IMPROVING THE HIERARCHICAL CLASSIFICATION OF PROTEIN FUNCTIONS WITH SWARM INTELLIGENCE

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT AT CANTERBURY
IN THE SUBJECT OF COMPUTER SCIENCE
FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY.

BY
NICHOLAS HOLDEN
AUGUST 2008

ABSTRACT

This thesis investigates methods to improve the performance of hierarchical classification. In terms of this thesis hierarchical classification is a form of supervised learning, where the classes in a data set are arranged in a tree structure. As a base for our new methods we use the TDDC (top-down divide-and-conquer) approach for hierarchical classification, where each classifier is built only to discriminate between sibling classes.

Firstly, we propose a swarm intelligence technique which varies the types of classifiers used at each divide within the TDDC tree. Our technique, PSO/ACO-CS (Particle Swarm Optimisation/Ant Colony Optimisation Classifier Selection), finds combinations of classifiers to be used in the TDDC tree using the global search ability of PSO/ACO.

Secondly, we propose a technique that attempts to mitigate a major drawback of the TDDC approach. The drawback is that if at any point in the TDDC tree an example is misclassified it can never be correctly classified further down the TDDC tree. Our approach, PSO/ACO-RO (PSO/ACO-Recovery Optimisation) decides whether to redirect examples at a given classifier node using, again, the global search ability of PSO/ACO.

Thirdly, we propose an ensemble based technique, HEHRS (Hierarchical Ensembles of Hierarchical Rule Sets), which attempts to boost the accuracy at each classifier node in the TDDC tree by using information from classifiers (rule sets) in the rest of that tree. We use Particle Swarm Optimisation to weight the individual rules within each ensemble.

We evaluate these three new methods in hierarchical bioinformatics datasets that we have created for this research. These data sets represent the real world problem of protein function prediction.

We find through extensive experimentation that the three proposed methods improve upon the baseline TDDC method to varying degrees. Overall the HEHRS and PSO/ACO-CS-RO approaches are most effective, although they are associated with a higher computational cost.

ACKNOWLEDGEMENTS

This thesis has only been possible due to the help and support from several people:

I thank my supervisor Alex Freitas. His dedication to research and science has been an inspiration to me. I couldn’t have wished for a better supervisor!

For their unwavering emotional and sometimes financial support I thank my family and Maria greatly.

I would also like to thank my friends and colleagues, Dan for the chats about swarms, Alex for his first class programming tips, Mudassar who has been an excellent office mate, Elon and Andy for being the voices of PhD wisdom. Also, to all the other people who have donated their time to me – in particular my supervisory panel.

I especially thank the XPS project and the University of Kent for providing my bursary and so the wonderful opportunity to conduct this work.

CONTENTS

Chapter 1. Introduction 1

1.1. Motivation 2

1.2. Aim and Objectives 4

1.3. Preview of Contributions 6

1.4. Thesis Organisation 8

Chapter 2. Bioinformatics 10

2.1. Proteins 12

2.2. Protein Function Prediction 15

2.3. Sequence Alignment 17

2.4. Motif Detection using Alignment-Based Similarity Search 22

2.4.1. Regular Expressions 23

2.4.2. Profiles 25

2.4.3. Hidden Markov Models (HMM) 26

2.4.4. Fingerprints 29

2.5. Motif Databases 32

2.5.1. PROSITE 33

2.5.2. Pfam 33

2.5.3. PRINTS 34

2.5.4. InterPro 34

2.6. Biological Databases 37

2.6.1. UniProt 37

2.6.2. GPCRDB 38

2.6.3. Enzyme Nomenclature 39

2.7. Summary 40

Chapter 3. Data Mining 41

3.1. The Classification Task of Data Mining 43

3.2. Three Conventional Types of Algorithm for Classification 45

3.2.1. Rule/Tree Induction 45

3.2.2. Bayesian Classification 51

3.3. Ensembles of Classifiers 53

3.3.1. Bagging 53

3.3.2. Boosting 56

3.3.3. Stacking 57

3.4. Particle Swarm Optimisation 60

3.4.1. Binary and Discrete PSO Algorithms 63

3.5. Ant Colony Optimisation 65

3.6. Particle Swarm Optimisation and Ant Colony Optimisation for Classification 66

3.6.1. The Ant-Miner Algorithm 66

3.6.2. Particle Swarm Optimization for Classification 70

3.7. Hierarchical Classification 71

3.7.1. Flattening Hierarchical Classes 74

3.7.2. Top-Down Divide-and-Conquer Approach 77

3.7.3. “Big Bang” Approach 83

3.7.4. Measuring Hierarchical Classification Performance 86

3.8. Summary 88

Chapter 4. A Hybrid Particle Swarm Optimisation/Ant Colony Optimisation Algorithm for Rule Induction 89

4.1. Introduction 89

4.2. The New PSO/ACO-RI Algorithm 92

4.2.1. PSO/ACO-RI’s Sequential Covering Approach 93

4.2.2. Adding Continuous Terms to the Rule using PSO 94

4.2.3. Pruning the Discovered Rule 97

4.3. The Part of the PSO/ACO-RI Algorithm Coping with Nominal Data in Detail.... 99

4.3.1. Pseudocode... 102

4.3.2. Pheromone Updating Procedure 104

4.3.3. Measuring Rule Quality.. 107

4.4. Motivations for PSO/ACO-RI and Discussion 109

4.5. Computational Results 112

4.6. Summary 115

Chapter 5. Particle Swarm Optimisation/Ant Colony Optimisation for Classifier Selection and Misclassification Recovery in Hierarchical Classification 116

5.1. Introduction 116

5.2. Global Search-Based Classifier Selection with a Particle Swarm Optimisation/Ant Colony Optimisation Algorithm 117

5.3. Recovering from Misclassifications at Parent Classifier Nodes in the Top-Down Divide-and-Conquer Tree 121

5.3.1. Deciding when to Recover from Parent Misclassifications with the PSO/ACO-RO (Recovery Optimisation) Algorithm 125

5.3.2. Combining Classifier Selection and Misclassification Recovery with PSO/ACO-CS-RO 127

5.4. Experimental Setup 128

5.4.1. The Creation of the Bioinformatics Datasets 128

5.4.2. Data Set Partitioning and Baseline Algorithms 130

5.5. Computational Results for Classifier Selection 132

5.6. Computational Results for Misclassification Recovery Approaches.................. 135

5.6.1. Comparing Standard Top-Down Divide-and-Conquer Against the Basic (Always On) Recovery Approach 135

5.6.2. Comparing the Standard Top-Down Divide-and-Conquer Approach against the PSO/ACO-Optimised Recovery Approach (PSO/ACO-RO) 141

5.6.3. Discussion of the Effectiveness of the PSO/ACO-CS, Greedy Selective and PSO/ACO-RO Approaches 144

5.7. Summary 147

Chapter 6. Hierarchical Ensembles of Hierarchical Rule Sets (HEHRS) 149

6.1. Introduction 149

6.2. Building a Hierarchical Ensemble of Hierarchical Rule Sets (HEHRS) 150

6.2.1. Overview 150

6.2.2. Technical Details of the HEHRS Method 151

6.3. Combining the Predictions from the Multiple Rules in HEHRS using Voting ... 155

6.3.1. Weighted Voting for HEHRS 155

6.3.2. Optimising HEHRS' Rule Weights with PSO 158

6.4. Stacking for the Hierarchical Ensemble of Hierarchical Rule Sets (SHEHRS) .. 161

6.5. Rule-Based Extended Multiplicative Method 168

6.6. The Creation of the Bioinformatics Data Sets 170

6.7. Computational Results 172

6.7.1. Voting Schemes for HEHRS and Extended Multiplicative Method Results 175

6.7.2. Stacking for HEHRS Results 180

6.8. Summary 185

Chapter 7. Conclusions 187

7.1. Contributions 187

7.2. Future Work 191

References 194

Publications from the work in this Thesis 210

Appendix A. Comparing the Performance of PSO/ACO and Binary PSO in Benchmark Binary Optimisation Problems 212

LIST OF TABLES

Table 2.1: An Example protein sequence alignment 18

Table 2.2: Constituent InterPro databases 35

Table 4.1: PSO/ACO-RI Pheromone Updating Scenarios 106

Table 4.2: An Example Single Class Data Set, R's are Records, A_n's are Nominal Attributes 111

Table 4.3: Predictive Accuracy and Rule Set size of PSO/ACO-RI and PART in UCI Data Sets, with Standard Deviation and Student T-Test Shadings 113

Table 4.4: Summation of the number of statistically significant results of PSO/ACO-RI against PART according to the Student T-Test (out of 27 Data Sets). 114

Table 5.1: Main characteristics of the datasets used in the experiments 129

Table 5.2: Predictive accuracy (%) for each approach in the Prints data set. 133

Table 5.3: Predictive accuracy (%) for each approach in the Interpro data set. 133

Table 5.4: Predictive accuracy (%) for each approach in the Pfam data set. 133

Table 5.5: Predictive accuracy (%) for each approach in the Prosite data set. 134

Table 5.6: Summation of the number of statistically significant results according to the Student T-Test 134

Table 5.7: Predictive accuracy (%) for each approach in the Prints data set. 138

Table 5.8: Predictive accuracy (%) for each approach in the Interpro data set 138

Table 5.9: Predictive accuracy (%) for each approach in the Pfam data set. 139

Table 5.10: Predictive accuracy (%) for each approach in the Prosite data set 139

Table 5.11: Summation of the number of statistically significant results according to the Student's T-Test, for the recovery always on approach against the recovery always off approach, for each labelled approach 140

Table 5.12: Predictive accuracy (%) for each approach in the Prints data set (recovery optimised by PSO/ACO) 142

Table 5.13: Predictive accuracy (%) for each approach in the Interpro data set (recovery optimised by PSO/ACO) 142

Table 5.14: Predictive accuracy (%) for each approach in the Pfam data set (recovery optimised by PSO/ACO) 142

Table 5.15: Predictive accuracy (%) for each approach in the Prosite data set (recovery optimised by PSO/ACO) 143

Table 5.16: Summation of the number of statistically significant results (recovery optimised by PSO/ACO) according to the Student's T-Test 143

Table 5.17: Reproduction of the totals from Table 5.6 and Table 5.16, in terms of numbers of significant wins over the baseline classification algorithms 144

Table 5.18: Comparing the predictive accuracy (%) of approaches using Recovery Optimisation against approaches not using Recovery Optimisation on the Prints dataset 145

Table 5.19: Standard deviations of base algorithm mean performance in each labelled dataset (not including ConjunctiveRule) 146

Table 6.1: Values (Classes) taken by variable *c* at each level *i* used to construct the Rule Sets in E_s ID:1, in Figure 6.1 153

Table 6.2: Meta-example generated by SHEHRS for the set of sibling classes S = {1, 2}, at level 1 in the class hierarchy 164

Table 6.3: Meta-example generated by SHEHRS for the set of sibling classes S = {1.1, 1.2}, at level 2 in the class hierarchy 164

Table 6.4: A set of rule weight-based SHEHRS meta-examples 166

Table 6.5: Main characteristics of the datasets used in the experiments 172

Table 6.6: Predictive accuracy (%) with Prints attributes and GPCR classes 176

Table 6.7: Predictive accuracy (%) with InterPro attributes and GPCR classes 176

Table 6.8: Predictive accuracy (%) with Prosite attributes and GPCR classes 176

Table 6.9: Predictive accuracy (%) with Prints attributes and Enzyme classes 177

Table 6.10: Predictive accuracy (%) with Pfam attributes and Enzyme classes 177

Table 6.11: Predictive accuracy (%) with Prosite attributes and Enzyme classes 177

Table 6.12: Summation of the number of statistically significant results according to the Student's t-test, when comparing the proposed approaches to the baseline . 177

Table 6.13: The Un-weighted Misclassification cost, comparing each proposed approach against the baseline 179

Table 6.14: The Weighted Misclassification cost, comparing each proposed approach against the baseline 179

Table 6.15: Predictive accuracy (%) with Prints attributes and GPCR classes 181

Table 6.16: Predictive accuracy (%) with InterPro attributes and GPCR classes........... 181

Table 6.17: Predictive accuracy (%) with Prosite attributes and GPCR classes............. 181

Table 6.18: Predictive accuracy (%) with Prints attributes and Enzyme classes............ 182

Table 6.19: Predictive accuracy (%) with Pfam attributes and Enzyme classes............. 182

Table 6.20: Predictive accuracy (%) with Prosite attributes and Enzyme classes.......... 182

Table 6.21: Summation of the number of statistically significant results, according to the Student's t-test, when comparing each proposed approach to the Baseline . 182

Table 6.22: The Un-weighted Misclassification cost, comparing each proposed approach against the baseline 184

Table 6.23: The Weighted Misclassification cost, comparing each proposed approach against the baseline 184

Table A.1: Number of iterations to find the maximum valued bitstring in each benchmark problem for PSO/ACO and Binary PSO 216

LIST OF FIGURES

Figure 2.1: Alpha Helix 13

Figure 2.2: Beta Sheet 13

Figure 2.3: Part of a multiple sequence alignment of Muscarinic Acetylcholine receptor GPCR proteins using ClustalW 21

Figure 2.4: An approximate phylogenetic tree derived from the multiple sequence alignment of Muscarinic Acetylcholine receptor GPCR proteins using ClustalW 21

Figure 2.5: Sequence for ACM1_DROME Muscarinic acetylcholine receptor from Drosophila Melanogaster (Fruit fly). 24

Figure 2.6: Part of a profile from Prosite [85] 25

Figure 2.7: A profile HMM showing match states (M), delete states (D) and insert states (I), with transition probabilities shown as arrows. The beginning state is M_0 and the end state is M_{i+1} adapted from [97]. 27

Figure 2.8: A Pfam search against ACM1_Drome showing the HMM hits with the most likely hit and other possible hits based on E-value. 28

Figure 2.9: Part of the alignment for ACM1_Drome against the 7tm_1 (seven trans-membrane helices) Pfam HMM. 29

Figure 2.10: A graphical view of the GPCRRHODOPSN Fingerprint – shown in red, the Fingerprint corresponds to the seven Trans-membrane helices of GPCRs 31

Figure 2.11: A graphical representation of the 7 motifs from the GPCRRHODOPSN Fingerprint covering the ACM1_Drome GPCR protein 32

Figure 2.12: An example Interpro entry matching the ACM1_Drome GPCR sequence (only part of the relationships cell is shown) 36

Figure 2.13: An example InterPro search for the ACM1_Drome GPCR sequence showing from top to bottom, PRINTS, Pfam, PROSITE profile and PROSITE pattern entries matching the search sequence 36

Figure 3.1: An example decision tree 50

Figure 3.2: Training and Testing Phases in Bagging 55

Figure 3.3: Training and testing phases in boosting 56

Figure 3.4: Training and testing stages in stacking 58

Figure 3.5: Ring (Local) Topology for PSO Particles 60

Figure 3.6: Global Topology for PSO Particles 60

Figure 3.7: Von-Neumann Topology for PSO Particles 61

Figure 3.8: The effect of *k* on the binary PSO $s(v_{id})$ function 64

Figure 3.9: Pheromone trails in natural ant colonies 65

Figure 3.10: An example hierarchical classification class structure 72

Figure 3.11: An example class-tree based hierarchical classification problem shown in the form of a Venn diagram 72

Figure 3.12: Reducing a hierarchical classification problem into a flat classification problem 74

Figure 3.13: Reducing a hierarchical classification problem into a set of flat classification problems 75

Figure 3.14: The Top-Down Divide and Conquer method to deal with a hierarchical classification problem 77

Figure 3.15: A TDDC tree using classification algorithm selection 80

Figure 3.16: Classifier interaction scenario where $|B \cap C| > |A \cap C|$ 81

Figure 3.17: Classifier interaction scenario where $|B \cap C| < |A \cap C|$ 81

Figure 3.18: A class tree used to illustrate the discussion on classifier interaction 82

Figure 3.19: The Big Bang approach to deal with a hierarchical classification problem, notice that the classifier can predict any node within the class hierarchy in just one step 84

Figure 4.1: The outline of the probability distribution for particle seeding at the lower bound of an attribute value 96

Figure 5.1: An encoded particle with *n* dimensions, each with *k* classifier ids 119

Figure 5.2: An example being redirected to the classifier discriminating between classes 2.1 and 2.2 by a classifier discriminating between classes 1.1, 1.2 and class 2. 123

Figure 6.1: Hierarchical Ensemble of Hierarchical Rule Sets (HEHRS) 152

Figure 6.2: An Example Classification in SHEHRS 163

Figure A.1: The average performance of the best particle during 100 runs of the All Same First benchmark function, for both PSO/ACO and BPSO 216

Figure A.2: The average performance of the best particle during 100 runs of the All Same benchmark function, for both PSO/ACO and BPSO 217

LIST OF ALGORITHMS

Pseudocode 3.1: Sequential Covering Approach 47

Pseudocode 3.2: Computing the Value of a Particle's Position in Binary PSO 63

Pseudocode 3.3: The Ant-Miner algorithm adapted from [119] 67

Pseudocode 4.1: Sequential Covering Approach used by the Hybrid PSO/ACO-RI Algorithm 94

Pseudocode 4.2: The Part of the PSO/ACO-RI Algorithm Coping with Nominal Data 102

Pseudocode 5.1: The Hybrid PSO/ACO-CS Algorithm for Classifier Selection 120

Pseudocode 6.1: The testing phase of Weighted Voting for HEHRS 157

Pseudocode 6.2: The training phase of SHEHRS 162

Pseudocode 6.3: The testing phase of SHEHRS 165

Pseudocode A.1: One Max 213

Pseudocode A.2: One First 213

Pseudocode A.3: One Max Noisy 214

Pseudocode A.4: All Same 214

Pseudocode A.5: All Same First 215

Chapter 1. Introduction

This thesis proposes new methods to improve the predictive accuracy of hierarchical classification algorithms. From a machine learning and data mining perspective classification involves a form of supervised learning, where a learning algorithm creates a classifier which is trained using examples from a data set with known class labels [164]. The classification model produced by this algorithm during the training phase is then used to predict what class label examples from a test set (unseen during training) have.

The type of classification problem dealt with in this thesis is hierarchical – as the classes form a hierarchical structure [149]. An example of a hierarchical classification problem might be the prediction of what species and then breed a pet is. In the first case we wish to know whether the given animal is of the class (species) dog or cat, and in the second case if the animal is of the class (breed) Burmese, British Blue, Jack Russell or Golden Retriever. In this thesis the species would be considered the first class level and the breed the second class level.

The methods we propose during the course of this thesis have been applied to challenging bioinformatics data sets. Our goal is to improve our ability to predict what functions proteins have, using only information about the protein's biochemical composition.

To aid us in dealing with the complex nature of the hierarchical classification problems examined in this thesis we use swarm intelligence based techniques, including several versions of a new PSO/ACO (Particle Swarm Optimisation/Ant Colony Optimisation) algorithm we have developed. Swarm intelligence based techniques have been the basis of effective optimisation algorithms in the past. However, there has been relatively little investigation into their application to data mining problems – notable examples of work in this area can be found in [1] [63]. We will show during the course of this thesis that swarm intelligence algorithms are also effective in hierarchical classification problems in the context of data mining and bioinformatics.

The rest of this chapter introduces the subject matter of this thesis, including our motivations for carrying out this work. The aims, contributions and structure of the remainder of this document are also discussed.

1.1. Motivation

The amount of information being produced by biologists is increasing massively. The automation of the collection and storage of this information has led, in part, to this increase, and so the demand for automated methods to analyse this information has also dramatically increased. Probably the most well known source of biological data has been from the sequencing of the human genome. However, many more genomes have been sequenced, hundreds of organisms have had their genetic code revealed with base pairs of nucleotides (the building blocks of DNA) numbering in the hundreds of thousands of millions [157]. An example of where computational methods have been essential to biological problems is where algorithms were developed to piece together the fragments of DNA sequences. During genomic sequencing fragments are discovered using a process called “shotgun sequencing”. During this process random segments of DNA are sequenced, with many overlapping segments, which are then knitted together automatically by an algorithm. With such a large amount of sequence data it would be impossible to use more traditional and labour intensive processes. In other words, it has become essential, and now commonplace, for biologists to embrace computational tools.

The prediction of protein function can be considered one of the most important challenges faced by biologists in the current “post-genome” era. The challenge arises, in part, from the fact that the number of proteins discovered each year is growing at a near exponential rate [156]. Potential proteins are automatically decoded from DNA, revealing a huge number of proteins with totally unknown functional characteristics [157].

In addition, advances in the understanding of protein function are critical for the unravelling of biological processes, more effective diagnosis and treatment of disease, also helping in the design of more effective medical drugs, etc. While the amount of time it takes to discover and sequence a protein has decreased massively (the discovery of a protein sequence was once considered worthy of a doctorate alone), discovering its function remains relatively difficult and long winded via traditional biological methods.

When using data mining techniques it is possible to predict the function of many proteins, quickly and with reasonable accuracy. Also, the patterns discovered in the process can potentially be used to gain insight into the nature of proteins under investigation.

Protein function prediction is particularly attractive from a data mining research perspective. This is due to the difficulty of the problem involved, the need for the validation of any predictions and the potentially great relevance of knowledge to be discovered from the biological data. There is a lot of data readily available on the web; this information is in large complex databases, with many records and many possible attributes. Although this data is readily available in raw form, extracting the required data in a form that can be processed by a data mining algorithm can be a very difficult and time consuming task. Care must be taken to weed out unwanted data, database cross-referencing systems are not always straight forward, conventions for identifying attributes and giving descriptions are not always obvious, records might have many missing attribute-values, duplicate or near duplicate records are commonplace. Also, the data available in these biological databases often involves hierarchical and multi-label class structures (where a record can belong to more than one class and the classes are structured in a hierarchy). These class hierarchies are often ignored by researchers [15] [85], possibly because at first glance they seem irrelevant, possibly because few data mining tools exist to take advantage of them. We will show that if there is a hierarchical class structure present in a data set it can be used to the advantage of the researcher. We also present methods that take advantage of hierarchical class structures in order to improve the predictive accuracy of the hierarchical classification of protein function.

Swarm intelligence algorithms are derived from the way in which relatively unintelligent entities (at the level of the individual) perform intelligent tasks (at the level of the swarm) in nature. Part of the reason we chose to use swarm intelligence based techniques with hierarchical classification problems is that swarm intelligence based approaches have shown to be effective for some data mining tasks (including classification) in the past. Also, swarm intelligence algorithms provide a great deal of flexibility – there are many different types of algorithms suiting different applications. Furthermore, due to the way in which the "intelligent" behaviour comes from the relatively unintelligent entities, the basic principles of swarm intelligence tend to transfer readily to computational algorithms. This is especially true for those algorithms designed to deal with problems that do not require a large amount of background knowledge about the target problem. Significantly, although the effectiveness of swarm intelligence approaches has been shown in conventional "flat" classification, swarm intelligence methods for hierarchical classification problems have not previously been investigated. This, broadly speaking, characterises the originality of the research described in this thesis, from a computer science point of view. In addition, this thesis also presents an original contribution to bioinformatics, since there has been no previous investigation of swarm intelligence methods for the hierarchical prediction of protein functions (again, to the best of our knowledge).

1.2. Aim and Objectives

The overall aim of this thesis is to develop swarm intelligence-based methods that use the extra information present within a class hierarchy to improve predictive accuracy. To be able to do this we must first identify which areas of the hierarchical classification problem can be improved upon.

During this thesis we take, as a baseline approach to be improved upon, a type of hierarchical classification method that is already generally considered to be effective, the top-down divide-and-conquer approach (TDDC). In this context, our overall aim consists of several more precise objectives:

Firstly, it should be noted that different classifiers may perform differently on each individual classification problem (each "divide") within the TDDC tree. This problem has already been investigated by other researchers (Secker et al. [140]) to some degree.

Our first objective is to propose and evaluate a new swarm intelligence method to improve upon Secker et al.'s method.

Secondly, the TDDC approach is based on the principle that only sibling class nodes need to be considered at any point in the class tree. So, at the first set of sibling class nodes (e.g., cat or dog), if we decide cat, then at the second set of class nodes we must only decide between the sibling class nodes Burmese or British Blue. This approach has a major drawback. If the pet is in fact a dog we are guaranteed to guess the breed wrong if we predict cat at the first class level. This problem is known as the "blocking" problem in the literature [152], where higher level classifiers block the ability of lower level classifiers to make correct predictions.

The second objective of this thesis is to propose and evaluate a swarm intelligence-based method that attempts to reduce this drawback.

Thirdly, the TDDC approach involves dividing the hierarchical classification problem into a set of flat classification problems for each classifier to deal with. In this sense the system is still, to some extent naive of the class hierarchy. The only part of the TDDC approach that is aware of the hierarchy is the simple top-down algorithm which guides each test example to the next child classifier during the training phase. It seems that by creating a method that is more aware of the hierarchy it should be possible to boost the classification accuracy of the entire system.

Our third objective is to use the multiple class levels in the class tree to guide the classification of each example in the test set in a more "intelligent" manner – using the information from each of these class levels (we also aim to evaluate this approach). This objective is pursued by proposing an ensemble based method whose performance is improved by a proposed swarm intelligence based method. Stacking and bagging (two well-known types of ensemble methods) are explored and adapted in this thesis in an attempt to improve hierarchical classification accuracy. The standard TDDC approach is extended by sampling the examples based on the native class hierarchy of the data, which can be considered, at a high level of abstraction, a variation of bagging in the context of hierarchical classification. We also propose a meta-learning approach conceptually similar to stacking.

Fourthly, it is common for researchers to infer protein function based on protein biochemical similarity alone [3], without using a data mining algorithm to induce a classification model from the data. By building a higher level classification model rather than relying on low level biochemical information it should be possible to increase predictive accuracy. We create new (hierarchical protein function) data sets with this in mind and try and identify which method is best to classify proteins in each data set.

Indeed, *our fourth objective* can be seen as the application of the methods created, as a result of the first three objectives, to the prediction of hierarchical protein function.

1.3. Preview of Contributions

The main original contributions of this thesis are:

- A new hierarchical classifier selector technique (PSO/ACO-CS) which uses PSO/ACO to try to find a hierarchical combination of classifiers (a tree of classifiers) with, overall, a higher predictive accuracy than the tree of classifiers found by Secker's greedy selective approach [140]. This relates to the first objective of this thesis; selecting the near optimal classifier at each classifier node (each "divide" within the TDDC tree) is a task which benefits from the global search performed by PSO/ACO-CS.

- A misclassification-recovery optimisation technique (PSO/ACO-RO) which uses PSO/ACO to try to mitigate a major drawback of the top-down divide-and-conquer (TDDC) approach. This relates to the second objective of this thesis and allows lower level classifiers to attempt to correct the errors of higher level classifiers (within the TDDC tree).
- An ensemble technique (Hierarchical Ensemble of Hierarchical Rule Sets) which attempts to boost the accuracy of individual classifier nodes within the TDDC tree. This relates to the third objective of this thesis. The HEHRS technique is an attempt to make a more "intelligent" hierarchical classification system, in that it uses information derived from the class hierarchy to guide predictions. We also propose the use of PSO to automatically adjust some parameters of the proposed ensemble method which led to an improvement in the predictive accuracy of the latter.

The more minor (secondary) original contributions are:

- A new hybrid Particle Swarm Optimisation/Ant Colony Optimisation (PSO/ACO) algorithm that deals directly with both categorical/nominal and numerical/continuous data. PSO/ACO is used to create a proof-of-concept "flat" classification algorithm for rule induction (PSO/ACO-RI) whose basic principles were used in the previously-mentioned swarm intelligence based algorithms for hierarchical classification (major contributions of this thesis).
- The creation of several hierarchical protein function data sets. Specifically, data sets containing data for the hierarchical prediction of GPCR function and data sets containing data for the hierarchical prediction of enzyme functions. These data sets are freely available on request.

1.4. Thesis Organisation

The remainder of this thesis is organised as follows. Chapter 2 provides a background on the biological knowledge required to understand the protein function prediction problem tackled in this thesis. We introduce the features of proteins in general, and how it is possible to predict what function a protein might have. We also introduce the approaches that have classically been taken by previous researchers for protein function prediction. The online protein databases that we use in the creation of our bioinformatics data sets are also described.

Chapter 3 introduces background information on data mining. It describes the approaches taken by researches interested in hierarchical classification. It also discusses how swarm intelligence algorithms have been applied to data mining problems in the past, focusing on Ant Colony Optimisation and Particle Swarm Optimisation. We also summarise previous work on ensemble-based techniques, focusing on the methods that are used in the approaches proposed in this thesis.

Chapter 4 describes our new hybrid Particle Swarm Optimisation/Ant Colony Optimisation (PSO/ACO) algorithm. As a proof of concept we implement a standard "flat" classification algorithm based on our PSO/ACO technique and test its performance in standard benchmark data sets. The concepts and principles incorporated into this PSO/ACO algorithm for flat classification are then used to produce more sophisticated PSO/ACO methods for hierarchical classification in the following chapters.

Chapter 5 describes the blocking problem in detail and proposes our solution to try and mitigate its effect – a PSO/ACO algorithm for misclassification-recovery optimisation. This is where examples can have their misclassification at higher class levels corrected by lower level classifiers. We also propose a classifier selection technique based on Secker's [140], which uses global (swarm-based) search to improve on the performance of his technique. We evaluate our proposed methods, along with several baseline approaches, using four new bioinformatics data sets. These challenging real world data sets were constructed for the hierarchical prediction of GPCR (G-Protein Coupled Receptors) protein function.

Chapter 6 proposes an ensemble-based approach for hierarchical classification. This uses information from all levels within the class hierarchy to guide the classification of each test example at each individual classification. We propose approaches that combine the prediction of the classifiers using standard ensemble-based techniques (voting, stacking) and also a PSO-based technique. We evaluate our proposed methods, along with an adapted baseline technique, in 6 data sets involving protein function (Enzyme and GPCR proteins) prediction. These data sets were created by us to test the effectiveness of our approaches in real world problems.

Chapter 7 draws general conclusions based on the analysis of the results of the experiments performed during this thesis and describes possible future research directions.

Chapter 2. Bioinformatics

> *"Bioinformatics is:*
>
> *(i) the development of computational methods for studying the structure, function, and evolution of genes, proteins, and whole genomes;*
>
> *(ii) the development of methods for the management and analysis of biological information arising from genomics and high-throughput experiments."*
>
> *P. G. Higgs and T. K. Attwood [78]*

The field of bioinformatics encompasses many different disciplines, where the application of techniques from artificial intelligence, statistical analysis, applied mathematics and algorithmic engineering sit side by side with biochemistry. The techniques from these fields are applied to an array of different biological problems. Some of the most important problems faced by bioinformaticians are described in this section.

Sequence alignment, with algorithms such as the well known BLAST [3], are in part taken from the world of pattern matching in strings. Sequence alignment is concerned with matching two or more strings, possibly with omissions of substrings and possibly with different length, in an optimal fashion.

The identification of active coding regions of DNA is also an important area of bioinformatics research. A coding region of DNA is defined as any sequence within the DNA that performs a useful function. Conventional wisdom is being challenged in this area (with respect to the nature of non-coding DNA), i.e., it is now considered possible that very little DNA has no useful role in the organism [55] [102].

Another important area encompassed in bioinformatics is protein-protein interaction [101] [125]. This involves the analysis of sets of proteins in an attempt to detect whether they directly interact with one another. This is very important in the understanding of cellular biology as proteins interact to form complex metabolic pathways. Unravelling these “computational” pathways gives us a better understanding of the mechanisms of life [13]. It is one thing to understand these pathways, in the sense of understanding which elements interact, it is another to understand the properties of the proteins themselves.

Another major task in the field of bioinformatics is protein function prediction (the subject of this thesis), which is the prediction of what function a protein might have using computational means. The function of a protein can be found experimentally in a “wet-lab” but this is usually a laborious process. Much more commonly the function of a protein is predicted computationally (the subject of this thesis) and this prediction can be validated, or at least used as a starting point in the lab. One way of understanding (to some extent) how a protein performs a function is to calculate or observe its structure; its structure being the 3-dimensional placement of each atom. Calculating the structure of any long protein is extremely computationally intensive [98] and the observation of protein structure is a difficult process requiring X-ray crystallography or protein nuclear magnetic resonance spectroscopy. However, the structure of a protein can be estimated using heuristic approaches, and this is also an active area of research [100] [136].

Another active area of research is evolutionary analysis, using computational methods to identify the way in which organisms have evolved, identifying common ancestors (common DNA), speciation events, lineage, building phylogenetic trees [78] etc.

The rest of this chapter is organised as follows. Section 2.1 gives an overview of what proteins are and the features that make it possible to predict their function. Section 2.2 introduces the topic of protein function prediction. Section 2.3 describes sequence alignment and how it can be used to infer homology. Section 2.4 describes methods that use motifs to detect homology. Section 2.5 presents the protein motif databases used for creating attributes for the protein datasets used in the experiments in this thesis. Section 2.6 introduces the biological databases used to create the records of the data sets investigated in this thesis, along with the two protein families which are the focus of this research, namely G protein-coupled receptors (GPCRs) and enzymes.

2.1. Proteins

The cell is a common element in almost all life; these cells contain extremely complex chemical reactions. These complex chemical reactions are protected by a semi-permeable membrane (cell membrane). Another common element of life is a “blueprint”, a method of storing the information needed to create the enclosed chemical reaction, such a blueprint is found in the form of DNA. This DNA is used to create progeny – descendants that have their own and possibly distinct DNA.

It is very probable that all cells (and so creatures) have one original ancestor. Therefore, common patterns in genes are likely to appear within the DNA and the subsequently decoded proteins of living organisms. This leads to the concept of homology, where two genes have evolved from a similar ancestral gene. Homology occurs across different species and also between different genes within the same genome. This is because one of the major ways in which creatures evolve is by a process of gene duplication where another copy of an existing gene is added to their genome and subsequently mutated. When two genes have a common ancestor between species due to split in species they are said to be orthologous. When two genes sequences have a common ancestry and occur within the same genome (in a single species) – due to a gene duplication process – they are said to be paralogous. Orthologous genes usually have similar function, i.e., they are used for a similar purpose but just in different species. This is not always true for paralogous genes, as they probably have mutated to perform a different function because of evolutionary pressures. Due to homology, proteins can often be organised into evolutionary families which can have similar function. Due to the nature of evolution these families often form hierarchies. Of course proteins do not have to be related in any way to perform a similar function. Analogous proteins may, however, still share some similarity; the amount of similarity is obviously dependant on how easily different proteins can fulfil the same role. Evolution often finds a slightly different answer to the same question; for instance, the eye has independently evolved many times in different but still identifiably similar ways.

Almost all of the cell's structure and functions rely on proteins, from cell walls to energy generation to biological motors to move the cell around. Proteins are generated from DNA, which is firstly transcribed into Messenger RNA and then translated into protein by the ribosome. Protein structure can be divided into four main levels of structure: primary, secondary, tertiary and quaternary [2]. Their primary structure, the one decoded from DNA, is formed from a sequence of amino acids which are held together by covalent bonds (strong bonds). This chain is built one amino acid at a time by the ribosome.

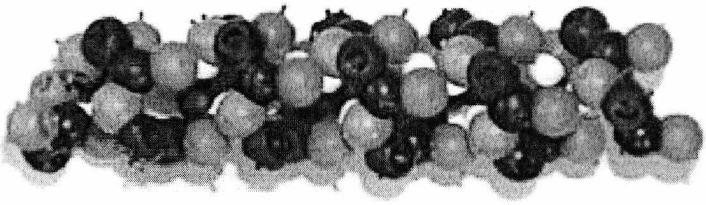


Figure 2.1: Alpha Helix

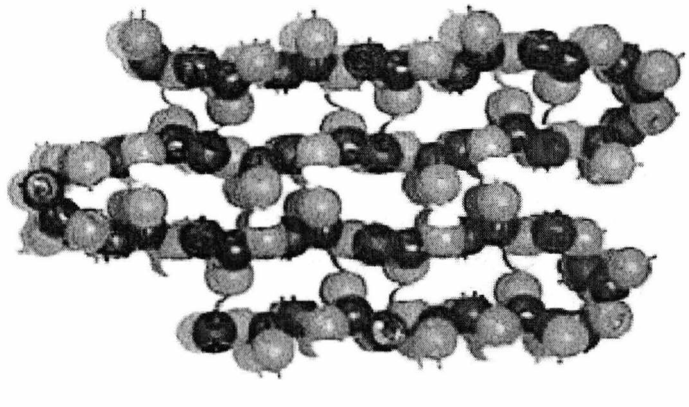


Figure 2.2: Beta Sheet

A protein forms a secondary structure after it has been constructed which is held together by hydrogen bonds (weaker bonds). The secondary structure consists of local three dimensional structures. Two common secondary structures that form based on the amino acid backbone are alpha helices and beta sheets, shown in Figure 2.1 and Figure 2.2. These may occur multiple times within each protein.

A protein typically folds to form a tertiary structure. This is the overall shape of the protein after the secondary structures have "joined" together. Each different protein will have a different tertiary structure. At this level of structure, proteins are held together by a variety of different interacting chemical bonds that are generally weaker than those in the first two structures.

Some proteins also form quaternary structures. This is the overall shape when two or more polypeptide chains join together, forming a complex, to produce a larger molecule. These multimeric complexes can be grouped in two main categories, globular and fibrous. Multimeric complexes may comprise of multiple copies of the same polypeptide chain, or multiple different polypeptides. Examples of fibrous proteins are fibroin in silk and spider webs, and keratin α in hair and fingernails. Examples of globular proteins (which are ball shaped) are haemoglobin and most enzymes.

Another level of organization somewhat separate from the above levels consist of protein domains. They are described as any segment of the primary structure that forms stable and compact structures and usually consist of around 100 amino acids [2]. They are modular elements that often perform a specific function. A protein may have one or many domains and they are often common between proteins. These domains or motifs are often highly conserved throughout evolution and between homologous proteins. They can be independent and functionally important and so are considered discrete evolutionary building blocks. In fact it has been argued that domains, not genes, are the fundamental units of evolution [114].

2.2. Protein Function Prediction

The prediction of protein function is a major task in bioinformatics and is the subject of this thesis. If done successfully and accurately it can produce very important results, for instance the ability to automatically find proteins related to known disease processes in cells [114]. The faster these proteins can be found, the faster new drugs can be created to act upon them.

As discussed in section 2.1, evolutionary processes mean that homologous proteins often share some similarity at the amino acid level (primary sequence). However, certain evolutionary mechanisms mean that the detection of similarity can be very difficult. Mutation often occurs at functionally insignificant sites. One amino acid can sometimes be swapped for another similar amino acid while conserving function. Whole sections (domains) can be swapped between genes creating a completely different function, or having no effect at all. These factors mean that detecting such meaningful similarity (especially in distantly related proteins) is a very challenging problem.

It may initially seem as though if two sequences have similar amino acid composition then they should be homologous. However, when searching large protein databases the chance of detecting similarity due only to chance increases. In fact this is another very challenging problem, especially when dealing with millions of sequences. However, these problems are not necessarily intractable. As stated previously, domains are often very highly conserved throughout evolution, with only minor changes taking place. Identifying regions of primary sequence - such as domains - that might be highly conserved and using them to search for homology has proved to be effective (as will be discussed in the following sections). Also, it is possible to take into account the likelihood that one amino acid has been exchanged for another within a protein sequence, as a result of an evolutionary process. This likelihood can be calculated from observed amino acid substitutions across homologous proteins or from the properties of the amino acids themselves.

