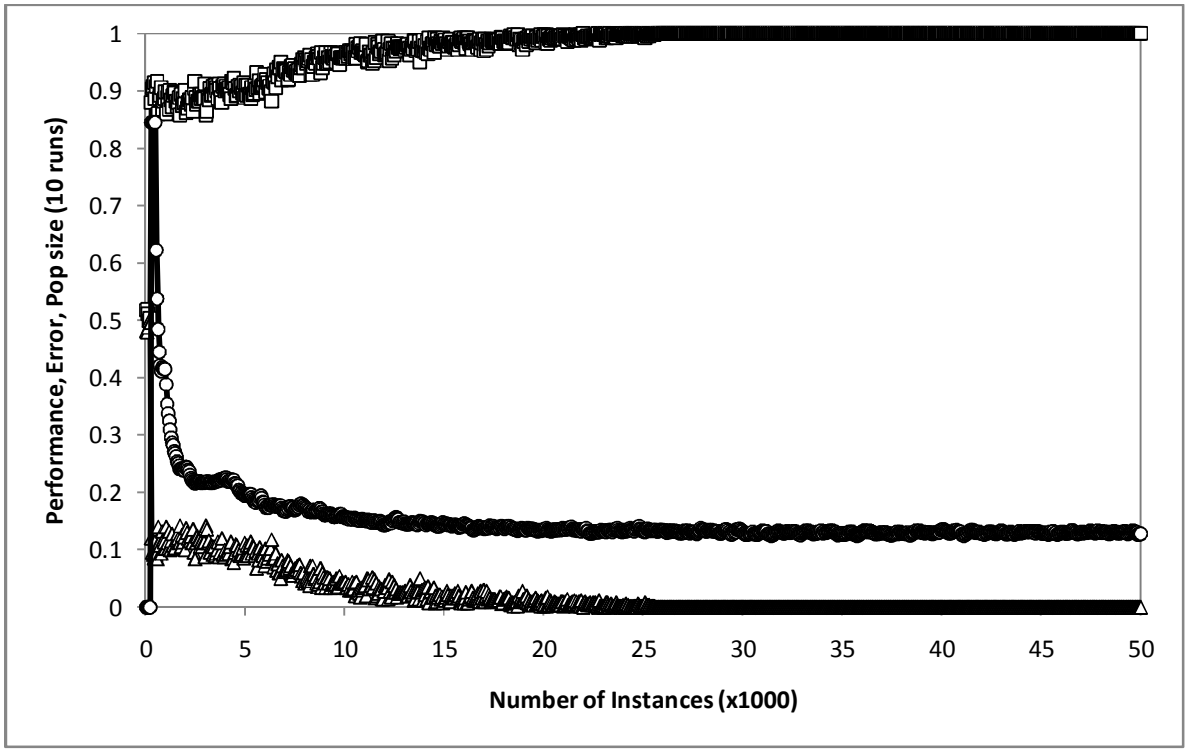

Figure 52: Results from MILCS applied to the 6 bit even parity

Figure 53, Figure 54 and Figure 55 show results from XCS, UCS and MILCS applied to the 7 bit even parity. Once again UCS converges the fastest in terms of performance while MILCS converges to the smallest final rule set size. XCS converges faster than MILCS but has the largest final population size.

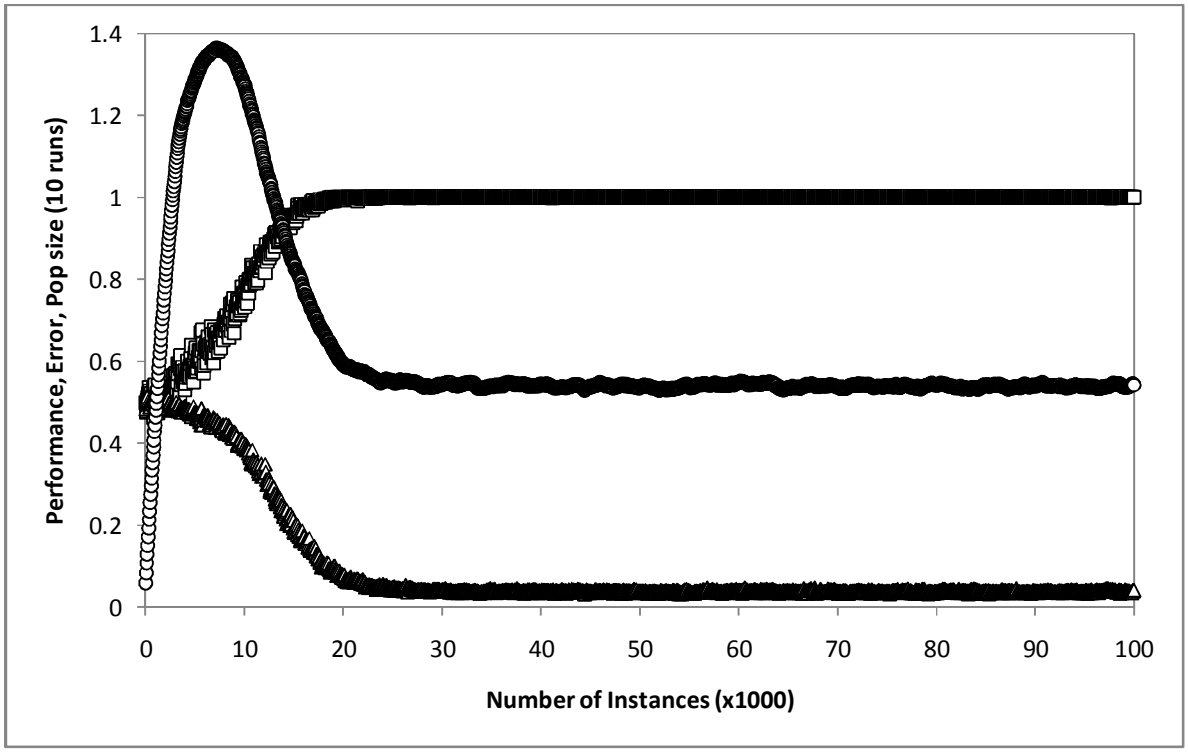

Figure 53: Results from XCS applied to the 7 bit parity

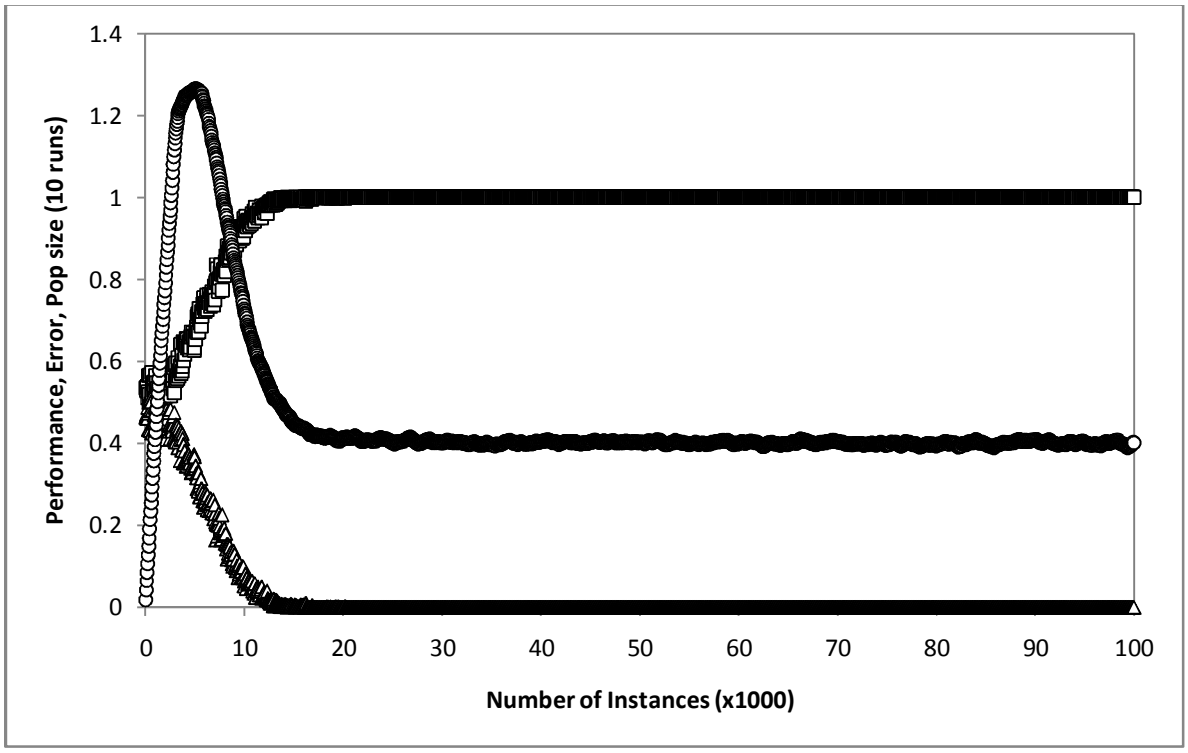

Figure 54: Results from UCS applied to the 7 bit parity

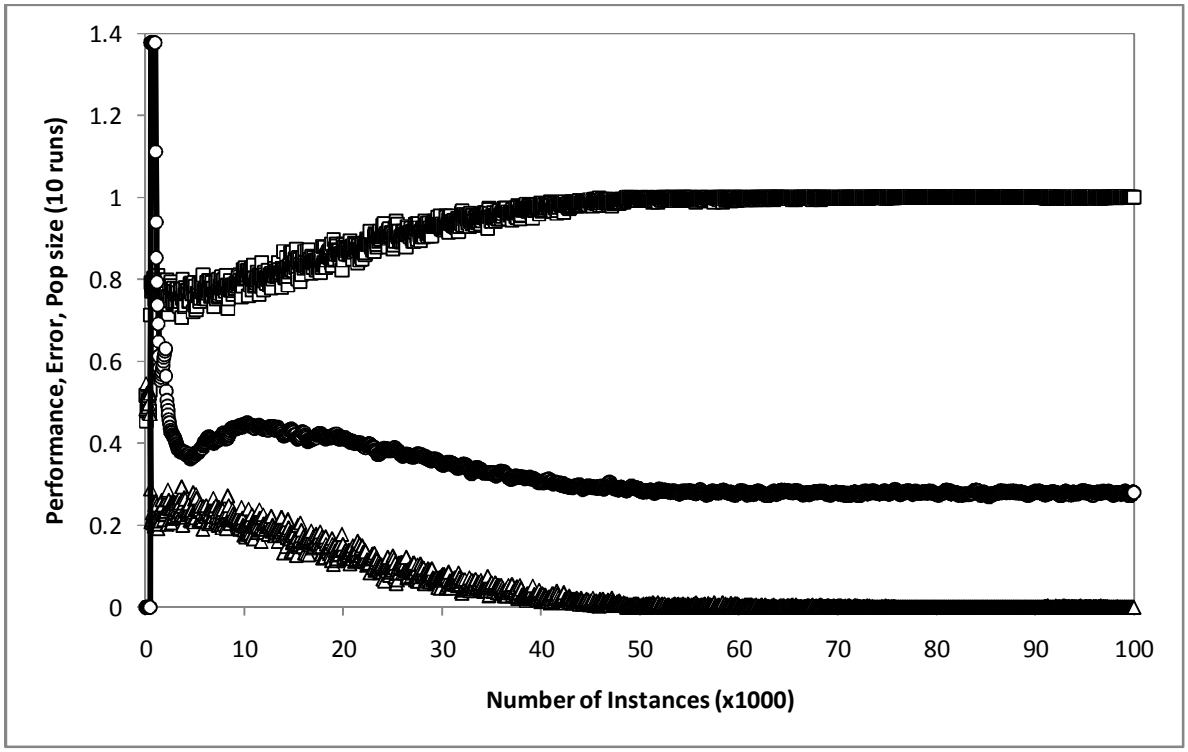

Figure 55: Results from MILCS applied to the 7 bit parity

Overall, the performances between UCS and XCS are similar and the former has a smaller rule set than the latter due to learning the best action map rather than the complete action map. This is consistent with the results reported in [8]. Although MILCS only converges the fastest at the 5 bit parity, it has the most compact final population size of all. This is expected because of the lack of "voting" in the action selection in MILCS and the concept of "structural learning" embedded in this system.

5.3 Summary of chapter

This chapter presents the first part of the experimental results of the thesis.

Section 5.1 reveals the scale-up of MILCS and compares it to XCS and UCS. The multiplexer problems are tested and the explanatory power of the final rule set are visualised using the tool introduced in Chapter 4. The benchmark results suggest that the scale-up of MILCS is slightly worse than that of XCS and UCS. However, MILCS manages to evolve a parsimonious rule set with strong explanatory power. This is visualised and measured using the tool introduced in Chapter 4. The effect of deletion parameters has also been studied.

Section 5.2 studies another benchmark problem: parity. Three tests with increasing problem length have been tested on MILCS, XCS and UCS, and MILCS is shown to converge to the most compact final population of all.

Chapter 6

Knowledge Discovery in Protein Structure Prediction

In this chapter, the task of data mining is further elaborated in terms of a process of knowledge discovery. The basic knowledge discovery process is studied followed by its development in the field of machine learning and GBML. Protein structure prediction, the chosen real-world knowledge discovery application, is tested on MILCS along with other machine learning techniques.

The section is structured as follows. Section 6.1 gives the necessary background on knowledge discovery through machine learning. Section 6.2 introduces the domain of protein structure prediction and describes the chosen experiment suite, followed by a results analysis. Section 6.3 provides a summary of this chapter.

6.1 Knowledge discovery and machine learning

Subsection 2.3.4 describes the urgent need for data mining is to extract useful information from the ever increasing accumulated data collected across a wide variety of fields. However the definition of data mining is often overloaded. Knowledge discovery, also known as *knowledge discovery in databases* (KDD), on the other hand, refers to the overall process of discovering useful knowledge from data whereas data mining is a particular step of the process [36]. In addition to the data mining step itself, KDD also includes several other steps. To simply the entire process, it can be roughly categorised into data pre-processing and discovered-knowledge post-processing. The KDD process is inherently iterative, as illustrated in Figure 56.