Protein function prediction methods can be broadly divided into two main categories [62]. Firstly the methods which seek to predict function via homology detection alone, which is in turn usually found via a search for similar sequences in a database of proteins with known function. Secondly, the prediction of function based on a model induced from a protein data set using machine learning-based model induction techniques [117]. Obviously a model can be induced, and utilised for function predicted (to varying degrees of success) from any set of features relating to the individual proteins. Common methods involve either inducing the model from a set of sequence similarities or alignments (as in this work), or reducing the sequence to a set of other non-alignment based attributes.

One possibility for generating a model which is "alignment-free" is to use proteochemometrics. The methods based on proteochemometrics usually aim to reduce the sequence data to a set of numeric attributes [140]. Techniques that rely on these metrics as a source of attribute data can be relatively effective. However, the comprehensibility of the produced model is questionable.

In the case of the alignment based techniques the abstraction provided by inducing a model based on sequence similarity (rather than simply inferring homology and so function directly from similarity) is an important one. As Freitas et al. [62] state, the abstraction mitigates three important problems associated with relying on sequence similarity alone:

> *"First, it is well-known that two proteins might have very similar sequences and perform different functions, or have very different sequences and perform the same or similar function (Syed & Yona, 2003), (Gerlt & Babbitt, 2000). Second, the proteins being compared may be similar in regions of the sequence that are not determinants of their function (Schug et al., 2002). Third, the prediction of function is based only on sequence similarity, ignoring many relevant biochemical properties of proteins (Karwath & King, 2002), (Syed & Yona, 2003)."*

Gerlt, J. A. & Babbitt, P.C. (2000). Can sequence determine function? Genome Biology 1(5), 1-10.

Karwath, A. & King, R. D. (2002). Homology induction: the use of machine learning to improve sequence similarity searches. BMC Bioinformatics 3:11.

Schug, J., Diskin, S., Mazzarelli, J., Brunk, B.P. & Stoeckert Jr., C.J. (2002). Predicting gene ontology functions from ProDom and CDD protein domains. Genome Research 12, 648-655.

Syed, U. & Yona, G. (2003). Using a mixture of probabilistic decision trees for direct prediction of protein function. In Proceedings of the 2003 Conference on Research in Computational Molecular Biology (RECOMB-2003) (pp. 289-300).

The rest of this chapter describes methods for finding similarity between proteins based on sequence alignment, from some of the most basic techniques (Section 2.3) to some of the most advanced (Section 2.4). The different approaches that are described in the remainder of this chapter have corresponding online databases (Section 2.5) where similarity information is stored. These databases are used extensively in this thesis for attribute creation.

2.3. Sequence Alignment

Determining how similar a protein sequence is to another is a cornerstone of bioinformatics. Similarity, however, needs to be defined to understand how it can be useful in determining if one protein is related to another in some way. Firstly one must decide whether to measure global (Needleman and Wunsch [110]) or local similarity (Smith and Waterman [145]); that is the similarity of two whole sequences or just smaller regions within those whole sequences. Global alignment is useful when determining if two sequences are related, whereas local alignment is often used to discover highly conserved regions. Primary sequence regions are likely to be highly conserved if there is an evolutionary pressure to maintain those regions; this is often the case in functionally important regions (domains). One must also consider that protein sequences are almost always of different lengths. Given that fact, the problem becomes an issue of alignment; how one can align the sequences so that they are the most similar.

	Original Sequence	**Alignment 1**	**Alignment 2**	**Alignment 3**
Sequence 1	ADCCCACDD	**ADCC**CACDD	**ADCCCAC**DD	**ADCCCACD**D
Sequence 2	CACADCCACEC	CAC**ADCC**ACEC	CAC**ADCC–AC**EC	CAC**ADCC–ACE**C

Table 2.1: An Example protein sequence alignment

Table 2.1 shows three possible alignments for two hypothetical sequences 1 and 2. A possible alignment is shown in the column labelled alignment 1. There are four amino acids in a row that match each other. Notice that after the matching sequence (ADCC) there are two more potentially matching amino acids A and C. Unfortunately there is an additional C in sequence 1 that prevents these two extra amino acids from matching. However, it is often the case in nature that amino acids are deleted or inserted (indels) due to mutation in the coding DNA; sometimes these will cause a functional difference, but if by adding a "gap" in the alignment enough extra amino acids align then it is likely that the sequences can still be considered similar. Indeed in functionally unimportant regions of the protein many indel events may have taken place across homologous proteins, but functionally important areas tend to be well preserved. How much of a gap to allow and how many extra amino acids should align is taken into account during the scoring of the match by the alignment algorithm. When performing local alignment it is also possible to start a totally new alignment at any position in the sequences, rather than adding a gap and continuing the current alignment. With global alignment gaps are almost always a necessity.

Alignment 2 shows the alignment with the added gap, we assume here that the extra AC adds enough to the alignment to negate the penalty of adding a gap (denoted by "–"). It may also be the case that rather than adding or removing an amino acid in the sequence, nature has changed it for a functionally very similar one. Alignment 3 shows such a scenario, where the system has considered Aspartic acid (D) and Glutamic acid (E) interchangeable enough to add one more amino acid to the alignment.

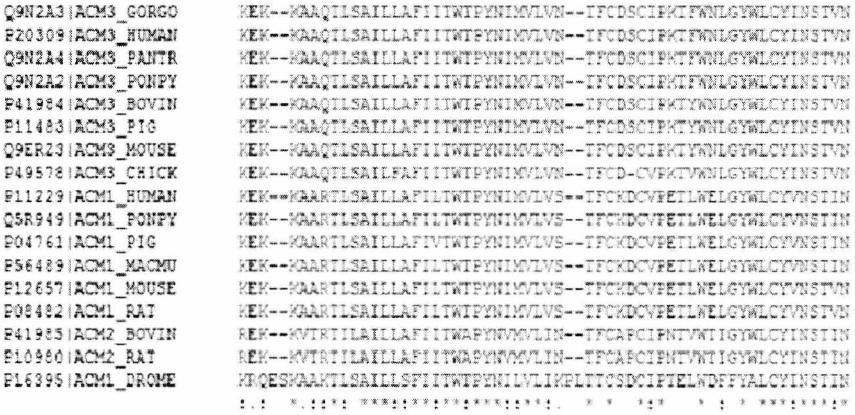
The method for sequence alignment seems relatively simple but leaves us with two important questions: when should the system stop allowing gaps and substitutions for the alignment, and how to calculate which amino acids substitute for others well. These questions are answered by employing a scoring system relying on a matrix of substitutions. Such a matrix will have entries corresponding to every possible amino acid substitution. Two common schemes for scoring an alignment are the BLOSUM [44] and PAM [77] matrices. Using these schemes it is possible to give each position in the alignment a score – presumably a positive one for a matching amino acid, and sometimes a negative one for non-matching amino acid. These matrices can be derived by looking at known conserved regions of pairs of proteins and seeing how many times each substitution takes place – high scores for pairs of amino acids that are often substituted for each other, and low scores for amino acids that are rarely seen substituting each other. Given this scoring system one can see how an optimal sequence alignment is computed; the addition of the individual scores for each pair of amino acids in the alignment minus the cost of adding gaps to the alignment. A common method of penalising the alignment for adding a gap is called the affine gap penalty, this works by deducting a fixed amount for opening a gap, and then penalising a further amount proportional to the length of the gap.

One naïve algorithmic method of finding an optimal alignment would be to generate and then score all possible alignments, returning the best one(s). This would obviously take a prohibitively long time even on modern computers and in fact there are much more efficient algorithms. In global alignment the Needleman and Wunsch dynamic programming algorithm [110] finds an optimal alignment in $O(mn)$ time where m and n are the lengths of the two sequences respectively. At a basic level, to achieve this the algorithm breaks the problem into a set of similar sub-problems and then recursively examines them to find the optimal solution. The Smith and Waterman algorithm [145] uses a similar dynamic programming technique to find the optimal local alignment also in $O(mn)$ time.

These algorithms, while efficient, are not fast enough for searching large databases for related proteins. Algorithms such as BLAST [3] and FASTA [121] allow the user to find related sequences amongst large numbers of sequences in good time. To achieve this increase in speed, the proteins are broken into sub-sequences and these are compared against each other (using hash values [121]). Then, larger (non-gapped) segments are found using a scoring matrix, and these segments are finally joined together using a dynamic programming algorithm. Importantly it must be possible to measure how significant each alignment is. The Z-value [121] gives such a measure where the alignment against the potential match is statistically compared against that of random proteins.

A pair-wise comparison of proteins can sometimes be limiting. For instance, consider the scenario where a biologist is investigating if and how multiple proteins are related. When examining a family of proteins that are all descendants of a common ancestor, it might be useful to find out which regions are conserved – and so are likely functionally important within that family. Although dynamic programming techniques have in some sense conquered pair-wise global and local alignment, the same cannot be said for multiple sequence alignment. A dynamic programming approach would take $O((2L)^n)$ time, with n sequences and L being the average sequence length [57]. For this reason many heuristic-based approaches have been proposed [155]. With some of the more recent work conducted in this field using naturally-inspired algorithms such as evolutionary algorithms and particle swarm optimisation [134][143].

Perhaps the most famous and widely used multiple sequence alignment algorithm is the ClustalW algorithm [154]. A multiple sequence alignment performed by ClustalW is shown in Figure 2.3, where the protein identifiers run down the far left column, and the colours represent certain properties of the amino acids in the sequences such as whether they are acidic. The symbols at the bottom "*", ":" and ".", show that the amino acid is always identical across all proteins, that substitutions have taken place and that less compatible substitutions have taken place (respectively).



```
Q9N2A3|ACM3_GORGO      KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTFWNLGYWLCYINSTVN
P20309|ACM3_HUMAN      KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTFWNLGYWLCYINSTVN
Q9N2A4|ACM3_PANTR      KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTFWNLGYWLCYINSTVN
Q9N2A2|ACM3_PONPY      KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTFWNLGYWLCYINSTVN
P41984|ACM3_BOVIN      KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTYWNLGYWLCYINSTVN
P11483|ACM3_PIG        KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTYWNLGYWLCYINSTVN
Q9ERZ3|ACM3_MOUSE      KEK--KAAQTLSAILLAFIITWTPYNIMVLVN--TFCDSCIPKTYWNLGYWLCYINSTVN
P49578|ACM3_CHICK      KEK--KAAQTLSAILFAFIITWTPYNIMVLVN--TFCD-CVPKTVWNLGYWLCYINSTVN
P11229|ACM1_HUMAN      KEK--KAARTLSAILLAFILTWTPYNIMVLVS--TFCKDCVPETLWELGYWLCYVNSTIN
Q5R949|ACM1_PONPY      KEK--KAARTLSAILLAFILTWTPYNIMVLVS--TFCKDCVPETLWELGYWLCYVNSTIN
P04761|ACM1_PIG        KEK--KAARTLSAILLAFIVTWTPYNIMVLVS--TFCKDCVPETLWELGYWLCYVNSTIN
P56489|ACM1_MACMU      KEK--KAARTLSAILLAFILTWTPYNIMVLVS--TFCKDCVPETLWELGYWLCYVNSTIN
P12657|ACM1_MOUSE      KEK--KAARTLSAILLAFILTWTPYNIMVLVS--TFCKDCVPETLWELGYWLCYVNSTVN
P08482|ACM1_RAT        KEK--KAARTLSAILLAFILTWTPYNIMVLVS--TFCKDCVPETLWELGYWLCYVNSTVN
P41985|ACM2_BOVIN      REK--KVTRTILAILLAFIITWAPYNVMVLIN--TFCAPCIPNTVWTIGYWLCYINSTIN
P10980|ACM2_RAT        REK--KVTRTILAILLAFIITWAPYNVMVLIN--TFCAPCIPNTVWTIGYWLCYINSTIN
P16395|ACM1_DROME      KRQESKAAKTLSAILLSFIITWTPYNILVLIKPLTTCSDCIPTELWDFFYALCYINSTIN
                       :.:  *.::*: ***::**:**:***::**:.   * *   *:*    * : * ******:*
```

Figure 2.3: Part of a multiple sequence alignment of Muscarinic Acetylcholine receptor GPCR proteins using ClustalW

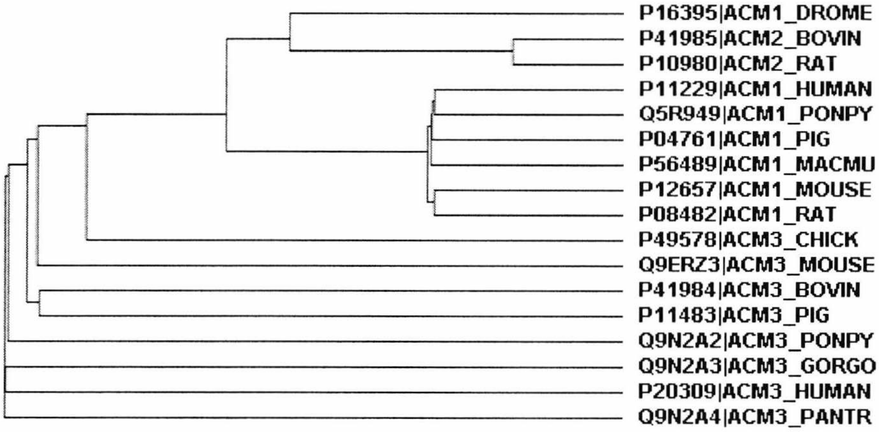


Figure 2.4: An approximate phylogenetic tree derived from the multiple sequence alignment of Muscarinic Acetylcholine receptor GPCR proteins using ClustalW

Although many techniques have been devised to perform multiple sequence alignment in reasonable time the basic principle usually involves first building a phylogenetic tree [154]. This tree of evolutionary relationship is constructed by finding the most closely related protein sequences using pair-wise sequence alignment. After all sequences have been aligned a matrix can be created showing the most related proteins. An approximate phylogenetic tree or guide tree (an example is shown in Figure 2.4) can then be constructed using hierarchical clustering to guide the pair-wise alignments in the multiple sequence alignment. For instance, in Figure 2.4 ACM2_Bovin is considered to be most closely related to ACM2_Rat, these sequences are aligned, and then the alignment for that two-protein cluster is aligned against ACM1_Drome, which the algorithm considers the second most related sequence.

2.4. Motif Detection using Alignment-Based Similarity Search

Although sequence alignment on its own is a good way to detect homology amongst proteins, it is not always flexible, sensitive (detecting as many homologous proteins as possible) or specific (not misclassifying non-homologous proteins as homologous) enough - especially when the percentage of similarity between homologous proteins is low [114]. For this reason more advanced techniques have been devised including Hidden Markov Models (HMMs), profiles, Fingerprints and regular expressions or patterns. HMMs, profiles and Fingerprints are used to identify homology based on the entire sequence, whereas patterns are used to detect smaller and somewhat independent functional units (domains).

All the techniques mentioned in this sub-section rely on sequence alignment in one way or another. This reliance on sequence alignment – the fact that any prediction made will directly relate to sets of amino acids within the search sequence – gives the prediction a certain level of transparency and comprehensibility. This is obviously important; it is not always enough to merely give an answer, the reason for the answer may be equally important. This is especially true in proteomics as the grouping of similar amino acids may often give insights into how the functionalities of proteins are conserved throughout protein families. However, this kind of information is at a relatively low level of abstraction (at the level of individual proteins), whereas information gained at a higher level of abstraction (at the level of a classification model) may reveal broader patterns, for instance, information about how protein domains interact.

2.4.1. Regular Expressions

Regular expressions or patterns are good at detecting the most conserved regions of proteins and usually detect a sub-sequence of around 10-20 amino acids [103]. Because of their small size they can detect very distantly related proteins via their highly conserved functional regions. As they are so specific they must be very carefully crafted to maintain sensitivity and specificity, for this reason the regular expressions used for patterns are often created semi-manually. To create a regular expression, a common starting point consists of using a multiple sequence alignment, or using some information about commonalities amongst a protein family. The basic components of a regular expression [17] are:

- Each position is separated by a hyphen
- An uppercase character only matches itself
- An x means any amino acid
- [] brackets contain multiple amino acids that can match
- [R]* matches any number of amino acids
- {} brackets contain amino acids that are not allowed
- () brackets contain the number of repeats

For instance, the regular expression (pattern) PS00237 found in the PROSITE [85] database (to be discussed later) is as follows:

[GSTALIVMFYWC] - [GSTANCPDE] - {EDPKRH} - x - {PQ} - [LIVMNQGA] - {RK} - {RK} - [LIVMFT] - [GSTANC] - [LIVMFYWSTAC] - [DENH] - R - [FYWCSH] - {PE} - x - [LIVM]

This expression was created to describe a sub set of GPCR (G protein-coupled receptor) proteins and covers this section of the primary sequence of the ACM1_Drome protein (Fruit fly GPCR) sequence:

ASVLNLLIISFDRYFSV (From the entire sequence):

MEPVMSLALAAHGPPSILEPLFKTVTTSTTTTTTTTTSTTTTTASPAGYSPGYPGTTLLT

ALFENLTSTAASGLYDPYSGMYGNQTNGTIGFETKGPRYSLASMVVMGFVAAILSTVTVA

GNVMVMISFKIDKQLQTISNYFLFSLAIADFAIGAISMPLFAVTTILGYWPLGPIVCDTW

LALDYLASN**ASVLNLLIISFDRYFSV**TRPLTYRAKRTTNRAAVMIGAAWGISLLLWPPWI

YSWPYIEGKRTVPKDECYIQFIETNQYITFGTALAAFYFPVTIMCFLYWRIWRETKKRQK

DLPNLQAGKKDSSKRSNSSDENTVVNHASGGLLAFAQVGGNDHDTWRRPRSESSPDAESV

YMTNMVIDSGYHGMHSRKSSIKSTNTIKKSYTCFGSIKEWCIAWWHSGREDSDDFAYEQE

EPSDLGYATPVTIETPLQSSVSRCTSMNVMRDNYSMGGSVSGVRPPSILLSDVSPTPLPR

PPLASISQLQEMSAVTASTTANVNTSGNGNGAINNNNNASHNGNGAVNGNGAGNGSGIGL

GTTGNATHRDSRTLPVINRINSRSVSQDSVYTILIRLPSDGASSNAANGGGGGPGAGAAA

SASLSMQGDCAPSIKMIHEDGPTTTAAAPLASAAATRRPLPSRDSEFSLPLGRRMSHAQ

HDARLLNAKVIPKQLGKAGGGAAGGGVGGAHALMNARNAAKKKKKSQEKRQESKAAKTLS

AILLSFIITWTPYNILVLIKPLTTCSDCIPTELWDFFYALCYINSTINPMCYALCNATFR

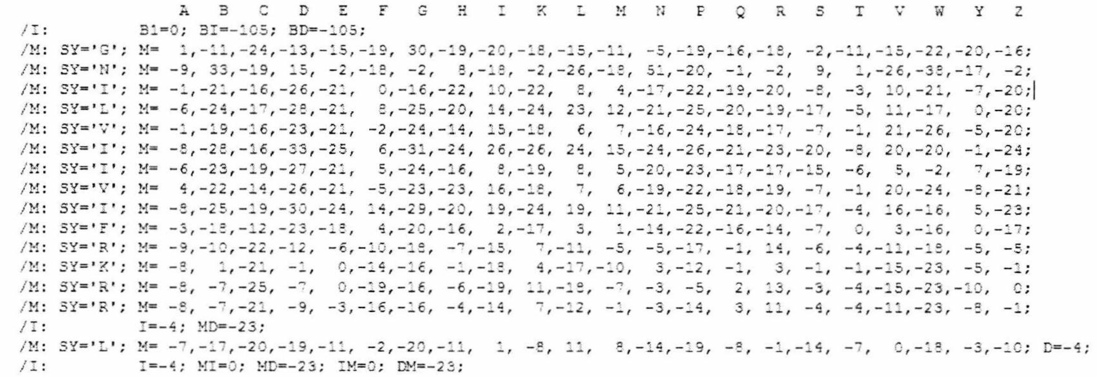
RTYVRILTCKWHTRNREGMVRGVYN

Figure 2.5: Sequence for ACM1_DROME Muscarinic acetylcholine receptor from Drosophila Melanogaster (Fruit fly).

During the creation of such a pattern a database of proteins must be iteratively searched to try and improve/expand the pattern. Initially a candidate pattern might cover proteins that are not of the correct type (false positives), in which case the pattern must be expanded to make it more specific. It is important that when making the pattern more specific the modifications do not exclude too many proteins (false negatives) from the target set decreasing the sensitivity. This procedure must be repeated until the pattern covers only its target proteins, or until no further improvement is possible.

2.4.2. Profiles

Profiles [26] are matrices derived from a multiple sequence alignment. After a multiple sequence alignment is performed it is possible to scan sequences characterised by that alignment and construct a scoring matrix based on how many times each amino acid occurs at each position. For instance, in Figure 2.3 at the first position within the alignment, Lysine (K) is very common and there are two instances of Arginine (R). Therefore these two amino acids would have a positive score in the first row in the matrix. Other amino acids would have scores depending on how similar or dissimilar they are to these two amino acids, this scoring system could be based on BLOSUM [44] and PAM [77].



```
                    A   B   C   D   E   F   G   H   I   K   L   M   N   P   Q   R   S   T   V   W   Y   Z
/I:             B1=0; BI=-105; BD=-105;
/M: SY='G'; M=  1,-11,-24,-13,-15,-19, 30,-19,-20,-18,-15,-11, -5,-19,-16,-18, -2,-11,-15,-22,-20,-16;
/M: SY='N'; M= -9, 33,-19, 15, -2,-18, -2,  8,-18, -2,-26,-18, 51,-20, -1, -2,  9,  1,-26,-38,-17, -2;
/M: SY='I'; M= -1,-21,-16,-26,-21,  0,-16,-22, 10,-22,  8,  4,-17,-22,-19,-20, -8, -3, 10,-21, -7,-20;
/M: SY='L'; M= -6,-24,-17,-28,-21,  8,-25,-20, 14,-24, 23, 12,-21,-25,-20,-19,-17, -5, 11,-17,  0,-20;
/M: SY='V'; M= -1,-19,-16,-23,-21, -2,-24,-14, 15,-18,  6,  7,-16,-24,-18,-17, -7, -1, 21,-26, -5,-20;
/M: SY='I'; M= -8,-28,-16,-33,-25,  6,-31,-24, 26,-26, 24, 15,-24,-26,-21,-23,-20, -8, 20,-20, -1,-24;
/M: SY='I'; M= -6,-23,-19,-27,-21,  5,-24,-16,  8,-19,  8,  5,-20,-23,-17,-17,-15, -6,  5, -2,  7,-19;
/M: SY='V'; M=  4,-22,-14,-26,-21, -5,-23,-23, 16,-18,  7,  6,-19,-22,-18,-19, -7, -1, 20,-24, -8,-21;
/M: SY='I'; M= -8,-25,-19,-30,-24, 14,-29,-20, 19,-24, 19, 11,-21,-25,-21,-20,-17, -4, 16,-16,  5,-23;
/M: SY='F'; M= -3,-18,-12,-23,-18,  4,-20,-16,  2,-17,  3,  1,-14,-22,-16,-14, -7,  0,  3,-16,  0,-17;
/M: SY='R'; M= -9,-10,-22,-12, -6,-10,-18, -7,-15,  7,-11, -5, -5,-17, -1, 14, -6, -4,-11,-18, -5, -5;
/M: SY='K'; M= -8,  1,-21, -1,  0,-14,-16, -1,-18,  4,-17,-10,  3,-12, -1,  3, -1, -1,-15,-23, -5, -1;
/M: SY='R'; M= -8, -7,-25, -7,  0,-19,-16, -6,-19, 11,-18, -7, -3, -5,  2, 13, -3, -4,-15,-23,-10,  0;
/M: SY='R'; M= -8, -7,-21, -9, -3,-16,-16, -4,-14,  7,-12, -1, -3,-14,  3, 11, -4, -4,-11,-23, -8, -1;
/I:             I=-4; MD=-23;
/M: SY='L'; M= -7,-17,-20,-19,-11, -2,-20,-11,  1, -8, 11,  8,-14,-19, -8, -1,-14, -7,  0,-18, -3,-10; D=-4;
/I:             I=-4; MI=0; MD=-23; IM=0; DM=-23;
```

Figure 2.6: Part of a profile from Prosite [85]

Figure 2.6 shows part of Prosite profile PS50262 for GPCR proteins. This scoring matrix has scores for each amino acid (columns) for each position in the alignment (rows). The first two characters in each row identify whether the position in the alignment is either a match (/M) or an insert (/I), with the SY identifier showing the original amino acid in the alignment. There are default costs associated with certain operations such as going from a match to an insert (MI) or having a delete (D) but they can be overridden by the parameters in each individual insert row [85].

Obviously the quality of the original multiple sequence alignment is very important for the specificity and sensitivity of the final profile. To find if a protein sequence matches this profile well, the search sequence must first be aligned against the profile alignment and then scored according to the generated scoring matrix. It is therefore possible to calculate an E-value or the expected value – the likelihood that the match is by chance alone. This value is also dependant on the database size, so large databases correlate to a higher E-value. A low E-value is indicative of a good match with a high probability that the sequences are really homologous.

2.4.3. Hidden Markov Models (HMM)

A HMM (developed by Leonard Baum and others [132]) is a probabilistic construct that consists of a set of linked states; each state is a set of probabilities that relate to the chance of emitting a particular value. In the case of a basic profile HMM [97] (a HMM for the characterisation of a set of proteins) the states are inserts, deletions and matches. A profile HMM is in some ways conceptually similar to a standard profile, in that each match state in the HMM functions in the same way as each column in a profile - holding a set of probabilities, one for each particular amino acid and for each insert. The insert state holds probabilities for inserting any amino acid, and the delete state always emits a delete value representing a deletion in the sequence. Connecting these states (Figure 2.7) are transition probabilities, the probability of going from one state to another. In this way a profile HMM can be considered a probabilistic sequence generator starting sequence generation at the non-emitting dummy state M_0 and ending at the dummy state M_{i+1}. Following the topology in Figure 2.7 it can be observed that the model may emit any number of inserted amino acids between each emitted match amino acid, however it may not add a delete and an insert for the same column (of the original alignment) – as this would simply remove the last insert.

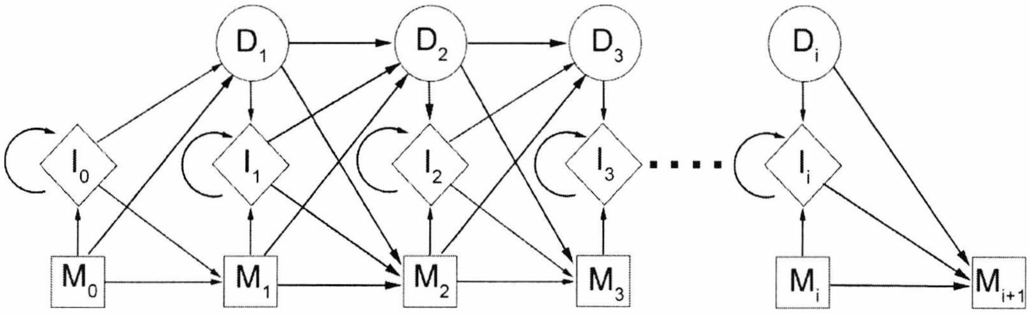
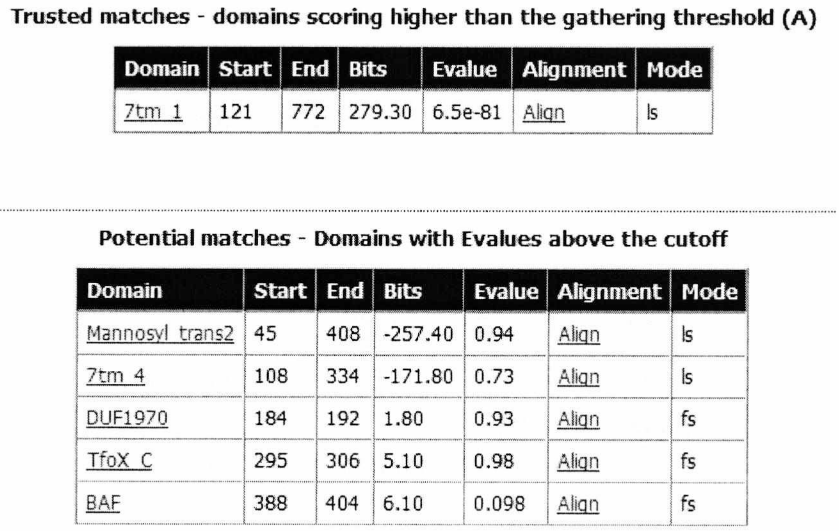


Figure 2.7: A profile HMM showing match states (M), delete states (D) and insert states (I), with transition probabilities shown as arrows. The beginning state is M_0 and the end state is M_{i+1} adapted from [97].

To create a profile HMM for a set of proteins, a good initial seed multiple sequence alignment is needed. Although it is possible to train the model on unaligned sequences, this is not done in practice as a suboptimal model is more likely to be produced. The number of match states can then be deduced along with the transition probabilities and emission probabilities from the observed number of amino acids in the aligned sequences [78]. This model can then be improved by aligning new sequences to it using the Viterbi algorithm [160] and then again using the Viterbi algorithm to retrain the HMM. This process can be iterated until the desired set of proteins are covered and the parameters of the HMM become stable. Of course it is important to monitor the HMM during its training to ensure specificity is maintained. HMMs are some of the best ways of detecting homology (distant or otherwise), tending to be very accurate.

HMMER [53] is a popular suite of programs used to create and use profile HMMs and is the basis of the well-known Pfam database [7]. It is possible to find the HMM that most likely produced a given sequence from a set of HMMs [53]. If each HMM characterises a family of proteins then it is likely that a "hit" against one of these HMMs means that a given sequence belongs to that family.



Trusted matches - domains scoring higher than the gathering threshold (A)

Domain	Start	End	Bits	Evalue	Alignment	Mode
7tm_1	121	772	279.30	6.5e-81	Align	ls

Potential matches - Domains with Evalues above the cutoff

Domain	Start	End	Bits	Evalue	Alignment	Mode
Mannosyl_trans2	45	408	-257.40	0.94	Align	ls
7tm_4	108	334	-171.80	0.73	Align	ls
DUF1970	184	192	1.80	0.93	Align	fs
TfoX_C	295	306	5.10	0.98	Align	fs
BAF	388	404	6.10	0.098	Align	fs

Figure 2.8: A Pfam search against ACM1_Drome showing the HMM hits with the most likely hit and other possible hits based on E-value.

A Boolean answer cannot be given as to whether a sequence was generated by a given HMM (unlike with regular expressions). However, E-values can be calculated and below a certain threshold a good certainty is obtainable. Figure 2.8 shows a search performed against Pfam HMMs with the ACM1_Drome sequence. As can be seen the search has resulted in one highly likely match, the HMM (7tm_1) characterising the family with 7tm or seven trans-membrane helices usually associated with GPCRs. The other top HMMs (representing different types of domains) are also displayed but have a relatively high E-value and so are more likely chance matches, although 7tm_4 is displayed that is of a similar type to the protein.

```
                 *->GNlLVilvilrtkklrtptnifilNLAvADLLflltlppwalyylvg
                    GN++V++ +   k+l+t +n+f+++LA+AD+ +++  +p ++++ +
ACM1_DROME   121    GNVMVMISFKIDKQLQTISNYFLFSLAIADFAIGAISMPLFAVTTIL 167

                 gedWpfGsalCklvtaldvvnmyaSillLtaISiDRYlAIvhPlryrrrr
                 g +Wp+G+++C+++ ald++++ aS+l+L+ IS+DRY ++++Pl yr++r
ACM1_DROME   168 G-YWPLGPIVCDTWLALDYLASNASVLNLLIISFDRYFSVTRPLTYRAKR 216
```

Figure 2.9: Part of the alignment for ACM1_Drome against the 7tm_1 (seven trans-membrane helices) Pfam HMM.

Even though an HMM is essentially a set of numbers whose interpretation is not easy, it is possible to see, in part, how each HMM relates to the search protein via the alignment. Figure 2.9 shows part of such an alignment, with the original search sequence, on the bottom rows, labelled as ACM1_Drome. The numbers next to the ACM1_Drome label are the positions in the search sequence. The *-> symbol is the start of the alignment (notice that the alignment does not start at the beginning of the search sequence). The top rows are the most likely output of the HMM with capital letters showing highly probable emission values (which are likely conserved regions) and lower case showing less probable values. The middle line shows the exact matches following the same upper and lower case format as the top line. The + symbol means that the alignment has a positive score at that position. On the bottom set of rows an indel has been emitted and is represented by "-".

2.4.4. Fingerprints

Fingerprints [120] work in a similar way to a set of patterns, but rather than identifying a protein using a regular expression it uses a set of frequency matrices. Firstly, a multiple alignment is taken from a seeding sub-set of proteins from the set that are to be characterised. As this multiple alignment is taken and more distant homologous proteins are added, gaps appear. Rather than just allowing a single alignment with gaps, the alignment is split into separate motifs. An un-weighted frequency matrix is constructed from the motifs containing the number of times each amino acid at each position within the motifs occurs throughout the set of proteins. This matrix is considered un-weighted as only this frequency of observed amino acids is taken into account and not other more general scoring matrices, e.g., PAM and BLOSUM. These motifs are intended to correspond to important conserved regions through the sequences. Once the motifs and frequency matrices have been constructed the software scans a database of proteins to find all proteins that match all the motifs and orders them by score – according to the frequency matrix. This entire process of motif and frequency matrix construction is repeated on part of the new set of top scoring proteins in an attempt to make the motifs more general. This is continued until the set of proteins does not change between iterations. As a Fingerprint is constructed from multiple motifs it is possible to create a hierarchy within them, with some proteins not having all the motifs from a particular Fingerprint still considered part of the set of proteins characterised by it (but possibly only belonging to a different sub-family).

To find out which fingerprint best matches a given sequence involves using a different matching algorithm than the one used to create them. Fingerprint-scan [139] increases the sensitivity of the scanning process when compared to the algorithm for fingerprint creation. It achieves this by converting the motifs into un-gapped profiles using a choice of different scoring matrices (usually BLOSUM). The only real restriction in combining the predictions of the individual motifs is that the order in which the motifs occur in the profile is the same as the order in which they occur in the original fingerprint. Search hits are returned sorted by E-value and probability.

Figure 2.10 is a three-dimensional graphical view of a fingerprint (GPCR-RHODOPSN) taken from the PRINTS database [4] [129]. The areas that are covered by the motifs in the fingerprint are displayed in red. As can be seen these areas are the seven trans-membrane helices indicative of GPCR proteins. GPCRRHODOPSN is one of the Fingerprints which cover the ACM1_Drome GPCR protein and the one that gives the lowest E-value for that protein, out of all Fingerprints in the PRINTS database. Figure 2.11 details the GPCRRHODOPSN fingerprint match found. Each motif from the fingerprint is shown numbered on the right hand side, with the position of the match for each motif represented by a box. The box height corresponds to the similarity between the search sequence and the labelled motif. Also labelled is the numerical position (in the protein sequence) of the motif match under each box. Notice that for the first motif there are two potential matches within the sequence.

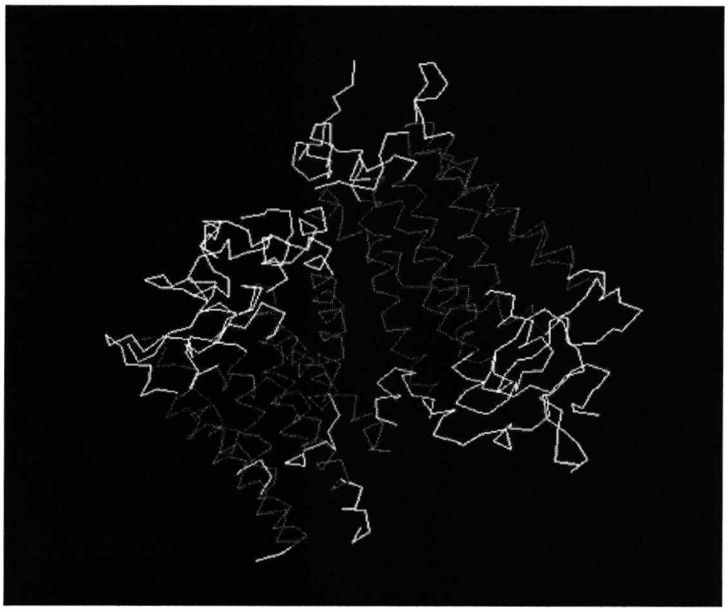


Figure 2.10: A graphical view of the GPCRRHODOPSN Fingerprint – shown in red, the Fingerprint corresponds to the seven Trans-membrane helices of GPCRs

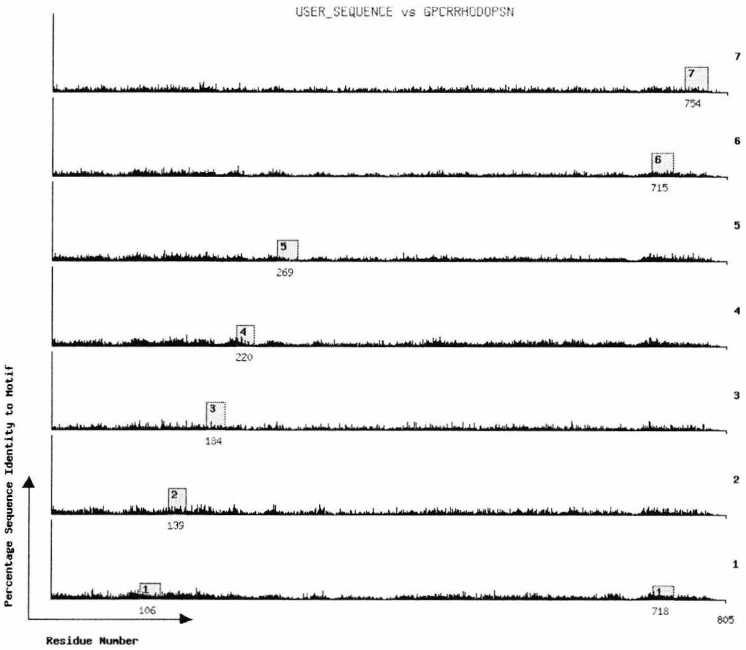


Figure 2.11: A graphical representation of the 7 motifs from the GPCRRHODOPSN Fingerprint covering the ACM1_Drome GPCR protein

2.5. Motif Databases

Motif databases are an invaluable resource for bioinformaticians and biologists alike. As their name might suggest they store motifs and related information so that each person who is interested in using the technology does not have to spend time generating motif data themselves (which given the number of known and potential proteins could be a very time consuming process). This section presents a brief overview of a subset of the numerous motif databases. More specifically, this section focuses on the motif databases used in generating the datasets investigated in the experiments presented in this thesis. This particular subset of possible motif databases (PROSITE, Pfam, PRINTS, InterPro) was chosen to explore as broad a set of motif generation approaches as possible.