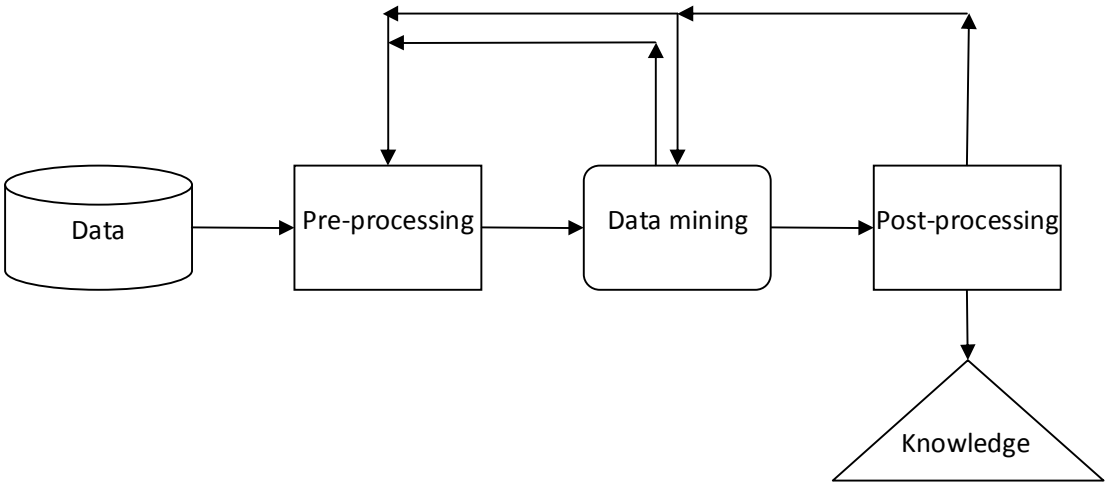

Figure 56: Knowledge discovery process

Data pre-processing involves the following basic steps (among others):

1) Data integration – A necessary step if the data comes from several different sources. It removes inconsistencies in the attribution names throughout the entire data sets.
2) Data cleaning – Basic operations include removing noise if appropriate, collecting the necessary information to model or account for noise, filling in missing values, etc. This step has a strong overlap with data integration. It is often common to bring in external domain knowledge from a user to achieve maximum accuracy of the source data.
3) Discretisation – A process to transform continuous attributes into nominal attributes, taking on only a few discrete values. It is an essential step for data mining algorithms that cannot cope with real-valued attributes.
4) Attribute selection – This consists of selecting, among all the attributes of the data set, a subset of attributes relevant for the target data mining task. The motivation for this step is the fact that irrelevant attributes can somehow "confuse" the data mining algorithm, leading to the discovery of inaccurate or useless knowledge.

Although several high-level knowledge representations, among which is the prediction rules, can express discovered knowledge to a level that it is comprehensible for the user (explanatory power, see Chapter 4), knowledge "interestingness" is often difficult to define and quantify due to its subjective nature. This is the motivation for discovered-knowledge post-processing. There are two mainly step in post-processing prediction rules discovered by LCSs.

1) Rule compaction – This is covered in Subsection 4.2.
2) Interesting-rules extraction – As mentioned early, many data mining algorithms have been designed to discover accurate, comprehensible rules but they were not designed to discover interesting rules, which is a rather more difficult and ambitious goal. However, it can still be achieved by the involvement of a user, i.e., the subjective way. By contrast, an objective way is data-driven and domain-independent. The basic idea is that the interestingness of a rule depends

not only on the quality of the rule itself, but also on its similarity to other rules. By comparing one rule against others is believed to yield some measurement in its interestingness [39].

The discovery of accurate knowledge has long been the goal of machine learning. Data mining has been particularly popular among the LCSs community. A first empirical comparison between LCSs and other machine learning techniques were made in 2001 and some competitiveness of LCSs was shown when applying to real-world classification tasks [9]. Similar results were obtained in [4]. Llora et al. implemented NAX, a LCS specialist in diagnosis of prostate cancer. The system has an overall performance similar to human examination, the current gold standard of care [73]. Recall Section 4.2, although several approaches have been attempted to develop an effective compaction algorithm for LCSs, it is believed the resulting rule set might suffer from over-fitting[13]. This is caused by extracting a minimal set of rules that covers the original dataset and ignores a large part the discovered domain knowledge achieved by LCSs. To overcome this, a new rule-driven approach has been introduced for better knowledge extraction and it was shown to highlight many potentially interesting rules that describe the problem space efficiently [58].

6.2 Protein structure prediction (PSP)

Proteins are heteropolymer molecules constructed as a chain of amino acids. There are 20 possible types of amino-acids that appear in nature, so one can see a protein as a string drawn from a 20-letter alphabet. The chain, however, does not have a linear structure. While being constructed, this chain folds to create a complex 3-D structure. This structure is very difficult to determine experimentally. Therefore, it needs to be predicted. This is the aim of the protein structure prediction field.

Protein structure prediction is a classification problem of significant, current scientific interest. While the thrust of this thesis is in developing a new machine learning methodology, it is a part of a larger EPSRC-funded project on such problems. Moreover, such problems are significantly complex, noisy, real-world tests for any classification algorithm. This section provides the necessary overview of the problem. For a more in-depth overview of protein structure prediction, see [79].

The prediction of the 3D structure of proteins remains a fundamental and difficult problem in computational biology after several decades of research. One of the ways in which this problem can be solved is by dividing it into several, easier but by no means trivial, sub-problems. A popular approach is to predict some specific attributes of a protein, such as the secondary structure, the solvent accessibility or the coordination number.

- **Secondary structure**: Proteins are heteropolymer molecules constructed as a chain of *residues*, which are amino acids of 20 different types. This string of amino acids is known as primary sequence. In a native state, the chain folds to create a 3-D structure. It is thought that this folding process has several steps. The first step, called secondary structure, consists of local structures such as *alpha helix*es, *beta strands* and coiled *coils*. These local structures can group in several conformations or domains, forming a *tertiary structure*.

[13] The trained rules that only perform well on the given dataset but fail to discover the domain knowledge

- **Coordination number**: For a given residue, CN is the number of residues from the same protein that are in contact. Two residues are said to be in contact when the distance between the two is below a certain threshold. Prediction of CN has been widely studied, since it can potentially constrain the search space to be explored using *de novo* approaches to protein structure prediction. A simplified protein model, the HP model, has been used to understand protein structure prediction. This model represents the sequence of a protein using two residue types: hydrophobic (H) and polar (P), based on its physico-chemical characteristics.
- **Relative solvent accessibility**: RSA, i.e., the exposure of residues to the solvent where the protein chain is located, is another important topological feature of a protein's native state. The hydrophobic effect has long been recognized as one of the principal effects that influence the ultimate structure of a protein. It is commonly assumed that the more hydrophobic residues will tend to be segregated in the inside of the structure and consequently will be less accessible to the solvent [72].

6.2.1 Coordination number prediction (CN)

6.2.1.1 Problem description

In order to convert real-valued CN definitions into a set of discrete states, so that they can be used as a classification dataset, Kinjo et al. [59] proposed a method to convert them into N class dataset. Because the current implementation of MILCS can only deal with two-class problems, source data of a two-state CN problem of real proteins using HP representation (ten pairs of training and test set, called Real-HP dataset, provided by Stout et al. in [106]) were used. As mentioned in Subsection 5.2, each residue is assigned a value of either hydrophobic (H) or polar (P). Windows were generated for one, two, and three residues at each side of a target residue and the CN class of the target residue assigned as the class of the instance. The dataset uses a WEKA format, ARFF [116]. A typical 3-window training set of the dataset is shown as follows,

```
@relation HP+CN_Q2
@attribute AA_-3 {h,p,x}
@attribute AA_-2 {h,p,x}
@attribute AA_-1 {h,p,x}
@attribute AA {h,p}
@attribute AA_1 {h,p,x}
@attribute AA_2 {h,p,x}
@attribute AA_3 {h,p,x}
@attribute class {0,1}
@data
x,x,x,h,p,h,h,1
x,x,h,p,h,h,p,0
x,h,p,h,h,p,p,1
h,p,h,h,p,p,h,1
p,h,h,p,p,h,h,0
h,h,p,p,h,h,h,0
…
…
h,p,p,h,p,h,p,0
p,p,h,p,h,p,h,0
p,h,p,h,p,h,x,0
h,p,h,p,h,x,x,0
p,h,p,h,x,x,x,0
```

Figure 57: Sample data of a two-state three-window Real-HP dataset

In Figure 57, the first few lines (starts with @) describe each attribute of the data representation and the actual data starts after "@data". "x" indicates "end of protein chain", "h" is hydrophobic and "p" is polar. Since it is a three-window representation, there are three residues at each side of the target residue, resulting a seven residue peptide. The dataset contains a balanced classification task. That is, the total numbers of antecedents belonging to each class are the same. However, being a real-world problem, the proportions of the two classes for each antecedent (determined by the class of the target residue) are different. The distribution of the target residue classes are shown in Figure 58.

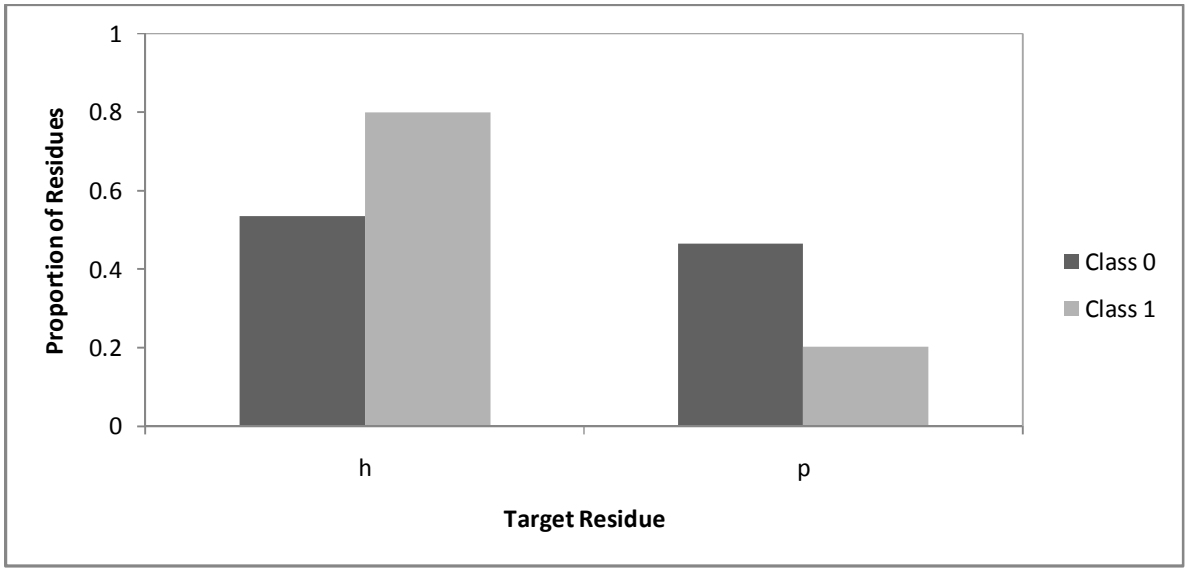

Figure 58: Distribution of hydrophobic/polar target residues classes in the Real-HP dataset: h=hydrophobic, p=polar

6.2.1.2 Parameter setting

This experiment has been tested on MILCS, XCS and UCS, along with published results from GAssist [106]. The GAssist results are used here because these published results were yielded from a very well tuned system for this type of prediction problem.

Three different window sizes for two-state predictions have been tested on all three systems.

MILCS parameter settings are shown in Table 14 and XCS and UCS parameters are shown in Table 15. The deletion for MILCS is again solely relied on $freq_{del}$.

Window size	1	2	3
N	100	200	300
N_I	50	50	50
$P_{\#}$	0.66	0.66	0.66
θ_{GA}	25	25	25
θ_{sub}	20	20	20
mat_{act}	150	150	150
mat_{del}	300	300	300
mat_{sub}	300	300	300
del_{range}	2500	2500	2500
$freq_{del}$	0.0004	0.0004	0.0004
$freq_{inc}$	0.0	0.0	0.0
$freq_{max}$	1.0	1.0	1.0
α_F	1.0	1.0	1.0
α_P	1.0	1.0	1.0
P_I	0.0	0.0	0.0
F_I	0.01	0.01	0.01
χ	1.0	1.0	1.0
μ	0.01	0.01	0.01
crossoverType	two-point	two-point	two-point
θ_{perf}	1.0	1.0	1.0
tournamentSize	0.4	0.4	0.4
selectTolerance	0.001	0.001	0.001

Table 14: MILCS parameters for the CN problem

With the increasing window size, only maximum population size is increased whereas the other parameters stay the same. This is somewhat different from the scale-up of the multiplexer problems. The CN problem using HP representation can be seen as a 3, 5 or 7 bit problem (for three different window sizes). Comparing to the 6, 11 and 20 bit multiplexer problems, the scale-up of the CN problem with HP representations does not have a major impact on the parameters.

System	XCS	UCS
N	100/200/300	100/200/300
β	0.2	0.2
α	0.1	0.1
ε_0	0.01	N/A
ν	5	10
γ	0.71	N/A
θ_{GA}	25	25
χ	0.8	0.8
μ	0.04	0.04
θ_{del}	20	20
δ	0.1	0.1
θ_{sub}	20	20
$P_{\#}$	0.33	0.33
P_I	10.0	N/A
ε_I	0.0	N/A
F_I	0.01	0.01
P_{explr}	0	N/A
θ_{mna}	2	2
crossoverType	two-point	two-point
tournamentSize	0.4	0.4
selectTolerance	0.001	N/A
doGASubsumption	Yes	Yes
doActionSetSubsumption	Yes	N/A
acc_0	N/A	0.99
initializePopulation	no	N/A

Table 15: XCS and UCS parameters for the CN problem

In order to be consistent with the GAssist rule presentation, a sparse encoding of rules have been implemented.

In XCS, UCS and MILCS, the representation is designed to be three condition bits each for the “H”, “P”, and “end of chain” conditions of the neighbouring residues, and two bits each for the “H” and “P” conditions of the target residue. The following table summarises the encoding,

	Neighbouring residues	Target residue
h	100	10
p	010	01
x (end of chain)	001	N/A

Table 16: HP problem rule translation

For instance, a typical window size 1 (1 neighbouring residue on each side of the target residue) input instance,

h,p,h,1

Using the above encoding gives,

10001100 1

It is a 7-bit condition, given that there are two neighbouring residues (3 bits each), and the target residue (2 bits). However, the weakness of this encoding lies on the subsumption algorithm. For example, consider the following two rule conditions:

1##01100

10#01100

These two rules conditions do not subsume each other if one uses the traditional subsumption algorithm. However, in this case, either of them could subsume the other, because both 1## and 10# can only match one residue, which is h. For this reason, a representation-specific subsumption algorithm has been implemented in XCS, UCS and MILCS, for the best results and fair comparison to the similar rule representation in GAssist.