2.5.1. PROSITE

Prosite [85] [126] is a database that contains entries to try and help identify which family unknown proteins belong to. These entries are known as Prosite motifs and contain patterns or profiles (as discussed in section 2.4) to identify one protein family from others along with descriptions of the nature of the pattern or profile. The patterns detect regions that code for sites such as [126]:

- Enzyme catalytic sites.
- Amino acids involved in binding a metal ion.
- Regions involved in binding a molecule.

Each Prosite entry contains a description of the family it covers, along with a list of protein matches from Swiss-Prot [166]. It also contains the number of true positives, false positives, true negatives and false negatives showing how sensitive and specific that particular entry is. Release 20.31 (April 2008) contains 1514 documentation entries that describe 1318 patterns and 784 profiles, and 784 ProRules (rules designed to increase the performance of the motifs).

2.5.2. Pfam

Pfam [7] [123] is another database designed to assign protein family. It uses profile HMMs (discussed in section 2.4) to perform this task and, like the other databases mentioned in this section, a user supplied protein sequence can be submitted to it. It returns the set of HMMs most likely to characterise that sequence ordered by E-value. There are two parts to Pfam, Pfam-A and Pfam-B. Pfam-A contains HMMs built from automatically generated seed alignments that are often hand-edited to maintain the quality of the produced HMM. Pfam-B is completely automatically generated, using seed alignment derived from the ProDom [25] database (a database containing automatically generated protein families). For this reason Pfam-B may, in some cases, be unreliable. Most Pfam functionality is based on the HMMER [53] suite of programs. Each entry in Pfam contains similar – but less well annotated – information when compared to Prosite. However, links to other well annotated databases are provided (such as InterPro) for each family characterised by each HMM. Release 22.0 (July 2007) contains 9318 families for which HMMs have been constructed.

2.5.3. PRINTS

Prints [4] [129] is another freely available database that contains entries to try and identify a protein family based on its sequence. In each FingerPrint (discussed in section 2.4) entry, the set of motifs, along with good annotation and information relating to the sensitivity and specificity, are present. A hierarchy is also provided that allows the user to see the connections between those FingerPrints that do not possess all the motifs of their parent FingerPrint. Release 38.1 (June 2007) of PRINTS contains 1904 entries, encoding 11,451 individual motifs.

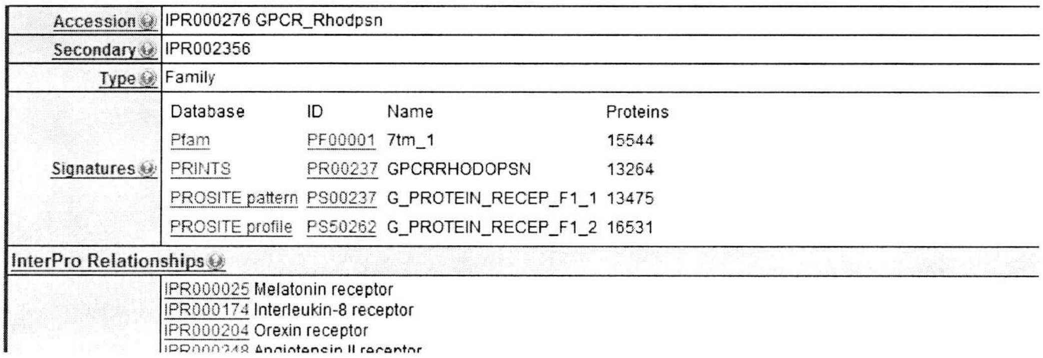
2.5.4. InterPro

InterPro [86] [109] is different from the motif databases mentioned so far in that it is a “composite protein pattern database”. Different large motif databases will inevitably have at least some overlap, covering the same families of proteins. Also, each database has its own advantages and disadvantages in certain situations. Obvious examples are the way in which Prosite with its regular expressions identifies families with small but highly conserved regions, Pfam with its HMMs is good at identifying proteins with small amounts of meaningful similarity and PRINTS is good at identifying smaller families – due to its hierarchical nature. Due to these facts InterPro was created to combine multiple motif databases into one easy to use database. Table 2.2 shows an overview of InterPro’s constituent databases. The database name is given in the first column, the second column shows the release version number, the third shows the total number of entries in that database, the fourth column shows the number of signatures that have been fully integrated into the InterPro entries and the fifth column gives a brief description of the type of approach the database uses.

Signature Database	Version	All Signatures	Integrated Signatures	Description
PANTHER	6.1	30128	2061	Hand edited protein functional classes characterised by HMMs
Pfam	21.0	8957	8957	See section 2.5.2
PIRSF	2.68	1748	1499	Provides detailed relationships between family members
PRINTS	38.0	1900	1898	See section 2.5.3
ProDom	2005.1	3538	1041	Automatically assigned protein families.
PROSITE patterns	20.0	1319	1319	See section 2.5.1
SMART	5.1	724	721	Automatically identifies the type of genetically mobile domain and provides automatic annotation. It also provides an analysis of domain architectures.
TIGRFAMs	6.0	2949	2933	Can predict function based on homology using HMMs
Gene3D	3.0.0	2147	783	Facilitates the prediction of structure using a Markov clustering algorithm
SUPERFAMILY	1.69	1538	463	HMMs of proteins with known structure

Table 2.2: Constituent InterPro databases

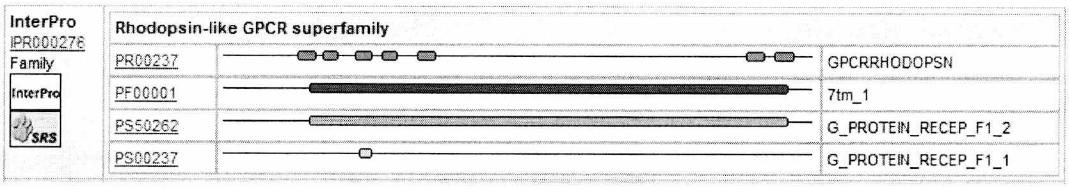
The process involved in amalgamating these different databases is obviously quite complex, but basically consists of creating relationships between the individual entries from the different constituent databases. The relationships can take the form of a hierarchy, where each child entry only has one parent. However, entries from more than one database can make up a single InterPro entry. There are also possibly overlapping entries, where more than one InterPro entry covers one protein. The annotations from each entry from each database are merged to form the annotation in the InterPro entry, making the annotation as comprehensive as possible.



Accession	IPR000276 GPCR_Rhodpsn			
Secondary	IPR002356			
Type	Family			
Signatures	Database	ID	Name	Proteins
	Pfam	PF00001	7tm_1	15544
	PRINTS	PR00237	GPCRRHODOPSN	13264
	PROSITE pattern	PS00237	G_PROTEIN_RECEP_F1_1	13475
	PROSITE profile	PS50262	G_PROTEIN_RECEP_F1_2	16531
InterPro Relationships				
	IPR000025 Melatonin receptor IPR000174 Interleukin-8 receptor IPR000204 Orexin receptor IPR000248 Angiotensin II receptor			

Figure 2.12: An example Interpro entry matching the ACM1_Drome GPCR sequence (only part of the relationships cell is shown)

The searching mechanisms from each constituent database are maintained in InterPro using InterProScan [168]. Essentially, this tool wraps the individual search programs used for each database and provides some filtering of the returned search results. A user can search InterPro for motif matches using any combination of the motif databases included in InterPro. The results of a search performed on InterPro can be seen in Figure 2.12, where all constituent databases were selected and the top hit was the InterPro entry "IPR000276" (not that it is possible for more than one InterPro entry to match a given protein sequence). This contained the motifs from Pfam, PRINTS and the PROSITE pattern and profile.



InterPro IPR000276 Family	Rhodopsin-like GPCR superfamily	
	PR00237	GPCRRHODOPSN
	PF00001	7tm_1
	PS50262	G_PROTEIN_RECEP_F1_2
	PS00237	G_PROTEIN_RECEP_F1_1

Figure 2.13: An example InterPro search for the ACM1_Drome GPCR sequence showing from top to bottom, PRINTS, Pfam, PROSITE profile and PROSITE pattern entries matching the search sequence

If we observe the way in which the individual entries from some of the constituent InterPro databases match the search sequence (Figure 2.13) it becomes clear the different ways in which they work. At the top the PRINTS Fingerprint covers the seven transmembrane helices using multiple motifs. The HMM from Pfam covers much of the sequence along with the profile from PROSITE. Notice that the pattern from PROSITE only covers a very small region of the primary sequence. InterPro release 17.0 (March 2008) contains a total of 16583 entries covering 79% of all proteins contained in UniProtKB (an extremely large protein database – see Section 2.6.1). InterPro clearly has the largest number of entries out of any motif database examined.

2.6. Biological Databases

2.6.1. UniProt

UniProt [157] [166] is a large protein database freely available on the web. It is formed from a consortium of EBI (European Bioinformatics Institute), SIB (Swiss Institute of Bioinformatics), and PIR (Protein Information Resource) to create as comprehensive and complete as possible protein resource (based on the best current knowledge). It contains sequence data, citation information (bibliographical references), taxonomic data (description of the biological source of the protein) along with other extensive annotations. It also has cross references for other major biological databases such as Prosite, Prints, Pfam and InterPro, etc. UniProt consists of two separate databases: Swiss-Prot and TrEMBL. Swiss-Prot is a database of high quality manually annotated proteins. TrEMBL entries are automatically annotated with most of the sequences themselves being automatically generated by simulated translations from genetic code held in the EMBL database [38]. UniProt is extensively used in this project, in particular as a source of data for creating the data sets used in our experiments.

The first release of Swiss-Prot in 1986 contained 3939 sequences, whereas in February 2008 (release 55) it contained 356194. TrEMBL (release 38 – February 2008) contains 5395414 entries, a 23% increase from the last release [157]. TrEMBL has grown at a near exponential rate since it was created in 1996. From these figures it is clear to see the growing gap between the ability to manually annotate proteins and the number of potential proteins being found.

2.6.2. GPCRDB

G-protein-coupled receptors (GPCRs) [60] are proteins involved in signalling. They span cell walls so that they influence the chemistry inside the cell by sensing the chemistry outside the cell. More specifically, when a ligand (a substance that binds to a protein) is received by the part of the GPCR on the outside of the cell, it (usually) causes an attached G-protein to activate and detach. This is a mechanical biological switch that causes the released G-Protein to affect other reactions within the cell. More than one GPCR can bind to more than one G-Protein, causing a complex set of pathways that can perform intricate functions within the cell. Common amongst all GPCRs are the seven trans-membrane α-helices (mentioned previously), with three intracellular loops and three extracellular loops. GPCRs are particularly important for medical applications because it is believed that 40%-50% of current drugs target GPCR activity [60]. The types of function GPCRs facilitate are extremely varied, from detecting light to managing brain chemistry.

GPCRDB [70] provides a classification system for GPCRs. It arranges them into a hierarchy based on a function. Each functional class has one and only one parent class with each child class becoming more specific. For instance the functional class 1.1.1.1 belongs to classes:

- Class 1 (Class A Rhodopsin like).
- Class 1.1 (Amine).

2.7. Summary

This chapter has presented the fundamental biological concepts required for this thesis. It has explored some of the technologies used to detect similarity between proteins. It has provided a survey of the motif and protein databases used to generate the data sets used in the experiments found in this thesis. Also, the difference between function inferred directly from similarity and function predicted using a model has been defined.

- Class 1.1.1 (Muscarinic acetylcholine).
- Class 1.1.1.1 (Musc. acetylcholine Vertebrate type 1).

The ACM1_Drome protein belongs to class 1.1.1.101, which has the same parents as 1.1.1.1, but rather than being of type "Musc. acetylcholine Vertebrate type 1", it is of type "Musc. acetylcholine Non Vertebrate" (Drome is the code given to fruit flies).

2.6.3. Enzyme Nomenclature

Enzymes are another subset of proteins. They are catalysts which are used to speed up most of the chemical reactions that take part within the cell, without being altered themselves during the reaction. They are the target of a further 28% of current drugs [114]. They are usually very specific and only catalyse certain types of reaction within the cell. Often other molecules known as inhibitors may slow down the reactions of a particular enzyme; conversely activators increase the rate of reaction. This is used to control both the speed of reaction and the course of overall reaction pathways that take place within the cell. Examples of well known enzymes are those in biological washing powder.

The Enzyme Nomenclature [16] was developed as a classification system for this family of proteins. It defines what type of reaction each enzyme catalyses in a hierarchical manner. For instance, an enzyme with the EC code (Enzyme Commission) of 1.1.1.1 belongs to:

- Class 1 (Oxidoreductases).
- Class 1.1 (acting on the CH-OH group of donors).
- Class 1.1.1 (with NAD or NADP as acceptor).
- Class 1.1.1.1 (Alcohol Dehydrogenase).

Each class has one and only one parent class, with the root node being all enzymes.

Chapter 3. Data Mining

Data mining is an interdisciplinary field involving mainly machine learning and statistics. At this point in time the ability of machines to learn in a similar way to humans is a distant ambition. Therefore, in this context, the word learn is relaxed to encompass a more achievable target, for machines to be able to "improve through experience" [106].

At the most basic level data mining involves finding useful and nontrivial patterns within data. To quote Witten and Frank [164]:

> *"People have been seeking patterns in data ever since human life began. Hunters seek patterns in animal migration behaviour, farmers seek patterns in crop growth" ..... "In data mining, the data is stored electronically and the search [for patterns] is automated – or at least augmented – by computer. Even this is not particularly new. Economists, statisticians, forecasters, and communication engineers have long worked with the idea that patterns in data can be sought automatically, identified, validated, and used for prediction. What is new is the staggering increase in opportunities for finding patterns in data."*

While the pursuit of finding patterns within data is not a new one the availability and quantity of this data is. Also, as the complexity of the data becomes greater the ability of humans to understand and utilise it in its raw form becomes less. Therefore, the importance of data mining (as a means of dealing with large quantities of complex data) has increased; it is a necessity in a modern world where we are swamped with data.

For the purposes of this thesis the data to be mined is considered to be contained in structured data sets. A data set is simply a set of examples, each example consisting of a set of attribute-value pairs. A data set containing information about credit might have attributes: Mortgage, Wage, Sex and Credit_Rating. A single example in this data set might be the following:

```
Yes, 21000, Male, Good
```

Where Mortgage can take the values *Yes* or *No*, Wage may take any continuous value, Sex can have the nominal/categorical values *Male* or *Female* and Credit_Rating can take the nominal/categorical values *Good, Average* or *Bad.*

It should be noted that the careful preparation of a data set for data mining is crucial and usually very time consuming [130]. The data miner must (iteratively) become familiar with the data sources, integrate the sources, transform the data, cleanse the data (removing errors, dealing with missing values, duplicates etc) and select which attributes/class labels to use before the data can be interpreted by a learning algorithm.

Many different machine learning methods have been developed to find and represent patterns in data. In general these methods can be split into three distinct groups, supervised, unsupervised and semi-supervised.

In *supervised learning* (the subject of this thesis) the class label of the example is known during the training phase, for instance the class label in the previous example might be the Credit_Rating, if we wish to know what credit rating to assign a person based on certain attribute-value pairs.

In *unsupervised learning* the class label is unknown during the run of the algorithm. This can be useful when trying to discover relationships between attribute-value pairs that are not necessarily dependant on a given class label. In fact, in unsupervised leaning the learning procedure generates its own class labels. If the example data set were used for unsupervised learning then the class label Credit_Rating would not be considered when discovering patterns.

In *semi-supervised* learning only some of the class labels are known during training, this is often the case when it is difficult or laborious to assign a class label to an example.

3.1. The Classification Task of Data Mining

Supervised learning consists of two main tasks. Firstly the classification task [164], where the predictions consist of nominal/categorical class labels. Secondly the regression task [14], where the predictions are continuous values (e.g., Wage from credit example).

It is possible to make predictions based on discovered patterns as they describe relationships between attribute-value pairs and class labels. Therefore, given a set of attribute-value pairs for a given example it is possible to use the discovered patterns (classification model) to predict which class label the example is likely to have. One of the simplest ways of assessing how good discovered patterns are is by measuring their predictive accuracy. This is achieved by splitting the data set into two separate partitions, a training set used to build the classification model, and a test set used to measure the predictive accuracy of that model. Examples belonging to the training set must not exist in the test set and vice-versa. If examples belonged to the training set and test set then the measured accuracy would not reflect predictive accuracy well. This is due to the need for the model to generalise well. It would be relatively easy to generate a model that matched the training set exactly (e.g., a simple mapping from each example's attribute-value pairs to its class label) but this model would have no generalising power, it could not cope with simple variations in new examples with an unknown class. After the patterns have been extracted from the training set they are then applied to the test set to predict which class label each example might have (in the test set the example class labels are unseen by the classification algorithm). The predictive accuracy is then simply the number of examples in the test set that have the same predicted and true class labels divided by the total number of examples in the test set.

Aside from predictive accuracy another two useful metrics are *recall* and *precision.* Recall is the number of examples whose class label was predicted correctly for a given class divided by the total number of examples in that class. Note that it is simple to obtain 100% recall by always predicting the given class.

Precision is the number of examples whose class label was predicted correctly for a given class divided by the total number of predictions made. Note that making only one prediction and getting it correct would be enough to maximise precision.

It is important to use precision and recall together as it is usually easy to maximise either one individually.

It is important to note that typically the data set's records are a sample of a much larger population of possible records. The aim of the measure of predictive accuracy is to give an indication of how well the model represents the base data, and how well it will generalise to future data. Therefore, the sampling of the training and test sets is important. When randomly choosing examples for the training and test sets bias will be introduced [164]. In an attempt to reduce the bias experienced during the testing procedure it is usual to perform multiple iterations and find the mean predictive accuracy across all iterations. A common method to achieve this goal is cross-validation [164], where the data set is split into *n* partitions of approximately the same size and *n* classification models are generated, one per iteration. At each iteration (*i*) all but one of the partitions (*n*-1) are combined and used as the training set with the *i*th (*i*=1,...,*n*) partition being used as the test set. The predictive accuracy over the *n* iterations can then be averaged, producing the overall measure of predictive accuracy reported to the user.

There are many other ways of measuring how good a particular model is (many are dependant on the application) [37] [164]. Witten states that a general goal of data mining is to discover knowledge that is not only accurate, but also comprehensible [164], so the model representation medium is important. A model that is not easily (or at all) human comprehensible is termed a black box model, often used when predictive accuracy is the only consideration. Even if the user is not interested in the model itself the ability to easily validate predictions made is lost when using a black box approach. Indeed, for this reason, an area of research is devoted to converting incomprehensible models to comprehensible models. For instance, extracting comprehensible models from Neural Networks [65] [87], Support Vector Machines [113] and for fuzzy rules [141]. Therefore, another important way of assessing a classification model is its comprehensibility. Comprehensibility is not easily reduced to one single measurement, but in general it is standard practice in the literature to consider that the simpler and more compact the model is the more comprehensible it is.

3.2. Three Conventional Types of Algorithm for Classification

This section reviews three conventional types of learning algorithm and the classifiers that are created by them. These are the approaches that are particularly relevant to this thesis, namely: rule induction, decision tree and Bayesian.

3.2.1. Rule/Tree Induction

3.2.1.1. Rule Induction

In the classification task, rule induction algorithms generate rules representing patterns in the training set. These rules are then applied to a test set with examples of unseen (by the classification algorithm) or unknown classes. In classification the knowledge or patterns discovered in a data set can be represented in terms of a set of rules. A rule consists of an antecedent (a set of attribute-values and logical operators) and a consequent (class):

```
  IF <attrib operator value> AND ...
AND <attrib operator value> THEN <class>
```

The consequent of the rule is the class that is predicted by that rule. The antecedent consists of a set of terms, where each term is essentially an attribute-value pair. More precisely, a term is defined by a triple *<attribute, operator, value>*, where *value* is a value belonging to the domain of *attribute*. The *operator* typically is "=" in the case of categorical attributes, or "<" and "≥" in the case of continuous attributes. The logical operation that is performed on these terms is AND in this example, however, it could be any other operation, i.e., NAND, XOR etc. The knowledge representation in the form of rules has the advantage of being intuitively comprehensible to the user. An example rule might be:

```
IF (Salary > 30k) AND (Mortgage = No) THEN (Good Credit)
```

A rule that is to be considered good must not only correctly match (via its antecedent) as many examples in its predicted class as possible, but also *not* match (cover) examples in classes different from its predicted class. These two objectives are often at odds with each other and a trade off often occurs between recall and precision.

Another issue to be considered while generating classification rules is how complicated they are (comprehensibility is a goal of data mining as stated earlier) and how to measure this complexity. The complexity of a rule set is usually measured by two factors, namely the number of discovered rules and the number of terms per rule. In general, the lower these numbers the simpler and so more comprehensible the rule set is considered to be. Note that the number of discovered rules is related to the recall of those rules. After all, if each rule has a large recall (i.e., covers many examples), the number of rules necessary to cover all the examples will be relatively small, in comparison to a scenario where rules have a lower recall.

In a seminal paper on cognitive science, George A. Miller [105] describes the problems humans have when trying to comprehend sets of things with greater than seven components. This can be used as a rough guide as to the simplicity of a rule and it can be presumed that rules with tens of terms will tend to be incomprehensible to the average user. Obviously it would not be useful to have a very simple rule set that did not represent the data set well, so a notion of accuracy and comprehensibility is essential during rule evaluation.

Pseudocode 3.1 describes the standard sequential covering approach used to build a set of classification rules covering a given training set [118] [164]. Note that Pseudocode 3.1 represents a very generic approach, which can be instantiated in many different ways to produce a number of very different rule induction algorithms as discussed later.

```
RS = ∅ /* initially, Rule Set is empty */
TS = {all training examples}
FOR EACH Class C
    WHILE (number of uncovered training examples of class C >
    MaxUncovExampPerClass)
           Discover the best rule predicting class C, called
              BestRule
           RS = RS ∪ {BestRule}
           TS = TS - {training examples correctly covered by
              BestRule}
    END WHILE
END FOR
```

Pseudocode 3.1: Sequential Covering Approach

In Pseudocode 3.1 the algorithm starts by initialising the rule set (RS) with the empty set and initialising the current training set with all the training examples. Then, for each class the algorithm performs a WHILE loop. Each iteration of this loop performs one run of the rule discovery algorithm, returning the best discovered rule predicting examples of the current class (C). This rule is added to the rule set, and the examples correctly covered by that rule are removed from the training set (TS). An example is said to be correctly covered by a rule if that example satisfies all the terms (attribute-value pairs) in the rule antecedent ("IF part") and it has the class predicted by the rule ("THEN part"). This WHILE loop is performed as long as the number of uncovered examples of the class C is greater than a user-defined threshold, the maximum number of uncovered examples per class (MaxUncovExampPerClass). After discovering rules for all classes, the algorithm returns RS, the discovered rule set.

Pseudocode 3.1 gives a basic method of creating an unordered rule set. The rule set is unordered in the sense that the rules can be applied to the test examples in any order and the model will still function as intended. The rule set generated by Pseudocode 3.1 produces an unordered rule set due to the way in which only correctly covered examples are removed from the training set at every while loop iteration. If all examples were removed (including the ones incorrectly covered) ordering the discovered rule set in a different way could have unforeseen consequences.

Many variations on the basic sequential covering principle are found in rule induction algorithms, including pruning the rule (removing terms to increase rule compactness and/or accuracy) after creation, pruning all rules after rule set creation (possibly removing rules entirely), discovering ordered rule lists, discovering rules not on a per class basis but on the basis of quality i.e., finding the best rule for any class rather than the best rule for a particular class. Note that, the variations mentioned so far do not include methods for actually discovering the rules themselves. Many algorithms have been created for this purpose, using a variety of different search methods and heuristic measures [118] [164].

Two of the most successful and commonly used rule induction algorithms are C4.5Rules [131] and RIPPER [39]. RIPPER (Repeated Incremental Pruning to Produced Error Reduction) incrementally produces rules starting with a rule for the least prevalent class and ending with a rule for the most, pruning them at each stage using a separate pruning set (the training set is divided into a "growing" and pruning set). The idea of growing and then pruning rules was based on IREP [66] but RIPPER includes 3 modifications, namely changing the heuristic function used to prune rules, changing the stopping criterion for adding rules to the rule set and optimising the entire rule set after its creation. These modifications make RIPPER extremely competitive in terms of rule set compactness, accuracy and running time.

C4.5Rules uses decision trees built by the widely used C4.5 decision tree induction algorithm (discussed in the next section). Note that this is essentially an advanced pruning algorithm, as rules and trees are interchangeable (as will be discussed in the next subsection). It firstly converts the decision tree to a set of rules, where every path from the root node to a leaf node is a single rule (note that the use of a decision tree negates the need for a sequential covering type approach for discovering rules). Then the algorithm removes terms that do not seem to affect the rule's performance in discriminating between the consequent class and the other classes. Then any rules that do not affect the performance of the rule set are removed.

Finally, the set of rules are ordered to minimise false positive errors (where examples are covered by the rule antecedent but do not have the class predicted by the rule). They are ordered to produce a decision list. This is where an example is considered covered by only the first rule that matches it in the sequential list of rules (decision list). This can be considered a very basic conflict resolution scheme – how to decide which class an example belongs to if two rules with different consequent classes match it.

In general decision trees are a comprehensible knowledge representation as discussed in the next sub-section. Hence, one may question the need to convert between human comprehensible decision trees and rule sets but as Quinlan [131] p.45 states:

> *"Large decision trees are difficult to understand because each node has a specific context established by the outcomes of tests at antecedent nodes."*

Indeed the rules from a rule set are in some sense modular (especially in unordered rule sets) and so can be interpreted in their own right. C4.5Rules retains almost all of the performance of the decision tree algorithm with (arguably) a simplification in the classification model representation.

3.2.1.2. Decision Tree Induction

Decision trees are another human comprehensible classification approach. They contain two features, decision nodes where a test is performed on an example with a corresponding subtree for each outcome of the test, and leaf nodes which indicate a class. Using these two features an example is inevitably classified as it moves through the tree in a top-down fashion, ending at a leaf node. An example decision tree is shown in Figure 3.1. The first test (at the first decision node) is made based on the Wage attribute: if the example's Wage value is less than or equal to 30k then it is sent to the second level left hand node, if it is greater than 30k is sent down the right hand edge and arrives at the leaf Credit=*Good*. This means that in the example decision tree any example having Wage > 30k will be assigned to the class Good Credit, whereas if an example has Wage ≤ 30K its class will be *Bad* or *Good* Credit, depending on whether or not the example's Mortgage value is *yes* or *no*, respectively.

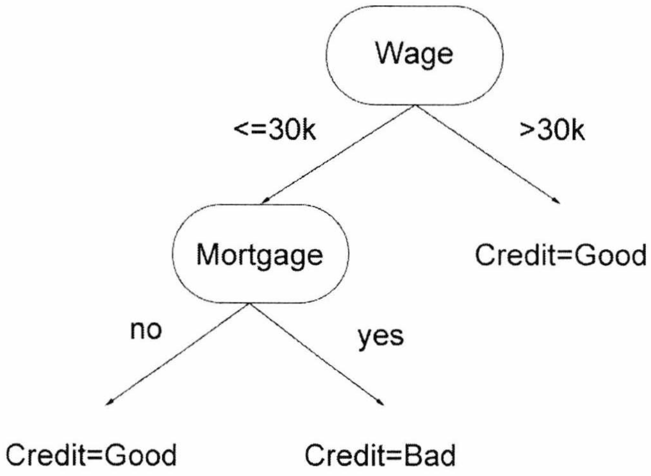


Figure 3.1: An example decision tree

Decision trees and rule sets are interchangeable (as stated previously). Converting a decision tree to a rule set simply requires generating one rule per possible path from the root node to each leaf.

For instance the decision tree shown in Figure 3.1 could be converted to the following (un-pruned and unordered) rule set:

```
          IF (Salary > 30k) THEN (Good Credit)
IF (Salary ≤ 30k) AND (Mortgage = No) THEN (Good Credit)
IF (Salary ≤ 30k) AND (Mortgage = Yes) THEN (Bad Credit)
```

A common method of generating decision trees which has been explored for many years is the divide and conquer principle. The divide and conquer approach involves calculating which test will produce the best divide of the training set, hopefully splitting examples belonging to one class from examples belonging to other class (or allowing this to happen in subsequent tests). Many algorithms use a greedy approach (e.g., C4.5 [131]) to calculate the best way to divide the training set, that is the interactions between the current test and subsequent tests are not considered fully. Another thing to be considered is the complexity of the tree, simpler (simple but not overly simple) trees are generally thought of as being more comprehensible. Simply using the divide and conquer principle to generate trees has a tendency to generate very complex and fragmented trees, which is not desirable. Therefore, a pruning procedure is normally used to reduce the size of the trees, often by substituting a subtree for a leaf. The subtree should be replaced if it produces more classification error than an appropriate leaf node. Although greedy algorithms such as C4.5 are relatively successful, much work has been conducted in producing "globally" optimised trees, using genetic algorithms and other global search heuristics [11] [61] [115].

3.2.2. Bayesian Classification

Bayesian classification algorithms are based on Bayesian probability theory. Naïve Bayes is one of the best known and widely used classification algorithms for real world problems, due in part to its simplicity and effectiveness. Its effectiveness is somewhat surprising due to its naivety; the basic assumption it makes about the lack of interaction between predicting attributes [135] given a class label. The Naïve Bayes classification algorithm uses Equation 3.1:

$$P(C_j \mid X) \propto P(C_j)\prod_{i=1}^{n} P(x_i \mid C_j)$$

Equation 3.1

Where X is a vector representing the set of attribute values (x_1,... x_n), n is the number of attributes, C_j is the jth class label, and the left hand side is proportional to the right hand side as the right hand side would have to be divided by a normalising constant in order to produce a probability value. Equation 3.1 states that the probability of observing a certain class C_j given a set of attribute values X can be calculated by multiplying two terms: the prior probability of the class, P(C_j), and the likelihood of X given C_j which is calculated as the observed probability (in the training set) of each attribute value (x_i), given that class, multiplied together. The way in which the algorithm is Naïve becomes clear; there is an assumption that the probability of each attribute value is completely independent from other attribute values given any particular class label. To turn Equation 3.1 into a functional classification algorithm, one must simply compute the probability for each class (C_j) given a set of attribute values (X), and take the maximum value, i.e., the most probable class.

Naïve Bayes algorithm makes the assumption about independence as computing the necessary sets of probabilities without that assumption would be problematic given a data set with a large number of attributes and values – where there would tend to be very few, or no examples for certain combinations of attribute values.

It is possible to still attempt to account for attribute interaction using more sophisticated Bayesian classification methods like Bayesian Networks, which rely on a Directed Acyclic Graphs (DAGs) where dependencies between attributes are represented by parent-child relationships. However, Bayesian Networks are not a “magic bullet”. The number of possible DAGs for a data set of reasonable size is massively large and in general finding the correct DAG for a given data set is an NP-Hard problem [40]. Bayesian Networks have another advantage over the naïve approach in that the induced DAG is a graphical representation and therefore is human comprehensible (as long as the DAG is not very large and complex). It can be used to gain knowledge about the interactions of attributes within a data set. Bayesian networks are beyond the scope of this thesis, but it is worth briefly mentioning that swarm intelligence methods can also be used to find Bayesian networks [43].

3.3. Ensembles of Classifiers

Ensemble classifiers normally try to combine the predictions from separate classifiers (classification models) in order to increase predictive accuracy. However, with this increase of accuracy there is usually a loss of comprehensibility. It may be very difficult for a human to fully understand the nuances of many interacting classification models produced from a single data set such as the ones used by ensembles.

There has been a large amount of research conducted in the field of ensembles and many different approaches have been developed. A particularly productive field of ensemble research has been conducted in the difficult area of handwriting recognition [71], where accuracy is of paramount importance and comprehensibility, in general, is of lesser importance. The most common ensemble techniques are [47]:

bagging (bootstrap aggregation), boosting, feature subspace re-sampling and stacking (stacked generalisation), as will be discussed in the remainder of this section.

3.3.1. Bagging

Ensemble classifiers can be categorised by the way in which their base classifiers are constructed and the way in which the predictions of these base classifiers are combined. In bagging [23] the training set is re-sampled several times in some way to generate separate classifiers, so that each classifier is trained with a different training set. The predictions of these classifiers are classically combined by a voting scheme which may or not be weighted. Bagging is useful when the learning algorithm used is unstable; where a small change in the training set causes a large change in the classifier created. Bagging relies on the premise that by sampling the data set multiple times and building classifiers for each one of these samples, the resulting combination of classifiers will be stable, with the instability of each component classifier averaged out across the multiple classifiers.

It is possible to re-sample the initial data set in many different ways. To make samples with the same (n) number of examples as the original training set, n examples are randomly chosen (from the training set) and added to the sample whilst allowing repeats. This method has the advantage of being able to create as many samples as are needed without reducing the number of examples in each sample. It is also possible to simply divide the training set into different partitions and use each as a sample. However, this limits the number of possible samples, as if there are too many, the number of examples in each sample will become too small to effectively train the classifier with.

There are many methods to try and increase the performance of the classifier prediction combining scheme used in bagging (although classical bagging simply uses unweighted voting). Unweighted voting is simply where the most frequently chosen class (chosen by the constituent classifiers) is assigned to the example, if there is a tie than an class must be chosen at random. In [46] the authors optimise the weights in the voting scheme to minimise variance (using SVMs as the base classifiers). Dietterich [47] provides a good review of alternative methods, including other combining schemes. Some of the most advanced and successful methods use genetic algorithms [72], [122], [133], [147]. It is also possible to use other advanced and iterative methods when sampling the data set, including samples based on clusters produced by clustering algorithms such as K-means [48].

Feature subspace re-sampling [48] can be considered a variation of bagging. It creates diverse classifiers by only giving (during the creation of one classifier within the ensemble) the classification algorithm a subset of the entire set of features available in the original dataset for each example. In other words, conventional bagging uses different training examples but the same features (attributes) in different runs of the classification algorithm, whilst feature subspace re-sampling uses different features but (possibly) the same training examples in different runs of the classification algorithm. The subsets of features given to different runs of a classification algorithm can be optimised based on several metrics [24] including the diversity of the classifiers created and the predictive accuracies.

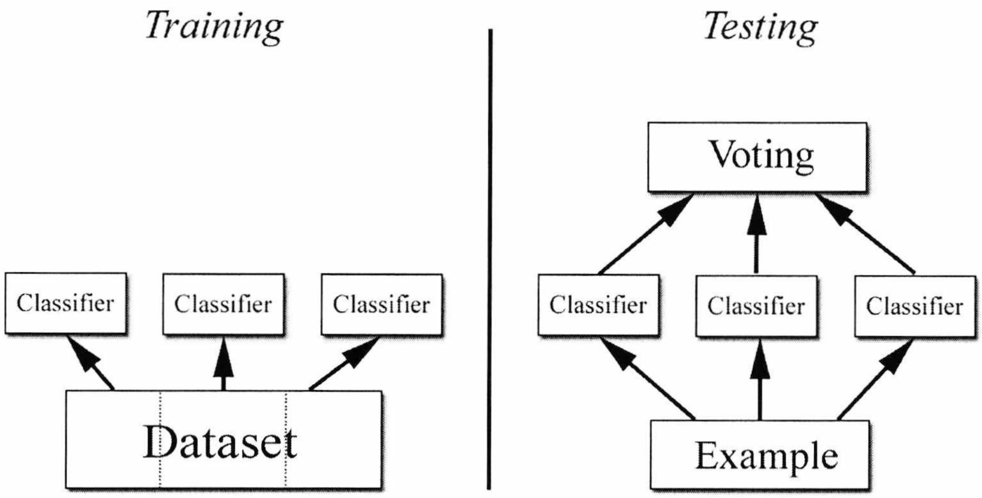


Figure 3.2: Training and Testing Phases in Bagging

The training phase of bagging is shown in Figure 3.2, where the data set is re-sampled in some way to produce separate classifiers. During the testing phase, when an example of unknown class is shown to the classifiers, a vote based on the predictions of the component classifiers is carried out to assign the final predicted class.

3.3.2. Boosting

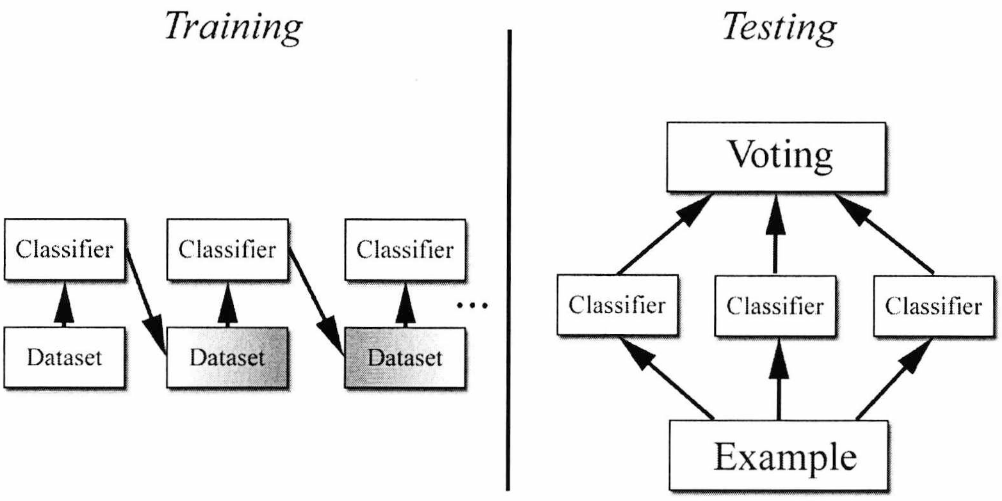


Figure 3.3: Training and testing phases in boosting

Whereas in bagging the classifiers can be built in parallel, in boosting the classifiers are built in series; each new classifier built is dependant on the previous classifier. Adaboost (Adaptively resample and combine) [64] is a commonly used boosting algorithm and works by assigning weights to examples. More precisely, examples that are considered hard to classify are assigned greater weights than examples that are easy to classify. It updates the example weights after each iteration, creating classifiers that concentrate on the hard examples. This creates a good set of specialised "experts" on particular areas in the data. This is possible because certain examples will not always be considered hard or always easy during the classifier-construction process, in fact the difficulty of classifying an example is dependant on the classifier. C4.5 is often used as the learning algorithm for Adaboost as it already can consider weighted examples when dealing with missing values. Some implementations of Adaboost use re-sampling rather than weighting to build the ensemble of classifiers. This is useful when the base algorithm does not easily support weighting. It has been suggested that the main strength of algorithms of the type of Adaboost lies not in the voting procedure, but in the "adaptive re-weighing of instances" [10].

The sequential nature of the training phase in boosting is shown in Figure 3.3. The changing gradient of grey shown on the "Dataset" boxes represent the changing weights associated with the examples in the training set – each example is a static point within the "Dataset" box and their shade (weight) changes after each of the three iterations shown in the diagram. The testing phase is usually much the same as with bagging, as shown in the right part of Figure 3.3.

3.3.3. Stacking

Unlike bagging and boosting, which classically combine the predictions of multiple applications of the same classification algorithm on different subsets of data, stacking [165] usually combines the predictions of multiple distinct/parametrically altered classification algorithms trained on the same data. These base classifiers are known as "level-0" classifiers. Stacking utilises another level of classification which combine the predictions from the "level-0" classifiers, this level is known as the "level-1" classifier.