Also, in order to show a fair comparison, UCS has been modified to use 'Tertiary Representation" rather than the default "Real-valued Representation" when learning with datasets. In this way, the subsumption algorithm and the human readability of final rule sets is the same as in XCS, MILCS and GAssist.

6.2.1.3 Results and analysis

Performance results presented here use the same training and testing procedures as those in [106]. A ten-fold cross validation has been carried out on each algorithm. Averaged results are shown in Table 17. The GAssist results were generously provided by the authors of [106].

Window size	Method	Performance		Solution size	Max evals
1	XCS	58.0% ±5.4%	•	58.1±3.1	300000
1	UCS	62.1% ±1.8%		47.6±7.0	300000
1	MILCS	63.4% ±0.7%		4.4±2.1	300000
1	GAssist	63.6% ±0.6%		4.0±0.0	8000000
1	C4.5	63.6%±0.6%		12.0±0.0	N/A
1	Naïve Bayes	63.6%±0.6%		N/A	N/A
2	XCS	60.1% ±2.9%	•	151.2±5.3	300000
2	UCS	60.6% ±3.9%	•	139.6±8.5	300000
2	MILCS	63.8% ±0.6%		8.8±2.1	300000
2	GAssist	63.9% ±0.6%		4.0±0.0	8000000
2	C4.5	63.9%±0.6%		12.0±0.0	N/A
2	Naïve Bayes	63.9%±0.6%		N/A	N/A
3	XCS	61.9% ±2.6%		255.2±6.4	300000
3	UCS	60.3% ±3.3%	•	248.6±5.9	300000
3	MILCS	63.9% ±0.8%		17.3±5.9	300000
3	GAssist	64.4% ±0.5%		5.1±1.0	8000000
3	C4.5	64.4%±0.5%		24.9±4.5	N/A
3	Naïve Bayes	64.3%±0.5%		N/A	N/A

Table 17: Results of XCS, MILCS, UCS, GAssist, C4.5 and Naïve Bayes on the two-state CN problem using HP representation of various window sizes. A • means that MILCS is statistically better than the algorithm to the left. Student T-test with 95% confidence level have been applied

Note that for each algorithm, "Max Evals" is the number of evaluations in a run, and this most likely represents a high upper bound on the number of evaluations required to get results of the indicated quality. However, these numbers do indicate the relative order of magnitudes of convergence times for results shown. Rule set size reflects the number of rules participating in action selection at the end of each run. The performance accuracy is shown along with its standard deviation followed by "±".

As shown in Table 17, the performance accuracy difference of window size 1 between XCS and MILCS is significant. For window size 2, the accuracy of MILCS is significantly better than that of XCS and UCS. UCS performs significantly worse than MILCS for window size 3. What is more, for all three window sizes, MILCS statically ties with GAssist, C4.5 and Naïve Bayes in terms of performance accuracy and the difference between the latter three and the other algorithms are statically significant for all three window sizes. In terms of final rule set size (of learning classifier systems), the difference between MILCS and GAssist for window size 1 is not significant.

However, for the other two window sizes, MILCS tends to have larger rule sets than GAssist, but converges an order of magnitude faster[14]. Final rule set sizes from XCS and UCS are much larger than those of MILCS and GAssist's. This is expected, as the prediction error of maximally accurate classifiers no longer reaches the threshold, so that the strong distinction between accurate and inaccurate classifiers does not apply. This means that subsumption does not apply, so the population size stays larger, even if the problem is successfully learned [16]. Overall, MILCS performance is statistically similar to GAssist, which provides the statistically best results, but with far greater learning steps.

For both MILCS and GAssist, it can be observed that the performance accuracy increases with the increasing window size. This can be explained by Table 18. The redundancy rate indicates the proportion of copies of the same instance in the test set and the inconsistency rate shows the proportion of instances with the same attributes (antecedent) but different class. Therefore, with the decreasing redundancy and inconsistency rate when the window size increases, larger window size yields better performance accuracy. Note that the systems still can learn even with 100% inconsistency rate for window size 1 is because of the different proportion of the two classes for each antecedent (See Figure 58).

Window size	Redundancy	Inconsistency
1	99.99%	100.00%
2	99.94%	92.50%
3	99.75%	81.71%

Table 18: Redundancy and inconsistency rate of the test set of Real-HP dataset

6.2.2 Relative solvent accessibility prediction (RSA)

6.2.2.1 Problem description

The RSA dataset is provided by Bacardit [7]. For each residue, the actual solvent accessibility is computed using the DSSP program [57] and it is normalised by dividing the value by the maximal accessibly of the residue's AA type [91], creating the RSA values. Afterwards, the RSA domain is divided into two states by placing a threshold at 25% relative accessibility, effectively converting the RSA prediction into a classification problem. ECGA is used to reduce the twenty-letter AA alphabets to a binary alphabet using the same procedure reported in [6]. Figure 59 shows some sample data from the RSA dataset (also in the WEKA ARRF format):

[14] However, it is worth mentioning that GAssist works in a different way from the Michigan LCSs. Thus it is a rough approximation of translated performance level.

```
@relation AA+SA_Q2
@attribute AA_-4 {0,1}
@attribute AA_-3 {0,1}
@attribute AA_-2 {0,1}
@attribute AA_-1 {0,1}
@attribute AA {0,1}
@attribute AA_1 {0,1}
@attribute AA_2 {0,1}
@attribute AA_3 {0,1}
@attribute AA_4 {0,1}
@attribute class {0,1}
@data
1,1,1,1,1,1,0,0,1,1
1,1,1,0,1,0,0,1,0,1
1,1,0,1,0,0,1,0,0,0
…
…
1,1,0,1,1,1,0,1,1,1
1,0,1,0,1,0,1,1,1,1
0,1,0,1,0,1,1,1,1,1
```

Figure 59: Sample data of binary attributes window size four AlphaRed_SA dataset

The data is organised in the same structure as in the CN dataset. Window size four is used which means there are four neighbouring residues at each side of the targeting residue, resulting a nine-residue peptide. Target residue with attribute value 0 is AA alphabet group of ACFILMVWY and 1 is GHNRSTDEKPQ. Neighbouring residues with attribute value 0 indicates ACFILMVWYGHNRST and 1 indicates DEKPQX. Approximately 57% of the instances are class 0 and the rest are class 1, resulting a slight class imbalance. Class imbalance problems have been seen as a challenge to learning systems because they tend to be biased towards the majority class and leave a poor generalisation for the minority class. However, only mild class imbalance exists in this problem; therefore it is not significant. Similar to the CN dataset, the proportions of the two classes for each antecedent are different. The distribution of the target residue classes are shown in Figure 58.

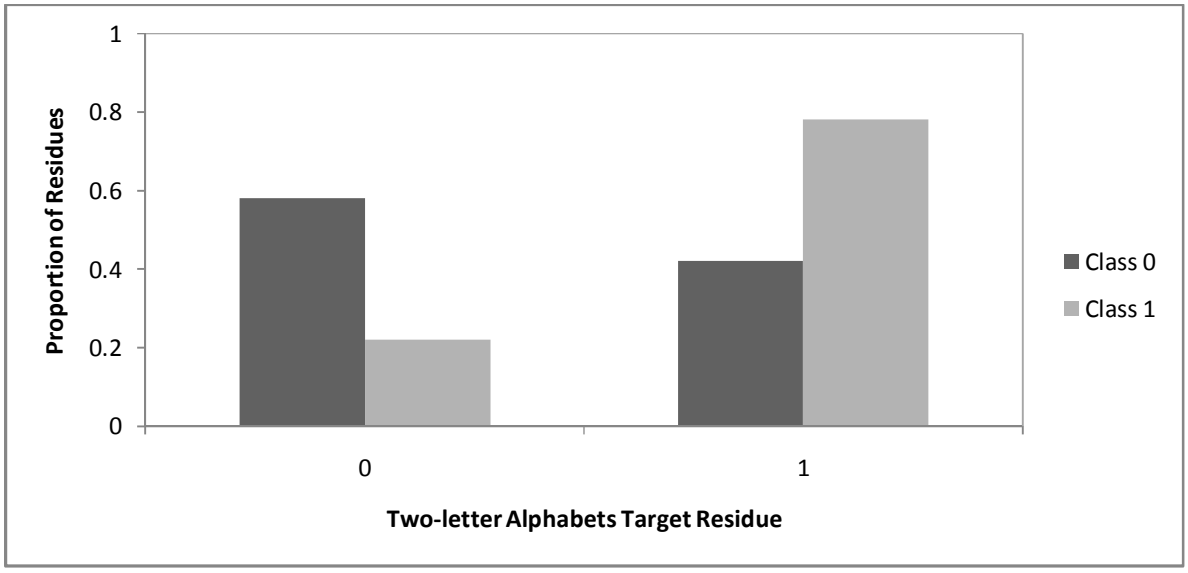

Figure 60: Distribution of two-letter alphabets target residues classes in the AlphaRed_SA dataset

6.2.2.2 Parameter setting

MILCS parameter settings are shown in Table 19 and XCS and UCS parameters are shown in Table 20.

Window size	4
N	2000
N_I	500
$P_{\#}$	0.1
θ_{GA}	60000
θ_{sub}	20
mat_{act}	450
mat_{del}	850
mat_{sub}	450
del_{range}	512000
$freq_{del}$	0.00001953
$freq_{inc}$	0.000001
$freq_{max}$	0.02
α_F	0.001
α_P	0.1
P_I	0.0
F_I	0.01
χ	1.0
μ	0.01
crossoverType	two-point
θ_{perf}	1.0
tournamentSize	0.4
selectTolerance	0.001

Table 19: MILCS parameters for the RSA problem

When compared to the CN problem (Section 6.2.1), the RSA problem is more difficult to learn, due to the fact that it is impossible to capture all the information needed for a proper RSA prediction in only two letters. Therefore the transformation into a binary alphabet (from a twenty-letter alphabet) makes the problem extremely noisy because too much information is lost in the process. While an exhaustive evaluation of the MILCS parameter space for this problem has not been studied, for learning such problems, θ_{GA} should be set to a large number for the mutual information fitness to stabilise first. The high del_{range} and low $freq_{\text{del}}$ settings are used to avoid immature deletion. However, it is not the case with the new subsumption and deletion algorithms (see Chapter 7).

System	XCS	UCS
N	2000	2000
β	0.2	0.2
α	0.1	0.1
ε_0	0.01	N/A
ν	5	10
γ	0.71	N/A
θ_{GA}	25	25
χ	0.8	0.8
μ	0.04	0.04
θ_{del}	20	20
δ	0.1	0.1
θ_{sub}	20	20
$P_{\#}$	0.33	0.33
P_I	0.01	N/A
ε_I	0.0	N/A
F_I	0.01	0.01
P_{explr}	0	N/A
Θ_{mna}	2	2
crossoverType	two-point	two-point
tournamentSize	0.4	0.4
selectTolerance	0.001	N/A
doGASubsumption	Yes	Yes
doActionSetSubsumption	Yes	N/A
acc_0	N/A	0.99
initializePopulation	no	N/A

Table 20: XCS and UCS parameters for the RSA problem

6.2.2.3 Results and analysis

BioHEL results were given by Bacardit [7] and the results are shown in Table 21. Once again UCS was modified to force using "Tertiary Representation" for a fairer comparison (See Subsection 6.2.1.2).

Method	Performance	Solution size	Max evals
XCS	66.1% ±1.6%	1094.6±44.0	1600000
UCS	67.1% ±6.6%	566.7±28.7	1600000
MILCS	65.6% ±1.2%	74.7±12.7	1600000
BioHEL	66.6% ±0.4%	33.4±4.8	125950000
C4.5	67.4%±0.2%	61.2±8.6	N/A
Naïve Bayes	67.2%±0.2%	N/A	N/A

Table 21: Results of XCS, UCS, MILCS, BioHEL, C4.5 and Naïve Bayes on a two-state RSA problem

It is interesting to see a similar performance among all six algorithms. The performance accuracy differences between MILCS and both non-learning classifier systems are statistically significant. There are noticeable differences in the final rule set size (of the learning classifier systems). Both BioHEL and MILCS have much smaller final rule set sizes, compared to the XCS and UCS. The high standard deviation of UCS performance accuracy indicates that the system is not stable enough to solve this problem reliably. Perhaps some fine tuning to the parameters would improve the stability, but that is not within the scope of this thesis. "Max evals" for BioHEL is an approximation of converge time translated from the iterative learning approach.

6.2.2.4 Default hierarchies

DHs (see Subsection 2.3.5) have been observed in this problem using the explanatory power visualisation tool. However, because the hierarchical structure of not-matched condition (everywhere outside a circle) and not-matched action is difficult to define, default hierarchies of matched condition (area of a circle) and action is considered. Figure 61 is the visualisation of final rule set derived from one run:

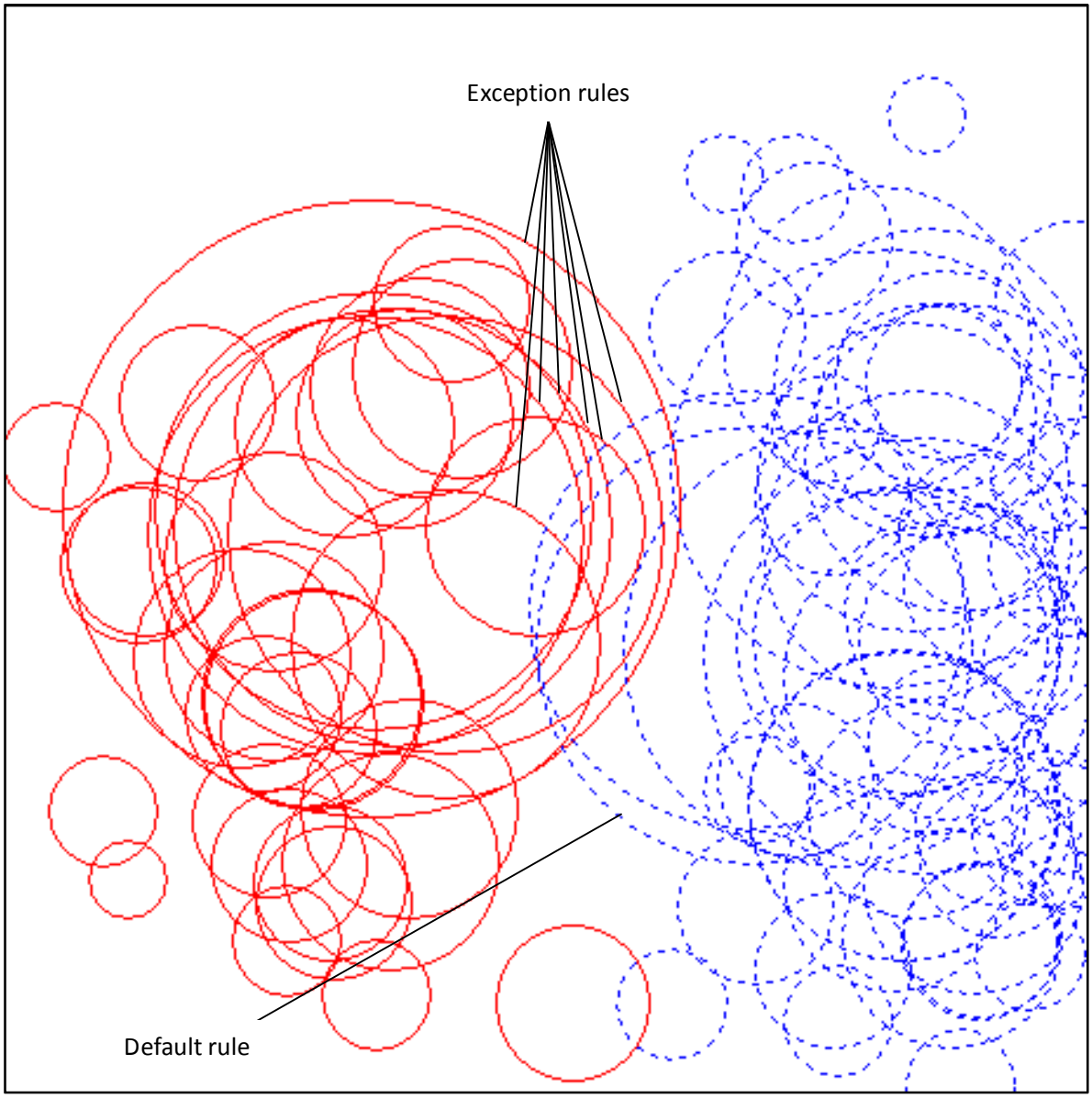

Figure 61: Default hierarchies of MILCS rules for the RSA problem

The overlaps of red and blue circles are highly likely to be DHs. However, this is not certain unless the classifiers really form hierarchical structures. For this reason, DH factors of each classifier are calculated for both the matched action and not-matched action.

$$DH\ factor(+) = \%Corr(Fired)(+) - \%Corr(Matched)$$

$$DH\ factor(-) = \%Corr(Fired)(-) - \%Corr(NotMatched)$$

where (+) indicates a positive firing and (-) indicates a negative firing.

If either DH factor of a classifier is a large positive value, it indicates a hierarchical structure. By using the visualisation tool, it is found that the default rule ######111 has a high DH factor (+19%)

and the red overlapping circles are the exception rules. The conditions and actions of these rules are shown in the following table:

	Cond	*Act*	*Pre*	*DH factor(+)*	*DH factor(-)*
Default rule	######111	1	600	0.1917	0.04790
Exception rules	##1#0##1#	0	707	-0.1911	-0.0516
	11#00#1##	0	662	-0.1780	-0.4282
	##0#0##1#	0	727	-0.1342	never fired
	01##011##	0	735	-0.1095	never fired
	#10#0###1	0	685.5846	-0.0918	never fired
	###001###	0	770	-0.0701	never fired
	0#0#0####	0	795	-0.0448	never fired
	0#100#1##	0	804	-0.0438	never fired
	#0##0####	0	793	-0.0386	never fired

Table 22: Default hierarchies of MILCS rules for the RSA problem

According to the conflict resolution used in MILCS, the action of the highest prediction value is used. As one can see the default rule has the lowest prediction of all. Therefore, whenever both the default rule and the exception rule match the same instance, the action of the exception rule is fired. This demonstrates the expected default hierarchical structure encouragement in MILCS, which is a unique contribution this system. However, this is also expected because each rule is evaluated for mutual information against all other rules, thus encouraging the development of DHs.

However, DHs have not been observed in the multiplexor because non-hierarchical rule sets are easily derived and problems like the multiplexers probably are most likely too simple to test the DH capacity of the system. This is further confirmed in Chapter 7 and the chapter also shows that DHs can be discovered in CN.

6.3 Summary of chapter

This chapter briefly reviews knowledge discovery through machine learning and reveals the ability of MILCS and other machine learning techniques to handle a couple of real-world applications.

Section 6.1 further extends Subsection 2.1.3 about classification and 2.3.4 on data mining. The process of KDD is covered in a simplified form and literature on KDD through machine learning has been reviewed.

Section 6.2.1 shows the performance of MILCS on a real-world protein structure prediction problem and compares it against XCS, UCS and GAssist. There is no significant statistical difference among MILCS, GAssist, C4.5 and Naïve Bayes in terms of performance accuracy. However, results of MILCS are statistically better than those of XCS and/or UCS. In terms of final solution size, MILCS is similar to GAssist, which is the most compact, but achieved in far greater number of learning steps.

Section 6.2.2 introduces a tougher PSP problem called RSA. Although there is no statistically significant difference between MILCS and the other LCSs, both C4.5 and Naïve Bayes performs significantly better than MILCS. What is more, MILCS manages to evolve a second most compact rule set after BioHEL. It can also be seen that the convergence time difference between MILCS and BioHEL is large. MILCS also shows some signs of hierarchical structures in the final rule set using the visualisation tool.

Chapter 7

Encouraging DHs in MILCS

To further investigate MILCS's DHs ability, a few simple test problems have been designed for MILCS. Since the discovery of the hierarchical RSA results, new subsumption and deletion algorithms have been included in MILCS for promoting DHs. Previous experiments have been re-run for performance comparison.

7.1 New subsumption and deletion algorithms

In addition to the subsumption and deletion algorithms described in Subsection 3.3.5, a couple of new algorithms have been included in MILCS. These algorithms are believed to further encourage hierarchical rule sets. In order to avoid confusion, the old subsumption is now named *subsumption by prediction* due to its use of a classifier's *Pre* and *~Pre*. Similarly, the old deletion method is named *deletion by acting* because the least active classifier is the first one to be deleted. The following sections describe the new algorithms:

7.1.1 Subsumption by acting

A couple of classifier parameters and a MILCS parameter have also been added for this algorithm.

Classifier parameters:

- *Perf, ~Perf* Performance metrics used for "subsumption by acting"

MILCS parameter:

- α_{del} Maximum reward multiplier for "deletion by prediction" and "subsumption by acting"

Perf is calculated by,

$$Perf = \frac{number\ of\ errors\ when\ matched}{number\ of\ matches}$$

Similarly,

$$\sim Perf = \frac{number\ of\ errors\ when\ not\ matched}{number\ of\ not\ matches}$$

Both *Perf* and *~Perf* are updated at the end of the supervised learning component and reset to 0 once the classifier has participated in the GA (at the same time when *MI* metrics are being reset).

Imagine C_1 is subsuming C_2; the following criteria are shared by both old and new subsumption algorithms:

- Both C_1 and C_2 have the same *act* and *~act*
- Both C_1 and C_2 have to be mature enough. In other words, both rules have to be exposed to at least mat_{sub} number of input instances.

Here is the new part:

- Both C_1 and C_2 have *Pred* and *~Pred* greater than $\alpha_{del} \times maximum\ reward$
- Either C_1 has an equal or higher *Perf* than both *Perf* and *~Perf* of C_2, or C_1 has an equal or higher *~Perf* than both *~Perf* and *Perf* of C_2

7.1.2 Deletion by prediction

Similar to deletion by acting, this method also has two modes and only applied on rules with *mat* greater than mat_{del}.

- Single rule deletion

The algorithm searches for classifiers with both *Pre* and *~Pre* less than $\alpha_{del} \times maximum\ reward$. The classifier with the lowest of all is selected to be deleted.

- Batch rule deletion

Once the system performance reaches $\theta_{perf} \times maximum\ reward$, all rules with both *Pre* and *~Pre* less than $\alpha_{del} \times maximum\ reward$ are removed from the population.

One will notice that the parameter α_{del} has a crucial influence on the performance of the new algorithms. It is highly problem dependant and a high setting will dramatically increase the chance of deletion but reduces the chance of subsumption. Therefore, it is recommended to perform several runs of experiments with different settings.

7.2 **Default hierarchy test problems**

As mentioned in Subsection 2.3.5, default hierarchy structure is a very import advantage of LCS. Previous studies have shown that DHs is very difficult to encourage in a LCS. While the core of MILCS is mutual information based fitness, it is built in a way to potentially encourage hierarchical structures. To further investigate MILCS's DHs ability, a few simple test problems have been designed.

7.2.1 Problem description

Imagine the following situation: where a small box is within a bigger box. The area of the bigger box is of class 0 except the area inside the smaller box which is of class 1. Another set of boxes (which are mirror image of the former) are placed next to them. Figure 62 is the graphical description of the problem,

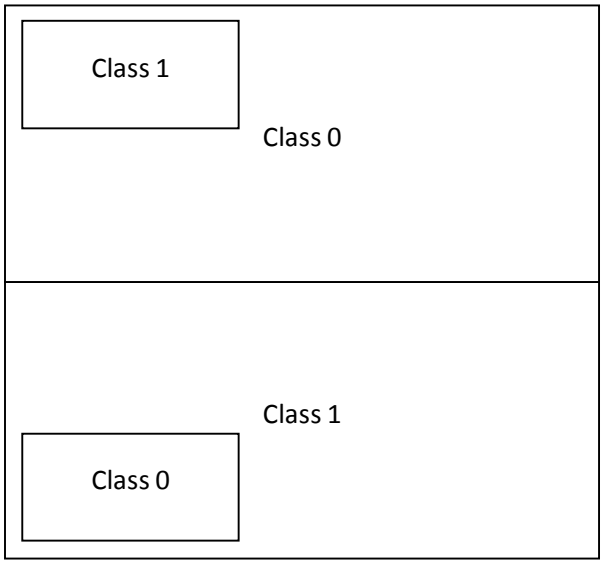

Figure 62: Default hierarchy test problem

To encode the problem into 6 bit binary problem,

	condition	class
top small box	000000	1
top big box	0#####	0
bottom small box	100000	0
bottom big box	1#####	1

The small boxes act as exception rules and the big boxes act as default rules. If a LCS is able to derive hierarchical structure, the final solution should only consist the above four rules. More small boxes, i.e., exception rules can be added to the problem to test the DHs ability further. Therefore, MILCS is tested on 6 bit DHs with 2, 4 and 6 exception rules.

7.2.2 Parameter settings

The parameter settings for MILCS are shown in Table 23 whereas the XCS and UCS parameter settings are in Table 24. Note that the *initializePopulation* parameter is set to *yes* here for the first time. Without initialising population, XCS is not able to achieve the results shown in Table 25 even after millions of runs.

System	MILCS
N	1600
N_I	1500
$P_{\#}$	0.33
θ_{GA}	25
θ_{sub}	20
mat_{act}	256
mat_{del}	320
mat_{sub}	320
del_{range}	1024
$freq_{del}$	0.0068
$freq_{inc}$	0.0
$freq_{max}$	0.0
α_F	1.0
α_P	1.0
P_I	0.0
F_I	0.01
χ	0.8
μ	0.04
crossoverType	uniform
deletionType	prediction
subsumptionType	acting
α_{del}	0.9
θ_{perf}	1.0
tournamentSize	0.4
selectTolerance	0.001

Table 23: MILCS parameters for the DHs problems

System	XCS	UCS
N	1600	1600
β	0.2	0.2
α	0.1	0.1
ε_0	0.01	N/A
ν	5	10
γ	0.71	N/A
θ_{GA}	25	25
χ	0.8	0.8
μ	0.04	0.04
θ_{del}	20	20
δ	0.1	0.1
θ_{sub}	20	20
$P_{\#}$	0.33	0.33
P_I	10.0	N/A
ε_I	0.0	N/A
F_I	0.01	0.01
P_{explr}	0	N/A
Θ_{mna}	2	2
crossoverType	uniform	uniform
tournamentSize	0.4	0.4
selectTolerance	0.001	N/A
doGASubsumption	Yes	Yes
doActionSetSubsumption	Yes	N/A
acc_0	N/A	0.99
initializePopulation	yes	N/A

Table 24: XCS and UCS parameters for the DHs problems

7.2.3 Results and analysis

Number of exceptional rules	Methods	Final rule set size	Max evals
2	XCS	161.2±10.3	20000
2	UCS	78.7±11.6	20000
2	MILCS	7.6±1.6	20000
4	XCS	143.0±8.7	20000
4	UCS	83.0±13.2	20000
4	MILCS	7.6±1.8	20000
6	XCS	134.6±9.7	20000
6	UCS	89.7±16.9	20000
6	MILCS	13.5±2.5	20000

Table 25: Results of XCS, UCS and MILCS on the DHs problems

As mentioned in Subsection 7.2.2, XCS has difficulties learning this problem without *initializePopulation* turned *on*. A possible explanation for this is that XCS finds the exceptional cases difficult to capture with the only help of covering algorithm. Covering a small input space with lower "hit rate", i.e., exceptional conditions, can be quite tricky. Therefore, initialising the population provides XCS with a better view of the overall input space. While UCS is performing better than XCS, MILCS is the only LCS achieving a hierarchical final population with the smallest size. Note that MILCS has not achieved the optimal rule set sizes which are 4, 6 and 8 respectively. This is expected due to the difference between *matact* and *matdel*. Rules with *mat* ranging between these two values always remain in the system along with the optimal solution. They can be regarded as "buffer rules".

Another test for investigating the DHs complexity further was also proposed. Imagine in Figure 62, there is a third box of the same class of the outer box but inside the inner box. In other words, it is a layered default hierarchical structure. Unfortunately, MILCS failed to capture the inner most boxes thus its ability is limited to the current level.

7.3 **Default hierarchies in the PSP problems**

Although MILCS is able to solve both CN and RSA problems with competitive results, human observation of the evolved rules reveals that it does not discover the entire input space. In the case of the CN problem, there exist some "end of chain" conditions which are of very small proportion of the training and test data. The new deletion and subsumption algorithms have been used to test the system once again to compare the difference. Note that both multiplexer and parity problems have also been tested but there was no noticeable difference. This can further confirm that these problems are too "flat" to form hierarchical structures.