The level-1 classifier makes decisions based on the predictions of the level-0 classifiers. In stacking a meta-data set is created with meta-attributes corresponding to the class predictions of the base, level-0 classifiers within the ensemble. This meta-data set is usually trained on a hold-out partition of the training set, which was not used in the construction of the level-0 classifiers. This process is used in order to reduce the risk of overfitting [164].

Using this meta-data set it is possible to build the level-1 classifier and predict which class an example with an unknown class belongs to by converting it to a meta-example. This meta-example will have as many meta-attributes as there are base classifiers, with values corresponding to the class they predict. This approach allows the level-1 classifier to learn from the correct predictions of the level-0 classifiers and also learn from the incorrect predictions of these classifiers.

The level-0 classifiers may be created in parallel as they can be independent, and schemes can be used [24] to try and insure the diversity of the base classifiers. It is possible to use stacking with classifiers generated by different algorithms, different parameter settings of the same algorithm, and also different samples or variations of data [29]. Furthermore, the level-1 classifier may also be a voting scheme or use Bayesian probability (in a similar way to the Naïve Bayes classifier). A comparison of these types of approaches can be found in [167], which also includes optimised weighted voting using a Particle Swarm Optimisation (PSO) algorithm (discussed in section 3.4).

The PSO algorithm adjusts the weights given to each voting classifier to increase accuracy. They conclude that given a large enough data set the PSO optimised weighted voting is the superior approach (when compared to using un-weighted/probabilistic types of approaches).

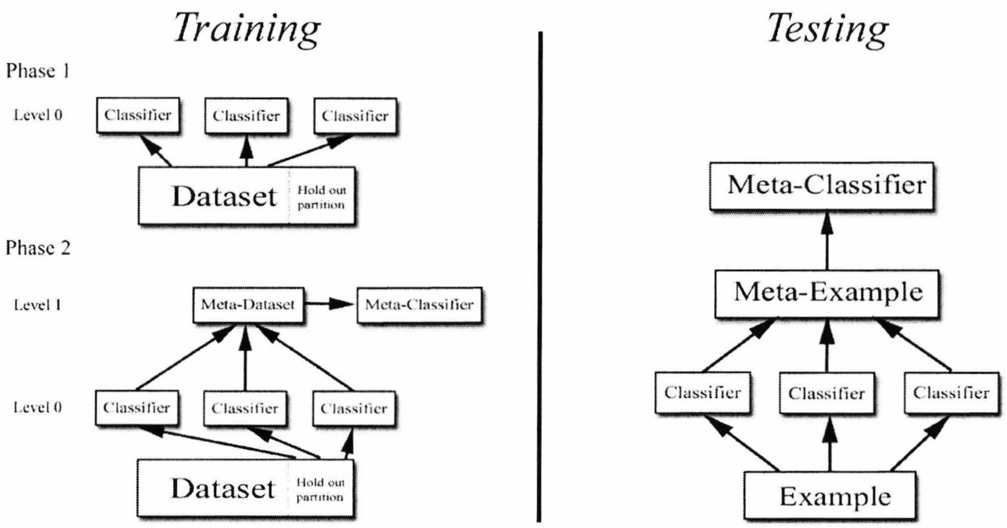


Figure 3.4: Training and testing stages in stacking

Figure 3.4 shows the training and testing stages of stacking. During phase 1 of the training stage the base classifiers are built as normal from the training set, with the caveat that a subset of the training set is used as a hold out set. In phase 2 the hold out training subset is classified by the level-0 base classifiers built during phase 1. These classifications are used to create the meta-data set and the level-1 combining classifier as discussed above. During the testing phase an example with unknown class is classified by the level-0 classifiers. These predictions are used to generate a meta-example which can then be classified by the level-1 meta-classifier to assign a final predicted class to the example.

Skalak [144] suggests that there should be three main considerations when designing an ensemble of classifiers: the accuracy of the component classifiers, the diversity of the component classifiers and the computational efficiency – i.e., the processing time required to build the entire ensemble of classifiers and also to use it to classify examples.

When using a single classifier the predictive accuracy is one of the most important factors. However with ensembles of classifiers each classifier does not need to be especially accurate for the ensemble to make an accurate prediction. Skalak [144] discusses an example of this phenomenon where a classifier that is 69% accurate is combined with classifiers that are 23% accurate and 25% accurate, this boosts overall accuracy to 88%.

The diversity of the component classifiers is very important in ensemble approaches; component classifiers must make different errors (from each other) in order for the overall ensemble to be more accurate than either of the base classifiers [9] [24]. There is often a trade-off between accuracy and diversity when building classifiers, as it is often easier to make more diverse (uncorrelated) classifiers when the restrictions on accuracy are lowered. Unless the time taken to build the classifiers (training) or to use them to predict the class of the examples (testing) is prohibitive, computational efficiency is probably the least important factor when designing an ensemble of classifiers.

3.4. Particle Swarm Optimisation

PSO is a meta-heuristic that maintains a population of particles – each of them a candidate solution to the target problem – that iteratively move around the (normally continuous) search space [21] [95]. Each particle is also part of a structured communication system as detailed below.

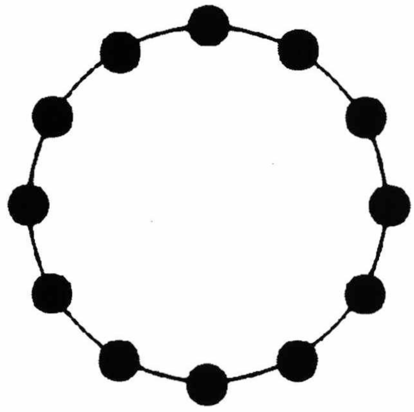


Figure 3.5: Ring (Local) Topology for PSO Particles

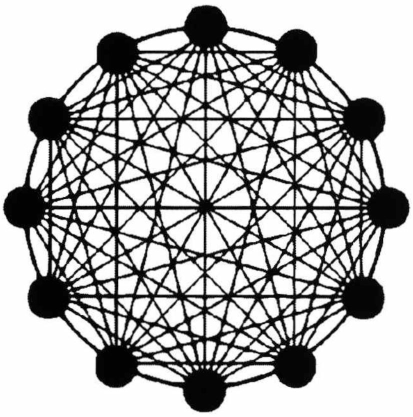


Figure 3.6: Global Topology for PSO Particles

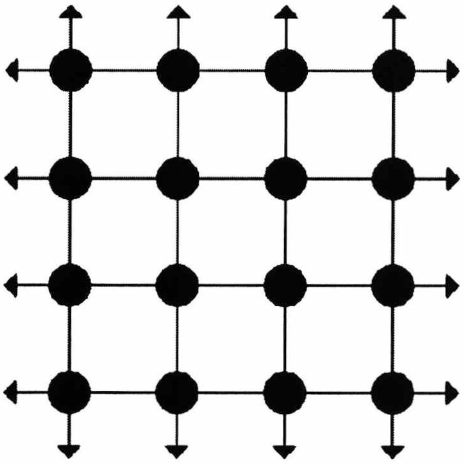


Figure 3.7: Von-Neumann Topology for PSO Particles

Figure 3.5, Figure 3.6 and Figure 3.7 show three commonly used topologies for particle communication in PSO. Figure 3.5 shows a local topology where a particle's only neighbours are to the left and right of it (also referred to as ring topology). Figure 3.6 shows the global topology where every particle is a neighbour of every other particle.

Figure 3.7 shows Von-Neumann topology where each particle has four neighbours in a 2-D grid layout.

The level of “connectedness” in the topology plays an important role in the PSO algorithm. Topologies that involve very well connected particles (such as the global topology) tend to converge to a solution much faster than those that are not so well connected (local topology). Although well connected topologies converge to a solution faster they are more likely to converge to a local maximum (according to the fitness fucntion) rather than the global maximum value [96]. The slower converging/less well connected topologies spend more time searching and so are more likely to perform better (in terms of the maximum or minimum value found). Depending on the problem difficulty and requirements it may not be necessary to select a very slow converging topology as this may simply waste computational time.

The position of a particle in the search space represents the contents of its candidate solution and so moving each particle corresponds to generating a new candidate solution. Each particle keeps track of the best position it has ever held, according to the fitness (evaluation) function. At each iteration, each particle finds its best neighbour (in a local or global neighbourhood). The particle then moves towards a combination of the best position any of its neighbours have ever held and its own best position, with a velocity calculated as shown in Equation 3.2. This process is repeated until a stopping criterion is met. To calculate the velocity and new position of a particle, Equation 3.2 and Equation 3.3 are often used (respectively) for each dimension of the current particle's position, although several variations have been proposed in the literature [124]:

$$v_{id}(t) = W\times(v_{id}(t-1))+\varphi_1\times Rand()\times(p_{id}-x_{id}(t-1))+\varphi_2\times Rand()\times(p_{gd}-x_{id}(t-1))$$

Equation 3.2: A particle's velocity at time *t*

$$x_{id}(t) = x_{id}(t\text{-}1) + v_{id}(t)$$

Equation 3.3: A particle's position at time *t*

Where x_{id} is the particle *i*'s position in dimension *d*, *t* is the iteration (time) index, v_{id} is particle *i*'s velocity in dimension *d*, *W* is an inertial constant to prevent the particle gaining too much speed. φ_1 and φ_2 are user-defined personal (to adjust the influence of the particle's own position on its next position) and social (to adjust the influence of the neighbour's position on the particle's next position) learning constants, respectively. p_{gd} is the best position of the particle's neighbours in dimension *d* and p_{id} is the best position particle *i* has ever held in dimension *d*. In addition to *W*, an optional maximum velocity is also used to prevent the particle from flying too far out of the search space. *Rand*() generates a random number in [0, 1].

Several studies have shown PSO to be a powerful optimisation algorithm, often outperforming more conventional population-based meta-heuristics such as evolutionary algorithms [75] [94] [107].

3.4.1. Binary and Discrete PSO Algorithms

The original PSO algorithm – designed mainly to optimise functions with continuous attributes – was extended to optimise functions with binary attributes [93]. The binary PSO algorithm does not use a particle's past position and velocity directly to calculate its new position for fitness evaluation. Rather, a particle's velocity in each dimension is interpreted as the probability that the particle will take the value 1 (rather than 0) in that dimension. More precisely, once the velocity of particle *i* in dimension *d* (denoted v_{id}) has been calculated, it is first converted into a number between 0 and 1 by Equation 3.4, where the constant *k* is used to determine how "deterministic" the search is:

$$s(v_{id}) = 1/(1 + \exp(-kv_{id}))$$

Equation 3.4: Converting from Velocity to Probability

Next, the value of the position d, denoted by x_{id}, is computed by Procedure 1, where *Rand*() is a random number between 0 and 1:

```
IF $Rand() < s(v_{id})$
  THEN $x_{id}(t) = 1$
  ELSE $x_{id}(t) = 0$
```

Pseudocode 3.2: Computing the Value of a Particle's Position in Binary PSO

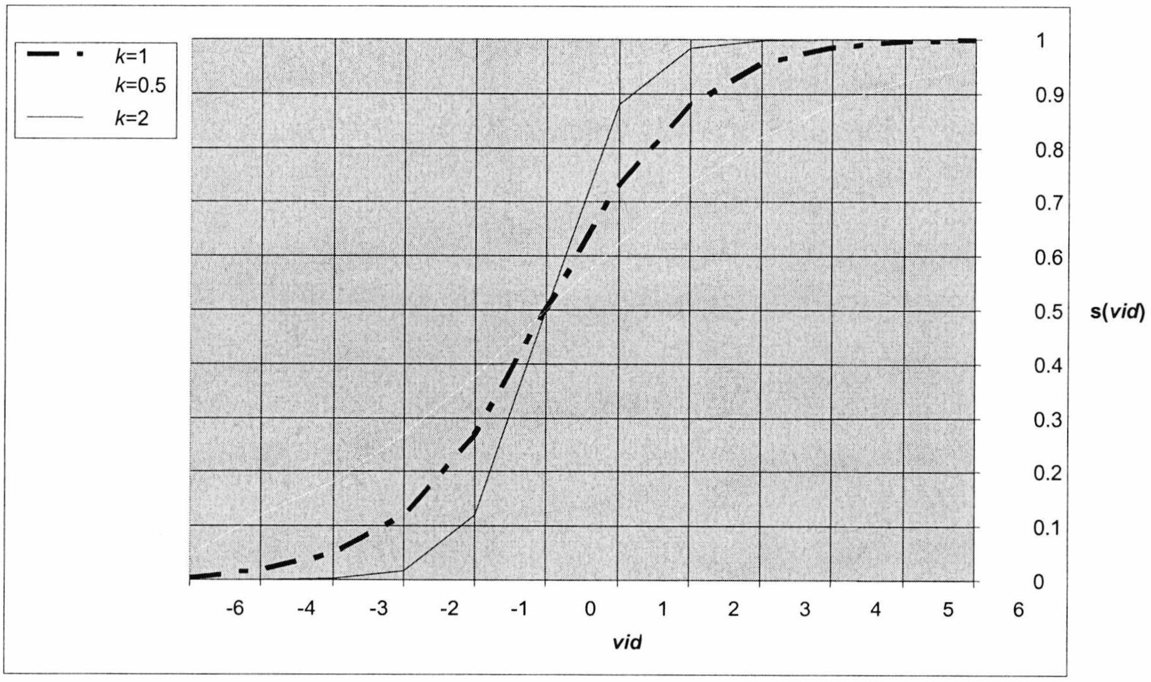


Figure 3.8: The effect of k on the binary PSO $s(v_{id})$ function

This modification to the original PSO algorithm is quite subtle in its workings. Each time a particle chooses a position 0 or 1, if it is found to be a position of lesser fitness when compared to its previous position and neighbourhood best, in the next iteration the particle will be accelerated in the opposite direction, increasing the probability of choosing the other value. As it gains velocity it becomes more unlikely that the particle will choose the position it is moving away from. If it chooses the position it is moving away from and finds it suboptimal again, it will accelerate away from this position, and so it eventually converges to a single solution. Maximum velocity can be used to limit the minimum probability of choosing 0 or 1, as the faster the particle gets in a dimension the less chance there is of choosing the position opposite to the one it is travelling towards.

Figure 3.8 shows the effect the constant *k* has on the search, the higher *k* is the more deterministic the search becomes. In other words, a higher value for *k* corresponds to a lower velocity being needed to cause a high probability of either choosing position 0 or 1. This variability is useful, for instance, when dealing with a noisy fitness function. In this scenario a lower value of *k* is often appropriate, as it prevents fast convergence to a false peak in the search space.

3.5. Ant Colony Optimisation

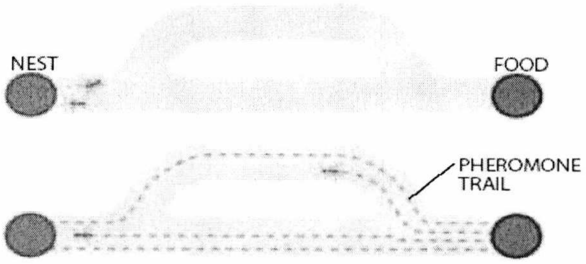


Figure 3.9: Pheromone trails in natural ant colonies

Ant Colony Optimisation (ACO) is a meta-heuristic inspired by the "intelligent" behaviour of real ant colonies. The pioneers in understanding how ant foraging works were Jean-Louis Deneubourg et al. [6]. They suggested that the reason ants are seen creating "highways" to and from their food is because of a chemical pheromone. Each ant lays down an amount of pheromone along its route and the other ants are attracted to the strongest scent. As a result, ants tend to converge to the shortest path. This is because a shorter path is faster to transverse, so if an equal amount of ants follow a long path and a short path, the ants that follow the short path will make more trips to the food and back to the colony. If the ants make more trips when following the shorter path, then they will deposit more pheromone over a given time period when compared to the longer path. This is a type of positive feedback and the ants following the longer path will be likely to change to follow the shorter path, where scent from the pheromone is stronger [49] [50]. Due to the evaporation of pheromone and the persistent small chance than an ant will take a random path, if a new shorter path is added the ant may eventually move to this new path. This feature makes ACO useful for dynamic problems such as network routing [50]. Figure 3.9 [20] shows a graphical representation of the way in which pheromone builds up on the shortest path.

ACO has been very successful in several types of combinatorial optimisation problems [50]. An obvious application is the travelling salesman problem, but in other combinatorial optimisation problems, such as sequential ordering problems, resource constraint project scheduling problems and the open-shop scheduling problem ACO methods are considered the "state of the art" [51].

3.6. Particle Swarm Optimisation and Ant Colony Optimisation for Classification

3.6.1. The Ant-Miner Algorithm

The Ant-Miner classification algorithm [119] takes the basic ideas from the Ant Colony paradigm and applies them to the field of data mining. Instead of foraging for food the ants in the Ant-Miner algorithm forage for classification rules and the path they take is described in terms of attribute-value pairs. Many variations of the original Ant-Miner algorithm have been proposed. Variations include fuzzy rule based approaches [67] [68] and a multi-label variant [28]. [63] provides a discussion about several other variations found in the literature.

The Ant-Miner classification algorithm described in Pseudocode 3.3 works in the following way. Firstly an ant starts off with an empty rule. It then iteratively adds one attribute-value pair (term) at a time to the rule, using a probabilistic procedure, where the selection of terms is based on the amount of virtual pheromone and on the value of a heuristic function that measures the information gain of each attribute-value pair.

```
TrainingSet = {all training examples};
DiscoveredRuleList = [ ];  /* rule list is initialized with an empty list */
WHILE (TrainingSetSize > MaxUncoveredExamples)
    $Ant_i$ = 1; /* ant index */
    NumConverged = 1; /* convergence test index */
    Initialize all trails with the same amount of pheromone;
    WHILE($Ant_i$ < MaxAnts AND NumConverged < MaxConverged)
        $Ant_i$ starts with an empty rule and incrementally constructs a classification rule $R_t$ by adding one term at a time to the current rule;
        Prune rule $R_t$;
        Update the pheromone of all trails by increasing pheromone in the trail followed by $Ant_i$ (proportional to the quality of $R_t$) and decreasing pheromone in the other trails (simulating pheromone evaporation);
        /* update convergence test */
        IF ($R_t$ is equal to $R_t$ - 1)
        THEN
            NumConverged = NumConverged + 1;
        ELSE
            NumConverged = 1;
        END IF
        $Ant_i$ = $Ant_i$ + 1;
    END WHILE
    Choose the best rule $R_{best}$ among all rules $R_t$ constructed by all the ants;
    Add rule $R_{best}$ to DiscoveredRuleList;
    TrainingSet = TrainingSet - {set of examples correctly covered by $R_{best}$};
END WHILE
```

Pseudocode 3.3: The Ant-Miner algorithm adapted from [119]

The larger the amount of pheromone and the larger the information gain for an attribute-value pair, the more likely that the attribute-value is chosen to be added to the current rule. The ant is considered to have completed its rule when adding any term to the rule would make the rule cover less than *Min_examples_per_rule* examples, a user-defined threshold. The reason for using this threshold is to avoid the generation of rules covering too few examples, which are unlikely to generalize well to examples in the test set, unseen during training.

Once a rule has been constructed, it is pruned by removing elements that are unnecessary or make the rule worse (in terms of rule quality). The pheromone matrix is then updated by increasing the amount of pheromone of the attribute values that occur in the rule the ant has just created. For each of the attribute values, pheromone is increased in proportion to the quality of the rule. This matrix can be considered a discrete landscape on which the ants travel, although it is not spatial in the sense that the values stored in the matrix do not map to coordinates. Once this has finished, the next ant creates a new and separate rule based on the pheromone trails of the previous ants (in addition to the information gain based heuristic). This means that eventually the ants will converge on a good classification rule as the pheromone for particular "good" attribute-value pairs will be much stronger than the pheromone for "bad" attribute-value pairs.

There is no part of the algorithm that explicitly makes the pheromone evaporate. However the probabilities stored in the matrix are normalized directly after updating the pheromone based on the last constructed rule. This normalisation has the side effect of decreasing the pheromone value of the attribute-value pairs that have not been updated after the candidate rule has been constructed.

$$Q = \mathrm{Sensitivity} \times \mathrm{Specificity}$$

Equation 3.5: The Rule Quality measure used for the Ant-Miner algorithm

Equation 3.5 gives the rule quality *Q* for a rule generated by an ant. The higher the rule quality ($0 \leq Q \leq 1$) the better the rule [74]: Sensitivity = TP / (TP + FN) and Specificity = TN / (TN + FP), where:

- True Positives (TP) are the number of examples that match the rule antecedent (attribute-values) and also match the rule consequent (class). These are desirable correct predictions.
- False Positives (FP) are the number of examples that match the rule antecedent but do not match the rule consequent. These are undesirable incorrect predictions.
- False Negatives (FN) are the number of examples that do not match the rule antecedent and do match the rule consequent. These are undesirable uncovered examples and are caused by an overly specific rule.
- True Negatives (TN) are the number of examples that do not match the rule antecedent and do not match the rule consequent. These are desirable and are caused by a rule's antecedent being specific to its consequent class.

When the rule generation loop finishes (i.e., the condition Ant_i < MaxAnts AND NumConverged < MaxConverged is not satisfied), the best rule is selected from all the generated rules (based on rule quality) and added to the discovered rule list. The rule needs to predict a class to be useful, and is assigned the class which leads to the best rule quality, given the just-generated rule antecedent.

After each rule is generated it is pruned. The rule pruning procedure is iterative and at each iteration it tries to remove, tentatively, each term of the current rule antecedent. The quality of the rule is measured after each term (*j*th value of the *i*th attribute) is tentatively removed from it. The term which when removed increases the rule quality the most is then removed permanently. Note that the class of the rule may be changed after each term *ij* is removed. This is continued until no term *ij* can be removed that would increase the quality of the rule.

To test the rules discovered from the training set, they are applied to the test set in the order they were created (this is the most basic rule conflict resolution scheme, as discussed in Section 3.2.1.1). That is, for each example in the test set, the algorithm scans the list of discovered rules until it finds a rule covering that example. To be considered a correct attempt at classifying a test example, the antecedent must match the attribute-value pairs stored in that example and the class of that example must match the one predicted by the rule. If there are examples that are not covered by any rule generated, then a default rule is used. This default rule simply classifies all the uncovered examples as the most frequent class in the training set.

3.6.2. Particle Swarm Optimization for Classification

Sousa proposed the first PSO-based rule induction algorithm [146], which uses PSO as the mechanism for searching for candidate rules. The algorithm functions in a similar way to Ant-Miner, in that it uses the sequential covering approach, the same rule quality measure to assess candidate rules and a similar rule pruning mechanism. However, rather than generating the rule of best quality first regardless of class (as in Ant-Miner), it generates rules for the majority class first. Also, ACO can cope with categorical attributes natively (as was shown with the Ant-Miner example in Section 3.6.1) whereas the Binary PSO used in [146] cannot. For this reason nominal (or categorical) attributes having more than two values must first be encoded in a binary form (with each value assigned a consecutive binary number index) and then each rule can be represented as a binary vector (to be optimised by PSO). If this was a straight forward conversion from attribute values to binary vector then each rule would always contain one value for each attribute. This would not be a good situation so instead an extra bit (for each attribute) is used to denote whether the attribute value is present or not in the decoded rule. The authors find that the proposed PSO approaches are at least competitive with (and often beat) J48 [164] (a Java implementation of C4.5) and Genetic Algorithm-based classification algorithms.

Note that this paper only addresses the problem of classifying data sets with categorical/discrete attributes.

Other works investigating PSO for classification include [58]. In this paper, rather than discovering rules, the PSO algorithm finds centroids in the attribute-value space, one per class. To classify an example a function is used to assess the distances between each centroid and the example. The centroid that is closest indicates that the example belongs to the corresponding class. The authors find that PSO is at least competitive with a wide range of commonly used techniques. Note that this paper only investigates data sets with continuous attribute values. A limitation of this approach is that in some problems a single class can consist of examples that are concentrated in two or more groups that are far apart in the search space, so that two or more centroids would be necessary for properly representing each class. The use of PSO for training artificial neural network classifiers is explored in detail in [12]. In this case the PSO algorithm is used to optimise the weights associated with the artificial neural network.

Another area of active research is in using PSO for discovering fuzzy rules [31] [108], however this topic is out of the scope of this thesis.

3.7. Hierarchical Classification

This thesis focuses on hierarchical classification problems [62] where the classes to be predicted are organized in the form of a tree, hereafter referred to as a class tree. Figure 3.10 shows an example 4 level hierarchical classification problem. The first class level has class nodes 1 and 2, class node 1 has two child nodes 1.1 and 1.2 at the second class level. All examples belonging to class nodes 1.1 and 1.2 also belong to class node 1, however there may be examples that only belong to class 1 and not classes 1.1 or 1.2 $((1.1 \cup 1.2) \subseteq 1)$. Notice that in Figure 3.10 each class node has only one parent class, with class nodes at level 1 having an *imaginary* parent root node (not shown for the sake of simplicity) which completes the tree structure and which does not correspond to any class.

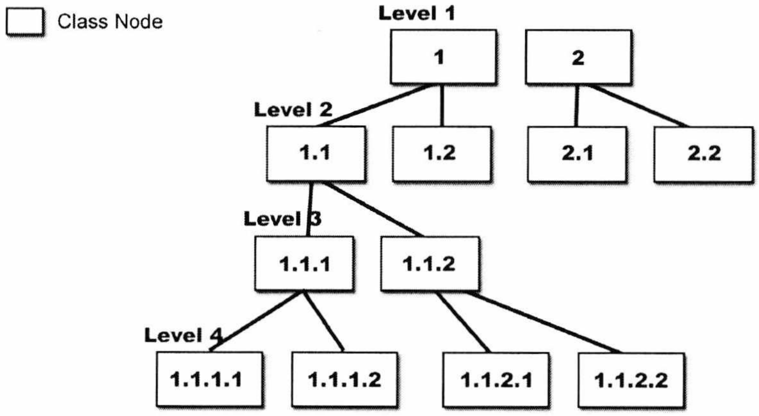


Figure 3.10: An example hierarchical classification class structure

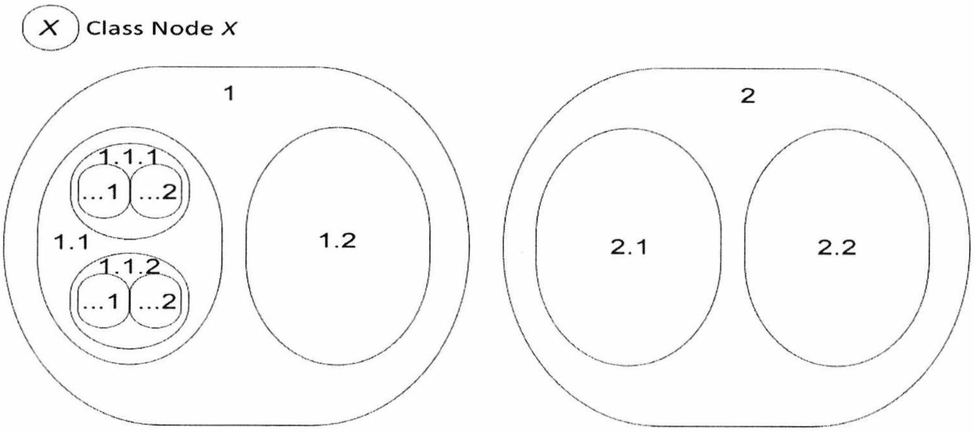


Figure 3.11: An example class-tree based hierarchical classification problem shown in the form of a Venn diagram

In general, the class structure for hierarchical classification problems can be split into two main categories:

- A tree structure, where each class node only has one parent class node (which is the focus of this thesis and shown in Figure 3.10).
- A Directed Acyclic Graph (DAG), where each class node may have one or more parent class nodes.

A commonly used DAG classification scheme in bioinformatics is the Gene Ontology (GO) [69]. GO comprehensively describes the relationships between "gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner" [69]. DAG based classification is not addressed in this thesis but good discussions can be found in [112] (which focuses on text mining) and [158] (which addresses GO classification).

Beyond the two types of class structures there is the possibility that the problem is multi-label – where each example in the data set may belong to one or more class nodes at each class level. These types of problem are known as Hierarchical Multi-Label (HML) problems. HML is not the focus of this thesis but good discussions of the problem can be found in [19] and [33].

Hierarchical classification with a tree based class structure can be considered a special case of the more general multi-label classification problem (where an example can be assigned more than one class label). The example shown in Figure 3.10 is shown in the form a Venn diagram in Figure 3.11. Figure 3.11 illustrates the way in which multi-label classification problems and hierarchical classification problems relate to each other. For instance, class-tree based hierarchical classification examples can be considered multi-label examples in the sense that an example belonging to class 1.1.1.1 can be said to belong to classes 1,1.1,1.1.1 and 1.1.1.1. However, not all multi-label classification problems are hierarchical. Importantly hierarchical classification problems tend to have different properties to "flat" (non-hierarchical) multi-label problems. Firstly, there tend to be classes with very small numbers of examples towards the bottom of the class tree, as at each progressive level the examples are split between sibling class nodes. Secondly, there tend to be many more classes, especially when dealing with bioinformatics data (the focus of this thesis).

3.7.1. Flattening Hierarchical Classes

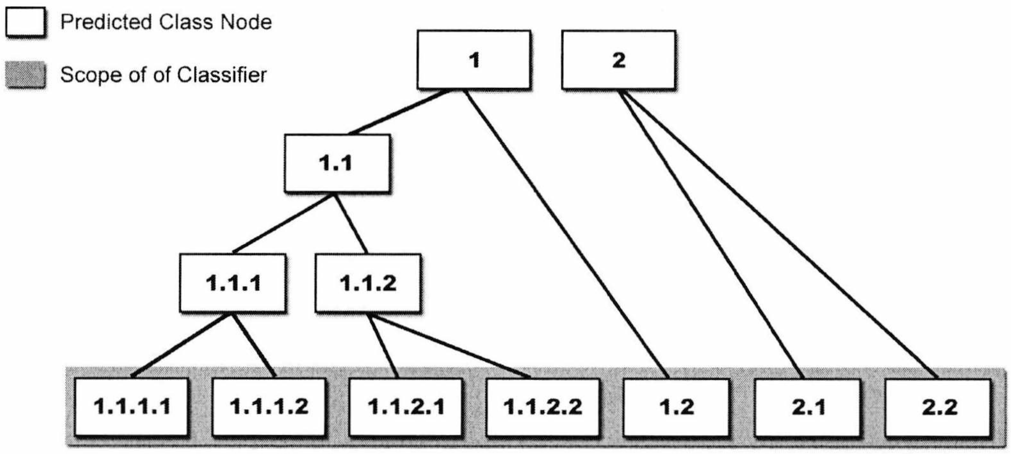


Figure 3.12: Reducing a hierarchical classification problem into a flat classification problem

Figure 3.12 shows the simplest (and most naïve) way of dealing with hierarchical classification problems, which is to ignore the class hierarchy completely and so only predict classes at the bottommost class level. When the bottommost classes are predicted the classes at higher levels are indirectly predicted. For instance, in a four level classification problem, if the algorithm predicts for a given example the class 1.1.1.2, it is also predicting class 1 at the first level and class 1.1 at the second level and class 1.1.1 at the third level. This approach avoids the complexity associated with a truly hierarchical classification algorithm (any generic classification algorithm can process a hierarchical data set by disregarding the hierarchical structure) at the expense of not discovering more generalised knowledge expressed by higher class levels. Flattening the classification problem causes only lowest level, specific knowledge to be discovered. Furthermore such specific classification models tend to be less accurate than more generic models (predicting classes at higher levels of the hierarchy) [42]. This is due to the specific model usually covering a smaller number of examples per class than a high-level, generic model. Also, flattening makes the task of building an accurate model more difficult for the classification algorithm; there are more classes to discriminate between and so more chance of errors occurring.

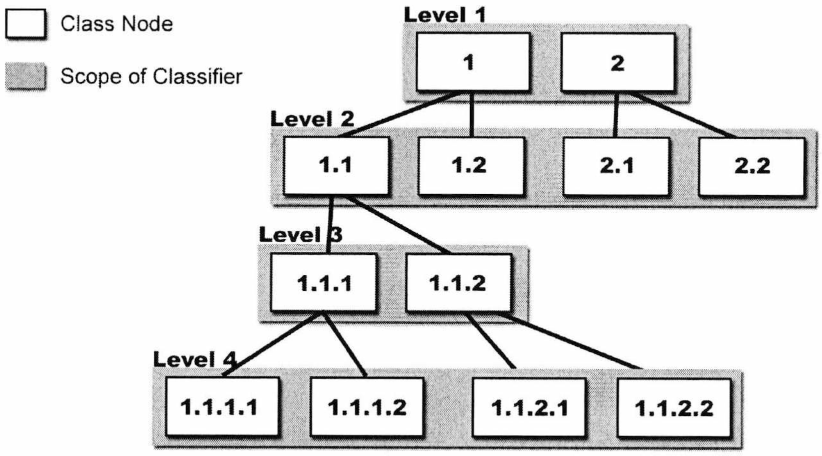


Figure 3.13: Reducing a hierarchical classification problem into a set of flat classification problems

The second simplest approach, as shown in Figure 3.13, is to build an independent classifier for each class level [62]. The user can then select the granularity of the prediction. This approach has the advantage of being simple to implement as only basic data set pre-processing is needed to convert the hierarchical classification problem into this form, e.g., creating four separate datasets where an example of class 1.1.1.1 is found as an example of class 1 in the data set corresponding to class level 1 predictions, class 1.1 in the data set corresponding to class level 2 predictions, 1.1.1 in the data set corresponding to class level 3 predictions and so on. This approach has the disadvantage of it being possible to get conflicting predictions about the class label of an example. The class level 1 classifier might predict class 2 for a given example, but then the class level 2 classifier might predict class 1.1 for the same example. This may not be a major issue as, in general, the higher the class level prediction the more likely it is to be accurate. This is often due to the larger number of examples per class present at higher class levels (so there is more statistical evidence for the classifier), and the smaller number of classes (reducing the complexity of the problem for the classifier). Due to these reasons it would be wise to accept the higher classification as the correct one and simply stop the classification process when a conflicting classification occurs

This raises an important issue, whether the hierarchical classification process is of the type "mandatory leaf-node prediction" or "optional leaf-node prediction" [62]. The scenario previously described where the classification process is ceased at a class node that is not a leaf would be an "optional leaf-node prediction". The first flattening process described in this section is a "mandatory leaf-node prediction" scheme, as by design it always predicts a leaf node. Obviously having a correct leaf node prediction is advantageous as the user is returned the best possible information (most specific class) about an example. However, in this case the hierarchical classification algorithm must be intelligent enough to know when to give up, such as in cases where there are no longer enough examples to effectively train the classifier at the deepest class levels. This issue is somewhat subjective and dependant on the application, sometimes it may be more beneficial to attempt to classify an example all the way to a leaf node rather than give up whilst there is still a slim chance of correct classification.

Reducing the hierarchical classification problem into one or a set of flat classification problems is a technique that is often used in the bioinformatics literature. To give a small cross section of the literature, this technique has been used for

GPCRs [15] [73] [92] [116], enzymes [32] [88] [162] and proteins classified according to the Munich Information Centre for Protein Sequences (MIPS) [32]. It should be emphasized that in this technique the separate classifiers are completely independent and the hierarchy is still essentially ignored.

3.7.2. Top-Down Divide-and-Conquer Approach

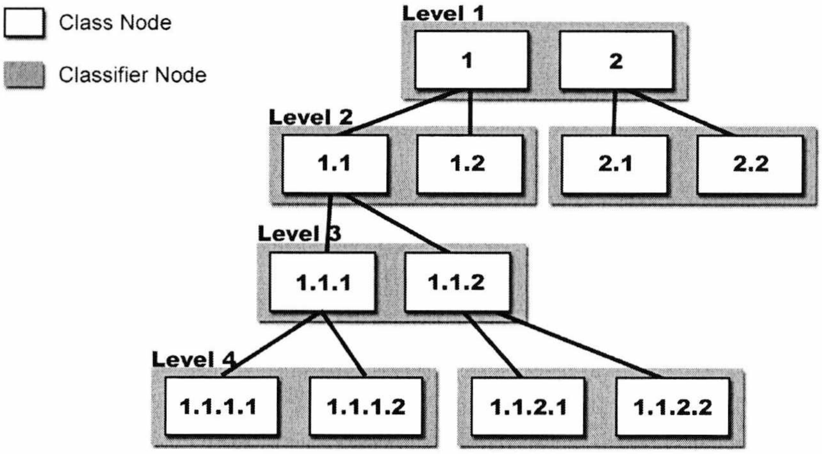


Figure 3.14: The Top-Down Divide and Conquer method to deal with a hierarchical classification problem

The first category of approaches that can be described as truly hierarchical is the Top-Down Divide-and-Conquer (TDDC) category [52] [149]. An example TDDC tree is shown in Figure 3.14, this approach being the focus of this thesis. This top-down approach has the important advantage of using information associated with higher-level classes in order to guide the prediction of lower-level classes. For instance, if class 1.X.X.X (where X denotes any digit) is predicted at the first level and the tree node for that class has only the child nodes 1.1.X.X and 1.2.X.X, only these two class nodes should be considered and not the children belonging to node 2.X.X.X. In general, any classification model constructed in the top-down divide and conquer tree only has to discriminate between sibling classes. In TDDC the models themselves form a tree, known as a classifier tree. This tree is populated with classifier nodes where a classifier discriminates between sibling classes.

As each classifier node in the TDDC tree only needs to discriminate between sibling classes, the training set used to generate each classifier node is different. For instance, to generate the classifier that discriminates between class nodes 1.1.X.X and 1.2.X.X, only examples belonging to class nodes 1.1.X.X and 1.2.X.X need to be present in the training set. To generate the classifier node that discriminates between class nodes 1.1.1.X and 1.1.2.X only examples belonging to class nodes 1.1.1.X and 1.1.2.X are present in the training set, and so on.

When it comes to the classification of an example, each classifier chooses which child classifier to send the example to, or if the classifier is at the leaf level what final class the example should be assigned to. For instance, the root classifier, which discriminates between classes 1.X.X.X and 2.X.X.X, will decide if an example should be sent to the classifier discriminating between the child classes of class 1.X.X.X or 2.X.X.X. If the example is first classified as 1.X.X.X then it will be sent to the classifier discriminating between classes 1.1.X.X and 1.2.X.X. This classifier will decide if the example should be sent to the classifier discriminating between the child classes of class 1.1.X.X or 1.2.X.X. If the example is then classified as 1.1.X.X then the next classifier (discriminating between 1.1.1.X and 1.1.2.X) will decide if it should be sent to the classifiers discriminating between the child classes of 1.1.1.X or 1.1.2.X and so on.

The TDDC approach has the advantage of being able to use any type of classifier. This is because the only way the classifiers interact is through their predictions. It has the disadvantage of taking a comparatively large amount of computational power to create all the necessary models – one per each set of sibling classes. Also due to the way in which classifications take place in the standard TDDC tree, i.e., a classifier only ever discriminates between sibling classes, once a misclassification has taken place it can never be corrected at a deeper-level classifier. This characteristic was dubbed blocking by Sun et al. [152]. In this paper Sun et al. propose three methods to try and reduce the impact of blocking in a scenario where more than one class label can be assigned to an example at each class level (i.e., a multi-label problem at each level). In their approach, each class node is associated with a binary classifier which predicts whether or not the current example is assigned to that class. If so, the example is further passed to all children of that class node, otherwise the example is blocked at that node. The three methods to cope with blocking proposed by Sun et al. are as follows:

- Firstly, the threshold reduction method, where examples are more easily passed to child classifiers (for further classification) according to some threshold, calculated on a per class level basis.
- Secondly, the restricted voting method, where separate classifiers are constructed to bypass the child classifier nodes and go directly to the child's child classifier node.
- Thirdly, the extended multiplicative method (originally proposed in [52] and extended by Sun), where a classification score is calculated for each leaf and non-leaf node. If the score is greater than a certain threshold then the example is assigned to that leaf or non-leaf node.

Sun also proposes another variation on TDDC [149], where each classifier node is associated with another classifier that decides if an example should be passed onto the child classifiers; this is an optional leaf-node prediction approach. Another more recent proposal to improve TDDC accuracy was Secker's Classifier Selection [140] to be discussed in the next sub-section.

3.7.2.1. Classifier Selection for the Top-Down Divide and Conquer Approach

In the conventional top-down approach for hierarchical classification, in general, the same classification algorithm is used for each classifier node. Intuitively, this is a suboptimal approach because each classifier node is associated with a different classification problem – more precisely, a different training set, associated with a different set of classes to be predicted. This suggests that the predictive accuracy of the classifier tree can be improved by selecting, at each classifier node, the classification algorithm with best performance in the classification problem associated with that node, out of a predefined list of candidate classification algorithms. Indeed it was found in [140] that by varying the classification algorithm at each classifier node, or divide, in the Top-Down Divide and Conquer (TDDC) approach classification accuracy could be somewhat improved.

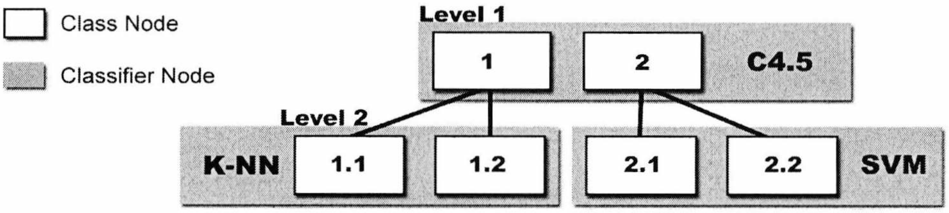


Figure 3.15: A TDDC tree using classification algorithm selection

In Secker's work the training set at each classifier node is divided into two non overlapping sub sets, a building set – used to train the classification algorithms – and a separate validation set – which is used to assess the predictive accuracy of the models constructed by the classification algorithms. At every classifier node in the TDDC tree, multiple classifiers are built using the building set, each using a different classification algorithm. The classification accuracy of each of these classifiers is measured using the validation set at each classifier node, and then the best classifier (according to classification accuracy in the validation set) is chosen. This process is repeated at each classifier node to select a set of classifiers to populate the TDDC classification tree, which is then used to classify the test instances (unseen during training). A simple example of a classification tree constructed by this method, showing a different classifier chosen at each node, is shown in Figure 3.15.

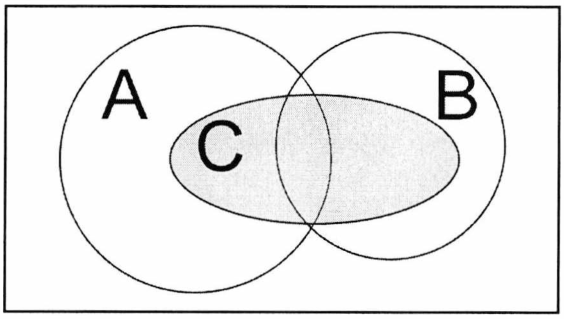


Figure 3.16: Classifier interaction scenario where |B∩C| > |A∩C|

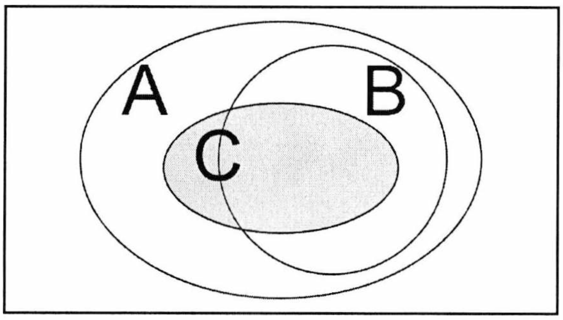


Figure 3.17: Classifier interaction scenario where |B∩C| < |A∩C|

In this way Secker's work uses a greedy selective approach to try and maximise classification accuracy. It is described as greedy because, when it selects a classifier at each classifier node, it maximises accuracy only in the current classifier node, using local data. Therefore, the greedy selective approach ignores the effect of this local selection of a classifier on the entire classifier tree. In other words, this procedure is "short sighted", and so it does not consider the interaction between classifiers at different classifier nodes.

Figure 3.16 and Figure 3.17 show two possible scenarios demonstrating interactions between classifiers at different classifier nodes during classifier evaluation. A and B are the two possible parent classifiers trying to discriminate between classes 1 and 2. C is the child classifier that attempts to discriminate between classes 1.1 and 1.2 – as shown in Figure 3.18. Figure 3.16 and Figure 3.17 show the sets of correctly classified examples for each classifier in the TDDC tree. Notice that $C \subseteq A \cup B$ for the three classifiers A, B and C. This is due to the fact that in the standard TDDC tree once a misclassification has been made, by classifier A or B at the first classifier node, it cannot be rectified by C at the child classifier node.

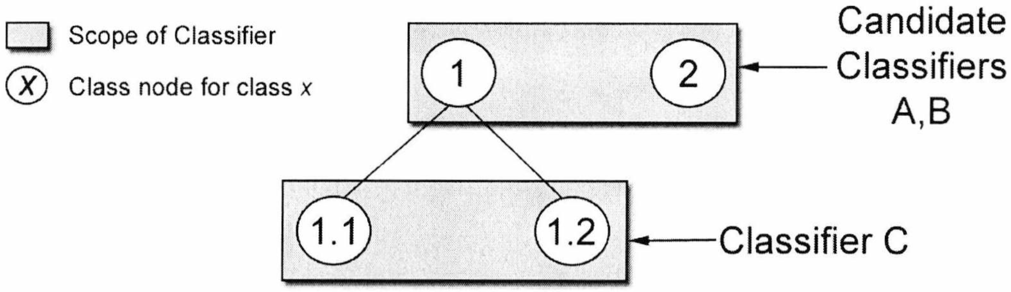


Figure 3.18: A class tree used to illustrate the discussion on classifier interaction

As mentioned earlier, the greedy approach chooses the best classifier at each node according to the classification accuracy, in the validation set, at that node. In the scenarios shown in both Figure 3.16 and Figure 3.17 classifier A would be chosen to discriminate between classes 1 and 2, as it is more accurate when compared to classifier B, i.e., its circle has a bigger area, denoting a greater number of correctly classified examples. Let us now discuss how appropriate the choice of classifier A (made by the greedy approach) is in each of the different scenarios shown in Figure 3.16 and Figure 3.17, taking into account the interactions between classifiers A and C, and between B and C, in the context of the class tree shown in Figure 3.18.

Recall that in the TDDC approach an example is correctly assigned to class 1.1 or 1.2 if and only if the two following events occur: the example is correctly classified by the root classifier (A or B); and the example is correctly classified by classifier C. Therefore, the individual accuracy of each classifier is not necessarily the most important factor when selecting a candidate classifier; rather it is the number of examples correctly classified by *both* the parent and child classifiers (the intersection between their sets of correctly classified examples). In the case of Figure 3.18, in order to maximise the classification accuracy at the leaf class nodes 1.1 and 1.2, if $|A \cap C| > |B \cap C|$ then classifier A should be chosen; if it is not, B should be chosen. For this reason, the greedy approach produces a good selection in the case of Figure 3.17, where $|A \cap C| > |B \cap C|$. However, the greedy approach would not produce an optimal selection in the case of Figure 3.16. This is due to the fact that although A has a greater area (higher accuracy) in Figure 3.16, $|B \cap C| > |A \cap C|$.

As the number of candidate classifier selections for any given classifier tree is k^n, where n is the number of classifier nodes and k is the number of candidate classifiers at each classifier node, it is impossible to exhaustively check all candidate solution for any data set with a reasonably large number of classes. Given this fact it is clear that a heuristic approach could be beneficial. Such a heuristic method, based on a robust global search algorithm, is proposed in Chapter 5.

3.7.3. "Big Bang" Approach

The second category of approaches that can be described as truly hierarchical is the "Big Bang" category [149]. A highly abstracted big bang approach is shown in Figure 3.19. Methods following the big bang approach usually take the form of modified classification algorithms, building classifiers that can predict any class node in one "step" and usually offering optional leaf-node prediction. The obvious disadvantage of such an approach is that the classifier produced is rather complex, as it must be able to assign one (or more) of possibly hundreds of class labels to an example in one step. While it is true that the single big bang classifier is more complex than each component classifier from the Top-Down Divide-and-Conquer (TDDC) approach, it could be argued that taken as a whole the many models from the TDDC approach are more complex. However, it is also arguable whether a complex hierarchical classification model would ever be fully comprehensible, and at least with the TDDC approach the problem is broken into smaller fragments that can easily be examined. Each fragment will almost certainly be simpler taken on its own, within its context (position in the TDDC tree), than the complete big bang model. Modularity is the key to comprehending any large system, a fact that is well known within software engineering circles.

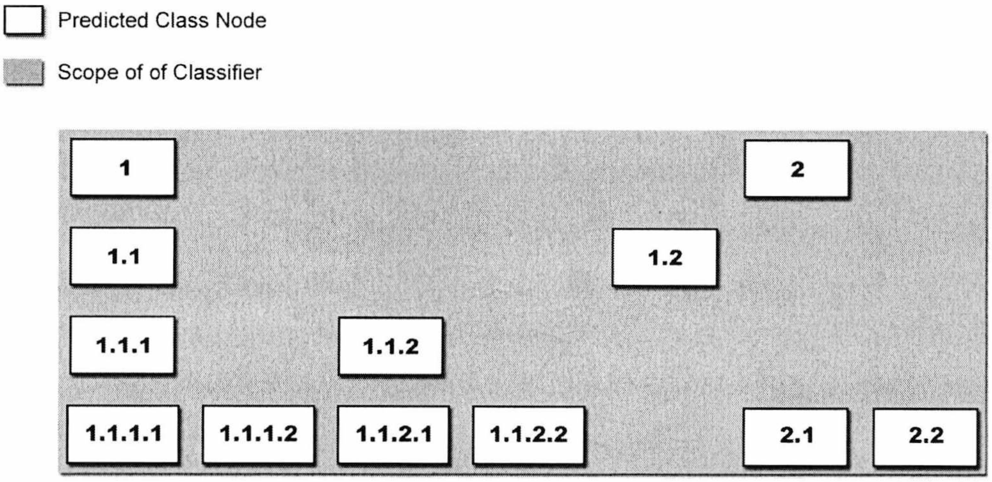


Figure 3.19: The Big Bang approach to deal with a hierarchical classification problem, notice that the classifier can predict any node within the class hierarchy in just one step

Initial big bang approaches were motivated by the need to classify data involving hierarchical text categories. Two examples being a modified RIPPER algorithm [138] and an algorithm that produces hierarchical association rules [161]. Another algorithm focusing on hierarchical multi-label classification (HMC) is discussed in [137]. It uses a kernel based [45] method to classify examples, and the classification hierarchy that is formed is represented by a Hidden Markov Tree.

Hierarchical C4.5 (HC4.5) [33] is another example of a big bang approach, but was motivated by the need to classify hierarchical multi-label gene data. A similar algorithm which is probably one of the most successful (producing human comprehensible models) big bang approaches is a recent variant of the Clus algorithm – Clus-HMC [18] [148]. Clus-HMC is another hierarchical decision tree algorithm which was shown to significantly outperform HC4.5 and a basic TDDC approach [18].

HC4.5 [33] is an extension of the original C4.5 decision tree induction algorithm that deals with hierarchical multi-label problems. The modifications needed for the original C4.5 algorithm are: a way of recording the hierarchical multi-label relationships between classes, a way of testing membership of a given class, a new way of finding which class (or classes) each node should be labelled with and a modified version of the entropy calculation (used to decide which decision node to place at a point in the decision tree). The relationships between classes are recorded in a separate file (to the data set), and stored internally as a Boolean array. Each element in the array corresponds to a class label, so testing if an example belongs to a particular class, or set of classes, becomes trivial.

Clus and HC4.5 differ in the way each leaf node is labelled with a set of classes. However, the basic principle revolves around finding whether a node should become another test (to allow finer grained class predictions) or whether it should become a leaf node. In Clus, if it is decided that the node should become a leaf the decision about what set of classes should be assigned is determined by the distribution of class labels in the set of examples at that node. A distance measure is used to find an array of class labels that represents the example class labels well. In this way the authors consider Clus a predictive clustering tree; the leaf predictions are in some sense clusters.

Another recent approach that has been developed for hierarchical multi-label gene function prediction is described in [8]. This approach uses a support vector machine as a classifier for each class node, and combines their predictions in a hierarchy-aware manner using Bayesian networks.

3.7.4. Measuring Hierarchical Classification Performance

As stated in section 3.1 there are many measures of performance for standard flat classification. Indeed the same can be said for hierarchical classification with multiple different proposals [19] [149]. Freitas and Carvalho [62] divide the approaches into four categories: uniform misclassification costs, distance-based misclassification costs, semantics-based misclassification costs and the hierarchical misclassification cost matrix.

With uniform misclassification costs all misclassifications are given the same weight. The most basic uniform approach would mean that a mistake anywhere in the classification process would cause the system to consider the prediction totally incorrect. For instance, an incorrect prediction would be if an example's correct class was 1.1.1.1 and its predicted class was 1.1.1.2. Obviously this is very unreasonable as three correct predictions have been made (at the first, second and third class levels) even if the last one is incorrect. The system would also consider the following a misclassification, if the example's true class was 1.1.1.1 and the predicted class was 1.1 (or vice versa).

Another uniform measure can be attained by using standard classification accuracy. At each class level the number of correctly classified examples can be divided by the total number of examples. This gives a per-level breakdown of the predictive accuracy of the algorithm, which can be quite enlightening. However, "the waters are muddied" by a specific case, when for instance, an example's correct class is 1.1 and its predicted class is 1.1.1.1. Using the predictive accuracy measure this prediction would be correct at the first level, correct at the second level and then not counted at the third and fourth. This is due to the fact that the accuracy measure only usually counts the number of examples per class level in the denominator and not the total number of predictions made. Although this example is technically a misclassification it is not clear how serious of an error it would be to the user, and so how greatly it should be penalised. For the purposes of this thesis hierarchical classification accuracy will remain as number of correctly classified examples divided by the total number of examples.

With distance-based misclassification costs, the shortest path between the correct class node and the predicted class node is taken. For instance, if the correct class is 1.1.1.1 and the predicted class is 1.1.1.2 then the distance, or misclassification cost, is 2 because there are 2 edges between those class nodes in the class tree. This raises an interesting point, which is that an example with predicted class 1.2 and correct class 1.3 will also have the same misclassification cost 2. It seems rational to believe that the second misclassification is more severe than the first. In the first case there are three correct classifications (first, second and third class levels) whereas in the second case there is only one correct classification. Not only this but a misclassification at the second level is more severe than a misclassification at the fourth, since almost no knowledge is gained about the example's true class when a misclassification is made at the second class level. These problems can be somewhat overcome by using weighted distances [19] and possibly normalisation (to ensure each example is given the same weight independent of the number of classes it belongs to, in the case of multi-label classification).

Weighted distances involve the edges between higher level class nodes being assigned a greater weight than those at the lower class levels. Also the misclassification cost can be normalised for each example: the worst possible misclassification cost can be calculated for each example, and then the actual misclassification cost can be divided by this number. This performance measurement scheme is used in this thesis and a further discussion of the approach can be found in section 6.7.

Semantics-based misclassification costs are discussed for text mining in [149] [150]. These costs are calculated not by any feature of the class structure, but by the similarity between the predicted and correct classes. The classes can be represented by two vectors and then a distance measure can be applied to calculate a misclassification cost. This approach has the disadvantage of relying on a distance measure which is subject to bias. Also, the user may not subjectively agree with the objective measure. This is a problem in a bioinformatics context as the class structure is usually the subject of a considerable amount of care and effort by biologists, so completely ignoring it seems inappropriate in many applications.

Hierarchical misclassification cost matrices involve matrices specifying which cost each particular misclassification should have. In each entry in the matrix – corresponding to a particular predicted class node and correct class node – a misclassification cost value can be set by the user. So, for instance, the user could specify what value to assign the misclassification of an example having correct class 1.2 and predicted class 1.2.1.1. This has the advantage of being extremely flexible and has already had its usefulness for flat classification [164]. However, due to the measure's complexity, manually setting each cost (a task to be done by the user) could be very time consuming in problems with a large number of classes; also an oversight by the user could cause the performance measure to be very misleading. An automatic system that guided the user's input could greatly speed up the process, along with ensuring that there are no anomalous values.

3.8. Summary

This chapter has introduced the elements relevant to this thesis from the very large fields of data mining and optimisation. It has provided a discussion of the different techniques commonly used for class-tree based hierarchical classification along with the methods that can be used for their evaluation. An overview of ensemble classification techniques has also been presented. Also, the two "swarm intelligence" optimisation algorithms (Particle Swarm Optimisation and Ant Colony Optimisation) that are used in the approaches described in this thesis have been described in terms of their application to data mining. These facets of data mining and optimisation are brought together to create the approaches proposed in the following chapters.

Chapter 4. A Hybrid Particle Swarm Optimisation/Ant Colony Optimisation Algorithm for Rule Induction

4.1. Introduction

The focus of this chapter is on supervised learning, more specifically, the classification task of data mining. As discussed in Section 3.2.1 a classification rule consists of an antecedent (a set of attribute-values) and a consequent (class). For the purposes of this chapter, a term is defined by a triple <*attribute, operator, value*>, where *value* is a value belonging to the domain of *attribute*. The *operator* used in this chapter is "=" in the case of categorical/nominal attributes, or "≤" and ">" in the case of continuous attributes. Knowledge representation in the form of rules has the advantage of being intuitively comprehensible to the user. This is important, because the general goal of data mining is to discover knowledge that is not only accurate, but also comprehensible to the user [59] [164].

In this chapter we propose a hybrid Particle Swarm Optimisation/Ant Colony Optimisation (PSO/ACO) algorithm for the discovery of classification rules (recall that PSO and ACO were reviewed in Sections 3.4 and 3.5, respectively). The PSO/ACO classification algorithm proposed in this chapter is freely available on Sourceforge: http://sourceforge.net/projects/psoaco2/. PSO has been explored as a mean for classification in previous work (as discussed in Section 3.6.2) and shown to be rather successful. However, previous authors have never addressed the case where PSO is used for data sets containing both continuous and nominal attributes (as discussed in

Section 3.6.2). The same can be said for ACO, where no variants have been proposed that deal directly with continuous attributes [63].

ACO has been shown to be a powerful paradigm when used for the discovery of classification rules involving nominal attributes [119] and is considered the state of the art for many combinatorial optimisation problems [51]. Furthermore, ACO deals directly with nominal attributes rather than having to convert the problem first into a binary optimisation problem. When compared to other combinatorial optimisation algorithms (e.g., binary PSO) this reduces the complexity of the algorithm and frees the user from the issues involved in the conversion process. Note that, in the case of a nominal attribute containing more than two values the conversion of the nominal attribute into a binary one in order to use binary PSO is not trivial. For instance, consider the nominal attribute marital status taking on 4 values: "single, married, divorced, widow". One could convert this attribute into four binary attribute-values – each of them taking "yes" or "no" for each original nominal value – but this has the drawbacks of increasing the number of attributes (and so the dimensionality of the search space) and requiring a special mechanism to guarantee that, out of the 4 new attributes, exactly one is "turned on" (taking the value "yes") in each candidate classification rule. Alternatively, we could try to use a standard PSO for continuous attributes by converting the original nominal values into numbers, say "1, 2, 3, 4", but this introduces an artificial ordering in the values, whereas there is no such order in the original nominal values. Actually there is good evidence that "native" PSO – i.e., PSO coping with continuous attributes only – performs badly in combinatorial optimisation problems (where the variables are categorical/nominal) as shown in the quote from Maurice Clerk's book on PSO [36] pages 203-204:

> *"[A version of PSO using local search] makes it possible to find a solution [to a travelling salesman problem] in fewer than 5,000 evaluations, which is definitely more acceptable than failing after a million evaluations in "native PSO"! But for problems of more consequence, it is better to call upon a version of PSO taking directly into account the combinatorial aspects in the equations of displacement."*

In this quote Clerk is talking about the way in which he attempted to apply a standard PSO algorithm to a 17 node travelling salesman problem – a classic combinatorial optimisation problem. The algorithm could not find the optimal solution even after a million function evaluations. When augmenting the algorithm with a form of greedy search the PSO algorithm for TSP is more effective, but he still calls for a specialised PSO for dealing with combinatorial optimisation, where particle movement equations can directly cope with categorical/nominal values. As mentioned earlier, this is the research direction followed in this chapter, with the difference being that our proposed algorithm addresses the classification task of data mining, rather than TSP problems.

PSO/ACO uses ideas from ACO to cope directly with nominal attributes, and uses ideas from PSO to cope with continuous attributes, trying to combine "the best of both worlds" in a single algorithm.

We have shown [79] [80] in two of our previous papers that a previous version of the PSO/ACO algorithm proposed in this chapter is at least competitive with binary PSO in terms of a search mechanism for discovering classification rules. PSO/ACO is competitive with binary PSO in terms of accuracy, and often beats binary PSO when rule set complexity is taken into account. In this chapter we propose an improved PSO/ACO algorithm for Rule Induction (PSO/ACO-RI) and provide a comprehensive comparison between it and an industrial standard classification algorithm (PART [164]) across 27 data sets (involving both continuous and nominal attributes).

The PSO/ACO-RI algorithm is a proof-of-concept exploration of the hybrid PSO/ACO paradigm in the context of the classification task of data mining. Creating

PSO/ACO-RI gave us a valuable experience in swarm intelligence for classification. This experience was useful in the design of other PSO/ACO variants for hierarchical classification problems, discussed in Chapter 5.

The remainder of the chapter is organised as follows. Section 4.2 describes in detail the workings of the new algorithm's sequential covering approach, along with the part of the algorithm that deals with continuous attribute-values and rule pruning. Section 4.3 describes the part of the algorithm that supports nominal/categorical attribute-values. Section 4.4 discusses the reasons for the new PSO/ACO-RI algorithm. In section 4.5 we present the experimental set-up and results. In section 4.6 we summarise the main findings of this chapter.

The work presented in this chapter has been reported in a journal paper [84].

4.2. The New PSO/ACO-RI Algorithm

The proposed hybrid Particle Swarm Optimization/Ant Colony Optimization Rule Induction (PSO/ACO-RI) algorithm is a significant extension of our original PSO/ACO algorithm (here denoted PSO/ACO1) proposed in [79]. The PSO/ACO1 algorithm was designed to be the first PSO-based classification algorithm to natively support nominal data – i.e., to cope with nominal data directly, without converting a nominal attribute into a numeric or binary one and then applying a mathematical operator to the converted value, as is the case in [146] (recall that the motivation for natively supporting nominal data was discussed in Section 4.1). The PSO/ACO1 algorithm achieves a native support of nominal data by combining ideas from Ant Colony Optimisation [50] (the successful Ant-Miner classification algorithm, see section 3.6.1) and Particle Swarm Optimisation (discussed in section 3.4) to create a classification meta-heuristic that supports innately both nominal (including binary as a special case) and continuous attributes.

4.2.1. PSO/ACO-RI's Sequential Covering Approach

Both the original PSO/ACO1 algorithm and the new modified version (PSO/ACO-RI) use a sequential covering approach (discussed in section 3.2.1) to discover one-classification-rule-at-a-time. The original PSO/ACO1 algorithm is described in detail in [79] and [80], hereafter we describe how the sequential covering approach is used in PSO/ACO-RI as described in Pseudocode 4.1. The sequential covering approach is used to discover a set of rules. While the rules themselves may conflict (in the sense that different rules covering a given example may predict different classes), the "default" conflict resolution scheme is used by PSO/ACO-RI. This is where any new (test) example to be classified is only considered covered by the first rule that matches it from the ordered rule list. E.g., the first and third rule may cover an example, but the algorithm will stop testing after it reaches the first rule. Although the rule set is generated on a per class basis, it is ordered according to rule quality before it is used to classify new examples in the test set (to be discussed later in this chapter).

The sequential covering approach starts by initialising the rule set (*RS*) with the empty set. Then, for each class the algorithm performs a WHILE loop, where *TS* is used to store the set of training examples the rules will be created from. Each iteration of this loop performs one run of the PSO/ACO-RI algorithm which firstly discovers a rule based on nominal attributes and then adds terms involving continuous attributes, returning the best discovered rule *(Rule)* predicting examples of the current class (*C*). After this rule has been pruned the training examples correctly covered by that rule – i.e., the examples whose attribute-values satisfy the rule antecedent and have the class predicted by the rule consequent – are removed from the training set. This process is repeated as long as necessary to discover rules covering all training examples of the current class. The main steps of the PSO/ACO-RI algorithm are described in detail in the next subsections.

```
RS = ∅   /* initially, Rule Set is empty */
FOR EACH class C
    TS = {All training examples belonging to any class}
    WHILE (Number of uncovered training examples belonging to
            class C > MaxUncovExampPerClass)
        Run the PSO/ACO-RI algorithm to discover the best nominal
        rule predicting class C, called Rule
        Run the standard PSO algorithm to add continuous terms to
        Rule, and return the best discovered rule BestRule
        Prune BestRule
        RS = RS ∪ {BestRule}
        TS = TS - {training examples correctly covered by
        discovered rule}
    END WHILE
END FOR
Order rules in RS by descending Quality
Prune RS removing unnecessary terms or rules
```

Pseudocode 4.1: Sequential Covering Approach used by the Hybrid PSO/ACO-RI Algorithm

4.2.2. Adding Continuous Terms to the Rule using PSO

The rule returned by the nominal/categorical PSO/ACO algorithm is not (usually) complete as it does not include any terms with continuous values. For this to happen, the best rule discovered by the PSO/ACO-RI algorithm is used as a base for the discovery of terms with continuous values.

For the continuous part of the rule a conventional PSO algorithm (applied only to numeric attributes) with constriction is used [21] (section 3.4). The vector to be optimised consists of two dimensions per continuous attribute, one for an upper bound (*ub*) and one for a lower bound (*lb*) (the way in which these are initialised will be discussed in the rest of this subsection). At every particle evaluation the vector is converted to a set of terms (rule conditions) and added to *Rule* produced by the PSO/ACO-RI algorithm for fitness evaluation. For instance, if the data set contained one nominal attribute A_{n0} and one continuous attribute A_{c0} the PSO/ACO-RI algorithm might produce a rule like: IF A_{n0} = <*value*> THEN class *C*. The standard PSO algorithm would then attempt to improve this rule by adding terms: $x_{ub0} > A_{c0}$ AND $x_{lb0} \leq A_{c0}$, which effectively corresponds to a term of the form: $x_{ub0} > A_{c0} \geq x_{lb0}$, where a single particle's position would be the vectors $\vec{x}_{lb}, \vec{x}_{ub}$. The rule for evaluation purposes would be:

```
IF $A_{n0}$ = <value> AND $x_{ub0}$ > $A_{c0}$ AND $x_{lb0}$ ≤ $A_{c0}$ THEN Class C
```

If the two bounds cross over (i.e., $x_{lb0} \geq x_{ub0}$) both terms are omitted from the decoded rule but the *Personal Best* (see Section 3.4) position is still updated in those dimensions.

To improve the performance of the PSO algorithm the upper bound for each dimension is initialised (seeded) in the following manner. Each example in the training set is examined to find the lowest and highest value that each continuous attribute takes. From these values the range of values for each continuous attribute is found. Then each particle's initial position (for the upper bound dimension) is set to a uniformly distributed position between the value of a randomly chosen seed example's continuous attribute and that value added to the range for that attribute. For the lower bound, the same procedure is also conducted except that the position is initialised at a uniformly distributed position between an example's value (for that attribute) and an example's value minus the range for that attribute. This seeding procedure will produce some seeding positions outside the range of the values seen within the data set. This is an intended feature as for some attributes it might never be beneficial to set lower or upper bounds on their values.

The most likely place a particle will be seeded is around the lowest and highest values the seeding examples have (for lower and upper bounds respectively).

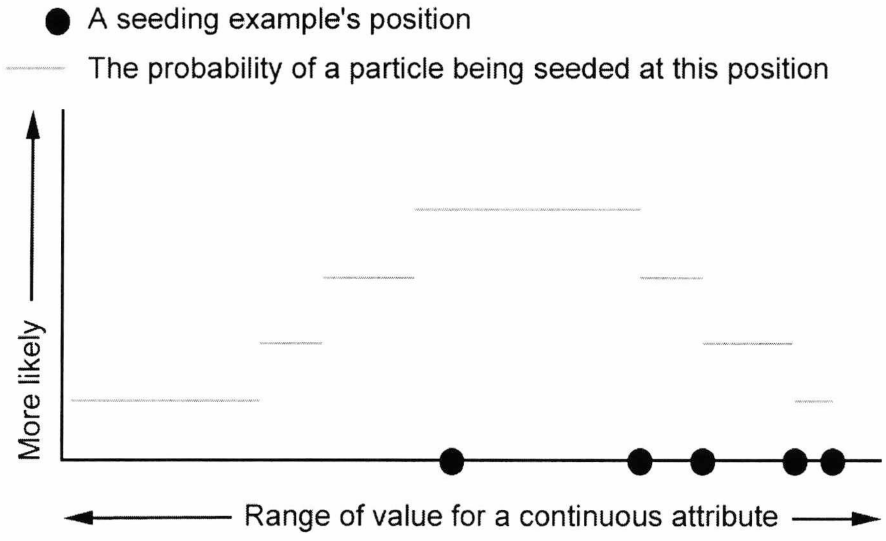


Figure 4.1: The outline of the probability distribution for particle seeding at the lower bound of an attribute value

Figure 4.1 shows the way in which the likelihood of a particle being seeded at a particular position changes (for the lower bound only). It shows the probability distribution (the grey line) for a particle's seeding value for one lower bound of one attribute. The black dots show the values of this particular attribute for the five examples in this class. The most likely place that the particle will be seeded in this dimension (bound) is between the lowest attribute-value and second to lowest attribute-value (shown by the black dots furthest to the left) present in any example in the class.

It is equally likely that any given example will be used as a seeding example for a particle. For the lower bound seeding value, a particle's seeding position is uniformly distributed between the seeding example's value and the seeding example's value minus the range of the values for that attribute. Overall, there is a cumulative probability distribution, with the most likely seeding position being around the position of the seeding value with the lowest value. For the lower bound, this means that no particles will be seeded with values greater than the value of the example with the highest value (as can be seen in Figure 4.1).

This type of seeding distribution will hopefully give the particles a head start at finding good bound values. In an ideal data set there will be non-overlapping groups (one group per class) of continuous values. If this is the case then lower and upper bounds set as (respectively) the lowest and highest values present in the examples from one class will exactly distinguish that class from all others. Obviously this idealised example will not often occur in real data sets, but nevertheless, initially looking for this type of pattern remains a good starting point.

While the standard PSO algorithm attempts to optimise the values for the upper and lower bounds of these terms, it is still possible that the nominal part of the rule may change. The particles in the PSO/ACO-RI algorithm are prevented from fully converging using the Min-Max system (discussed in the next sub-section) used by some ACO algorithms, so that a mechanism for exploratory search remains for the nominal part of the rule. This is helpful for the search, as in combination with the continuous terms, some nominal terms may become redundant or detrimental to the overall rule quality. The exact mechanism of this exploratory search is discussed in Section 4.3.

4.2.3. Pruning the Discovered Rule

After the *BestRule* has been generated it is then added to the rule set after being pruned using a pruning procedure inspired by Ant-Miner's pruning procedure [119]. Ant-Miner's pruning procedure involves finding the term which, when removed from a rule, gives the biggest improvement in rule quality. When this term is found (by iteratively removing each term tentatively, measuring the rule's quality and then replacing the term) it is permanently removed from the rule. This procedure is repeated until no terms can be removed without loss of rule quality. Ant-Miner's pruning procedure attempts to maximise the quality of the rule in any class, so the consequent class of the rule may change during the procedure. The procedure is obviously very computationally expensive; a rule with n terms may require in the worst case – $\frac{n(n+1)}{2}$ i.e. $O(n^2)$ – rule quality evaluations before it is fully pruned [119]. For this reason the PSO/ACO-RI classification algorithm only uses the Ant-Miner pruning procedure if a rule has less than 20 terms (a value empirically determined in our preliminary experiments). If there are more than 20 terms then the rule's terms are iterated through once, removing each one if it is detrimental or unimportant for the rule's quality – i.e., if the removal of the term does not decrease the classification accuracy of the rule on the training set. Also, for reasons of simplicity the rule's consequent class is fixed throughout the pruning procedure in PSO/ACO-RI. These alterations were observed (in initial experiments) to make little or no difference to rule quality.

After the pruning procedure the examples covered by that rule are removed from the training set (*TS*). Recall that an example is said to be covered by a rule if that example satisfies all the terms (attribute-value pairs) in the rule antecedent ("IF part") (discussed in Section 3.2.1). This WHILE loop is performed as long as the number of uncovered examples of the class *C* is greater than a user-defined threshold, the maximum number of uncovered examples per class (*MaxUncovExampPerClass*). After this threshold has been reached *TS* is reset by adding all the previously covered examples. This process means that the rule set generated is unordered – it is possible to use the rules in the rule set in any order to classify an example without unnecessary degradation of predictive accuracy. Having an unordered rule set is important because after the entire rule set is created the rules are ordered by their quality and not the order they were created in. This is a common approach often used by rule induction algorithms [131] [118]. Also, after the rule set has been ordered it is pruned as a whole. This involves tentatively removing terms from each rule and verifying whether each term's removal affects the accuracy of the entire rule set on the training set. If that individual term's removal does not affect the accuracy then it is permanently removed. If it does affect the accuracy then it is replaced and the algorithm moves onto the next term, and eventually the next rule. After this process is complete the algorithm also removes whole rules that do not contribute to the classification accuracy. This is achieved by classifying the training set using the rule list and if any rules do not classify any examples correctly then they are removed.

4.3. The Part of the PSO/ACO-RI Algorithm Coping with Nominal Data in Detail

The PSO/ACO-RI algorithm initially generates the nominal part of the rule, by selecting a (hopefully) near optimal combination of attribute-value pairs to appear in the rule antecedent (the way in which rules are assessed is discussed in Section 4.3.3). The PSO/ACO-RI algorithm generates one rule per run and so must be run multiple times to generate a set of rules that cover the entire training set. The sequential covering approach, as described in Section 4.2.1, attempts to ensure that the set of rules cover the training set in an effective manner. This section describes in detail the part of the PSO/ACO-RI algorithm coping with nominal data, which is the part inspired mainly by ACO. The part of the PSO/ACO-RI algorithm coping with continuous data is essentially a variation of standard PSO where each continuous attribute is represented by two dimensions, referring to the lower and upper bound values for that attribute in the rule to be decoded from the particle, as explained in Section 4.2.1.

To understand – in an intuitive and informal way – why the PSO/ACO-RI algorithm is an effective rule discovery meta-heuristic, it may be useful to first consider how one might create a very simple algorithm for the discovery of classification rules. An effective rule should cover as many examples as possible in the class given in the consequent of the rule, and as few examples as possible in the other classes in the data set. Given this fact a good rule should have the same attribute-value pairs (terms) as many of the examples in the consequent class. A simple way to produce such a rule would be to use the intersection of the terms in all examples in the consequent class as the rule antecedent. This simple procedure can be replicated by an agent based system, as follows.

Each agent has the terms from one example from the consequent class (it is seeded with these terms), each agent could then take the intersection of its terms with its neighbours and then keep this new set. If this process is iterated, eventually all agents will have the intersection of the terms from all examples in the consequent class.

This simple procedure may work well for very simple data sets, but we must consider that it is highly likely that such a procedure would produce a rule with an empty antecedent (as no single term may occur in every example in the consequent class). Also, just because certain terms frequently occur in the consequent class does not mean that they will also not frequently occur in other classes, meaning that our rule will possibly cover many examples in other classes.

PSO/ACO-RI was designed to "intelligently" deal with the aforementioned problems with the simple agent based algorithm by taking ideas from PSO and ACO. From PSO: having a particle network, the idea of a best neighbour and best previous position. From ACO: probabilistic term generation guided by the performance of good rules produced in the past, and directly coping with nominal attribute-values without converting them into binary or integer values. PSO/ACO-RI still follows the basic principle of the simple agent based system, but each particle takes the intersection of its best neighbour's and previous personal best position's terms in a selective (according to fitness) and probabilistic way.

Each particle in the PSO/ACO-RI population is a collection of *n* pheromone matrices (each matrix encodes a set of probabilities) where *n* is the number of nominal attributes in a data set. Each particle can be decoded probabilistically into a rule with a predefined consequent class. Each of the *n* matrices has two entries, one entry represents an *off* state and one entry represents an *on* state. If the *off* state is (probabilistically) selected the corresponding (seeding) attribute-value pair will not be included in the decoded rule. If the *on* state is selected when the rule is decoded the corresponding (seeding) attribute-value pair will be added to the decoded rule. Which value is included in this attribute-value pair (term) is dependant on the seeding values.

The seeding values are set when the population of particles is initialised. Initially, each particle has its past best state set to the terms from a randomly chosen example from class *C* – the same class as the predefined consequent class for the decoded rule. From now on the particle is only able to decode to a rule with attribute-values equal to the seeding terms, or to a rule without some or all those terms. This may seem at first glance counter-intuitive as flexibility is lost – each particle cannot be translated into any possible rule, the reasons for this will be discussed later.

Each pheromone matrix entry is a number representing a probability and all the entries in each matrix for each attribute add up to 1. Each entry in each pheromone matrix is associated with a minimum, positive, non-zero pheromone value. This prevents a pheromone from dropping to zero, helping to increase the diversity of the population (reducing the risk of premature convergence).

For instance, a particle may have the following three pheromone matrices for attributes *Colour, Has_fur* and *Swims.* It was seeded with an example: *Colour=Blue, Has_fur=False, Swims=True, Class=Fish:*

Colour		**Has_fur**		**Swims**	
(on) Colour=Blue	Off	(on) Has_fur=False	off	(on) Swims=True	off
0.01	0.99	0.8	0.2	0.9	0.1

The probability of choosing the term involving the attribute colour to be included in the rule is low, as the *off* flag has a high probability in the first pheromone matrix (0.99). It is likely that the term *Has_fur=False* will be included in the decoded rule as it has a high probability (0.8) in the second pheromone matrix. It is also likely that the term *Swims=True* will be included in the decoded rule.