7.3.1 Default hierarchies in the CN problem

The new parameters are shown in Table 26.

Window size	**1**	**2**	**3**
N	100	200	1000
N_I	90	190	900
$P_\#$	0.66	0.66	0.66
θ_{GA}	25	25	25
θ_{sub}	20	20	20
mat_{act}	128	150	300
mat_{del}	192	300	450
mat_{sub}	192	300	300
del_{range}	2500	2500	2500
$freq_{del}$	0.0004	0.0004	0.0004
$freq_{inc}$	0.0	0.0	0.0
$freq_{max}$	1.0	1.0	1.0
α_F	1.0	1.0	1.0
α_P	1.0	1.0	1.0
P_I	0.0	0.0	0.0
F_I	0.01	0.01	0.01
χ	1.0	1.0	1.0
μ	0.01	0.01	0.01
crossoverType	two-point	two-point	two-point
deletionType	prediction	prediction	prediction
subsumptionType	acting	acting	acting
α_{del}	0.6	0.6	0.6
θ_{perf}	1.0	1.0	1.0
tournamentSize	0.4	0.4	0.4
selectTolerance	0.001	0.001	0.001

Table 26: MILCS DHs parameters for the CN problem

This set of parameter uses "deletion by acting" and "subsumption by acting", which are potentially encouraging the hierarchical structure. With this set of parameters, MILCS is able to discover rules of the entire input space, and achieve even better performance, but with a slightly larger solution size. The new performance and solution size are shown in Table 27. Note that the larger final rule

set size is expected, because of all the exception conditions missed by the previous runs have been discovered.

Window size	Method	Performance		Solution size	Max evals
1	XCS	58.0% ±5.4%	•	58.1±3.1	300000
1	UCS	62.1% ±1.8%		47.6±7.0	300000
1	MILCS	63.4% ±0.7%		4.4±2.1	300000
1	GAssist	63.6% ±0.6%		4.0±0.0	8000000
1	C4.5	63.6%±0.6%		12.0±0.0	N/A
1	Naïve Bayes	63.6%±0.6%		N/A	N/A
1	DHs_MILCS	63.6%±0.6%		7.2±1.6	300000
2	XCS	60.1% ±2.9%	•	151.2±5.3	300000
2	UCS	60.6% ±3.9%	•	139.6±8.5	300000
2	MILCS	63.8% ±0.6%		8.8±2.1	300000
2	GAssist	63.9% ±0.6%		4.0±0.0	8000000
2	C4.5	63.9%±0.6%		12.0±0.0	N/A
2	Naïve Bayes	63.9%±0.6%		N/A	N/A
2	DHs_MILCS	63.9%±0.6%		16.3±3.1	300000
3	XCS	61.9% ±2.6%		255.2±6.4	300000
3	UCS	60.3% ±3.3%	•	248.6±5.9	300000
3	MILCS	63.9% ±0.8%		17.3±5.9	300000
3	GAssist	64.4% ±0.5%		5.1±1.0	8000000
3	C4.5	64.4%±0.5%		24.9±4.5	N/A
3	Naïve Bayes	64.3%±0.5%		N/A	N/A
3	DHs_MILCS	64.1%±0.5%		24.3±6.0	300000

Table 27: DHs results of MILCS on the two-state CN problem comparing to previous results

7.3.2 Default hierarchies in the RSA problem

The new parameters are shown in Table 28. However, these settings are quite different from Table 19. It is suggested that there was a sign of over-learning judging by the results in Table 29. Using the new algorithms, MILCS is able to learn more accurately, much faster and to a smaller solution size. It is now out-performed BioHEL in terms of accuracy and achieved the smallest solution size of all.

Window size	**4**
N	100
N_I	50
$P_{\#}$	0.66
θ_{GA}	25
θ_{sub}	20
mat_{act}	128
mat_{del}	135
mat_{sub}	135
del_{range}	2500
$freq_{del}$	0.0004
$freq_{inc}$	0.0
$freq_{max}$	1.0
α_F	1.0
α_P	1.0
P_I	0.0
F_I	0.01
χ	1.0
μ	0.01
crossoverType	two-point
deletionType	prediction
subsumptionType	acting
α_{del}	0.7
θ_{perf}	1.0
tournamentSize	0.4
selectTolerance	0.001

Table 28: MILCS DHs parameters for the RSA problem

Method	Performance	Solution size	Max evals
XCS	66.1% ±1.6%	1094.6±44.0	1600000
UCS	67.1% ±6.6%	566.7±28.7	1600000
MILCS	65.6% ±1.2%	74.7±12.7	1600000
BioHEL	66.6% ±0.4%	33.4±4.8	125950000
C4.5	67.4%±0.2%	61.2±8.6	N/A
Naïve Bayes	67.2%±0.2%	N/A	N/A
DHs_MILCS	66.7%±0.4%	10.2	100000

Table 29: DHs results of MILCS on the RSA problem comparing to previous results

7.4 Summary of chapter

This chapter investigates the default hierarchy ability of MILCS. The new subsumption and deletion algorithms have been very successful and the potential of MILCS's DHs ability has been further released.

Section 7.1 gives details of the new algorithms included in MILCS for encouraging DHs.

Section 7.2 describes a series of designed DHs test problems. They have been run on MILCS and the results are compared to XCS and UCS. While evolving the most compact final rule set, MILCS is able to form hierarchical structure. However, its complexity (in terms of "box within box") is limited to the current level.

Section 7.3 shows how the new algorithms have affected the performance of MILCS on the PSP problems. Performances of both CN and RSA have been improved comparing to the previous results. MILCS has even out-performed BioHEL in terms of both accuracy and solution size.

This section concludes the contribution of this thesis. A global conclusion and further work is discussed in the next chapter.

Chapter 8

Conclusions and Further Work

8.1 Global conclusions of the thesis

Before extracting conclusions from the research, it is necessary to summarise the work done so far.

Chapter 2 contains all the necessary background material of the primary subject of this thesis. It has five sections covering machine learning, GA, GBML, ANN and Shannon's information theory respectively. In this chapter, it is revealed that the focus of this thesis is supervised learning, especially classification, a data mining task. After reviewing the most widely-used knowledge representations in the literature, a comparison is drawn, showing that the rule-based learning classifier system has a significant asset in its advanced structural learning via evolutionary computation. The associated generalisation, parsimony and explanatory power, is another contribution the thesis seeks to provide. Several state-of-the-art LCSs from different approaches have been studied and their applications in the field of data mining have been reviewed. Default hierarchies, an important feature of LCSs, has also been discussed. What is more, an ANN architecture and learning algorithm called CC is also studied as a structural learning algorithm. Mutual information, which is central to the contributions of Shannon's information theory, also plays a vital part in the learning system this thesis presents. Several entropy-based machine learning systems have also been covered.

The main contribution of this thesis is presented in Chapter 3. After drawing an analogy between XCS and CCN, it is found that the difference lies in the origin of LCS: reinforcement learning. Since CCN also has structural learning ability, the motivation for an integration of the two is clear. However, it is revealed that correlation used in CCN has certain weaknesses. Therefore, mutual information is suggested as a well-founded metric for dependence, and it is used as the fitness function in the suggested system. By integrating CC and LCS, and using MI as fitness, this thesis presents an innovative GBML system called MILCS. All four stages of its learning processes are described in detail and an algorithmic description of the system can be found in Appendix.

As previously stated, explanatory power is important for machine learning systems. In order to analyse the explanatory power of the results yielded from different learning systems examined in this thesis, a new method to visualising classifiers is introduced in Chapter 4. A metric is suggested to measure the visualisation in a quantitative way. On the other hand, due the complete map approach used by XCS often resulting a large population, several rule compaction algorithms in the literature are covered in this chapter. However, most of these algorithms are able to achieve such a goal by sacrificing performance and possibly over-fitting to the training data.

A couple of basic benchmark problems are presented in Chapter 5. The scale-up of MILCS is compared to XCS and UCS. The multiplexer problems are tested and the explanatory power of the final rule set are visualised. These benchmark results suggest that the scale-up of MILCS is slightly worse than that of XCS and UCS. However, it is not of significant order and the tool reveals that MILCS manages to evolve a parsimonious rule set with strong explanatory power. This is expected from the concept of "structural learning" embedded in this system. A second problem, named even parity, is tested on MILCS, XCS and UCS to further study their performance and explanatory power. Once again, although MILCS converges slower than both XCS and UCS with larger problem sizes, it always results the most compact final population.

Chapter 6 reveals the ability of MILCS in discovering useful knowledge from real-world data source. Knowledge discovery is a long goal in the machine learning field. A simplified process of knowledge discovery is covered in this chapter along with a literature review of such application using machine learning techniques. Protein structure prediction is a fundamental and difficult problem in computational biology and of significant, current scientific interest. A couple of classification tasks have been selected and tested on MILCS against other machine learning techniques.

The first problem is called the CN problem. MILCS results are compared against XCS, UCS GAssist, C4.5 and Naïve Bayes. Results show that there is no significant statistic difference among MILCS, GAssist, C4.5 and Naïve Bayes, although MILCS has significantly less learning steps than GAssist. However, results of MILCS are statistically better than those of XCS and UCS. In terms of solution size, MILCS is similar to GAssist, which is the most compact.

A tougher PSP problem, called RSA, is the next to test. Although there is no statistically significant differences in classification performance between MILCS and the other LCSs (XCS, UCS and BioHEL), the difference between MILCS and the rest (C4.5 and Naïve Bayes) is significant. Any comparison of accuracies alone is not an entirely fair if the final population size is not taken in account. One of the aims of this thesis is the demonstration of MILCS's explanatory power. MILCS manages to evolve the second most compact rule set after BioHEL. It can also be seen that the BioHEL takes far longer to converge than MILCS. What is more, DHs can also be observed from the final rule set evolved by MILCS using the visualisation tool. This superior effect is expected, given the firm information theoretic basis of the mutual information fitness function.

Chapter 7 further investigates the default hierarchy ability of MILCS. Modifications have been made to encourage DHs in MILCS. In order to explore the DHs complexity within MILCS, a suite of basic test problems were set up and tested on XCS, UCS and MILCS. MILCS is able to evolve hierarchical rule sets as expected with increasing number of exception rules. However, its complexity in terms of "box within box" is limited to the current level. All the previous experiments have been re-run and the results of both CN and RSA have been improved. However, there is no noticeable improvement on the multiplexer or the parity problem, possibly due to easily derived non-hierarchical rules. Note that the DHs is a unique contribution of MILCS, as encouraging the development of DHs LCS has proven difficult in past studies.

All the experimental results suggest the research on a new, innovative, mutual-information-based supervised learning classifier system is positive. The performance of MILCS is very competitive with several of the best machine learning systems in the literature. Computational cost tends to have a much larger impact on Pitt-styled and iterative learning LCSs. Also, MILCS yields a final rule set with a very compact size and strong explanatory power. Although certain compaction algorithms can further reduce the population size of XCS and UCS, they typically cause overfitting and a performance drop. DHs, one of the most anticipated advantages of LCSs, are also observed.

Nevertheless, these comparisons show that there is still room for possible improvement. Experimental run time can become an issue for the current version of MILCS when running large

problems, due to the rule size reduction (subsumption) algorithm and MI metrics update. MILCS can benefit enormously with a well designed data structure.

8.2 **Further work**

8.2.1 Multi-class problems

So far problems with two classes (0 and 1), i.e. *binary-class problems*, have been experimented on MILCS. However, many real world problems often contain a high number of classes (e.g. pattern recognition). Thus, they are called *multi- class problems*. Although none of the multi-class problems has been tested on MILCS, it is able to perform such tasks with no further change to the system if it is designed following the algorithmic description of MILCS found in the Appendix.

Recall that in the mutual information formula (see Subsection 3.2.3), the distribution of the error, X, has two values, "there is an error" or "there is no error". This is used for binary-class problems. For multi-class problems, X has multiple values. Each is associated with the degree of erroneousness. For example, imagine a three-class problem (0, 1, 2) and for a given state, the correct action/class is 0. The possible response of the system can be 0, 1 or 2. Thus X has three values which are "there is an error and it is off by 1", "there is an error and it is off by 2" or "there is no error". Along with the distribution of rule's response (two values, "match" or "not match"), Y, there are 6 (i.e., 3×2) combinations and the mutual information fitness the sum of them.

8.2.2 Beyond rule learning

The concepts in MILCS are not specific to the particulars of a rule learning system. Exploring a neural network system that employs a similar structural learning paradigm is also a promising direction for future investigation. Mutual Information Neuro-Evolutionary System (MINES) [100] is one of such learning systems. This approach automatically determines the optimal quality and connectively of the hidden layer of a three-layer feed-forward neural network using mutual information. Moreover, the use of mutual information in this fashion may also have application in supervised learning of other knowledge representations.

8.2.3 Reinforcement learning

While the focus of this work has been on supervised learning, it is possible that the system may be adapted to reinforcement learning. Note that to some extent, XCS already adapts reinforcement learning to supervised learning, in its tendency to learn a complete "model" of the long term payoff function across the state/action space. The mapping from state/action to payoff is a supervised learning problem, drawing on Bellman optimality and Q-learning the appropriate target values and error functions.

8.2.4 “Cascade” of CCN

The first ‘C’ in CCN is studied so far in this thesis, which is ‘Cascaded’. In CCN, hidden nodes are added to network in a cascaded manner. Could it be the same when inserting rules to the population in MILCS? How well does MILCS benefit from it? It will be interesting to investigate.

8.2.5 Improving the visualisation tool for explanatory power

Although the tool has been proved to be useful when visualising the classifiers generated from the multiplexer problems, for problems like CN, the technique seems incapable. This is because this technique is based on the diversity of the problem instances. If the problem instances have a similar pattern and a very subtle difference, the tool may not visualise the classifiers well and instead, all the classifiers are clustered together. Therefore some modification to overcome this problem is necessary. It will also be interesting to allow comparing explanatory power in a larger variety of knowledge representations other than rule-based.

Appendix

9.1 Algorithmic description of MILCS

This section presents the algorithms used in MILCS in addition to Section 3.3. The description starts from the top level. The overall execution cycle is described first and the individual parts are specified in more in the successive paragraphs. Each specific sub-procedure in this description is written in capital letters.