The most likely rule decoded from this set of pheromone matrices is:

```
IF Has_fur=False AND Swims=True THEN Class=Fish
```

4.3.1. Pseudocode

```
Initialize population
REPEAT for MaxInterations
    FOR every particle $x$
        /* Rule Creation */
        $C$ = The current class being predicted
        Set Rule $R_x$ = "IF $\varnothing$ THEN $C$"
        FOR every dimension $d$ in $x$
            Use roulette wheel selection to choose whether the
            state should be set to off or on. If it is on then
            the corresponding attribute-value pair set in the
            initialisation will be added to $R_x$; otherwise (i.e.,
            if off is selected) nothing will be added.
        END FOR
        Calculate Quality $Q_x$ of $R_x$
        /* Set the past best position */
        $P$ = $x$'s past best state
        $Q_P$ = $P$'s quality
        IF $Q_x > Q_P$
            $Q_p = Q_x$
            $P = x$
        END IF
    END FOR
    FOR every particle $x$
        $P$ = $x$'s past best state
        $N$ = the best state ever held by a neighbour of $x$
        according to $N$'s quality $Q_N$
        FOR every dimension $d$ in $x$
            /* Pheromone updating procedure */
            IF $P_d = N_d$ THEN
                pheromone_entry corresponding to the value of $N_d$
                in the current $x_d$ is increased by $Q_p$
            ELSE IF $P_d$ = off AND seeding term for $x_d \neq N_d$ THEN
                pheromone_entry for the off state in $x_d$ is
                increased by $Q_p$
            ELSE
                pheromone_entry corresponding to value of $N_d$ in
                the current $x_d$ is decreased by $Q_p$
            END IF
            Normalize pheromone_entries
        END FOR
    END FOR
END REPEAT
RETURN best rule discovered
```

Pseudocode 4.2: The Part of the PSO/ACO-RI Algorithm Coping with Nominal Data

Pseudocode 4.2 shows the new PSO/ACO-RI algorithm proposed in this chapter and utilised in Pseudocode 4.1 (sequential covering approach).

Firstly the population of particles is initialised. Each particle is seeded with terms from a randomly selected example, as described earlier. Initially, in each dimension the pheromone for the *on* state is set to 0.9 and the pheromone for the *off* state is set to 0.1. The first loop (REPEAT statement) iterates the whole population for *MaxIterations*. Then for each particle x a rule is built probabilistically from its pheromone matrices. For each dimension d in x, roulette-wheel selection – a well-known selection method in evolutionary algorithms [54] – is used to decide if the *on* or *off* state should be selected. Roulette wheel (fitness proportional) selection simply means that the probability of an attribute-value being picked is proportional to the corresponding value in each entry in the pheromone matrix. It would also be possible to use most other selection mechanisms [54] but as PSO/ACO-RI is inspired by the Ant-Miner algorithm it uses its selection mechanism. Investigation into the effects of using different selection mechanisms is a topic for future research.

In PSO/ACO-RI roulette selection simply involves the following rule:

```
IF rand() > pheromone in entry for on state THEN
     Select on state
ELSE
     Select off state
End IF
```

Where rand() returns a number from the interval [0..1] with a uniform probability distribution. If the *on* state is selected then the corresponding term is added to the antecedent of R_x. This is an attribute-value pair where the attribute corresponds to the dimension d and the value corresponds to the initial seeding value. After this process has been repeated for every dimension, the quality Q_x of the rule is calculated. If the new Q_x is greater than the previous best Q_p, then the particle's state is saved as P.

After the rule creation phase the pheromone is updated for every particle. Each particle finds its best neighbour's best state (*N*) according to Q_N (the quality of the best rule *N* has ever produced). The particles are in a static topology, so each particle has a fixed set of neighbour particles throughout the entire run of the algorithm. PSO/ACO-RI uses Von-Neumann [96] topology (Figure 3.7), where the particles are arranged in a 2D grid, each particle having four neighbours. This topology was chosen as it consistently performed the best in initial experiments. This is likely due to the level of connectivity present in this particular topology, i.e., enough connectivity but not too much (global topology was shown to be suboptimal due to premature convergence to a local optimum in the search space).

4.3.2. Pheromone Updating Procedure

The pheromone updating procedure is influenced by two factors, the best state a particle x has ever held (the state *P*), and the best state ever held by a neighbour particle *N*. As discussed previously each dimension can take two values and so it has two corresponding pheromone entries, one for the *on* state and one for the *off* state. These states are examined in every dimension d and the following rules are applied. If P_d is the same as N_d then an amount of pheromone equal to Q_p (the quality of *P*) is added to the pheromone entry in x_d corresponding to the value of P_d. In other words, if a particle and its neighbour have both found that a particular attribute-value is good then pheromone is added to the entry corresponding to it. If P_d and N_d disagree about that attribute-value then pheromone is removed from the corresponding entry. This is directly related to the principles of the simple algorithm discussed in Section 4.3, where the aim of the algorithm is to get the particles to agree on a single solution – by making it less likely for neighbouring particles to pick disagreeing attribute-values.

There is a caveat in this pheromone updating procedure given by the "ELSE IF" statement in Pseudocode 4.2. It states that if P_d is *off* and the current particle and its best neighbour do not have the same seeding terms, then increase the likelihood of choosing the *off* state (by adding pheromone to the pheromone entry corresponding to the *off* value). The reason for this is to maintain the idea from the simple agent based system described earlier in this section. That is, when two particles have different seeding terms then those terms should tend to be omitted. Without this caveat the opposite would happen, the probability of the term being omitted would become less, as pheromone would otherwise be removed from P_d (*off*) if P_d and N_d do not agree. A more detailed examination of the effect of the pheromone updating rules can be seen in Table 4.1, with this caveat being shown in the second row from the bottom.

The third pheromone updating rule (the ELSE statement) states that if P_d is not the same as N_d (and P_d is not *off*) then an amount of pheromone equal to Q_p is removed from the pheromone entry in x_d corresponding to the value of P_d.

If after this process is completed any pheromone entry is less than a predefined minimum amount then it is set to that amount (0.01). It is possible that adjusting this value might improve performance under the specific conditions of the given application – this is a topic left for future research, but careful experimentation would undoubtedly be a good starting point. Importantly, the minimum pheromone value allows the pheromone entry that is not the best state to increase due to the normalisation procedure. This increase will occur if pheromone is removed from a state. If this happens the amount of pheromone in the matrix becomes less than 1 and, as long as both entries have a greater than zero amount of pheromone, when the matrix is normalised both entries will increase. It also aids search in a conceptually similar way to mutation in GAs and the Min-Max system in the ACO literature [51].

In Table 4.1 the six possible scenarios for pheromone updating in the current particle x are described given the differing states of P_d, N_d and also the seeding term for x_d. The two highlighted rows show the only cases where pheromone is increased for selecting the *on* seeding value. These outcomes are controlled by the pheromone updating rules shown in Pseudocode 4.2 (discussed previously).

The first and last cases shown in the table are quite intuitive, if both P_d and N_d agree on a state that state is made more likely in the current particle. This allows the algorithm to converge on a good solution that the particles P_d and N_d agree on.

Cases of the second type are shown in the second and fifth rows, where P_d and N_d have different seeding terms. In these cases the current particle x makes it more likely that the conflicting term will be omitted from the decoded rule, by increasing the pheromone of the *off* state. This feature allows the particle to create rules that generalise well, covering more examples from the consequent class (discussed further in Section 4.4).

Seeding Term for x_d	P_d	N_d	**Outcome for entries in x_d**
<value>=w	(*on*) *<value>=w*	(*on*) *<value>=w*	*on* pheromone increased *off* pheromone decreased
<value>=w	(*on*) *<value>=w*	(*on*) *<value>≠w*	*off* pheromone increased *on* pheromone decreased
<value>=w	(*on*) *<value>=w*	*off*	*off* pheromone increased *on* pheromone decreased
<value>=w	*off*	(*on*) *<value>=w*	*on* pheromone increased *off* pheromone decreased
<value>=w	*off*	(*on*) *<value>≠w*	*off* pheromone increased *on* pheromone decreased
<value>=w	*off*	*off*	*off* pheromone increased *on* pheromone decreased

Table 4.1: PSO/ACO-RI Pheromone Updating Scenarios

Cases of the third type – which involve a disagreement between P_d and N_d about whether or not the seeded term should be used in the rule decoded from the current particle – are shown in the third and fourth rows. These cases bias the search towards N_d so that each particle tries to emulate its best neighbour. In the third row; if N_d = *off* (and N_d and x_d have the same seeding terms) then the probability of x_d decoding to *off* will be increased (by increasing the pheromone associated with the *off* state). In the fourth row; if $N_d = w$ (and N_d and x_d have the same seeding terms) the probability of x_d decoding to *on* will be increased. The cases of the third type allow the particles to come to a consensus about the best set of states. By trying to emulate its best neighbour each particle has the potential to create (in future iterations) a new past best state (*P*) based on a mix of its own current *P* and *N*.

4.3.3. Measuring Rule Quality

It is necessary to estimate the quality of every candidate rule (decoded particle). A measure must be used in the training phase in an attempt to estimate how well a rule will perform in the testing phase. Given such a measure it becomes possible for an algorithm to optimise a rule's quality (the fitness function). In our previous work [79] the Quality measure used was *Sensitivity* × *Specificity* (Equation 4.1) [74].

$$\text{Sensitivity} \times \text{Specificity} = (TP\ /\ (TP + FN)) \times (TN\ /\ (TN + FP))$$

Equation 4.1: Quality Measure used by PSO/ACO1 [79]

Where *TP*, *FN*, *FP* and *TN* are, respectively, the number of true positives, false negatives, false positives and true negatives associated with the rule [164]:

- True Positives (*TP*) are the number of examples that match the rule antecedent (attribute-values) and also match the rule consequent (class). These are desirable correct predictions.
- False Positives (*FP*) are the number of examples that match the rule antecedent but do not match the rule consequent. These are undesirable incorrect predictions.
- False Negatives (*FN*) are the number of examples that do not match the rule antecedent but do match the rule consequent. These are undesirable uncovered cases and are caused by an overly specific rule.

- True Negatives (*TN*) are the number of examples that do not match the rule antecedent and do not match the rule consequent. These are desirable and are caused by a rule's antecedent being specific to its consequent class.

In the new PSO/ACO-RI classification algorithm proposed in this chapter the quality measure is *Precision* with Laplace correction [34] [164], as per Equation 4.2. In initial experiments this quality measure was observed to lead to the creation of rules that were more accurate (when compared to the original quality measure shown in Equation 4.1).

```
Laplace-Corrected Precision = (1 + TP) / (1 + TP + FP)
```

Equation 4.2: New Quality Measure used by PSO/ACO-RI

We observed that in some cases (when using Equation 4.2 as a quality measure) rules would be generated covering very few examples. These cases were likely due to the way in which the *Laplace-Corrected Precision* measure penalises False Positives very severely (when compared to *Sensitivity* × *Specificity*). To stop this less than ideal situation we added the following conditional statement to the new quality measure:

```
IF TP < MinTP
     Rule Quality = Laplace-Corrected Precision * 0.01
ELSE
     Rule Quality = Laplace-Corrected Precision
END IF
```

Where *MinTP* is the least number of correctly covered examples that a rule has to cover before it is given a "normal" value, as computed by Equation 4.2. When a rule covers too few examples the quality is severely reduced (by a factor of 100). This procedure reduces the quality below the quality of any normal rule but still allows the particles covering fewer than *MinTP* examples to compare their solutions effectively. In the experiments reported in this chapter we set *MinTP* to 10, optimisation of this parameter is left for future research.

4.4. Motivations for PSO/ACO-RI and Discussion

PSO/ACO-RI improves upon our original preliminary PSO/ACO1 algorithm for classification. However, both algorithms are based on the same principle. They are both PSO based algorithms that have pheromone matrices instead of a single velocity value in each dimension of search space.

The modified algorithm (PSO/ACO-RI) discussed in this chapter differs from our original algorithm (PSO/ACO1) proposed in [79] [80] in five important ways.

- Firstly PSO/ACO1 attempted to optimise both the continuous and nominal attribute-values present in a rule antecedent at the same time, whereas PSO/ACO-RI takes the best nominal rule built by nominal PSO/ACO algorithm and then attempts to add continuous attributes to it using a conventional PSO algorithm.
- Secondly the original algorithm used a type of rule pruning to create seeding terms for each particle, whilst PSO/ACO-RI uses all the terms from an entire training example (record).
- Thirdly in PSO/ACO1 it was possible for a particle to select a value for an attribute that was not present in its seeding terms, whilst in PSO/ACO-RI only the seeding term values may be added to the decoded rule.
- Fourthly the pheromone updating rules have been simplified to concentrate on the optimisation properties of the original algorithm. In PSO/ACO1 pheromone was added to each entry that corresponded to the particle's past best state, its best neighbour's best state, and the particle's current state in proportion to a random learning factor. Now pheromone is only added to a pheromone matrix entry in the current particle x when N_d and P_d match, or taken away when they do not.

- Fifthly, the algorithm now prunes the entire rule set after creation, not simply on a per rule basis.

In PSO/ACO-RI the conventional PSO for continuous data and the hybrid PSO/ACO-RI algorithm for nominal data have been separated partially because they differ quite largely in the time taken to reach peak fitness. It usually takes about 30 iterations (depending on the complexity of the data set) for the pheromone matrices to reach a stable state in PSO/ACO-RI, whilst it tends to take considerably longer for the standard PSO algorithm to converge. Due to this fact the standard PSO algorithm's particles set past best positions in quite dissimilar positions, as their fitness is dependant on the quickly converging part of the PSO/ACO-RI algorithm coping with nominal data. This causes high velocities and suboptimal search, with a higher likelihood of missing a position of high fitness. Therefore, separating the rule discovery process into two stages – one stage for the part of the PSO/ACO-RI algorithm coping with nominal data and one stage for the part of the PSO/ACO-RI algorithm coping with continuous data (essentially a variation of a standard PSO) – provides a better control of the search and more consistent results.

Secondly, in the PSO/ACO1 algorithm, sets of seeding terms were pruned before they were used. This aggressive pruning algorithm used a heuristic to discard certain terms. This is less than ideal as the heuristic does not take into account attribute interaction, and so potentially useful terms are not investigated.

To understand the reasons behind the last two modifications it is important to understand how the algorithms find good rules. In both PSO/ACO1 and PSO/ACO-RI sets of terms are generated by mixing together the experiences of the particles and their neighbours. As the entries in the pheromone matrices converge and reach one (or zero), better rules should be generated more often. In PSO/ACO1 the levels of the pheromone in the matrices are influenced by three factors (current state, past best state and best neighbours' best state). If these factors do not agree then the pheromone matrix will be slow to converge. Slow convergence can sometimes be advantageous as the algorithm should not prematurely converge to a local maximum. However, in PSO/ACO1 the result of this slow convergence is usually destructive, as incompatible terms can be mixed together over and over again. Incompatible terms are terms that do not cover any of the same examples. For instance, in Table 4.2, incompatible terms are $A_{n1} = a$ and $A_{n2} = b$. A rule including both these terms would have a quality of zero as it would not cover any examples. This problem is addressed by the third modification in PSO/ACO-RI, now incompatible terms will not be mixed. This modification also ensures a particle will always cover at least one example (the seeding example) even if all the terms are included in the decoded rule. This was not the case in PSO/ACO1 as at the beginning of the search many incompatible terms could be mixed, creating many particles with zero fitness.

	A_{n1}	A_{n2}	A_{n3}
R_1	*a*	*a*	*a*
R_2	*a*	*a*	*b*
R_3	*a*	*a*	*b*
R_4	*b*	*b*	*b*
R_5	*b*	*b*	*b*
R_6	*b*	*b*	*b*

Table 4.2: An Example Single Class Data Set, R's are Records, A_n's are Nominal Attributes

In PSO/ACO-RI the pattern being investigated by the particles will likely include relatively general terms – an example might be a rule including the term $A_{n3} = b$ in Table 4.2. It is the job of the PSO/ACO-RI algorithm to find terms that interact well to create a rule that is not only general to the class being predicted (covering many examples of that class) but also specific to the class (by not covering examples in other classes). It is also the job of the PSO/ACO-RI algorithm to turn off terms that limit the generality of the rule without adding specificity to it. This trade-off between specificity and generality (or sensitivity) is calculated by the rule quality measure. It is clear, in Table 4.2, that including values for A_{n1} and A_{n2} will not ever lead to the most general rule (the optimal rule only has one term, $A_{n3} = b$). Due to the new pheromone updating procedures a particle would choose the *off* state for these conflicting attributes quickly.

4.5. Computational Results

For the experiments we used 27 data sets from the well-known UCI dataset repository [111]. We performed 10 fold cross validation [164], and ran the PSO/ACO-RI algorithm 10 times for each fold – with a different random seed for each run – as it is a stochastic algorithm.

Both the part of the PSO/ACO-RI algorithm coping with nominal data and the standard PSO algorithm (i.e. the part of PSO/ACO-RI coping with continuous data) had 100 particles, and these two algorithms ran for a maximum of 100 iterations (MaxIterations) per rule discovered. In all experiments constriction factor χ = 0.72984 and social and personal learning coefficients c1 = c2 = 2.05, as is standard in the literature [21].

MaxUncovExampPerClass was set to 10 as this is also standard in the literature [119]. As mentioned previously PSO/ACO-RI uses Von-Neumann topology, where each particle has four neighbours, with the population being connected together in a 2D grid. The corrected WEKA [164] statistics class was used to compute the standard deviation of the predictive accuracies and to apply the corresponding corrected two-tailed Student's T-Test – with a significance level of 5% – in the results presented in Table 4.3 and Table 4.4.

The algorithms compared in Table 4.3 are PSO/ACO-RI and PART. PART is WEKA's implementation of a variation of the well-known C4.5Rules rule induction algorithm [164]. PART extracts rules from decision trees created by J48 (WEKA's implementation of C4.5). We compared PSO/ACO-RI against PART as the latter is a very well regarded and commonly used classification algorithm.

Data Set	Predictive Accuracy (%)		Average Rule Size		Average Rule Set Length	
	PSO/ACO-RI	PART	PSO/ACO-RI	PART	PSO/ACO-RI	PART
Autos	76.63±8.36	79.83±11.43	2.8±0.17	2.54±0.24	16.0±1.25	14.2±2.74
balance-scale	82.72±4.77	79.38±7.81	2.56±0.17	3.13±0.16	26.6±1.07	38.9±3.25
breast-cancer	72.62±6.84	69.7±7.8	1.73±0.26	1.91±0.18	12.4±2.27	17.3±4.72
breast-w	93.42±3.79	93.7±4.05	1.17±0.09	1.01±0.03	9.9±1.6	10.4±3.03
credit-a	85.31±4.14	84.23±3.35	2.94±0.31	2.46±0.34	22.7±2.0	30.8±9.66
credit-g	67.9±5.82	69.7±4.4	4.23±0.19	3.01±0.25	54.3±1.89	77.0±4.57
Crx	85.6±2.84	84.54±2.8	2.94±0.28	2.44±0.31	22.5±3.1	29.9±8.67
diabetes	72.67±4.98	74.36±4.51	3.88±0.29	1.88±0.23	33.4±1.43	7.1±1.52
Glass	70.95±7.5	65.43±11.45	3.11±0.18	2.7±0.28	20.4±1.35	16.1±1.6
Heart-c	77.38±5.45	78.72±5.92	3.33±0.19	2.42±0.21	12.6±0.84	19.9±2.42
Heart-statlog	81.11±6.16	78.15±6.64	3.17±0.44	2.88±0.34	9.7±1.34	18.4±1.9
ionosphere	88.06±4.91	90.04±4.68	3.33±0.79	2.35±0.43	3.6±0.97	8.9±1.91
Iris	94.67±5.26	90.67±7.17	0.93±0.14	1.02±0.05	3.0±0.0	4.3±1.42
iris_d	94.67±6.13	94.0±5.84	0.68±0.04	0.76±0.06	3.2±0.42	4.4±0.97
lymph	83.05±6.67	83.19±9.47	1.89±0.15	2.26±0.42	14.7±2.0	10.0±1.25
mushroom	99.9±0.11	100.0±0.0	1.86±0.18	1.55±0.02	8.7±0.48	12.8±0.42
promoters	81.0±12.12	83.91±7.91	1.02±0.05	1.02±0.14	5.1±0.32	6.9±1.2
segment	96.67±1.17	96.67±0.84	2.8±0.27	3.07±0.17	21.9±0.99	26.3±1.7
Sonar	75.05±9.11	72.52±10.57	2.6±0.63	2.23±0.49	4.4±1.58	7.4±1.17
soybean	87.01±6.53	90.57±3.96	2.08±0.21	2.66±0.16	24.2±1.03	32.1±3.21
tic-tac-toe	100.0±0.0	93.85±2.7	2.67±0.0	2.65±0.11	9.0±0.0	38.3±3.06
vehicle	73.05±4.45	73.29±2.77	3.85±0.18	3.84±0.38	37.8±1.2	34.0±3.02
vowel	86.16±3.47	85.05±5.79	4.2±0.25	3.55±0.21	29.0±0.82	50.5±3.57
wisconsin	94.87±2.53	94.43±2.06	1.21±0.07	1.02±0.03	10.2±1.87	9.9±3.11
kr-vs-kp	99.47±0.51	99.37±0.29	2.25±0.15	3.03±0.35	18.7±2.0	22.7±1.34
Zoo	97.18±6.25	94.18±6.6	1.14±0.18	1.48±0.12	7.1±0.32	7.6±0.52
splice	93.48±1.24	92.79±1.65	3.0±0.07	2.65±0.1	88.0±2.91	99.6±6.1

Table 4.3: Predictive Accuracy and Rule Set size of PSO/ACO-RI and PART in UCI Data Sets, with Standard Deviation and Student T-Test Shadings

The first two columns (not including the data set column) in Table 4.3 show the percentage predictive accuracy of both algorithms – i.e., their average predictive accuracy over the 10 test sets associated with the cross-validation procedure. The second two columns show the average rule size (number of terms, or attribute-value pairs) for the rules generated for each data set. The third two columns show the average rule set size for each data set; this is simply the average number of rules in each rule set. The measures of average rule size and average rule set size give an indication of the complexity (and so indirectly comprehensibility) of the rule sets produced by each algorithm. The shading in these six columns denotes a statistically significant win or a loss (according to the corrected WEKA two tailed Student's T-Test), light grey for a win and dark grey for a loss against the baseline algorithm (PART). Table 4.4 shows the overall score of the PSO/ACO-RI classification algorithm against PART, considering that a significant win counts as "+1" and a significant loss counts as a "-1", and then calculating the overall score across the 27 data sets.

	Predictive Accuracy T-Test Results	**Average Rule Size T-Test Results**	**Average Rule Length T-Test Results**
Total	1	-6	14

Table 4.4: Summation of the number of statistically significant results of PSO/ACO-RI against PART according to the Student T-Test (out of 27 Data Sets).

It can be seen from Table 4.3 and Table 4.4 that in terms of accuracy PART and PSO/ACO-RI are quite closely matched. This is overall a good result as PART is already considered to be very good in terms of predictive accuracy and it is the result of decades worth of research into rule induction algorithms. Furthermore, there is only one result that is significant in terms of accuracy: the accuracy result for the tic-tac-toe data set. However, if one scans through the accuracy results it is clear that often one algorithm outperforms the other slightly. In terms of rule set complexity the algorithms are much less closely matched. When the average rule size results are taken as a whole PSO/ACO-RI generates significantly longer rules in 6 cases overall. Although the average rule size results are significant, the real impact of having a rule that is under one term longer is arguable (as is found in many cases). The most significant results by far are to be found in the rule set size columns. PSO/ACO produces significantly smaller rule sets for 14 data sets overall, sometimes having tens of rules less than PART. These improvements have a tangible effect on the simplicity of the rule set as a whole.

4.6. Summary

In this chapter we have proposed a new PSO/ACO (Particle Swarm Optimisation/Ant Colony Optimisation) algorithm for rule induction. We have conducted experiments on 27 public domain "benchmark" data sets used in the classification literature. We have also shown that PSO/ACO-RI is at least competitive with PART (an industry standard classification algorithm) in terms of accuracy, and that PSO/ACO-RI often generates *much* simpler (smaller) rule sets. This is a desirable result in data mining – where the goal is to discover knowledge that is not only accurate but also comprehensible to the user.

The proof-of-concept PSO/ACO-RI algorithm has provided promising results. We explore the PSO/ACO paradigm further in the next chapter, where we propose new PSO/ACO algorithms for hierarchical classification problems involving combinatorial optimisation.

Chapter 5. Particle Swarm Optimisation/Ant Colony Optimisation for Classifier Selection and Misclassification Recovery in Hierarchical Classification

5.1. Introduction

In this chapter we propose and evaluate two new methods to increase the accuracy of classification when using the top-down divide and conquer (TDDC) approach for hierarchical classification (as described in Chapter 3). The methods examined in this chapter are applied to datasets involving the hierarchical prediction of GPCR function (discussed in Chapter 2).

The new methods employ a swarm intelligence algorithm, more precisely a hybrid Particle Swarm Optimisation/Ant Colony Optimisation (PSO/ACO) algorithm derived from the PSO/ACO-RI classification algorithm described in Chapter 4. The first method involves using the PSO/ACO algorithm for classifier selection (denoted as PSO/ACO-CS) (Section 3.7.2.1). The second method involves recovering misclassifications made by parent classifier nodes in the TDDC tree. PSO/ACO is also used to improve the performance of this "recovery approach". The variant of PSO/ACO used to improve the performance of the recovery approach is denoted as PSO/ACO-RO in this thesis – where the "RO" stands for recovery optimiser. The two proposed methods are also combined to improve TDDC classification accuracy further.

The remainder of this chapter is organised as follows. Section 5.2 introduces the PSO/ACO-CS algorithm. Section 5.3 describes the recovery approach and how

PSO/ACO-RO is used to improve it. Section 5.4 introduces the hierarchical protein function datasets, the way in which they are partitioned for the experiments, and the base algorithms used in this investigation. Section 5.5 examines the results from the experiments involving PSO/ACO-CS. Section 5.6 examines the results from the experiments involving the recovery approach and the PSO/ACO-RO algorithm. Section 5.7 provides an overview of the findings from this chapter.

Part of the work reported in this chapter appeared in a conference paper [83] (which won best paper award).

5.2. Global Search-Based Classifier Selection with a Particle Swarm Optimisation/Ant Colony Optimisation Algorithm

Given the discussion in Section 3.7.2.1 it is quite clear that there is a potential to improve the classification accuracy of the entire classifier tree by using a more "intelligent" classifier selector – a classifier selector that (unlike the greedy one) takes into account interaction among classifiers at different classifier nodes. As there is an obvious objective function to be optimised – the classification accuracy of the entire TDDC tree on the validation set (part of the training set) – and also a collection of elements whose optimal combination has to be found – the type of classifier at each classifier node, it seems appropriate to use a combinatorial optimisation algorithm.

We propose to optimise the selection of a classifier at each classifier node with a PSO/ACO method, adapted from the PSO/ACO method described in Chapter 4. The choice of this algorithm was motivated by the following factors. Firstly PSO/ACO has been shown to be an effective classification-rule discovery method [79] [80] [81] across a wide variety of data sets involving mainly nominal attributes. Secondly, the PSO/ACO method can be naturally adapted to be used as a classifier selector, where instead of finding a good combination of attribute-values for a rule, it finds good combinations of classifiers for all the nodes of the classifier tree. This is possible because the combination of classifiers can be specified by a set of nominal values (types of classification algorithms).

To recap, the PSO/ACO method works with a population of particles. Each particle contains multiple pheromone vectors and each pheromone vector is used to probabilistically decide which value of a nominal attribute is best in each dimension of the problem's search space. In the original PSO/ACO method for discovering classification rules these dimensions correspond to predictor attributes of the data being mined, so there is one pheromone vector for each nominal attribute. The entries in each individual pheromone vector correspond to possible values the attribute can take, and each pheromone value denotes the "desirability" of including the corresponding attribute value in a rule condition. We now describe in detail how this algorithm was adapted to act as a classifier selector, rather than discovering classification rules.

To optimise the classifier selection at each classifier node the problem must be reduced to a set of dimensions and possible values in each dimension. Hence, in the proposed PSO/ACO-CS approach each decoded particle (candidate solution) consists of a vector with n components (dimensions), as follows:

$$\mathit{Decoded\ Particle} = w_1, w_2, \ldots, w_n$$

Where w_d $(d=1,..,n)$ is the classifier selected at the dth classifier node in the TDDC tree and n is the number of classifier nodes in the tree. Each w_d can take one of the nominal (classifier ids) values $c_1,..c_k$ where k is the number of different candidate classifiers at each node of the TDDC tree.

It must also be possible to assess how good an individual solution created from an individual particle is. To do this the validation set is classified by the TDDC tree composed of the classifiers specified by the particle, and that tree's average classification accuracy (the mean of the accuracy from each class level) on the validation set is taken.

The mean classification accuracy across all the class levels is used as the "fitness" (evaluation) function for evaluating each particle's quality.

Note that the only increase in computational time for this approach (over the greedy selective approach) is in the time spent classifying examples at each fitness evaluation. The classifiers are trained using the same data at each fitness evaluation and so can be cached and reused without the need for retraining.

Pseudocode 5.1 shows the PSO/ACO-CS algorithm. At each iteration each pheromone vector for each particle produces a state in a probabilistic manner. That is, the probability of choosing a given classifier (c_1,..c_k) for a given classifier node (w_1,..w_n) is proportional to the amount of pheromone (a number between 0 and 1) in the corresponding entry in the corresponding pheromone vector (τ_{pd} is the pheromone vector corresponding to particle *P* and classifier node *d*), see Figure 5.1. More precisely, the selection of a classifier at each classifier node is implemented by a fitness proportional (roulette-wheel) selection mechanism [54].

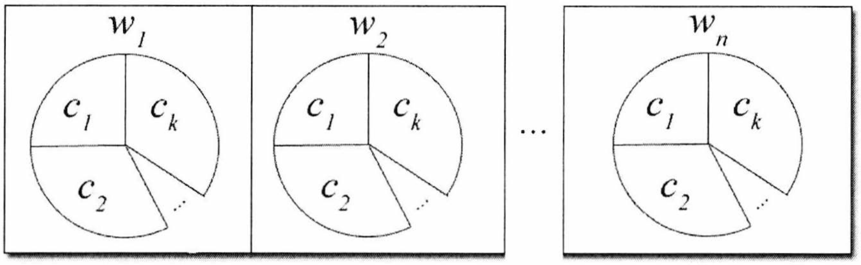


Figure 5.1: An encoded particle with *n* dimensions, each with *k* classifier ids

Figure 5.1 shows an encoded particle *P*. Each section labelled $c_1,c_2,..c_k$ (in each dimension $w_1,w_2,..,w_n$) represents an amount of pheromone. The probability of choosing each classifier $c_i\,(i=1,..,k)$ in each dimension $w_d\,(d=1,..,n)$ is proportional to the amount of pheromone (τ) in the corresponding pheromone entry τ_{pdi}.

```
Divide Training Set into Building and Validation sets
Initialize population
REPEAT for MaxInterations
  FOR every particle P
    /* Classifier Selection */
    FOR every dimension $w_d$ in $P$
      Use fitness proportional selection on pheromone vector
      corresponding to $w_d$ to choose which state (classifier id)
      $c_1,..c_k$ should be chosen for this $w_d$
    END FOR
    Construct a classifier tree from the Building Set by using
    the classifiers selected from the particle's pheromone
    vectors
    Calculate fitness $F$ of this set of classifiers $w_1,..w_n$ on the
    Validation Set
    /* Set the past best position */
    $P_b$ = $P$'s past best combination of classifiers
    $F_b$ = The quality of $P_b$
    IF $F > F_b$
      $F_b = F$
      $P_b$ = the current combination of classifiers $w_1,..w_n$
    END IF
  END FOR
  FOR every particle P
    Find $P$'s best Neighbour Particle $N$ according to each
    neighbour's best fitness ($F_b$)
    FOR every dimension $w_d$ in $P$
      /* Pheromone updating procedure */
      $f$ = $N$'s best fitness $F_b$
      $y$ = $N$'s best state $P_b$ in dimension $d$
      /* Add an amount of pheromone proportional to f to the
      pheromone entry for particle P corresponding to y (the best
      position held by P's best Neighbour) */
      $\tau_{pdy} = \tau_{pdy} + (f \times \alpha)$
      Normalize $\tau_{pd}$
    END FOR
  END FOR
END REPEAT
```

Pseudocode 5.1: The Hybrid PSO/ACO-CS Algorithm for Classifier Selection

The "decoded" state for all m dimensions of the particle is then evaluated, and if it is better than the previous personal best state (P_b), it is set as the personal best state for the particle. A particle finds its best neighbour (N) according to the fitness of each neighbour's best state (P_b). In this chapter the particles are arranged in a Von-Neumann topology (discussed in Section 3.4), so that each particle has four neighbours.

A slightly different pheromone updating approach is taken with the PSO/ACO-CS algorithm for classifier selection when compared to the PSO/ACO-RI algorithm for rule discovery. As detailed in the pheromone updating procedure in Pseudocode 5.1, the approach simply consists of adding an amount of pheromone proportional to f to the pheromone entry corresponding to τ_{pdy}, where f is the fitness of the best neighbour's best state, y is the best neighbour's best state ($c_1,..c_k$) in the particular dimension d ($w_1,..w_n$) and P is the current particle. The amount of pheromone added can be modified to slow down (or speed up) convergence, this is achieved using the constant α. The closer this constant is set to 0 the slower the convergence achieved. The pheromone vectors are normalised after pheromone has been added, so that the pheromone entries of each pheromone vector add up to 1. All the experiments reported in this chapter use a value α = 1, which effectively means the constant α has no influence in the results. Future research could perform experiments with different values of α, in order to study its influence on the performance of the algorithm.

5.3. Recovering from Misclassifications at Parent Classifier Nodes in the Top-Down Divide-and-Conquer Tree

As explained in section 3.7.2 one of the main problems with the standard TDDC approach is that once an example has been misclassified at one classifier node, it can never be correctly classified at a deeper classifier node. This sort of situation can potentially be predicted during the training phase and so corrected during the testing phase. When the examples of a given building set (a proper subset of the training set) are classified by a classifier node, it is possible to find out which examples have been misclassified and so would potentially be sent to the wrong child classifier node. For example, if a classifier discriminating between classes 1 and 2 is built, it is possible to classify the examples in the building set used to build this classifier and find the examples belonging to class 1 that are misclassified as class 2 and vice versa. It is then possible to use these misclassified examples (along with the examples actually belonging to the classes) to train the child classifiers in an attempt to correct the misclassifications of the parent node.

For example, suppose one parent classifier node discriminating between classes 1 and 2 misclassifies some examples belonging to class 2 as class 1. In the conventional TDDC approach the classifier discriminating between classes 1.1 and 1.2 would be trained only with examples of classes 1.1 and 1.2. By contrast, in the proposed "recovery approach" the classifier node discriminating between classes 1.1 and 1.2 receives the modified building set composed of the examples belonging to classes 1.1, 1.2 and the misclassified (misclassified as belonging class node 1) examples belonging to class node 2. As shown in Figure 5.2, during the classification of examples in the validation or test set, if an example is assigned class 2 by the classifier discriminating between 1.1, 1.2 and 2 then the system sends that example back to the classifier node discriminating between classes 2.1 and 2.2. Note that the example must not be sent to any ancestor node, as this may cause a loop with the same example being misclassified over and over again (if the classifiers are deterministic). For instance, it must not send the example now classified as 2 back to the node discriminating between 1 and 2, as it will likely be misclassified as class 1 again.

Other types of loops may also develop, so the simplest way to avoid these situations is to use a counter to limit the number of times an example can be "redirected" (or "recovered") to another classifier node. For instance, suppose an example is always sent to the classifier node discriminating between classes 2.1, 2.2 and 1 by the classifier node discriminating between 1.1, 1.2 and 2. If the node discriminating between classes 2.1, 2.2

and 1 also always sent the example to the node discriminating between 1.1, 1.2 and 2 an infinite loop would occur. We set the value for the maximum number of redirections to be ten (we consider this to be a safe number), investigation of this parameter is left for future research. Although it is in theory possible that complex redirections may form and be useful, in practice any example that is redirected more than ten times is likely to be stuck in a loop. It is of course possible to incorporate more intelligent loop detection and a much higher maximum number of redirections, but it is not clear that this would be advantageous.

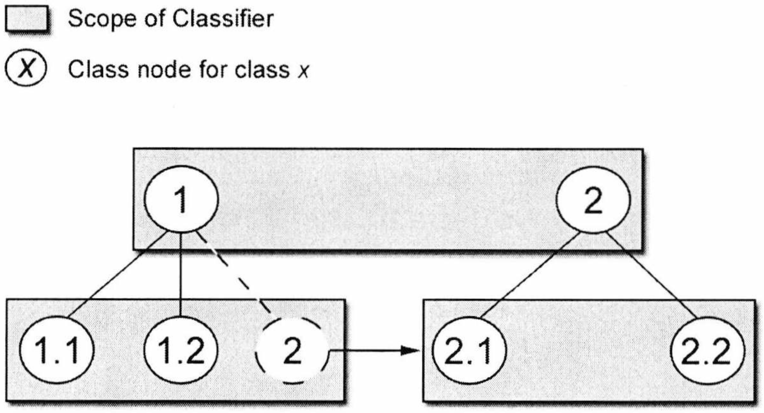


Figure 5.2: An example being redirected to the classifier discriminating between classes 2.1 and 2.2 by a classifier discriminating between classes 1.1, 1.2 and class 2.

Figure 5.2 shows an example of the recovery approach. The dashed line indicates that the classifier discriminating between classes 1.1 and 1.2 is also trained using examples misclassified as class 1 but belonging to class 2 (the dashed class 2 that appears as a child of class 1). In this way if an example is assigned to class 2 by the classifier discriminating between 1.1, 1.2 and 2, then it is sent to the classifier discriminating between classes 2.1 and 2.2 (denoted by the arrow).

Note that in Figure 5.2 the highest possible class is used to try and recover misclassifications, i.e., examples are redirected as class 2 rather than 2.1 and 2.2. This is intended as a way to simplify the problem for the classifier trying to recover the examples. It is more likely that the classifier will be successful when attempting to discriminate between three classes (1.1, 1.2 and 2) rather than four classes (1.1, 1.2, 2.1 and 2.2). Also a classifier has already been constructed to discriminate between the sibling classes of class node 2.

Having the recovery approach enabled for every classifier node in the TDDC tree is likely to produce suboptimal accuracy; the misclassified examples might reduce the accuracy of the local classifications significantly (in the validation or test sets) by making it harder to build an effective classifier. Discriminating just between classes 1.1 and 1.2 could yield a high accuracy, but when including class 2 this may drop as some examples from classes 1.1 and 1.2 can be misclassified as class 2. Indeed, by their nature misclassified examples are more likely to be difficult to classify. There is an obvious trade-off between the potential benefit of correcting the original misclassifications of examples that are recovered (e.g., belonging to class 2 that were originally misclassified as class 1) and the potential harm of misclassifying examples that belong to classes 1.1 and 1.2 as class 2.

It would be difficult to design an effective algorithm to sequentially decide if recovery should be enabled or disabled for each classifier node. This is because such a sequential approach could not take into account the accuracy of the entire classification tree, only one node at a time. For instance, consider an example belonging to class 2 that was misclassified as class 1 and then redirected to class 2. If the classifier discriminating between class 2.1 and 2.2 has recovery enabled it may classify the recovered example originally misclassified as class 1 to any of the extra recovery class nodes associated with it, including possibly class 1 (which would form a loop). Therefore, the decision as to whether to enable recovery for the classifier discriminating between class 1.1 and 1.2 is directly related to the decision as to whether to enable recovery for any classifier node redirecting examples to it, interacting with it directly or through intermediate classifier nodes. In this sense the nodes with recovery enabled form a type of network of interaction, with the possibility of having multiple non-interacting networks within each TDDC tree.

One method to find the optimal solution for the recovery problem (i.e., which nodes should have recovery enabled) for a given TDDC tree would firstly involve detecting the sets of interacting classifiers. Once the sets have been detected all possible combinations of states (where the states involve having recovery enabled or disabled at each classifier node) for each set should be assessed one by one to find the one that has maximal quality. Once this has been completed for each set of interacting classifier nodes, the solutions can be combined to produce the complete optimal solution for the TDDC tree with recovery. In the worst case scenario (where every classifier interacts with every other classifier) there are $2^{(n\text{-}1)}$ possible combinations where n is the number of classifier nodes (1 is taken away as it is not possible to enable recovery for the root classifier without generating a loop). The number of combinations will obviously be very large number for any reasonably sized data set and so it seems appropriate to use a heuristic search algorithm. Such an approach based on PSO/ACO (the PSO/ACO-RO algorithm) is discussed in the next subsection.

5.3.1. Deciding when to Recover from Parent Misclassifications with the PSO/ACO-RO (Recovery Optimisation) Algorithm

The problem of global recovery interactions can be reduced to a combinatorial optimisation problem involving a set of binary decisions. Therefore, we can also apply the hybrid PSO/ACO method to this optimisation problem. To apply the PSO/ACO method to this problem a particle (candidate solution) indicates which classifier nodes will have recovery enabled. More precisely, a particle is represented by a vector with n binary components, i.e.:

$$Particle\ = r_1, r_2, \ldots, r_n$$

Where n is the number of classifier nodes and d indexes the dimensions in the PSO/ACO's search space. Each r_d (d=1,...,n) can take the values of *on* or *off* depending on whether recovery should be enabled for the dth classifier node or not.

For this approach the training set is divided into a building set and a validation set, as explained earlier. To evaluate such a particle, the validation set would simply be classified by the modified TDDC tree – i.e., the TDDC tree constructed by using the "recovery approach" in the classifier nodes whose r_d flag was set to "*on*". The fitness of a particle is computed as the mean of the classification accuracies (on the validation set) for each class level.

It is also important to consider the way in which recovery has a downward influence in the TDDC tree during training. For instance, from the previous example shown in Figure 5.2, if the examples belonging to class 2 are first misclassified as class 1 and then misclassified again as, for instance, class 1.1, then the classifier node discriminating between 1.1.1, 1.1.2 and 2 can again try to correct the misclassification of these examples belonging to class 2. However, if recovery is turned off for the classifier node discriminating between class nodes 1.1 and 1.2 (previously the classifier node discriminating between class nodes 1.1, 1.2 and 2) the examples from class 2 that were misclassified as class 1 will not pass down to any descendant classifier nodes during training (this will not be the case during testing as the test example classes are unknown during classification). In this way the decision of whether to turn recovery on or off in a given classifier node will have an effect on its descendant classifier nodes. A full and rigorous investigation into whether this type of approach is more effective than always passing on misclassified examples is left for future research.

Note also that a particle represents a complete candidate solution to the problem of deciding which classifier nodes should use the recovery approach. This means that the evaluation of a particle is performed in a global fashion taking into account interaction between all classifier nodes. This procedure avoids the drawbacks of a greedy approach where the decision of tuning on/off recovery would be done sequentially for one classifier node at a time.

5.3.2. Combining Classifier Selection and Misclassification Recovery with PSO/ACO-CS-RO

Two new approaches have been discussed to try and improve classification accuracy in TDDC trees, namely PSO/ACO-RO and PSO/ACO-CS. As they work independently of each other, it is possible to combine them in an attempt to boost accuracy further, creating an extended swarm intelligence algorithm denoted as PSO/ACO-CS-RO.

In this case a particle consists of two sections:

$$Particle \;= w_1, w_2, \ldots, w_n, r_{n+1}, r_{n+2}, \ldots, r_{n+n}$$

Where n is the number of classifier nodes and d indexes the dimensions in the PSO/ACO-CS-RO search space. Each r_d (d=n+1,...,n+n) can take the values of *on* or *off* depending on whether recovery should be enabled for the d-nth classifier node or not. Each w_d (d=1,..,n) is the classifier selected at the dth classifier node in the TDDC tree. Each w_d can take one of the nominal (classifier ids) values c_1,..c_k, where k is the number of different candidate classifiers at each node. The particles used for this approach have a number of dimensions equal to twice the number of classifier nodes, two dimensions per node. Dimension w_d is used to select the classification algorithm used for the dth classifier node, and dimension r_d is used to decide whether misclassification recovery should be on or off for the (d - n)th classifier node.

In this chapter PSO/ACO-CS-RO will be compared with an approach that uses the greedy selective TDDC technique (section 3.7.2.1), the PSO/ACO-RO algorithm and the PSO/ACO-CS algorithm. However, it would be extremely computationally expensive to combine the use of the PSO/ACO-RO algorithm with the greedy approach for classifier selection in a tightly integrated approach. An example of such an approach would be where each particle would consist of $r_1, r_2, \ldots, r_n$ (indicating whether recovery is turned *on* or *off* for each classifier node) and the classification algorithm to be used at each classifier node would be selected by the greedy approach at each fitness evaluation. This would lead to a k times increase in computational time over the PSO/ACO-CS-RO algorithm discussed in this sub-section, where k is the number of different types of classifier. Instead, the classification algorithm is selected for each node of the TDDC tree using the greedy approach once, before the PSO/ACO-RO algorithm is run. Recovery is then optimised by the PSO/ACO-RO algorithm using this fixed selection of classification algorithms.

5.4. Experimental Setup

5.4.1. The Creation of the Bioinformatics Datasets

The hierarchical classification methods discussed in this chapter were evaluated in four challenging datasets involving the prediction of protein function. The protein functional classes to be predicted in these data sets are the functional classes of GPCRs (G-Protein-Coupled Receptors) as discussed in Section 2.6.2.

The GPCR functional classes are given unique hierarchical indexes by GPCRDB (Section 2.6.2). The GPCR class hierarchy originally had up to 5 class levels, but only 4 levels are used in the datasets created in this work, as the data in the 5th level is too sparse for training – i.e., in general there are too few examples of each class at the 5th level. In any case, it should be noted that predicting all the first four levels of GPCR's classes is already a challenging task. Indeed, most works on the classification of GPCRs limit the predictions to just one or two of the topmost class levels [15] [73] [92] [116].

The data sets used in our experiments were constructed from data in UniProt (discussed in 2.6.1) and GPCRDB. UniProt is a well known biological database, containing sequence data and a rich annotation about a large number of proteins. It also has cross-references for other major biological databases. It was extensively used in this work as a source of data for creating our data sets. Only the UniProtKB/Swiss-Prot was used as a data source, as it contains a higher quality, manually annotated set of proteins.

We performed experiments with four different kinds of predictor attributes, each of them representing a kind of "protein signature", or "motif", namely: FingerPrints from the Prints database, Prosite patterns, Pfam and Interpro entries (all discussed in Section 2.5). The four GPCR data sets each use predictor attributes from one of either the Prints, Prosite, Interpro or Pfam databases. They also contain two additional attributes, namely the protein's molecular weight and sequence length.

Any duplicate examples (proteins) in a data set are removed in a pre-processing step, i.e., before the hierarchical classification algorithm is run, to avoid redundancy. If there are fewer than 10 examples in any given class in the class tree that class is merged with its parent class. If the parent class is the root node, the entire small class is removed from the data set. This process ensures there is enough training and test data per class to carry out the experiments. (If a class had less than 10 examples, during the 10-fold cross-validation procedure there would be at least one iteration where there would be no example of that class in the test set).

After data pre-processing, the final datasets used in the experiments have the numbers of attributes, examples (proteins) and classes per level (expressed as level 1/ level 2/level 3/level 4) indicated in Table 5.1.

	GPCR/Prints	GPCR/Prosite	GPCR/Interpro	GPCR/Pfam
#Attributes	283	129	450	77
#Examples	5422	6261	7461	7077
#Classes	8/46/76/49	9/50/79/49	12/54/82/50	12/52/79/49

Table 5.1: Main characteristics of the datasets used in the experiments

5.4.2. Data Set Partitioning and Baseline Algorithms

Each data set was split into two main subsets at each iteration of the 10-fold cross validation process, one test set and one training set. The test set is used to assess the performance of the approach in question; therefore the true class of each test example remains unseen during the training process, only to be revealed to measure the predictive accuracy of the approach. The training set is split into a further two subsets. Firstly 75% of the training set was used as the building set; this building set is used to train the classifiers. Secondly the validation set, which consists of the remaining 25% of the training examples. The validation set is used to compute the quality of the classifiers, and so particle fitness in all variants of the PSO/ACO algorithm. After the best solution (according to accuracy in the validation set) has been found in a single PSO/ACO run, the classifiers at every classifier node specified in that best particle are trained using the entire training set. This procedure attempts to maximise the individual classifier's accuracy and so the final accuracy in the test set (unseen during the PSO/ACO run).

As a baseline it is important to evaluate the proposed method by comparing its predictive accuracy with the predictive accuracy of the greedy selective top-down approach. The baseline should also include each of the individual classification algorithms used in the greedy selective top-down approach. Therefore the first experiments are to build standard TDDC trees using one type of classification algorithm throughout.

The baseline classification algorithms used in the experiments presented in this chapter were implementations from the WEKA [164] package. These algorithms were chosen to include a diverse set of machine learning paradigms, while having high computational efficiency. The paradigms in question are rule induction, decision tree induction and Bayesian classification.

The five baseline algorithms used in the experiments were:

- HyperPipes is a simple algorithm that constructs a "hyperpipe" for every class in the data set; each hyperpipe contains each attribute-value found in the examples from the class it was built to cover. An example is classified by finding which hyperpipe covers it the best.
- NaiveBayes uses Bayes' theorem to predict which class an example most likely belongs to. It is naïve because it assumes attribute independence given the class.
- J48 is a decision tree algorithm, being WEKA's modified version of the very well known C4.5 algorithm.
- ConjunctiveRule is another simple algorithm that only produces two rules to classify the entire data set. A "default" rule is produced that predicts the class with the greatest numbers of records in the training set. The other rule is constructed using information gain to select attribute-values for the antecedent.
- BayesNet uses a Bayesian network to classify examples and can theoretically completely take into account attribute dependency.

Although some of these algorithms are clearly more advanced than the others, all were selected for some classifier nodes by the classifier selection method (greedy approach or PSO/ACO-CS) during training, confirming that all of them perform best in certain circumstances. All experiments were performed using 10-fold cross validation [164] with α (the constant used to either speed up or slow down particle convergence) set to 1 for the PSO/ACO algorithm.

The remainder of this chapter examines the effectiveness of the proposed PSO/ACO-CS (Section 5.5), the baseline recovery approach (Section 5.6.1) and PSO/ACO-RO approaches (Section 5.6.2).

5.5. Computational Results for Classifier Selection

The predictive accuracy for each method (the five baseline classifiers used throughout the TDDC tree, the greedy and PSO/ACO-CS approaches) are shown in Table 5.2 through Table 5.5 for each dataset. The values after the "±" symbol are standard deviations (calculated using the WEKA statistics classes). Table 5.2 through Table 5.5 are shown for the sake of completeness, but, to simplify the analysis we focus mainly on a summary of the results (reported in Table 5.6). Table 5.6 shows the summary of the number of cases where there is a statistically significant difference in the predictive accuracy of a classifier selection method and a baseline algorithm according to the WEKA corrected two-tailed student t-test (with a significance level 1%) [164]. Each cell of the last five columns show the number of times the labelled approach (Greedy or PSO/ACO-CS) significantly beats the corresponding baseline classification algorithm (HP – HyperPipes, NB – NaiveBayes, J4.8, CR – ConjunctiveRule, BN – BayesNet), in each data set across all four class levels. Note that in each cell (except the totals in the last two rows) the worst (best) possible result for a classifier selection approach is -4 (+4), since there are four class levels for each data set. Totals across all data sets are shown at the bottom of the table. In the total cells, the worst (best) possible result is -16 (+16), since the totals are calculated over four data sets.

Both the greedy and PSO/ACO-CS approaches were very successful in improving predictive accuracy with respect to four of the base classification algorithms (HP, NB, CR, BN), as shown by the totals in Table 5.6. These two approaches were less successful in improving accuracy with respect to J48, but even in this case the classifier selection approaches improved upon J48's accuracy several times, whilst never decreasing upon J48's accuracy. In this sense both the greedy and PSO/ACO-CS approaches are quite successful, often increasing and never decreasing predictive accuracy significantly below that of any base classifier.

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	90.76±0.34	76.79±0.55	49.99±1.1	75.42±2.11
NaiveBayes	87.74±0.71	72.72±1.11	41.3±0.99	63.85±1.89
J48	91.68±0.51	83.35±1.0	58.34±1.26	85.14±1.8
ConjunctiveRule	80.16±0.31	49.63±0.46	17.03±0.84	24.8±0.87
BayesNet	88.34±1.39	77.41±1.25	48.0±0.93	74.53±2.94
Greedy	91.68±0.51	83.06±0.88	58.21±1.23	84.66±2.09
PSO/ACO-CS	91.59±0.52	82.67±1.13	57.99±1.52	84.8±2.34

Table 5.2: Predictive accuracy (%) for each approach in the Prints data set.

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	83.74±1.14	73.77±1.01	48.21±0.95	82.62±2.5
NaiveBayes	87.88±0.59	74.78±0.78	38.59±1.07	51.25±1.85
J48	90.36±0.34	80.68±0.66	51.06±0.93	79.86±2.68
ConjunctiveRule	73.68±0.18	47.73±0.48	17.76±0.47	24.84±0.68
BayesNet	89.18±0.67	78.99±0.83	46.4±0.94	67.3±2.62
Greedy	90.36±0.34	80.41±0.81	54.36±1.33	83.58±2.46
PSO/ACO-CS	90.36±0.34	80.4±0.78	54.43±1.27	84.24±2.27

Table 5.3: Predictive accuracy (%) for each approach in the Interpro data set.

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	92.02±0.44	25.4±0.75	9.8±0.82	4.58±1.22
NaiveBayes	89.59±0.72	59.23±1.41	19.6±1.43	16.27±2.39
J48	92.98±0.48	70.77±1.39	37.03±1.07	48.97±3.98
ConjunctiveRule	75.55±0.13	51.4±0.53	13.49±2.0	6.97±4.63
BayesNet	90.35±1.1	62.7±1.45	23.25±1.46	23.43±2.42
Greedy	92.98±0.48	70.54±1.29	36.97±1.2	48.24±3.55
PSO/ACO-CS	92.98±0.48	70.5±1.35	36.97±1.21	48.5±3.58

Table 5.4: Predictive accuracy (%) for each approach in the Pfam data set.

TDDC Type	**Predictive accuracy at each level in the class hierarchy**			
	1st	**2nd**	**3rd**	**4th**
HyperPipes	82.14±0.71	46.03±1.28	23.1±1.62	32.16±2.82
NaiveBayes	85.34±1.14	60.63±1.25	24.86±1.3	23.94±2.11
J48	84.71±0.57	61.02±1.12	29.31±1.63	39.58±3.35
ConjunctiveRule	78.68±0.15	41.38±0.25	14.79±0.45	10.0±0.89
BayesNet	85.93±0.88	62.17±1.06	26.68±1.35	31.14±2.47
Greedy	85.93±0.88	62.54±0.91	31.46±1.25	40.73±4.21
PSO/ACO-CS	85.93±0.88	62.8±1.33	32.18±1.48	43.11±3.71

Table 5.5: Predictive accuracy (%) for each approach in the Prosite data set.

Dataset	**Classif. Selection Approach**	**Classification Algorithm**				
		HP	**NB**	**J48**	**CR**	**BN**
GPCR/Prints	Greedy	4	4	0	4	4
	PSO/ACO-CS	4	4	0	4	4
GPCR/InterPro	Greedy	3	4	1	4	4
	PSO/ACO-CS	3	4	2	4	4
GPCR/Pfam	Greedy	4	4	0	4	4
	PSO/ACO-CS	4	4	0	4	4
GPCR/Prosite	Greedy	4	2	1	4	2
	PSO/ACO-CS	4	3	3	4	2
Totals	**Greedy**	**15**	**14**	**2**	**16**	**14**
	PSO/ACO-CS	**15**	**15**	**5**	**16**	**14**

Table 5.6: Summation of the number of statistically significant results according to the Student T-Test

The PSO/ACO-CS approach significantly obtains a better result than the greedy approach in four cases overall, as follows. PSO/ACO-CS improves on the performance of J48 in five cases, three more than the greedy approach. These improvements are in the third and fourth level of the Prosite dataset and there is also an improvement in the InterPro dataset at the fourth level. As J48 is the hardest classification algorithm to beat, these results show the most difference. However, the PSO/ACO-CS algorithm also scores better against NaiveBayes when compared to the greedy approach in one case – in the Prosite dataset at the second class level.

The results imply that both the PSO/ACO-CS and greedy approaches benefit more from more "difficult" data sets. The data set in which the base classification algorithms perform worst is the Prosite data set. This data set also yields the biggest improvement in accuracies when using the greedy (1 significant win over J48), and more so the PSO/ACO-CS (3 significant wins over J48) approach. Indeed for either of these approaches to increase predictive accuracy above that of any base classifier, one of the base classifiers must make an error that is not made by another base classifier. In other words, if every base classifier was deterministic and the same then these approaches would never lead to any improvement. The more mistakes made by a certain classification algorithm (due to a more difficult data set) the higher the probability of another classification algorithm not making the same set of mistakes. Furthermore, it was observed that overfitting is sometimes a limiting factor with the PSO/ACO-CS approach, since increases in validation set accuracy (over the baseline classification algorithms) did not always result in a similar increase in test set accuracy.

5.6. Computational Results for Misclassification Recovery Approaches

5.6.1. Comparing Standard Top-Down Divide-and-Conquer Against the Basic (Always On) Recovery Approach

As discussed previously, the most basic recovery approach involves enabling recovery for every single classifier node, so that every node will always try and redirect misclassified examples to the correct classifier node. This is a relatively naïve approach and its effectiveness is analysed separate in this subsection mainly in order to provide a baseline for the more advanced recovery approach examined in section 5.6.2 (PSO/ACO-RO). The reason this basic recovery approach can be considered naïve is because it does not consider whether attempting to recover from misclassifications at a given classifier node will introduce more error than it corrects (error may be introduced by incorrectly redirecting previously correctly classified examples, as explained earlier). This basic recovery approach also does not consider the effect of attempting to recover the misclassified examples on the accuracy of the entire classifier tree.

Table 5.7 through Table 5.10 show the predictive accuracies achieved by each labelled approach in each dataset on a per class level basis. Note that the basic recovery approach can be instantiated in 7 different labelled approaches: 5 base classification algorithms, the PSO/ACO-CS approach and the greedy approach for classifier selection. The approaches without recovery enabled are included for comparative purposes. The shadings in the tables show whether the approach without recovery enabled performs significantly better (according to the WEKA corrected two-tailed Student's t-test with a significance level 1% [164]) than the corresponding approach with recovery always enabled. Dark grey indicates that the approach without recovery performs significantly better than its counterpart, light grey indicates that the approach without recovery performs significantly worse than its counterpart. For instance, in Table 5.7, there are two rows for the Hyperpipes algorithm. The first row (from the top of the table) with the label Hyperpipes shows the predictive accuracy obtained in the Prints dataset with the standard TDDC approach (using the HyperPipes classification algorithm). Therefore it includes the entry *No* in the *Recovery Enabled* column. The second row with the HyperPipes label is the entry for the approach utilising the recovery method and so includes the entry *Always* in the *Recovery Enabled.* The entry *Always* indicates that for *every* classifier node in the TDDC tree the algorithm always tries to recover misclassifications. The first HyperPipes row has two entries coloured dark grey. This indicates that the HyperPipes approach without recovery performs significantly better in the third and fourth class levels when compared to the HyperPipes approach with recovery always enabled. The corresponding shading is not included in second row for HyperPipes (or any of the other approaches with recovery always enabled) as it would simply be the opposite of the first row.

The greedy selective and PSO/ACO-CS approaches are also included in the comparisons in Table 5.7 through Table 5.11. As the PSO/ACO-CS approach allows the evaluation of the entire TDDC tree at once, it is possible to take into account the effect of having recovery enabled at each classifier node in the selection of the classifiers. This is due to way in which the PSO/ACO-CS approach defines a set of classifier types "all at once". As discussed previously the recovery method causes classifier nodes to interact through the redirection of misclassified examples. This means that it would impossible to evaluate this effect on a per classifier node basis (as is the case with the greedy selective approach). The overall effect of these redirections can only be taken into account if the TDDC tree is evaluated as a whole (as is the case with the PSO/ACO-CS approach). For this reason the greedy approach with recovery always enabled selects the types of classifier in the standard way (without considering the effect of misclassification recovery during the selection of each classifier). Only after the classifier type selections are made by the greedy approach is the flag at every classifier node set to having recovery enabled.

Table 5.11 shows a summary of the student's t-tests performed in all four data sets. Each cell shows the summation of the number of times the approaches with recovery always enabled significantly beat the corresponding approach without recovery (the standard TDDC approach) in each of the four datasets across the four class levels: That is, in each cell in the table, a score of +/-1 is added if the approach with recovery always on produced a significantly better or worse (respectively) result than having recovery always off for the algorithm in the cell's row, in the given data set of the cell's column. Note that (as expected for these baseline comparisons) in almost all cases the score is negative, indicating that having the recovery approach enabled for every classifier node produces a significantly lower accuracy than approaches without any recovery.

Approach	Recovery Enabled	Predictive accuracy at each level in the class hierarchy			
		1st	2nd	3rd	4th
HyperPipes	No	90.76±0.34	76.79±0.55	49.99±1.1	75.42±2.11
NaiveBayes	No	87.74±0.71	72.72±1.11	41.3±0.99	63.85±1.89
J48	No	91.68±0.51	83.35±1.0	58.34±1.26	85.14±1.8
ConjunctiveRule	No	80.16±0.31	49.63±0.46	17.03±0.84	24.8±0.87
BayesNet	No	88.34±1.39	77.41±1.25	48.0±0.93	74.53±2.94
Greedy	No	91.68±0.51	83.06±0.88	58.21±1.23	84.66±2.09
PSO/ACO-CS	No	91.59±0.52	82.67±1.13	57.99±1.52	84.8±2.34
HyperPipes	Always	89.86±0.95	77.54±1.03	48.48±1.07	72.49±2.64
NaiveBayes	Always	76.5±1.09	69.32±1.06	38.83±0.65	59.62±2.16
J48	Always	91.33±1.39	85.8±1.16	57.56±1.05	83.18±1.79
ConjunctiveRule	Always	47.66±1.42	47.25±0.71	14.8±0.66	11.37±0.4
BayesNet	Always	84.42±1.54	77.0±1.13	46.03±0.56	69.1±3.3
Greedy	Always	91.41±1.49	85.67±1.32	57.22±0.93	80.59±1.61
PSO/ACO-CS	Always	91.26±1.11	86.06±1.18	57.53±1.09	82.36±2.02

Table 5.7: Predictive accuracy (%) for each approach in the Prints data set.

Approach	Recovery Enabled	Predictive accuracy at each level in the class hierarchy			
		1st	2nd	3rd	4th
HyperPipes	No	83.74±1.14	73.77±1.01	48.21±0.95	82.62±2.5
NaiveBayes	No	87.88±0.59	74.78±0.78	38.59±1.07	51.25±1.85
J48	No	90.36±0.34	80.68±0.66	51.06±0.93	79.86±2.68
ConjunctiveRule	No	73.68±0.18	47.73±0.48	17.76±0.47	24.84±0.68
BayesNet	No	89.18±0.67	78.99±0.83	46.4±0.94	67.3±2.62
Greedy	No	90.36±0.34	80.41±0.81	54.36±1.33	83.58±2.46
PSO/ACO-CS	No	90.36±0.34	80.4±0.78	54.43±1.27	84.24±2.27
HyperPipes	Always	84.1±1.15	74.08±0.99	47.69±0.78	80.26±2.38
NaiveBayes	Always	82.12±1.35	71.53±1.13	37.71±0.57	48.34±2.29
J48	Always	79.49±0.92	73.81±1.08	41.79±0.84	68.19±2.47
ConjunctiveRule	Always	48.33±0.57	47.98±0.57	14.29±0.43	11.41±0.63
BayesNet	Always	86.66±0.44	76.54±0.66	45.26±1.01	61.35±1.78
Greedy	Always	90.47±0.75	83.67±0.78	52.2±1.45	78.83±2.28
PSO/ACO-CS	Always	90.54±0.71	83.86±0.71	52.2±1.47	80.07±2.99

Table 5.8: Predictive accuracy (%) for each approach in the Interpro data set

Approach	Recovery Enabled	Predictive accuracy at each level in the class hierarchy			
		1st	2nd	3rd	4th
HyperPipes	No	92.02±0.44	25.4±0.75	9.8±0.82	4.58±1.22
NaiveBayes	No	89.59±0.72	59.23±1.41	19.6±1.43	16.27±2.39
J48	No	92.98±0.48	70.77±1.39	37.03±1.07	48.97±3.98
ConjunctiveRule	No	75.55±0.13	51.4±0.53	13.49±2.0	6.97±4.63
BayesNet	No	90.35±1.1	62.7±1.45	23.25±1.46	23.43±2.42
Greedy	No	92.98±0.48	70.54±1.29	36.97±1.2	48.24±3.55
PSO/ACO-CS	No	92.98±0.48	70.5±1.35	36.97±1.21	48.5±3.58
HyperPipes	Always	92.27±0.3	25.35±0.66	7.99±0.68	1.72±0.89
NaiveBayes	Always	83.38±1.05	57.75±1.81	19.64±1.21	15.2±3.42
J48	Always	91.72±0.86	70.89±1.38	35.93±0.93	47.42±3.79
ConjunctiveRule	Always	63.37±1.17	48.58±0.58	15.97±0.67	10.28±0.85
BayesNet	Always	83.07±0.75	59.71±1.28	23.17±1.2	21.57±2.61
Greedy	Always	91.76±0.82	70.75±1.35	35.26±1.2	43.18±3.02
PSO/ACO-CS	Always	91.21±0.62	70.18±0.77	36.56±1.13	49.09±2.58

Table 5.9: Predictive accuracy (%) for each approach in the Pfam data set.

Approach	Recovery Enabled	Predictive accuracy at each level in the class hierarchy			
		1st	2nd	3rd	4th
HyperPipes	No	82.14±0.71	46.03±1.28	23.1±1.62	32.16±2.82
NaiveBayes	No	85.34±1.14	60.63±1.25	24.86±1.3	23.94±2.11
J48	No	84.71±0.57	61.02±1.12	29.31±1.63	39.58±3.35
ConjunctiveRule	No	78.68±0.15	41.38±0.25	14.79±0.45	10.0±0.89
BayesNet	No	85.93±0.88	62.17±1.06	26.68±1.35	31.14±2.47
Greedy	No	85.93±0.88	62.54±0.91	31.46±1.25	40.73±4.21
PSO/ACO-CS	No	85.93±0.88	62.8±1.33	32.18±1.48	43.11±3.71
HyperPipes	Always	82.05±0.74	45.46±1.41	22.13±1.53	30.26±2.86
NaiveBayes	Always	82.14±1.78	59.41±1.51	24.24±1.28	22.92±1.66
J48	Always	82.32±1.44	61.3±1.68	22.06±1.14	28.36±1.4
ConjunctiveRule	Always	54.41±28.03	33.02±11.74	12.12±3.43	9.93±0.71
BayesNet	Always	84.08±1.47	61.72±1.27	26.09±1.14	29.65±2.62
Greedy	Always	85.93±1.03	63.27±1.49	29.31±1.67	36.25±3.14
PSO/ACO-CS	Always	84.59±2.41	62.5±2.2	30.27±1.41	39.01±4.56

Table 5.10: Predictive accuracy (%) for each approach in the Prosite data set

Approach	Dataset				
	GPCR/ Prints	GPCR/ InterPro	GPCR/ Pfam	GPCR/ Prosite	Total
HyperPipes	-2	0	-2	-1	-5
NaiveBayes	-4	-2	-1	-2	-9
J48	1	-4	-1	-3	-7
Conjunctive Rule	-4	-3	-2	0	-9
BayesNet	-3	-4	-2	-2	-11
Greedy	0	-1	-3	-2	-6
PSO/ACO-CS	1	-1	-1	-1	-3

Table 5.11: Summation of the number of statistically significant results according to the Student's T-Test, for the recovery always on approach against the recovery always off approach, for each labelled approach

The PSO/ACO-CS approach with recovery enabled performs the best (in one case generating a positive result), due to the way in which it can tailor the selection of classifiers to the recovery scenario. It can tailor the selection of classifiers by determining (through fitness evaluations) which choice of classifiers will reduce the amount of error introduced by the recovery approach. E.g., it may be the case that having a J48 classifier at a particular classifier node decreases accuracy when compared to having a Conjunctive Rule classifier under normal TDDC circumstances; however, the opposite may be true when recovery is enabled.

There are some cases where enabling recovery for every classifier node is advantageous. These significant increases in accuracy can be seen in two of the datasets (Table 5.7 and Table 5.8) mainly for the PSO/ACO-CS and greedy selective approaches. These increases in accuracy are likely due to way in which PSO/ACO-CS and the greedy approaches use a selection of different classifiers. Different types of classifier are more likely to make different mistakes when classifying examples. For the types of approaches using the same type of classifier throughout the TDDC tree it is more likely that any attempt to recover misclassifications will lead to the same misclassifications being made again. Whereas, with TDDC trees with different types of classifiers (PSO/ACO-CS and greedy selective approaches) is it more likely that a classifier receiving misclassifications from a different type of classifier will be able to correct its mistake. Using the same classifier type to recover misclassified examples relies only on having a different distribution of data, whereas using a different type of classifier to recover misclassified examples relies on having a different data distribution and a different classification algorithm.

5.6.2. Comparing the Standard Top-Down Divide-and-Conquer Approach against the PSO/ACO-Optimised Recovery Approach (PSO/ACO-RO)

In this subsection the effect of using PSO/ACO for optimising recovery is examined. As discussed previously PSO/ACO can be used to decide whether recovery should be enabled or disabled at each classifier node. In this subsection this approach is combined with either the greedy selective approach or the PSO/ACO-CS approach (as discussed in Section 5.3.2). The two approaches examined in this section are: PSO/ACO-CS combined with PSO/ACO-RO (denoted as PSO/ACO-CS-RO) and the greedy selective approach combined with PSO/ACO-RO (denoted as Greedy-PSO/ACO-RO). The performance of these two approaches is compared against that of the five baseline classification algorithms. These baseline approaches use the standard TDDC approach without recovery enabled or optimised by PSO/ACO.

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	90.76±0.34	76.79±0.55	49.99±1.1	75.42±2.11
NaiveBayes	87.74±0.71	72.72±1.11	41.3±0.99	63.85±1.89
J48	91.68±0.51	83.35±1.0	58.34±1.26	85.14±1.8
ConjunctiveRule	80.16±0.31	49.63±0.46	17.03±0.84	24.8±0.87
BayesNet	88.34±1.39	77.41±1.25	48.0±0.93	74.53±2.94
Greedy-PSO/ACO-RO	91.44±1.06	83.48±1.21	58.31±1.51	84.46±2.38
PSO/ACO-CS-RO	91.52±0.44	83.98±1.65	58.5±1.34	84.4±1.59

Table 5.12: Predictive accuracy (%) for each approach in the Prints data set (recovery optimised by PSO/ACO)

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	83.74±1.14	73.77±1.01	48.21±0.95	82.62±2.5
NaiveBayes	87.88±0.59	74.78±0.78	38.59±1.07	51.25±1.85
J48	90.36±0.34	80.68±0.66	51.06±0.93	79.86±2.68
ConjunctiveRule	73.68±0.18	47.73±0.48	17.76±0.47	24.84±0.68
BayesNet	89.18±0.67	78.99±0.83	46.4±0.94	67.3±2.62
Greedy-PSO/ACO-RO	90.36±0.34	80.95±0.88	54.65±1.35	83.25±2.52
PSO/ACO-CS-RO	90.35±0.34	81.04±0.75	55.06±1.24	84.68±2.06

Table 5.13: Predictive accuracy (%) for each approach in the Interpro data set (recovery optimised by PSO/ACO)

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	92.02±0.44	25.4±0.75	9.8±0.82	4.58±1.22
NaiveBayes	89.59±0.72	59.23±1.41	19.6±1.43	16.27±2.39
J48	92.98±0.48	70.77±1.39	37.03±1.07	48.97±3.98
ConjunctiveRule	75.55±0.13	51.4±0.53	13.49±2.0	6.97±4.63
BayesNet	90.35±1.1	62.7±1.45	23.25±1.46	23.43±2.42
Greedy-PSO/ACO-RO	92.69±1.07	70.32±1.16	37.07±0.84	49.42±2.38
PSO/ACO-CS-RO	92.93±0.48	69.61±1.46	37.47±1.16	50.42±3.57

Table 5.14: Predictive accuracy (%) for each approach in the Pfam data set (recovery optimised by PSO/ACO)

TDDC Type	Predictive accuracy at each level in the class hierarchy			
	1st	2nd	3rd	4th
HyperPipes	82.14±0.71	46.03±1.28	23.1±1.62	32.16±2.82
NaiveBayes	85.34±1.14	60.63±1.25	24.86±1.3	23.94±2.11
J48	84.71±0.57	61.02±1.12	29.31±1.63	39.58±3.35
ConjunctiveRule	78.68±0.15	41.38±0.25	14.79±0.45	10.0±0.89
BayesNet	85.93±0.88	62.17±1.06	26.68±1.35	31.14±2.47
Greedy-PSO/ACO-RO	86.23±0.85	63.03±1.02	31.56±1.35	40.86±4.67
PSO/ACO-CS-RO	86.44±0.93	63.39±1.5	31.96±1.52	44.13±2.4

Table 5.15: Predictive accuracy (%) for each approach in the Prosite data set (recovery optimised by PSO/ACO)

Dataset	Classif. Selection Approach	Classification Algorithm				
		HP	NB	J48	CR	BN
GPCR/Prints	Greedy-PSO/ACO-RO	3	4	0	4	4
	PSO/ACO-CS-RO	4	4	0	4	4
GPCR/InterPro	Greedy-PSO/ACO-RO	3	4	2	4	4
	PSO/ACO-CS-RO	3	4	2	4	4
GPCR/Pfam	Greedy-PSO/ACO-RO	3	4	0	4	4
	PSO/ACO-CS-RO	4	4	0	4	4
GPCR/Prosite	Greedy-PSO/ACO-RO	4	4	3	4	3
	PSO/ACO-CS-RO	4	4	4	4	4
Totals	**Greedy-PSO/ACO-RO**	**13**	**16**	**5**	**16**	**15**
	PSO/ACO-CS-RO	**15**	**16**	**6**	**16**	**16**

Table 5.16: Summation of the number of statistically significant results (recovery optimised by PSO/ACO) according to the Student's T-Test

Table 5.12 through Table 5.15 show the predictive accuracies the labelled approaches attained in the four data sets used in this chapter. Table 5.16 shows the summation of the Student's t-tests in a similar way to the results included in Section 5.5 (Table 5.6). As per Section 5.5 results for each base classifier are shown for comparative purposes.

As can be seen from Table 5.16 both the Greedy-PSO/ACO-RO and PSO/ACO-CS-RO approaches increase classification accuracy beyond that of any base classifier type. These results are discussed further in the next subsection.

5.6.3. Discussion of the Effectiveness of the PSO/ACO-CS, Greedy Selective and PSO/ACO-RO Approaches

Both approaches with recovery optimised by PSO/ACO improve on the performance of the corresponding approaches without the recovery approach. This can be observed by comparing the summary of the results in Table 5.16 to the summation of the results in Table 5.6 (the table showing the summation of the results for the PSO/ACO-CS and greedy selective approaches). The totals from Table 5.6 and Table 5.16 are reproduced in Table 5.17 to allow easier comparisons.

Classif. Selection Approach	**Classification Algorithm**				
	HP	**NB**	**J48**	**CR**	**BN**
Greedy	15	14	2	16	14
PSO/ACO-CS	15	15	5	16	14
Greedy-PSO/ACO-RO	13	16	5	16	15
PSO/ACO-CS-RO	15	16	6	16	16

Table 5.17: Reproduction of the totals from Table **5.6** and Table **5.16**, in terms of numbers of significant wins over the baseline classification algorithms

As can be seen in Table 5.17 both PSO/ACO-CS and Greedy approaches benefit from being combined with PSO/ACO-RO. The increase in the number of significant wins when using Greedy-PSO/ACO-RO over the standard greedy selective are: 2 for Naïve Bayes, 3 for J48 and 1 for Bayesian Network (where J48 produces the most competitive classifiers). Compare these increases to PSO/ACO-CS-RO against PSO/ACO-CS: 1 for Naïve Bayes, 1 for J48 and 2 for Bayesian Network. When only considering improvements the Greedy-PSO/ACO-RO approach comes out on top. However, the law of diminishing returns should be considered, where the more competitive the approach the harder it is to increase on its accuracy. Indeed the PSO/ACO-CS approach is already quite effective – more so than the greedy selective approach. This makes it more difficult for the PSO/ACO-CS-RO approach to increase upon the accuracy of PSO/ACO-CS, when compared to the Greedy-PSO/ACO-RO and greedy selective approaches. Also the Greedy-PSO/ACO-RO approach actually performs worse than the greedy selective approach against Hyper Pipes (with Greedy-PSO/ACO-RO being significantly beaten by HyperPipes in two cases). This indicates that the PSO/ACO-CS-RO approach is more reliable when compared to the Greedy-PSO/ACO-RO approach – with PSO/ACO-CS-RO always at least equalling the number of significant wins of the PSO/ACO-CS approach.

Interestingly in a few cases the PSO/ACO-CS and greedy selective approaches with recover always enabled (also called basic recovery and examined in Section 5.6.1) seem to outperform the PSO/ACO-CS-RO and Greedy-PSO/ACO-RO approaches. For instance, in the Prints dataset, the greedy selective approach and PSO/ACO-CS with basic recovery seem to outperform the PSO/ACO-CS-RO and Greedy-PSO/ACO-RO approaches in the second class level.