9.1.1 Initialisation and main loop

When MILCS is started, the modules must first of all be initialised. The parameters in the environment must be set and any input data file must be read in. Meanwhile, the supervised program must be initialised. Finally, MILCS itself and the population [P] must be initialised. The population must be filled with the initial size N_I, generating each classifier with a random a condition and action and initial parameter values. After the initialization, the main loop is called.

MILCS:

1. Initialise environment *env*
2. Initialise supervised program *sp*
3. Initialise MILCS
4. RUN EXPERIMENT

The main loop *RUN EXPERIMENT* consists 2 parts, EXPLORATION and EXPLOITATION. The current situation is first sensed (received as input) and the EXPLORATION is run followed by EXPLOITATION. There is another loop *UPDATE MI METRICS* in EXPLORATION. In this loop, the first classifier is removed from the population [P] to form [P']. Secondly, the prediction array *PA'* is formed based on all the classifiers in [P']. *PA'* predicts for each possible action a_i *'* (both matched and not-matched) the resulting payoff. Based on *PA'*, one action is chosen for execution. Next, the winning action is executed and the current reward *p'* is used to update the mutual information MI metrics of the removed classifier. Then, this classifier must be added back to the population. This loop *UPDATE MI METRICS* must be repeated on all successive classifiers in [P] until all rules are updated. Once this loop is finished, *PA* is then formed using the entire population [P] and one action is chosen to execute. The winning action is executed and the current reward *p* is used to adjust [P] using supervised learning. Depending on the chosen action, either the match set or

the not-match set [M], is set. The action set [A] is formed, which includes all classifiers of either [M] that propose the chosen action. Fitness of classifiers of [A] are updated and GA is applied to them. EXPLOITATION is for performance monitoring. It only involves forming *PA* to select an action for execution and getting the reward *p*. The population [P] must not be modified during this stage. The main loop is executed as long as the termination criterion is not met. A termination criterion is, e.g., a certain number of trials or a 100% performance level. It will also turn on/off certain deletion algorithms in the EXPLORATION depending on the EXPLOITATION performance.

RUN EXPERIMENT ():

1. do{
2. $\sigma \leftarrow$*env*: get situation
3. EXPLORATION
4. EXPLOITATION
5. }while (termination criteria are not met)

EXPLORATION(σ):

1. UPDATE MI METRICS
2. GENERATE PREDICTION ARRAY *PA* out of [P]
3. *act* ← SELECT ACTION according to *PA*
4. *env*: execute action *act*
5. ADJUST POPULATION SET [P]
6. if *act* is matched action
7. GENERATE SET [M] out of [P] using σ
8. GENERATE ACTION SET [A] out of [M] according to *act*
9. else if *act* is not-matched action,
10. GENERAET SET [M] out of [P] by using σ
11. GENERATE ACTION SET [A] out of [~M] according to *act*
12. ADJUST ACTION SET [A] and subsuming in [A]
13. RUN GA in [A] and possibly deleting in [P]

EXPLOITATION(σ):

1. GENERATE PREDICTION ARRAY *PA* out of [P]
2. *act* ← SELECT ACTION according to *PA*
3. *env*: execute action *act*
4. *p* ← *sp*: get reward
5. PERFORMANCE MONITOR

9.1.2 Sub-procedures

This section covers all relevant processes specified in the main loop.

9.1.2.1 The mutual information metrics update

The *UPDATE MI METRICS* procedure removes one of the classifiers of [P] to form [P']. The prediction array *PA'* is then formed based on all the classifiers in [P']. *PA'* predicts for each possible action a_i' (both matched and not-matched) the resulting payoff. Based on *PA'*, one action is chosen for execution. Next, the winning action is executed and the current reward *p'* is used to update the mutual information MI metrics of the removed classifier. Then, this classifier must be added back to the population. This loop *UPDATE MI METRICS* must be repeated on all successive classifiers in [P] until all rules are updated.

UPDATE MI METRICS([P], σ):

1. for each classifier *cl* in [P]
2. remove *cl* from [P]
3. GENERATE PREDICTION ARRAY *PA* out of [P] using σ
4. *act* ← SELECT ACTION according to *PA*
5. *env*: execute action *act*
6. *p* ← *sp*: get reward
7. if (DOES MATCH classifier *cl* in situation σ)
8. for each payoff level *pol*
9. if (*p* = PayoffLevel(*pol*))
10. $MI[pol][1]_{cl}$++
11. else
12. for each payoff level *pol*
13. if (*p* = PayoffLevel (*pol*))
14. $MI[pol][0]_{cl}$++
15. add *cl* to [P]

The sub-procedure GENERATE PREDICTION ARRAY and DOES MATCH can be found in Subsection 9.1.2.2.

9.1.2.2 The prediction array

Given an input, MILCS makes a "best guess" prediction of the payoff to be expected for each possible action. These *system predictions* are stored in a vector called the Prediction Array *PA*. The system prediction for an action is the maximum prediction of all classifiers in [P] that advocate that action. If no classifier in [P] advocates a certain action, its system prediction is not defined, symbolised by null.

GENERAGE PREDICTION ARRAY([P], σ):

1. initialise prediction array *PA* to all null
2. for each classifier *cl* in [P]
3. if ($noOfTrainings_{cl} > mat_{act}$)
4. if (DOES MATCH classifier *cl* in situation σ)
5. //for matched actions
6. for each possible action *A*
7. if ($PA[A][1] < Pre_{cl}$)
8. $PA[A][1] \leftarrow Pre_{cl}$
9. else
10. //for not-matched actions
11. for each possible action *A*

12. if ($PA[A][0] < \sim Pre_{cl}$)
13. $PA[A][0] \leftarrow \sim Pre_{cl}$
14. Return *PA*

DOES MATCH(cl, σ):

1. for each attribute x in C_{cl}
2. if ($x \neq$ # and $x \neq$ the corresponding attribute in σ)
3. return *false*
4. return *true*

9.1.2.3 Choosing an Action

MILCS employs a simple deterministic action selection method. The action with the highest system prediction is chosen. Furthermore, the current iteration time is recorded for the firing classifier for the winning frequency *freq* calculation in Subsection 9.1.2.4. Note that match type *m* has only two values, 1 for matched and 0 for not-matched.

SELECT ACTION(PA, [P], itTime):

1. initialise maximum system prediction *max* to 0.0
2. initialise winning action *act* to 0
3. initialise action type *actType* to 1
4. for each match type *m*
5. for each possible action *A*
6. if ($max < PA[A][m]$)
7. $max \leftarrow PA[A][m]$
8. $act \leftarrow A$
9. $actType \leftarrow m$
10. else if ($max = PA[A][m]$ and $act \neq A$ && $actType = m$)
11. $act \leftarrow$ RandomNumber[0,1] * number of possible actions
12. else if ($max = PA[A][m]$ and $act \neq A$ && $actType \neq m$)
13. $act \leftarrow$ RandomNumber[0,1] * number of possible actions
14. $actType \leftarrow$ RandomNumber[0,1]
15. else if ($max = PA[A][m]$ and $act = A$ and $actType \neq m$)
16. $actType \leftarrow$ RandomNumber[0,1]
17. for each classifier *cl* in [P]
18. if ($actType = 1$ and $Pred_{cl} = max$)
19. record *itTime*
20. else if ($actType = 0$ and $\sim Pred_{cl} = max$)
21. record *itTime*
22. return *act*

9.1.2.4 Adjust Population Set via Supervised Learning

Being a supervised learning system, MILCS tunes actions and action predictions of its classifiers. In addition, it updates the winning frequency of each classifier for deletion, which is described in .

ADJUST POPULATION SET([P], reward, σ):

1. initialise reward ρ to 0.0
2. for each classifier *cl* in [P]

3. $noOfTrainings_{cl}$++
4. //The matched action
5. if (DOES MATCH classifier *cl* in situation σ)
6. $noOfMatches_{cl}$++
7. nom_{cl}++
8. if (*reward* = maximum reward)
9. nem_{cl}++
10. for each possible action *A*
11. $\rho \leftarrow sp$: get reward
12. $rewardSum[A][1]_{cl} \leftarrow rewardSum[A][1]_{cl} + \rho$
13. if (ρ = maximum reward)
14. $actCount[A][1]_{cl}$++
15. //The not-matched action
16. else
17. non_{cl}++
18. if (*reward* = maximum reward)
19. nen_{cl}++
20. for each possible action *A*
21. $\rho \leftarrow sp$: get reward
22. $rewardSum[A][0]_{cl} \leftarrow rewardSum[A][0]_{cl} + \rho$
23. if (p = maximum reward)
24. $actCount[A][0]_{cl}$++
25. $A_{cl} \leftarrow$ highest $actCount[A][1]$ for each action *A*
26. $\sim A_{cl} \leftarrow$ highest $actCount[A][0]$ for each action *A*
27. $Pre_{cl} \leftarrow (1 - \alpha_P) * Pre_{cl} + \alpha_P * (rewardSum[A_{cl}][1] / noOfMatches_{cl})$
28. $\sim Pre_{cl} \leftarrow (1 - \alpha_P) * \sim Pre_{cl} + \alpha_P * (rewardsum[A_{cl}][0] / (noOfTrainings_{cl} - noOfMatches_{cl})$
29. $Perf_{cl} \leftarrow nem_{cl} / nom_{cl}$
30. $\sim Perf_{cl} \leftarrow nen_{cl} / non_{cl}$
31. $noOfWins_{cl} \leftarrow$ number of wins in the last del_{range} iterations
32. $freq_{cl} \leftarrow noOfWins_{cl} / del_{range}$

9.1.2.5 Formation of the Match Set or Not-Matched Set

The GENERATE SET procedure gets its members from the current population [P] and under the current situation σ.

GENERATE SET([*P*], σ, *actType):*

1. initialise empty set [M]
2. if (*actType* = 1)
3. //match set
4. for each classifier *cl* in [P]
5. if (DOES MATCH classifier *cl* in situation σ)
6. add classifier *cl* to set [M]
7. else
8. //not-match set
9. for each classifier *cl* in [P]
10. if (DOES not MATCH classifier *cl* in situation σ)
11. add classifier *cl* to set [M]
12. return [M]

The sub-procedure DOES MATCH can be found in Subsection 9.1.2.1.

9.1.2.6 Formation of the Action Set

After the match/not-match set [M] is formed and an action is chosen for execution, the GENERATE ACTION SET procedure forms the action set out of [M]. It includes all classifiers that propose the chosen action for execution.

GENERATE ACTION SET([M], act):

1. initialise empty set [A]
2. for each classifier *cl* in [M]
3. if ($A_{cl} = act$)
4. add classifier *cl* to set [A]

9.1.2.7 Update Classifier fitness

The update procedure is applied in action set [A]. Each time a classifier enters the set [A], its experience *exp* and fitness *F* is updated. In addition, the procedure calls the DO ACTION SET SUBSUMPTION procedure to eliminate a large number of classifiers in the action set. Note that [A[then shrinks down to contain classifiers with *mat* greater than mat_{sub} only to allow more trained classifiers participate in the GA next.

ADJUST ACTION SET([A], [P]):

1. for each classifier *cl* in [A]
2. initialise *noOfTrainings* to 0
3. initialise *noOfMatches*[] to 0
4. initialise *noOfErrors*[] to 0
5. exp_{cl}++
6. for each match type *m*
7. for each possible payoff level *pol*
8. $noOfTrainings \leftarrow noOfTrainings_{cl} + MI[pol][m]_{cl}$
9. $noOfMatches[m] \leftarrow$ noOfMatches[*m*] + $MI[pol][m]_{cl}$
10. $noOfErrors[pol] \leftarrow$ noOfErrors[*pol*] + $MI[pol][m]_{cl}$
11. for each match type *m*
12. for each possible payoff level *pol*
13. $F_{cl} \leftarrow F_{cl} + (MI[pol][m]_{cl} / noOfTrainings) * \log2((MI[pol][m]_{cl} / noOfMatches[m]) / (noOfErrors[pol] / noOfTrainings))$
14. DO ACTION SET SUBSUMPTION in [A] updating [P]
15. for each classifier cl in [A]
16. if ($mat_{cl} < mat_{sub}$)
17. remove *cl* from [A]

9.1.2.8 The Genetic Algorithm in MILCS

The RUN GA sub-procedure is borrowed from XCS. It first checks the action set to see if the GA should be applied at all. In order to apply a GA the average time period since the last GA application in the set must be greater than the threshold θ_{GA}. Next, two classifiers (i.e. the parents) are selected by tournament selection based on fitness *F* and maturity mat_{GA}. Then the offsprings are created out of them. After that, the offspring are possibly crossed and mutated. If the offsprings are

crossed, their fitness are set to be the average of the parents' values. In addition, if the offsprings are crossed or mutated, their action counters *actCount*[] and winning frequency *freq* are set to 0. Finally, the offspring are inserted in the population, followed by corresponding deletion. The sub-procedure *INSERT IN POPULATION* and *DELETE FROM POPULATION* are defined in Subsection 9.1.2.9.

RUN GA([A], σ, [p], itTime):

1. if ($itTime - \sum_{cl \in [A]} ts_{cl} * num_{cl} / \sum_{cl \in [A]} num_{cl} > \theta_{GA}$)
2. for each classifier *cl* in [A]
3. $ts_{cl} \leftarrow itTime$
4. $parent_1 \leftarrow$ SELECT OFFSPRING in [A]
5. $parent_2 \leftarrow$ SELECT OFFSPRING in [A]
6. $child_1 \leftarrow$ copy classifier $parent_1$
7. $child_2 \leftarrow$ copy classifier $parent_2$
8. $Pre_{child1} = Pre_{child2} \leftarrow 0$
9. $\sim Pre_{child1} = \sim Pre_{child2} \leftarrow 0$
10. $num_{child1} = num_{child2} \leftarrow 1$
11. $exp_{child1} = exp_{child2} \leftarrow 0$
12. $noOfTrainings_{child1} = noOfTrainings_{child2} \leftarrow 0$
13. $noOfMatches_{child1} = noOfMatches_{child2} \leftarrow 0$
14. $noOfWins_{child1} = noOfWins_{chid2} \leftarrow 0$
15. $nom_{child1} = nom_{child2} \leftarrow 0$
16. $nem_{child}1 = nem_{child}2 \leftarrow 0$
17. $non_{child1} = non_{child2} \leftarrow 0$
18. $nen_{child1} = nen_{child2} \leftarrow 0$
19. $Perf_{child1} = Perf_{child2} \leftarrow 0$
20. $\sim Perf_{child1} = \sim Perf_{child2} \leftarrow 0$
21. $actCount[][]_{child1} = actCount[][]_{child2} \leftarrow 0$
22. $MI[][]_{child1} = MI[][]_{child2} \leftarrow 0$
23. $rewardSum[][]_{child1} = rewardSum[][]_{child2} \leftarrow 0$
24. if (RandomNumber [0, 1] < χ)
25. APPLY CROSSOVER on *child1* and *child2*
26. $F_{child1} = F_{child2} \leftarrow (F_{parent1} + F_{parent2}) / 2$
27. $actCount[][]_{child1} = actCount[][]_{child2} \leftarrow 0$
28. $freq_{child1} = freq_{child2} \leftarrow 0$
29. for both children *child*
30. if (APPLY MUTATION on *child*)
31. $actCount[][]_{child1} = actCount[][]_{child2} \leftarrow 0$
32. $freq_{child1} = freq_{child2} \leftarrow 0$
33. INSERT *child* IN POPULATION
34. DELETE FROM POPULATION [P]
35. for each classifier *cl* in [A]
36. $nom_{cl} \leftarrow 0$
37. $nem_{cl} \leftarrow 0$
38. $non_{cl} \leftarrow 0$
39. $nen_{cl} \leftarrow 0$
40. $MI[][]_{cl} \leftarrow 0$

Tournament Selection

As the name implies, tournament selection refers to a GA selection process in which tournaments are held among a subset of individuals of a population. Individuals that take part in the tournament

are selected at random from the population. A fixed tournament size is used and the individuals in the tournament with the highest fitness wins and is reproduced.

SELECT OFFSPRING([A]:

```
1.  initialise winnerSet [W]
2.  initialise fitness to -1.0
3.  initialise size to 0
4.  initialise winner to null
5.  while ([W] = null)
6.     for each classifier cl in [A]
7.        if ([W] = null or (fitness – selectTolerance) ≤ F_cl)
8.           for each copy of classifier cl
9.              if (RandomNumber[0,1] < tournamentSize)
10.                if ([W] = null)
11.                   INSERT cl IN [W]
12.                   fitness ← F_cl
13.                   size ← 1
14.                else
15.                   if (fitness + selectTolerance > F_cl)
16.                      INSERT cl IN [W]
17.                      size ++
18.                   else
19.                      [W] ← null
20. INSERT cl IN [W]
21.                      fitness ← F_cl
22.                      size ← 1
23.                   break
24. winner ← randomly select a winner from [W]
25. return winner
```

Crossover

There are three types of crossover in MILCS, uniform, one point and two point crossover. These are similar to the standard crossover procedure in Gas. Thus, the two-point crossover procedure is shown here. Note that classifier actions are not affected by crossover.

APPLY CROSSOVER(cl_1, cl_2):

```
1.  x ← RandomNumber [0,1] * (length of C_cl1 + 1)
2.  y ← RandomNumber [0,1] * (length of C_cl2 + 1)
3.  if (x > y)
4.     switch x and y
5.  i ← 0
6.  do {
7.     if (x ≤ i and i < y)
8.        switch C_cl1[i] and C_cl2[i]
9.     i++
10. } while(i < y)
```

Mutation

While crossover does not affect the action, mutation takes place in both the condition and the action. A mutation in the condition flips the attribute to one of the other possibilities. Mutation in action changes it equiprobably to one in which a die is flipped for each attribute.

APPLY MUTATION(cl):

1. $i \leftarrow 0$
2. *changed* $\leftarrow 0$
3. do {
4. if (RandomNumber [0,1] < μ)
5. C_{cl}[i] $\leftarrow$ a randomly chosen other two possibilities
6. *changed* $\leftarrow 1$
7. i++
8. } while (i < length of C_{cl})
9. if (RandomNumber [0, 1] < μ)
10. $A_{cl} \leftarrow$ a randomly chosen other possible action
11. *changed* $\leftarrow 1$
12. return *changed*

9.1.2.9 Insertion in and Deletion from the Population

This section covers processes that handle the insertion and deletion of classifiers in the current population [P]. The INSERT IN POPULATION procedure checks to see if the classifier to be inserted is identical in condition and action with a classifier already in the population. If so, the latter's numerosity is incremented; if not, the new classifier is added to the population.

INSERT IN POPULATION(cl, [P]):

1. for each c in [P]
2. if (c is equal to cl in condition and action)
3. num_c++
4. return
5. add cl to set [P]

There are two types of deletion algorithms in MILCS, named *DELETION BASED ON PREDICTION* and *DELETION BASED ON ACTING*. In addition, both algorithms can invoke single classifier deletion and batch classifiers deletion depending on the current system performance. Single deletion usually applies until the system performance reaches $\theta_{performance}$, at which point the batch deletion is triggered. Note that this is done in the PERFORMANCE MINITOR sub-procedure of EXPLOITATION.

The *DELETION BASED ON PREDICTION* procedure usually looks for the classifier with the lowest prediction values Pre and ~Pre and once batch deletion is triggered it deletes all classifiers with both prediction values lower than α_{del} * maximum reward. The *DELETION BASED ON ACTING* procedure works by deleting the classifier with the lowest *freq* and batch deletion removes all classifier with freq lower than $freq_{del}$. However, both procedures are implemented in DELETE CLASSIFER and triggered by *deleteType*. Note that this system parameter should not be confused with the *type* parameter for DELETION FROM POPULATION. *Type* is for switching between single deletion and batch deletion of the same algorithm. Each time if the batch deletion is triggered but no classifier gets deleted, $freq_{del}$ can be increased by $freq_{inc}$ until $freq_{max}$ is reached.

DELETE FROM POPULATION([P], type):

1. *flag* $\leftarrow 0$
2. //Batch deletion
3. if (*type* = 1)
4. while (DELETE CLASSIFIER([P])

5. $flag \leftarrow 1$
6. if ($flag = 0$ and $freq_{del} < freq_{max}$)
7. $freq_{del} \leftarrow freq_{del} + freq_{inc}$
8. //Single deletion
9. else
10. DELETE CLASSIFIER([P])

DELETE CLASSIFIER([P]):

1. initialise *c*
2. $deleted \leftarrow 0$
3. //Deletion based on prediction
4. if ($deleteType = 1$)
5. $min \leftarrow \alpha_{del}$ * maximum reward
6. for each classifier *cl* in [P]
7. if ($Pre_{cl} < min$ and $\sim Pre_{cl} < min$ and $noOfTrainings_{cl} \geq mat_{del}$
8. $min \leftarrow Pre_{cl}$
9. if ($\sim Pre_{cl} < Pre_{cl}$)
10. $min \leftarrow \sim Pre_{cl}$
11. $c \leftarrow cl$
12. $deleted \leftarrow 1$
13. //Deletion based on acting
14. else
15. $min \leftarrow freq_{del}$
16. for each classifier *cl* in [P]
17. if ($freq_{cl} < min$ and $noOfTrainings_{cl} \geq mat_{del}$
18. $min \leftarrow freq_{cl}$
19. $c \leftarrow cl$
20. $deleted \leftarrow 1$
21. if (c = null)
22. return *deleted*
23. else if ($num_c > 1$)
24. num_c --
25. else
26. remove classifier *c* from set [P]
27. return *deleted*

PERFORMANCE MONITOR(sysPerf):

1. print and record performance details
2. if ($sysPerf > \theta_{perf}$)
3. if ($freq_{del} < freq_{max}$)
4. $freq_{del} \leftarrow freq_{del} + del_{inc}$

9.1.2.10 Subsumption

THE ACTION SET SUBSUMPTION procedure takes place in every action set [A]. The set is searched for the most general classifier which also meets the "subsumber" criteria. Then all other classifiers (which also to meet the "subsumee" criteria) are tested against the general one to see if it subsumes them. Any classifiers that are subsumed are eliminated from the population. There exists two types of subsumption algorithms in MILCS. The SUBSUMPTION BY PREDICTION has a focus on action prediction whereas SUBSUMPTION BY ACTING is based on each individual's

performance with the rest of the classifiers in mind. Both subsumption procedures are implemented in SUBSUMES procedure triggered by system parameter *subType*.

DO ACTION SET SUBSUMPTION([A], [P]):

1. for each classifier *cl* in [A]
2. if (*cl* COULD SUBSUME)
3. for each classifier *c* in [A]
4. if (*cl* SUBSUMES *c*)
5. $num_{cl} \leftarrow num_{cl} + num_c$
6. remove classifier *c* from set [A]
7. remove classifier *c* from set [P]

COULD SUBSUME(cl):

1. if ($exp_{cl} > \theta_{sub}$)
2. if ($noOfTrainings_{cl} > mat_{sub}$)
3. return *true*
4. return *false*

SUBSUMES(cl1, cl2):

1. $ret \leftarrow 0$
2. if ($subType = 0$)
3. $min \leftarrow \alpha_{del}$ * maximum reward
4. $ret \leftarrow (Pre_{cl1} > min$ or $\sim Pre_{cl1} > min)$ and $(Pre_{cl2} > min$ or $\sim Pre_{cl2} > min)$ and $A_{cl1} = A_{cl2}$ and $\sim A_{cl1} = \sim A_{cl2}$ and *cl1* IS MORE GENERAL than *cl2* and $((Perf_{cl1} \geq Perf_{cl2}$ and $Perf_{cl1} \geq \sim Perf_{cl2})$ or $(\sim Perf_{cl1} \geq \sim Perf_{cl2}$ and $\sim Perf_{cl1} \geq Perf_{cl2}))$ and $noOfTrainings_{cl2} > mat_{sub}$)
5. else
6. $ret \leftarrow (A_{cl1} = A_{cl2}$ and $\sim A_{cl1} = \sim A_{cl2}$ and *cl1* IS MORE GENERAL than *cl2* and $((Pre_{cl1} \geq Pre_{cl2}$ and $Pre_{cl1} \geq \sim Pre_{cl2})$ or $(\sim Pre_{cl1} \geq \sim Pre_{cl2}$ and $\sim Pre_{cl1} \geq Pre_{cl2}))$ and $noOfTrainings_{cl2} > mat_{sub}$)
7. return *ret*

IS MORE GENERAL(cl1, cl2):

1. if (the number of # in $C_{cl1} \leq$ the number of # in C_{cl2})
2. return *false*
3. $i \leftarrow 0$
4. do {
5. if ($C_{cl1}[i] \neq$ # and $C_{cl1}[i] \neq C_{cl2}[i]$)
6. return *false*
7. i++
8. } while ($i <$ length of C_{cl1})
9. return *true*

References

[1] Adami, C. (1994). On Modelling Life. *Artificial Life*, 1, pp. 429–438.

[2] Aha, D. W., Kibler, D. F., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1), pp. 37–66.

[3] Bacardit, J. (2004). *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: representations, generalization, and run-time.* PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain.

[4] Bacardit, J., & Butz, M. (2007). Data Mining in Learning Classifier Systems: Comparing XCS with GAssist. *Learning Classifier Systems*. 4399/2007, pp. 282-290. Springer Berlin/Heidelberg.

[5] Bacardit, J., & Krasnogor, N. (2006). *Biohel: Bioinformatics-oriented hierarchical evolutionary learning. Nottingham eprint*. Technical Report, University of Nottingham.

[6] Bacardit, J., Stout, M., Hirst, J.D., Sastry, K., Llorà, X., & Krasnogor, N. (2007). Automated Alphabet Reduction Method with Evolutionary Algorithms for Protein Structure Prediction, In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO2007)*, pp. 346-353, ACM Press.

[7] Barcardit, J. (2007). Personal communication, March 23, 2007.

[8] Bernadó-Mansilla, E. (2003). Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation* 11(3).

[9] Bernadó-Mansilla, E., Llor`a, X., & Garrell, J. (2001). XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks. *Advances in Learning Classifier Systems: 4th international workshop (IWLCS 2001)*, pp. 115.

[10] Bonelli, P., Parodi, A., Sen, S., & Wilson, S. W. (1990). NEWBOOLE: A fast GBML System. In Porter, B. and Mooney, R., editors, *Seventh International Conference on Machine Learning*, pp. 153.159. Morgan Kaufmann.

[11] Butz, M. & Wilson, S. W. (2001). An Algorithmic Description of XCS. In *Revised Papers From the Third international Workshop on Advances in Learning Classifier Systems*. P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, Lecture Notes In Computer Science,1996, pp. 253-272. Springer-Verlag, London.

[12] Butz, M. V., Kovacs, T., Lanzi, P. L., and Wilson, S. W. (2001). How XCS Evolves Accurate Classifiers. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voigt, H., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., and Burke, E., editors, *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pp. 927-934. San Francisco, CA: Morgan Kaufmann.

[13] Butz, M. V., & Pelikan, M. (2001). Analyzing the evolutionary pressures in XCS. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 935-942.

[14] Butz, M. V. (2003). *Documentation of XCS+TS C-Code 1.2*. IlliGAL report 2003023, University of Illinois at Urbana-Champaign. (Source code: ftp://gal2.ge.uiuc.edu/pub/src/XCS/XCS1.2.tar.Z)

[15] Butz, M. V. (2008). Function Approximation With XCS: Hyperellipsoidal Conditions, Recursive Least Squares, and Compaction. *Evolutionary Computation*. 12(3). Pp. 355-376.

[16] Butz, M. V., Goldberg, D. E., and Tharakunnel, K. (2003). Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*. 11, 3 (Sep. 2003), pp. 239-277.

[17] Butz, M. V., Sastry, K., Goldberg, D. E. (2003). Tournament Selection in XCS. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*. pp. 1857-1869. Springer Verlag, Berlin (2003).

[18] Bull, L., & Hurst, J. (2002). ZCS Redux. *Evolutionary Computation*. 10, 2 (Jun. 2002), pp. 185-205.

[19] Bull, L., & O'Hara, T. (2002). Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith editors, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-2002*, pp. 11-18. Morgan Kaufmann.

[20] Bull, L. (2004). *Applications of Learning Classifier Systems*. Springer.

[21] Bull, L., Bernado-mansilla, E., Homes, J. (2008). *Learning Classifiers Systems in Data Mining*. Springer.

[22] Chang, C. C., & Lin, C. J. (2001). *LIBSVM: a library for support vector machines*. Department of Computer Science and Information Engineering, National Taiwan University. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[23] Chawla, N., Moore, T. E., Bowyer, K.W., Hall, L. O., Springer, C. & Kegelmeyer, W. P. (2001). Bagging-like effects for decision trees and neural nets in protein secondary structure prediction. In *BIOKDD01: Workshop on Data Mining in Bioinformatics at KDD01*, pp. 50-- 59.

[24] Clark, P., & Niblett, T. (1994). The CN2 induction algorithm. *Machine Learning* 15(2) pp. 201-221.

[25] Cuff, J. A., & Barton, G. J. (2000). *Application of Multiple Sequence Alignments Profiles to Improve Protein Secondary Structure Prediction*. *Proteins* 40. pp. 502-511.

[26] Dam, H. H., Abbass, H. A., Lokan, C., Yao, X. (2008). Neural-Based Learning Classifier Systems. IEEE Transactions on Knowledge and Data Engineering, 20 (1), pp. 26-39, January, 2008.

[27] Davis, L. (1989). Mapping Neural Networks into Classifier Systems. In J.D. Schaffer editors, *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 375-378. Morgan Kaufmann.

[28] DeJong, K. A., & Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 651–656. Morgan Kaufmann.

[29] Dixon, P. W., Corne, D., Oates, M. J. (2003). A ruleset reduction algorithm for the XCS learning classifier system. *Learning Classifier System*, pp. 20-39. Springer-Verlag, Berlin.

[30] Domingos, P. (1994). The RISE system: Conquering without separating. *Proceedings of the Sixth IEEE Inter-national Conference on Tools with Artificial Intelligence*, pp. 704–707. New Orleans, LA: IEEE Computer Society Press.

[31] Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, 42, pp. 149-160.

[32] Eiben, A.E. & Smith, J. E. (2007). Introduction to Evolutionary Computing, Springer, *Natural Computing Series*, Corr. 2nd printing.

[33] Fahlman, S. E. (1988). Faster-learning variations on back-propagation: An empirical study. In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *1988 Connectionist Models Summer School*, San Mateo, CA, 1988. Morgan Kaufmann.

[34] Fahlman, S. E., & Lebiere, C. (1990). The Cascade-Correlation learning algorithm. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann.

[35] Farmer, D. (1989). A Rosetta Stone for Connectionism. *Physica* D42, pp.153-187.

[36] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors. (1996) *Advances in Knowledge Discovery and Data Mining*. American Association for Artificial Intelligence.

[37] Fogel, L. J. (1964). *On the organization of intellect*. Doctoral dissertation, University of California, Los Angeles.

[38] Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *Proc. 15th International Conf. on Machine Learning pp*. pp. 144–151. Morgan Kaufmann, San Francisco, CA.

[39] Freitas, A. A. (2002). *Data mining and knowledge discovery with evolutionary algorithms*. Springer-Verlag.

[40] Frey, P. & Slate, D. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6, pp. 161.182.

[41] Friedman, N., Geiger, D., Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29, pp. 131-163.

[42] Fu, C. & Davis, L. (2002). A Modified Classifier System Compaction Algorithm, *GECCO-2002*, pp. 920–925.

[43] Gao, Y., Wu, L., Huang, J. Z. (2006). Ensemble Learning Classifier System and Compact Ruleset. *Simulated Evolution and Learning*, pp. 42-49.

[44] Giordana, A. & Neri, F. (1995). Search-intensive concept induction. *Evolutionary Computation* 3, pp. 375-416.

[45] Greene, D. P. & Smith, S. F. (1994). Using coverage as a model building constraint in learning classifier systems. *Evolutionary Computation* 2(1) pp.67-91.

[46] Goldberg, D. E. (1983). *Computer-aided gas pipeline operation using genetic algorithms and rule-learning*. Ph.D. Thesis, University of Michigan.

[47] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

[48] Heckerman, H. (1996). *A Tutorial on Learning with Bayesian Networks*. Technical Report, MSR-TR-95-06.

[49] Heyer, L., Kruglyak, S., and Yooseph, S. (1999) Exploring Expression Data: Identification and Analysis of Coexpressed Genes, *Genome Research,* 9, pp. 1106-1115. (Java implementation : http://icb.med.cornell.edu/wiki/index.php/QtClustering)

[50] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

[51] Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Hayes-

Roth, D., & Waterman, F. (Eds.), *Pattern-directed Inference Systems*, pp. 313–329. New York: Academic Press.

[52] Holland, J.H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell & T.M. Mitchell editors, *Machine learning, an artificial intelligence approach*. Volume II. Los Altos, California: Morgan Kaufmann.

[53] Holland, J. H. (1992). Genetic algorithms. *Scientific America* 267(1), pp. 66−72.

[54] Holland, J., Holyoak, K. J., Nisbett, R. E., and Thagard, P. (1986). *Induction: Processes of Inference Learning and Discovery*. Cambridge. MIT Press.

[55] Horn, J., Goldberg, D. E., and Deb, K. (1994). Implicit Niching in a Learning Classifier System: Nature's Way. *Evolutionary Computation*, 2(1), pp. 37-66.

[56] John, G. H., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence,* pp. 338–345. Morgan Kaufmann Publishers, San Mateo.

[57] Kabsch W., Sander C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* 22(12), pp. 2577–2637.

[58] Kharbat, F., Odeh, M., Bull, L. (2007). New approach for extracting knowledge from the XCS learning classifier system. *International Journal of Hybrid Intelligent Systems* 4. pp. 49-62. IOS Press.

[59] Kinjo, A.R., Horimoto, K., Nishikawa, K. (2005). Predicting absolute contact numbers of native protein structure from amino acid sequence. Proteins 58, pp. 158–165

[60] Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* 1, pp. 128–135. IEEE Press.

[61] Kovacs, T. (1997). XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions. In Roy, Chawdhry, & Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pp. 59-68. Springer-Verlag.

[62] Kovacs, T. (1999). Deletion Schemes for Classifier Systems. In Banzhaf, W., Daida, J., Eiben, A., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, (GECCO-99), pp. 329-336. Morgan Kaufmann.

[63] Kovacs, T. (2000). Strength or Accuracy? Fitness Calculation in Learning Classifier Systems. *Learning Classifier Systems: An Introduction to Contemporary Research*, pp. 143-160. Springer-Verlag.

[64] Kovacs, T. & Kerber, M. (2001). What Makes a Problem Hard for XCS? In Lanzi, P. Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifer Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pp 80-99. Springer-Verlag Berlin Heidelberg.

[65] Koza, J. R. (1992). *Genetic programmin*g. Cambridge, Massachusetts: The MIT Press.

[66] Langley, P. (1995). *Elements of machine learning*. Morgan Kaufmann Publishers Inc.

[67] Lanzi, P. L. (1997). A Study of the Generalization Capabilities of XCS. In Bäck, T., editor, *Proceedings of the Seventh International Conference of Genetic Algorithms (ICGA97)*, pp. 418-425. Morgan Kaufmann.

[68] Lanzi, P. L. & Riolo, R. L. (2003). Recent trends in learning classifier systems research. In *Advances in Evolutionary Computing: theory and Applications*, A. Ghosh and S. Tsutsui, Eds. Natural Computing Series, pp. 955-988. Springer-Verlag New York, New York, NY.

[69] Lanzi, P. L. (2005) xcslib: source code available at http://xcslib.sourceforge.net/.

[70] Lanzi, P. L., Loiacono, D., Wilson, S. W. and Goldberg, D. E. (2005). XCS with Computed Prediction for the Learning of Boolean Functions. *Evolutionary Computation* 1, pp. 588- 595.

[71] Larranaga, P., & Lozano, J., editiors. (2002). Estimation of Distribution Algorithms, A New Tool for Evolutionnary Computation. *Genetic Algorithms and Evolutionnary Computation*. Kluwer Academic Publishers.

[72] Lesk, A. M. (2001). *Introduction to Protein Architecture*. Oxford University Press.

[73] Llorà, X., Reddy, R., Matesic, B., and Bhargava, R. (2007). Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, England, July 07 - 11, 2007), GECCO '07, pp. 2098-2105. ACM, New York, NY.

[74] Loiacono, D., & Lanzi, P. L. (2006). Evolving Neural Networks for Classifier Prediction with XCSF, ECAI'2006, *Workshop on Evolutionary Computation*, pp. 36--40, ACM Press, New York, USA.

[75] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1, pp. 281–297, University of California Press.

[76] Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs*. Springer-Verlag.

[77] Michalski, R. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, A3, pp. 125–128.

[78] Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

[79] Moult, J. (2005). A decade of CASP: Progress, bottlenecks and prognosis in protein structure prediction. *Current Opinion in Structural Biology* 15(3 SPEC. ISS.), pp. 285-289.

[80] Mucchielli-Giorgi, M. H., Hazout, S., & Tuffery, P. (2002). Predicting the disulfide bonding state of cysteines using protein descriptors. *Proteins: Structure, Function and Genetics* 46. pp. 243-249.

[81] Naderi-Manesh, H., Sadeghi, M., Arab, S. and Moosavi Movahedi, A. A. (2001). Prediction of protein surface accessibility with information theory. *Proteins* 42. pp. 452-459.

[82] O'Hara, T., & Bull, L. (2004). *Prediction calculation in accuracy-based neural learning classifier systems*. Technical Report UWELCSG04-004.

[83] O'Hara, T., & Bull, L. (2007). Backpropagation in Accuracy-based Neural Learning Classifier Systems. In T. Kovacs, X. Llora, K. Takadama, P-L. Lanzi, W. Stolzmann & S.W. Wilson editors, *Learning Classifier Systems: International Workshops IWLCS 2003-2005*, pp. 26-40. Springer.

[84] Orriols-Puig, A. (2008). Personal communication, February 6 2008.

[85] Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

[86] Ravichandran, B., Gandhe, A., Smith, R., and Mehra, R. (2007). Robust automatic target recognition using learning classifier systems. *Inf. Fusion* 8(3), pp. 252-265.

[87] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Fromman-Holzboog Verlag.

[88] Riolo, R. (1991). Modeling simple human category learning with a classifier system. In: L, Booker & R.K. Belew editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 324-333. Morgan Kaufmann, Los Altos, CA.

[89] Rivest, R. L. (1987). Learning decision lists. *Machine Learning* 2(3), pp. 229–246.

[90] Robertson, G. & Riolo, R. (1988). A tale of two classifier systems. *Machine Learning* 3, pp. 139-159.

[91] Rost B., and Sander, C. (1994). Conservation and prediction of solvent accessibility in protein families. *Proteins* 20, pp. 216-226.

[92] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning Internal Representations by Error Propagation in Rumelhart, D. E. and McClelland, J. L., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press.

[93] Russel, S., & Norvig, P. (1995). *Artificial intelligence, a modern approach*. Englewood Cliffs, NJ: Prentice Hall.

[94] Selbig, J., Mevissen, T. & Lengauer, T. (1999). Decision tree-based formation of consensus protein secondary structure prediction. *Bioinformatics* 15, pp. 1039-1046.

[95] Sen, S. (1993). Improving classification accuracy through performance history. In: S. Forrest (ed.) *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 652. Morgan Kaufmann, Los Altos, CA.

[96] Shalizi, C. R. & Crutchfield, J. P. (2002). Information bottlenecks, causal states, and statistical relavance bases: How to represent relevant information in memoryless transduction. *Advances in Complex Systems*, 5(1), pp. 91–95.

[97] Shannon, C.E. (1948). A Mathematical Theory of Communication, *Bell System Technical Journal* 27, pp. 379–423 & 623–656, July & October.

[98] Shannon, C.E. (1949). Communication in the presence of noise, Proc. *Institute of Radio Engineers* 37(1), pp. 10-21, Jan.

[99] Smith, R. E. (1991). Default Hierarchy Formation and Memory Exploitation in Learning Classifier Systems. PhD thesis, The University of Alabama, USA.

[100] Smith, R. E., & Behzadan, B. (2008). Mutual Information Neuro-Evolutionary System (MINES), *IEEE Congress on Evolutionary Computation (CEC) 2009*, in press.

[101] Smith, R. E. & Cribbs, H. B. (1994). Is a classifier system a type of neural network? *Evolutionary Computation*, 2(1), pp. 19-36.

[102] Smith, R. E., Dike, B. A., Ravichandran, B., El-Fallah, A., and Mehra, R. K. (2001). Discovering Novel Fighter Combat Maneuvers in Simulation: Simulating Test Pilot Creativity. In Bentley P. J. and Corne, D. W. (eds.) *Creative Evolutionary Systems*, pp. 467-486. Morgan Kaufmann.

[103] Smith, R. E. & Jiang, M. K. (2007). A Learning Classifier System with Mutual-Information-Based Fitness, In *Proceedings of the 2007 Congress on Evolutionary Computation*.

[104] Smith, S. F. (1980). *A learning system based on genetic algorithms*. Doctoral dissertation, University of Pittsburgh.

[105] Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search.In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* pp. 421–425.

Los Altos, CA: Morgan Kaufmann.

[106] Stout, M., Bacardit, J., Hirst, J., Krasogor, N. & Blazewicz, J. (2006). From HP lattice models to real proteins: coordination number prediction using learning classifier systems. In *4th European Workshop on Evolutionary Computation and Machine Learning in Bioinformatics*.

[107] Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press/Bradford Books.

[108] Thompson, M. J. and Goldstein, R.A. (1996). Predicting solvent accessibility: higher accuracy using Bayesian statistics and optimized residue substitution classes. *Proteins* 25, pp. 38-47.

[109] Tishby, N., Pereira, F. C., and Bialek, W. (1999). The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, pp. 368–377.

[110] Wilson, S.W. (1987). Classifier systems and the animat problem. *Machine Learning* 2, pp. 199-288.

[111] Wilson, S.W. (1990). Perceptron redux: Emergence of structure. In S. Forrest editors, Emergent Computation: *Proceedings of the Ninth Annual Conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in natural and Artificial Computing Networks*, pp. 249-256. Amstredam: North-Holland.

[112] Wilson, S. W. (1994). ZCS: A Zeroth-Level Classifier System, *Evolutionary Computation* 2(1), pp. 1-18.

[113] Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation* 3(2), pp. 149-175.

[114] Wilson, S. W. (1998). Generalization in the XCS classifier system. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R. (eds.), Genetic Programming 1998: *Proceedings of the Third Annual Conference*, pp. 665–674. Morgan Kaufmann.

[115] Wilson, S. W. (2001). Function approximation with a classifier system. *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO-2001), L. Spector et al, eds. Morgan Kaufmann, San Francisco, CA, pp. 974-981.

[116] Witten, I. H., & Frank, E. (2000). Data Mining: practical machine learning tools and techniques with java implementations. Morgan Kaufmann.

[117] Wood, M. J., Hirst, J. D. (2004). Predicting protein secondary structure by cascade-correlation neural networks; *Bioinformatics* 20, pp. 419–420.