TDDC Type	**Recovery Type**	**Predictive accuracy at each level in the class hierarchy**			
		1st	**2nd**	**3rd**	**4th**
Greedy	Always On	91.41±1.49	85.67±1.32	57.22±0.93	80.59±1.61
PSO/ACO-CS	Always On	91.26±1.11	86.06±1.18	57.53±1.09	82.36±2.02
Greedy-PSO/ACO-RO	PSO/ACO	91.44±1.06	83.48±1.21	58.31±1.51	84.46±2.38
PSO/ACO-CS-RO	PSO/ACO	91.52±0.44	83.98±1.65	58.5±1.34	84.4±1.59

Table 5.18: Comparing the predictive accuracy (%) of approaches using Recovery Optimisation against approaches not using Recovery Optimisation on the Prints dataset

Although the approaches with recovery always enabled outperform the PSO/ACO-RO based approaches at this one class level it should be noted that PSO/ACO-RO is optimising the mean predictive accuracy of the TDDC tree over all four class levels. The mean accuracy of the PSO/ACO-CS approach with recovery always enabled is 79.3% whilst the mean accuracy of the PSO/ACO-CS-RO approach is 79.6%. This demonstrates that, in this case, the PSO/ACO-RO has affected a slight increase in mean performance.

The same pattern of improvement can be seen with the PSO/ACO-CS-RO and Greedy-PSO/ACO-RO approaches when compared to the standard PSO/ACO-CS and greedy selective approaches respectively. That is, both recovery optimisation approaches are more successful in data sets that are more difficult. For instance in the Prosite dataset the PSO/ACO-CS-RO approach always significantly improves on the performance of every base classification algorithm in all levels.

	Prints	**Interpro**	**Pfam**	**Prosite**
Standard Deviation of Classif. Accuracy	5.42	5.21	12.13	3.38

Table 5.19: Standard deviations of base algorithm mean performance in each labelled dataset (not including ConjunctiveRule)

The performance of the approaches in the Prosite dataset set can be differentiated because the performance of each individual classifier is more closely matched. In most of the other datasets J48 is always the clear winner in terms of classification accuracy. When J48 struggles to perform optimally (as is the case with the Prosite data set) and the classifiers perform more similarly, the base classifier errors are more readily corrected by the approaches discussed in this chapter. This effect can be seen in Table 5.19 which shows the standard deviations of the performance of every base classifier type in each dataset (except for ConjunctiveRule which was omitted as it performed equally badly in every dataset). As can be seen in Table 5.19 Prosite clearly has the lowest standard deviation. This correlates to the higher relative score (PSO/ACO-CS-RO beats all base classifiers at every level in the Prosite dataset) of the more advanced approaches discussed in this chapter when compared to the base classifiers. Interpro has the next lowest standard deviation which correlates to the second highest relative score of the more advanced approaches against the base classifiers.

5.7. Summary

Our experiments show that both the greedy and PSO/ACO-CS approaches significantly improve predictive accuracy over the use of any single fixed algorithm throughout the classifier tree, in the majority of cases involving our four protein data sets. Overall, the PSO/ACO-CS approach was somewhat more successful (significantly better in four cases) than the greedy approach.

PSO/ACO-CS-RO performs the best out of all approaches. In one dataset PSO/ACO-CS-RO always significantly outperforms all other base classifiers. Indeed, one of the main advantages of the approaches described in this chapter is their robustness; they never significantly lose to any base classifier.

Computational time is an issue, with the PSO/ACO-RO based approaches taking large amounts of computational time when compared to any of the other approaches. This is due to the way in which component classifiers have to be retrained at many of the function evaluations when using PSO/ACO-RO (as the building set changes when recovery is enabled or disabled for a given classifier node). This is dissimilar to the PSO/ACO-CS approach, as the classifiers need only be trained once during a single run (as the building set never changes during a single run). A typical single run of the PSO/ACO-CS algorithm takes up to 1 hour on a Pentium 4 3.2 GHz machine (for the largest data set), whereas a run involving PSO/ACO-RO can take up to 24 hours. This means that a complete PSO/ACO-RO experiment involving 10 folds of a cross validation procedure can take up to 10 days.

In terms of computational efficiency the greedy approach is the most efficient out of the more advanced approaches examined in this chapter. However, the only extra computational time spent when using PSO/ACO-CS (when compared to the greedy approach) is spent classifying the examples belonging to the validation set. Recall that when using the greedy approach the validation set is classified *k* times, where *k* is the number of possible types of classifiers. The PSO/ACO-CS approach requires that the validation set be classified a number of times equal to the number of function evaluations made by the PSO/ACO search algorithm.

Overall we believe that the use of the more advanced approaches discussed in this chapter is more beneficial in more difficult data sets, where classification algorithms are more likely to make mistakes. Estimating a priori how likely a classification algorithm is to make a mistake is an open problem and this topic is left for future research. However, in these experiments, the smaller the standard deviation of the means of the predictive accuracies obtained by each component classifier, the better the proposed approaches performed when compared to the base classifiers.

Chapter 6. Hierarchical Ensembles of Hierarchical Rule Sets (HEHRS)

6.1. Introduction

In this chapter we propose novel ensemble-based data mining methods tailored to the hierarchical classification problem and apply them to six protein data sets. The datasets examined in this chapter, as with many other bioinformatics datasets, pose a significant problem for any classification technique as they have a relatively large number of attributes, in this case ranging from 126 to 708 attributes. The large number of attributes increases the search space for the classification algorithm which may often lead to sub-optimal performance. Another challenge associated with these data sets is that they involve a large number of classes – ranging from 179 to 351 arranged in four hierarchical class levels.

Although some research has applied ensemble methods to protein data sets [27] [76] [153] and to hierarchical data sets [56], previous research concentrates on "classical" ensemble techniques such as bagging, or ignores any class hierarchy present. Some work has been conducted in the field of hierarchical multi-label protein function prediction [18] [33] but their approaches rely on modifying the base classification algorithm, rather than using ensemble techniques.

The remainder of this chapter is organised in the following way. Section 6.2 gives an overview of the HEHRS approach and describes how the ensembles of rule sets are generated during training. Sections 6.3 and 6.4 describe the two ways in which the predictions from the component rules are combined (using voting and stacking respectively). Section 6.4 proposes a new baseline approach (Rule Based Extended

Multiplicative Method) which we have adapted for hierarchical classification. Section 6.6 describes the data sets and how they were created. Section 6.7 discusses the computational result. Finally, section 6.8 gives a summary of our findings.

The work presented in this chapter has partially been reported in a journal paper [82].

6.2. Building a Hierarchical Ensemble of Hierarchical Rule Sets (HEHRS)

6.2.1. Overview

In essence, the proposed ensemble method for hierarchical classification can be considered a new variation of bagging adapted to the hierarchical classification problem. In this method, an ensemble of rules is created by varying the sets of positive and negative examples *according to the class hierarchy*. In order to classify test examples, the predictions made by the rules are combined by using either voting or stacking. Such a method should improve the accuracy beyond the use of a non-ensemble based technique as the errors in each model can be, to some extent, mitigated by combing the predictions made by multiple models. As the bioinformatics data sets examined in this chapter make it more difficult to induce accurate models – because of the high number of classes, attributes and the sparseness of the data at lower levels – the potential benefits from using such an error correcting technique become greater.

Let us first describe the basic idea of the proposed method at a high level of abstraction. Recall that in the standard top-down hierarchical classification approach a rule set is built to distinguish between a set of sibling class nodes, using the training examples belonging to those sibling class nodes. By contrast, in the proposed HEHRS method *K* rule sets will be built for each set of sibling nodes in the class tree, where *K* is the number of class levels between the current level (inclusive) and the deepest class level (inclusive) which is a descendant from either of the current class nodes. For instance for a non-leaf node in level 2 of the class tree and a class tree with 4 levels (not counting the root node which is at the 0th level), K=3 rule sets will be generated, namely one rule set for each of the class levels 2, 3 and 4. All these rule sets contain rules predicting classes at the second level of the class tree, but they are called here hierarchical rule sets because they have been produced from examples at different levels of the class tree. In addition, to continue with this example, these three rule sets also form an ensemble of rule sets, and the proposed method builds several ensembles like this, at different levels of the class tree. Therefore, the ensembles also form a hierarchy, namely a hierarchy of ensembles, where each ensemble consists of a hierarchical rule set. Hence, this approach is here called Hierarchical Ensemble of Hierarchical Rule Sets (HEHRS).

6.2.2. Technical Details of the HEHRS Method

Let us now describe HEHRS in more detail, starting with notation issues. In general an ensemble of rule sets created for a given set of sibling class nodes is denoted as $E_{s,}$ where S is a set of sibling class nodes. A rule set within this ensemble (one of the K rule sets) is denoted by the letter i, where i corresponds to the level at which the rule set is built. Therefore any rule set belonging to E_s at the level i, used to distinguish between a set of sibling class nodes S, is denoted by E_{si}. Note that each rule in E_{si} will predict one of the classes in S, so there will be one or more rules (i.e. a subset of the rules in E_{si}) predicting each class in S. A set of rules in E_{si} built at level i predicting a single class d in S is denoted as E_{sid}. As discussed previously E_s consists of K rule sets, each containing rules produced from a different level of the class tree. For each level i and for each class c which is a descendant class of d, the rule induction algorithm will discover rules predicting class d, using as positive examples the examples having class c, and using as negative examples the examples having any class different from c at level i that is a descendant of d in the class tree. These concepts are illustrated in Figure 6.1. Note that Figure 6.1 refers to a hierarchy of rule sets, rather than the class hierarchy. Hence, each node (*d*) in Figure 6.1 denotes an ensemble of rule sets for a given class (as explained next).

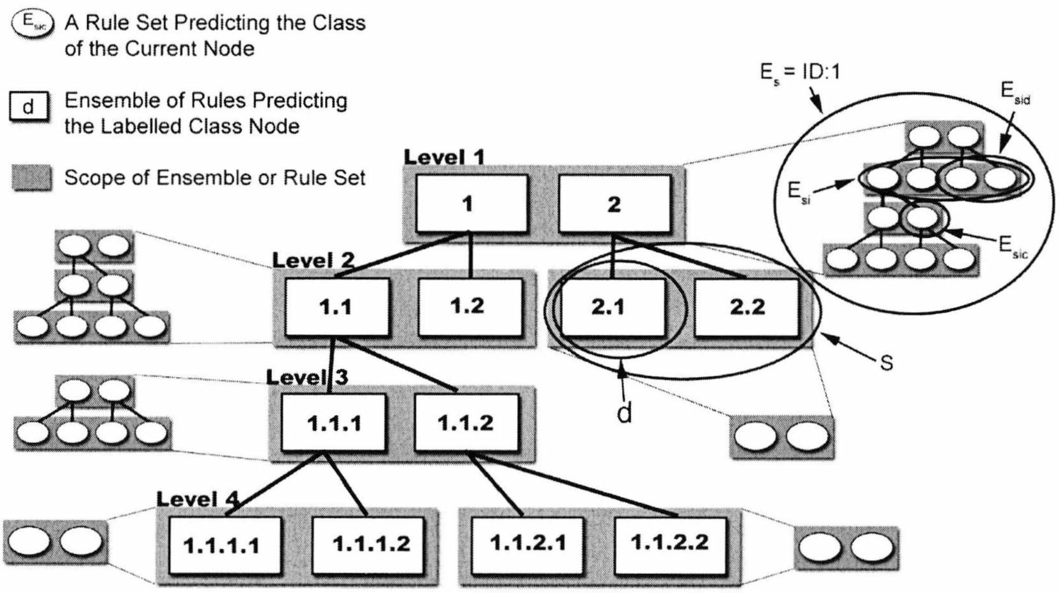

A Rule Set Predicting the Class of the Current Node
Ensemble of Rules Predicting the Labelled Class Node
Scope of Ensemble or Rule Set
Level 1
1
2
Level 2
1.1
1.2
2.1
2.2
Level 3
1.1.1
1.1.2
Level 4
1.1.1.1
1.1.1.2
1.1.2.1
1.1.2.2
d
S
Es = ID:1
Esid
Esi
Esic

Figure 6.1: Hierarchical Ensemble of Hierarchical Rule Sets (HEHRS)

In Figure 6.1 the grey boxes represent the scope of the classification performed by a given rule set (E_{si}) or ensemble (E_s). In the case of E_s the scope of the classification involves the sibling classes *S*. The main tree – i.e, the large tree at the centre of Figure 6.1 – shows the hierarchy of ensembles in a standard top-down approach. The expanded (smaller) trees show the rule sets (E_{si}) generated by HEHRS. For each set of sibling classes (*S*) in the main tree, there is a hierarchy of rule sets in the corresponding smaller tree, indicated by the presence of several grey boxes in the smaller tree. The label *S* in Figure 6.1 shows an example set of sibling classes (2.1 and 2.2) predicted by an ensemble, and the label *d* within *S* shows one of the classes (2.1) in the set *S*.

Table 6.1 shows in detail the variation in the sets of examples used at different class levels when inducing classification rules for HEHRS, with respect to Figure 6.1. For example, let us consider the construction of the ensemble of rule sets E_s labelled ID:1 in the top-right part of Figure 6.1. This ensemble will consist of rules predicting either class 1 or class 2, i.e., the set of sibling classes $S = \{1, 2\}$. So, the variable d, indicating the class to be predicted by a rule in E_s, will take on the value 1 or 2. This ensemble E_s will consist of four rule sets, each of them denoted E_{si}, i=1,...,4, where the i-th rule set is constructed from examples in the i-th level of the class tree.

Level (*i*)	**Class (*d*) in set of sibling classes *S***	
	$d = 1$	$d = 2$
1	1	2
2	1.1, 1.2	2.1, 2.2
3	1.1.1, 1.1.2	NA
4	1.1.1.1, 1.1.1.2, 1.1.2.1, 1.1.2.2	NA

Table 6.1: Values (Classes) taken by variable c at each level i used to construct the Rule Sets in E_s ID:1, in Figure **6.1**

As can be seen in Table 6.1 at the first level i is set to 1. The rule induction algorithm is given the training set with examples belonging to classes (c) 1 and 2, it then returns a rule set predicting classes (d) 1 and 2 for the first rule set $E_{si.}$. S={1,2}, i=1. At the second level i is set to 2. The rule induction algorithm is given the training set with examples belonging to classes (c) 1.1, 1.2, 2.1 and 2.2 (descendants of the classes in S). It then returns a rule set with rules discriminating between these classes. The rules predicting classes 1.1 and 1.2 (E_{sid} where i=2 and d=1) have their consequent changed to predict class (d) 1. The rules predicting classes 2.1 and 2.2 (E_{sid} where i=2 and d=2) are changed to predict class (d) 2 and are added, with the other rules now predicting class 1, to the second rule set $E_{si.,}$ S={1,2}, i=2. At the third level i is set to 3. As there are no third level descendant classes of class 2 (in the right-hand side of Table 6.1 the term "NA" means "not applicable") only rules predicting class 1 will be contained in this E_{si}. The rule induction algorithm is given a training set containing examples belonging to classes (c) 1.1.1 and 1.1.2. The rules predicting the classes 1.1.1 and 1.1.2 have their consequent class changed to predict class (*d*) 1. They are then added to E_{sid} where *i*=3 and *d*=1, which in this case is equal to E_{si} where *i*=3. An analogous procedure (as described in Table 6.1) is performed at level 4 ($i = 4$) where again, because this is quite an unbalanced class tree, there are only rules predicting class (*d*) 1 in the rule set E_{si} where *i*=4.

Note also that when the level *i* is set to 2 and the class (*d*) being predicted by the ensemble is 1, the rule induction algorithm produces a rule set discriminating between classes (*c*) 1.1 and 1.2 (along with 2.1 and 2.2), which at first glance seems counter-intuitive – as they are both descendants of class 1, the class (*d*) being predicted. The reason for this is to try and encourage diversity in the rules generated. As the rule induction algorithm is unaware of the hierarchical relationships between classes, the algorithm could produce the same (or very similar) rule sets for, say, the following two classification scenarios: (a) class 1 vs. class 2; and (b) class 1.1 vs. other non-descendant classes of class 1 (i.e., classes 2.1 and 2.2). Although it is still possible that the same or very similar rules will be generated between different levels and classes even when using the method described in this section, the probability (dependant on the make-up of the training set) of this happening is smaller when including the examples belonging to sibling classes as negative examples when inducing rules. Recall that to make an effective ensemble it is very important that the component classifiers be diverse, even if at the expense of some accuracy [9] [24] [144].

This section described how the ensemble of rules produced by HEHRS is built during the training phase of the algorithm. The next two sections describe two different approaches to combine the predictions of the ensemble of rules during the testing phase, namely an approach based on voting and one based on stacking, respectively.

6.3. Combining the Predictions from the Multiple Rules in HEHRS using Voting

6.3.1. Weighted Voting for HEHRS

After the entire hierarchical ensemble of hierarchical rule sets has been induced in the training phase, all the induced rules can be used to predict the class of a new example in the test set. In this testing phase, in order to combine the predictions of the rules in the hierarchical ensemble into a single predicted class at each level of the class tree for a given test example, each rule in the ensemble is assigned a weight. That is, for each ensemble of rule sets E_s, each rule in E_s is assigned a weight.

The weight of a rule is a measure of its classification accuracy, computed on the training set. When the class predicted by a rule is the majority class (a class having more examples than the rest of the training set combined) its weight is computed by the product of the rule's sensitivity and specificity [74], as shown in Equation 6.1, where *TP*, *FN*, *FP* and *TN* are, respectively, the number of true positives, false negatives, false positives and true negatives associated with the rule [164].

When a rule predicts a minority class (i.e., any class different from the majority class) the precision [74], shown in Equation 6.2, is used as the rule's weight. This approach, based on measuring rule quality either as the product of sensitivity and specificity or as precision, depending on the relative frequency of the class predicted by the rule, is an attempt to get a more "balanced" weight in extreme cases, as follows.

When there are a small number of examples in the class being predicted, when compared to the overall size of the training set, then the way in which specificity accounts for the number of false positives becomes problematic. This is because sensitivity multiplied by specificity weights the sensitivity (TP / (TP + FN)) and the specificity (TN / (TN + FP)) equally, ignoring the actual number of true positives and false positives. Therefore, in the case where the minority class is being predicted, it is possible to obtain a good rule quality even though the ratio of TP / (TP + FP) – i.e. the precision – is bad. Such a situation will likely produce a low accuracy as although a high sensitivity and relatively high specificity may be obtained, many examples may be misclassified as this minority class (due to the absolute number of false positives). The opposite is true for the majority class; the absolute number of false positives becomes less important for producing good accuracy as the number of true positives will likely be much higher. Sensitivity multiplied by specificity increases the importance of obtaining a low number of false positives when compared to precision. This is because it is more useful to consider the ratio *TN* / (*TN* + *FP*), rather than ratio of the large number of true positives to the low possible number of false positives (as it is implicitly the case with precision). For further discussions see [80].

$$\text{Sensitivity} \times \text{Specificity} = (TP\ /\ (TP + FN)) \times (TN\ /\ (TN + FP))$$

Equation 6.1: Rule Weight (Majority Class)

$$\text{Precision} = TP\ /\ (TP + FP)$$

Equation 6.2: Rule Weight (Minority Class)

The testing phase can be seen in Pseudocode 6.1. A test example is classified in a top-down fashion, as follows. Let *S* be the set of sibling classes out of which one class must be assigned to the example. Initially, *S* contains the set of classes in the first class level. For each class *d* in S, the weight of class *d* is given by the summation of the weights of all the rules in the ensemble of rule sets E_s that cover the test example and predict class *d*. The class with the greatest weight is assigned to the test example at the first level. Next the example is pushed down to the second level, where the set *S* is updated to contain the child classes of the class assigned to the example in the first level – the ensemble E_s is also updated accordingly. Again, for each class *d* in the current *S* the weight of class *d* is computed – adding the weights of all rules in the current E_s that cover the test example and predict class d – and the class with the greatest weight is assigned to the test example at the current (second) level, and so on. This process is repeated until the test example reaches a leaf node in the class tree.

```
FOR each example in test set
  S = The set of classes in first class level
  d = No class
  /* S will not contain classes when d is a leaf node */
  WHILE S contains classes
    FOR each rule set E_si in E_s for this set of sibling classes S
      Find rule from rule set in E_si that covers example
      Record which class (from S) the rule predicts and what
      weight is associated with the rule
    END FOR
    d = the class from S that has the highest accumulative weight
    S = the set of d's child classes
  END WHILE
  Classify example as having class d
END FOR
```

Pseudocode 6.1: The testing phase of Weighted Voting for HEHRS

To allow the user to interpret a prediction made by HEHRS for a given example a simple procedure can be implemented. After the example has been fully classified to the leaf level it is possible to examine all the rules that covered it. All the rules that have consequent classes that are parents of the final leaf classification can be used to present an overview of the classification process to the user. However, the issue of such interpretation by the user is out of the scope of this thesis.

6.3.2. Optimising HEHRS' Rule Weights with PSO

As computed by Equation 6.1 and Equation 6.2, the weight of a rule in HEHRS depends only on the predictive accuracy of that individual rule, and it does not take into account the complex interactions of the rules in an ensemble. It is possible to optimise the set of rule weights by taking rule interaction into account, by defining two elements:

(a) An evaluation function that measures the quality of a candidate set of rule weight values. The evaluation function to be maximised is the normalised total number of correct predictions made at each internal (non-leaf) class node and each leaf class node. This evaluation function is computed on the training set.

(b) An optimization method, which searches for the optimal set of rule weight values in the space of candidate weight values. In this work we use, as an optimization method, a Particle Swarm Optimization (PSO) algorithm.

Recall that PSO is a meta-heuristics that maintains a population of particles – each of them a candidate solution to the target problem – that iteratively move around the search space [95]. The position of a particle in the search space represents the contents of its candidate solution, and so moving the particles correspond to generating new candidate solutions. In this work, each particle's position corresponds to a set of rule weight values for the HEHRS method. Each particle is initialised with randomly deviating (±1) position generated from the rule weight equations and random velocity. The value ±1 is used to ensure that there is an even spread of initial positions over the range of the evaluation function ([0, 1]) with up to ±1 extra to stimulate exploration. The rest of the algorithm uses the standard PSO methodology (see Section 3.4).

The main motivations for using PSO is that it performs a global search (rather than the greedy search performed by local search algorithms), and has been empirically shown to be a powerful optimizer, often outperforming more traditional population-based optimizers such as evolutionary algorithms (EAs) [94], [107]. In any case, we do not claim that PSO is the "optimal" algorithm for our rule weight optimization problem. It produced very good results – as will be shown later – but it is possible that other global search optimization methods such as EAs would produce a similarly good result. The issue of comparing PSO and EAs is out of the scope of this thesis, and is left for future research.

Two versions of the PSO for rule weight optimization are proposed in this work, one where *negative weight values are allowed* and another one where *they are not*. In the former case, if a rule is extremely unreliable it may be assigned a negative weight, detracting from the class predicted by that rule. An example of where a negative value may be appropriate for a rule is where that rule covers more examples of other classes than its own consequent class and so, in fact, signals that other classes are more likely. In the version where negative values are not allowed, the lowest possible rule weight is 0, where a rule will not have any influence in the classification of a test example.

In some cases it does not matter what the weights associated with certain rules are during the training phase. For instance, if all examples are always correctly classified by all rules, then as long as the weights are all positive it does not matter what the weight values are. This can cause a problem, as even though all examples are correctly classified by all rules during the training phase they may not be during the testing phase. Therefore, during the testing phase the exact weights may become important. To combat this situation it is detected whether any rules do not take part in any contentions (where two or more rules predict different classes for any given example) during the training phase. If they do *not* they will *not* have their weights optimised by the PSO algorithm and default to the normal rule weights. Such contentions (or lack of) can be detected by assigning a flag to each rule (with a default value of *off*); the sets of rules covering each example can then be examined. If any set of rules contain rules covering a given training example with different consequent classes then the contention flag is set to *on* for those rules, meaning that the weight for those rules should be optimised. The rules left with a flag of *off* should not have their weight optimised.

The two main elements of the proposed PSO for rule weight optimization are the particle representation and the fitness function. The particle representation consists of a vector with n components, each of them denoted w_i, $i = 1,\ldots,n$, where w_i is the weight associated with i-th rule and n is the total number of rules. That is:

$$Particle \; = w_1, w_2, \ldots, w_n$$

The fitness function measures the quality of a particle, i.e., the quality of a candidate set of rule weights. In order to compute the fitness of a particle, for each example in the training set, the system extracts the rule weights from the particle and uses those weights to decide which class will be assigned to the example. This decision is made by computing, for each class, the total weight of rules that cover the example and have that class, as discussed earlier. The class chosen to be assigned to the example is the class with the largest total weight. After every training example has been completely classified (i.e., assigned a class at a leaf node in the class tree), the value of the fitness function for the current particle is the classification accuracy on the training set. This is the average accuracy across all four class levels.

Note that, ideally, the fitness function should be based on the classification accuracy on a hold out set, i.e. the original training set should be divided into a building set (used to build the rules) and a validation, hold out set, used to compute the classification accuracy to be used as the fitness of a particle. This would have the advantage of avoiding overfitting of the rule weights optimised by the PSO to the training set. However, it was not feasible to use such a hold out set in our experiments, due to the sparseness of data at lower levels of the class tree. It would be impossible to induce rules for some classes if examples from the training set were reserved for a hold out set. We consider the benefits of creating rules for all classes outweigh the problems due to possible overfitting from the lack of a hold out set.

6.4. Stacking for the Hierarchical Ensemble of Hierarchical Rule Sets (SHEHRS)

It is reasonable to think that out of all the rules built by the HEHRS method some of them will be redundant or detrimental to the overall performance of the system. The PSO-based HEHRS approaches should ideally consider these cases by setting the weights of the low quality rules to 0 (or negative values) and so minimise the influence of these detrimental rules. The SHEHRS (Stacking for the Hierarchical Ensemble of Hierarchical Rule Sets) method attempts to address these same issues in a more effective way by creating meta-rules [159]. Also, as a stacking method (see section 3.3 for a discussion of stacking), the SHEHRS method should be able to learn from the fact that some classifiers often misclassify some examples.

The meta-rules generated by SHEHRS are not derived from the features (attributes) in each original (base level) example, but instead are learnt from the decision making process of the original rules. In order to build these meta-rules, we first create a meta-data set where each meta-example corresponds to an original example. Each meta-example is described by a set of binary meta-features, each of which indicates whether or not the original example is covered by a given original (level-0) rule. More precisely, each meta-example consists of a vector with (n+1) components, as follows:

$$Meta\text{-}Example_z = R_{z1}, R_{z2}, ..., R_{zn}, C_z$$

Meta-example$_z$ is the zth meta-example in the meta-dataset, R_{zj}, j = 1,..,n (where n is the number of level-0 original rules), is a binary meta-variable taking in the value "yes" or "no" to indicate whether or not the jth rule covers the zth original example. C_z is the class of the original zth example.

```
FOR each example in training set
  S = The set of classes in first class level
  d = No class
  /* S will not contain classes when d is a leaf node */
  WHILE S contains classes
    /* Create the meta-data set If necessary */
    IF meta-data set for S does not exist
      Create a meta-data set, where each meta-feature corresponds
      to a rule in $E_s$
    FOR each rule set $E_{si}$ in $E_s$ for this set of sibling classes S
      Find rule from rule set $E_{si}$ that covers example and record
      it
    END FOR
    ADD new meta-example to meta-data set for S. Where meta-
    example's class is the true class of example (at the level of
    the classes in S) and all of its feature-values are "no"
    except for those corresponding to the rules that covered the
    example from the previous FOR loop - which are set to "yes"
    d = The true class of example (at the level of the classes in
       S)
    S = The set of d's child classes
  END WHILE
END FOR
```

Pseudocode 6.2: The training phase of SHEHRS

Recall that, during the rule discovery process of the HEHRS method, each rule in E_{si} is constructed to predict a single class from the set of sibling classes S at class level i. Note that it is clear which meta-features correspond to which level in the class hierarchy, or which rules are used to predict the classes in which level for a given original example. Hence (as shown in Pseudocode 6.2), in order to predict the classes in S we construct a meta-data set having only meta-features corresponding to the discovered rules in E_s. Note that this means we create only one meta-data set for each ensemble of hierarchical rule sets E_s. We now have a new secondary hierarchical classification problem, so it would be possible (though not necessarily useful) to recursively apply an extended version of this approach.

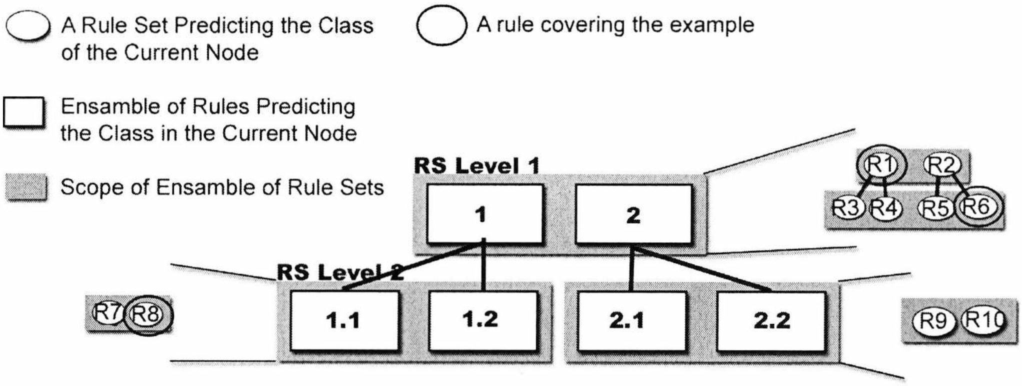

A Rule Set Predicting the Class of the Current Node
A rule covering the example
Ensamble of Rules Predicting the Class in the Current Node
Scope of Ensamble of Rule Sets
RS Level 1
1
2
RS Level 2
1.1
1.2
2.1
2.2
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10

Figure 6.2: An Example Classification in SHEHRS

To illustrate how SHEHRS uses the result of HEHRS, Figure 6.2 shows the ensembles of rules produced by HEHRS in a simple two level class hierarchy. In Figure 6.2, each of the ellipsis R1,…,R10 denotes a single classification rule. (Note that this notation is being used in the example of Figure 6.2 purely to keep the example and the figure as simple as possible, since in general in HEHRS each of those ellipses would represent a set of rules, as in the notation used in Figure 6.1.) Figure 6.2 also shows the rules that are covering a given training example, which are the rules R1, R6 and R8. Given the result of HEHRS shown in Figure 6.2, and the fact that the given training example has class 1.2, SHEHRS generates, during its training phase, the meta-examples shown in Table 6.2 and Table 6.3.

Meta-Features						**Actual Class**
R1	**R2**	**R3**	**R4**	**R5**	**R6**	1
yes	no	no	no	no	yes	

Table 6.2: Meta-example generated by SHEHRS for the set of sibling classes S = {1, 2}, at level 1 in the class hierarchy

Meta-Features		**Actual Class**
R7	**R8**	1.2
no	yes	

Table 6.3: Meta-example generated by SHEHRS for the set of sibling classes S = {1.1, 1.2}, at level 2 in the class hierarchy

For the sake of simplicity Table 6.2 and Table 6.3 show only one meta-example each. Of course, the meta-data set will contain a meta-example for each original (base level) example of the classes to be predicted. A separate meta-data set is generated at each set of sibling classes *S*. Table 6.2 shows the meta-example generated at level 1. The only meta-attributes with value *yes* for this meta-example are the ones referring to rules R1 and R6, since these are the only rules covering the original example at level 1.

Note that the class predicted by SHEHRS' level-1 classifier for the meta-example at class level 1 is not used to decide how the meta-example travels through the HEHRS tree when building the meta-dataset for class level 2; only the actual (true) class of the example does. Hence, Table 6.3 shows the meta-example generated at class level 2.

As shown in Pseudocode 6.3 – when an example of unknown class needs to be classified it must first be converted into a meta-example based on which rules cover it from the original ensemble of rule sets (E_s) produced to decide which class the example is assigned to at each divide. Once this conversion is done, the classification is performed using the standard top down hierarchical classification approach, using the meta-rules discovered in the training phase.

```
FOR each example in test set
  S = The set of classes in first class level
  d = No class
  /* S will not contain classes when d is a leaf node */
  WHILE S contains classes
    /* Creating the meta-example */
    FOR each rule set $E_{si}$ in $E_s$ for this set of sibling classes S
      Find rule from rule set $E_{si}$ that covers example and record
      it
    END FOR
    meta-example = New example where all feature-values are "no"
    except for those corresponding to the rules that covered the
    example from the previous FOR loop - which are set to "yes"
    /* Classifying the new meta-example */
    d = Classify meta-example using the meta-classifier (rule
        set) constructed from the meta-data set associated with
        S (constructed during the training phase) and return the
        predicted class label
    S = The set of d's child classes
  END WHILE
  Classify example as having class d
END FOR
```

Pseudocode 6.3: The testing phase of SHEHRS

Note that in the version of SHEHRS described so far each meta-feature is a binary one, indicating only whether or not the example is covered by the corresponding rule. Hereafter this version of SHEHRS will be referred to as Cov-SHEHRS (rule Coverage-based SHEHRS). An alternative to this approach is to use the rule weights from the HEHRS method as the set of meta-features, rather than using meta-features that just indicate whether or not an example is covered by each rule. In essence this is an extension of the voting scheme discussed in section 6.3.1, where rules are built to decide the vote. In this alternative approach a meta-example has the following structure:

$$\textit{Meta-Example}_z = N_{z1}, N_{z2},\ldots, N_{zn}, C_z$$

Where N_{zj}, j = 1,..,n, is the weight of the jth class (d) and n is the number of classes in the set of sibling classes S. This version of SHEHRS will be hereafter referred to as Wei-SHEHRS (rule Weight-based SHEHRS).

Meta-Example ID	**Meta-Features**		**Actual Class**
	Class_1	**Class_2**	
1	1.2	0.7	1
2	1	0.9	1
3	0.3	1.1	2
4	1.2	1	2
5	1.1	1	2

Table 6.4: A set of rule weight-based SHEHRS meta-examples

Table 6.4 shows a possible set of meta-examples for Wei-SHEHRS, where each meta-feature is the accumulative rule weight of all rules covering the base level example for each class d. This simple set of meta-examples might lead to the discovery of a meta-rule set such as:

```
IF Class_1 > 0.3 AND Class_2 < 1 THEN CLASS = 1
IF Class_2 ≥ 1 THEN CLASS = 2
```

Even in this simple example it is clear to see the potential advantage gained by using a more flexible approach (when compared to standard weighted voting) such as Wei-

SHEHRS. If the simple weighted voting scheme presented in Section 6.3.1 was used to classify these examples, meta-examples with ID 4 and 5 would be misclassified.

It is also possible to mix these two kinds of meta-features (Boolean rule coverage and real-valued rule weight) to form a meta-example with both binary and continuous meta-features. The motivation for using this approach is to give the meta-learning algorithm more information for discovering high-quality meta-rules. This approach including the two kinds of meta-features will be hereafter referred to as Cov-Wei-SHEHRS.

The main difference between SHEHRS and classical stacking (designed for flat classification) is the way in which the level-0 classifiers are constructed using the class hierarchy in SHEHRS. As discussed previously, the problem is broken down using the divide and conquer approach, building an ensemble for each set of sibling class nodes, rather than building a single level-1 classifier that makes a single complete prediction for each example (although this is a possible future research direction).

As a meta-data set is constructed any classifier-learning algorithm can be used to build a classifier to classify it. A rule induction algorithm has the advantage building a model that is comprehensible and has the possibility of only using a subset of the meta-features. In addition, rule induction algorithms are in general relatively computationally fast, by comparison with much slower types of algorithms such as artificial neural networks and support vector machines. This computational efficiency is important in the context of SHEHRS and the datasets used in our experiments, where there are a large number of classes to be predicted, as will be shown later.

As is common in the literature [159] we also compare the proposed SHEHRS using a rule induction algorithm against a Bayesian method which constructs the level-1 classifiers using a naive Bayesian classification algorithm [135], which is also a computationally efficient type of classification method.

6.5. Rule-Based Extended Multiplicative Method

This method is derived from a method proposed by Sun et al. [152] to reduce the problem of blocking in hierarchical multi-label classification. The blocking problem was described by Sun et al. in the following way. Each class node in the class tree is associated with a probabilistic classifier, learned during the training phase. In the testing phase, an example with unknown class is classified in a top-down fashion, as follows. For each class node in the first level of the class tree, the example is assigned that class if the corresponding classifier predicts that class with a probability greater than a predefined threshold. An example is said to be rejected by a classifier if the probability of the example having the class predicted by the classifier is smaller than or equal to the threshold. For each of the (parent) classes assigned to the example at the first level, the example is pushed down the class tree to the child class nodes of those parent classes. Then, for each of those child classes the example is either assigned that child class or is rejected by the corresponding classifier depending on the probability of that class as computed by the classifier (again, compared with a threshold), etc. This top-down classification process is repeated until the example reaches the leaf nodes of the class tree. In this context, blocking occurs when an example is wrongly rejected by a classifier in an internal (non-leaf) node of the class tree, and so the example can never be shown to the classifiers that are descendants of the classifier that made the wrong rejection. As a result, the example can never be correctly classified at class levels deeper than *d*, where *d* is the level of the classifier that wrongly rejected the example.

One of the methods proposed by Sun et al. to cope with the blocking problem consists of assigning an example to a leaf class in the class tree if the multiplied probabilities of the example belonging to the internal (non-leaf) classes along the path from the root node to the leaf class node exceed a certain threshold. The authors called their approach the Extended Multiplicative Method (EMM).

Note that in Sun et al.'s work an example can be assigned to more than one class at each hierarchical level, which is characteristic of multi-label classification problems. This is not the case in the data sets examined in this chapter, where a single class must be assigned for each level. In addition, EMM was proposed in the context of probabilistic classifiers, which again is not the case in our work, where the classifier consists of a set of IF-THEN classification rules.

Therefore, we adapted EMM to the context of our work, where the classification of test set examples is performed by classification rules and we must assign only one class label per hierarchical level to each example. In this context, there is no need for the threshold used by EMM, since a testing example is simply assigned the best predicted class at each hierarchical class level. In addition, note that different leaf class nodes can be at different depths in the class tree. Hence, just multiplying the probabilities along each path from the root to a leaf class node is not appropriate because, when we compare the probabilities associated with different leaf classes in order to choose the best leaf class to be assigned to the testing example, shallower leaf class nodes would have an advantage over deeper ones – given the reductive nature of multiplying positive numbers smaller than 1. Furthermore, there is no innate sense of probabilistic matching given a rule-based classifier, so it is natural to use a measure of rule quality instead of probabilistic matching.

Given this discussion, our variant of EMM, called Rule-based EMM, finds the best "path" consisting of a series of rules discovered by HEHRS – one rule for each class level. It considers every possible path by considering not only the best rule covering the current test example at each class level, but all possible rules that cover the current test example. The best path is considered to be the path with the highest value of the geometric mean of all the rule weights along the path from the root to the class leaf node, as given by Equation 6.3. The geometric mean is used as the rule qualities in this hierarchical setting have a multiplicative (rather than additive) nature – at each progressively lower level in the hierarchy the rule can only correctly classify the set of examples correctly classified by its parent classifier.

$$PathQuality = \sqrt[l]{w_1 \times w_2 \times ... w_l}$$

Equation 6.3: Rule Path Quality for Rule-Based EMM

Where *PathQuality* is the score for a certain path and $w_i, i = 1,...,l,$ is the weight associated with the rule covering the example at class level *i* in that path and *l* is the number of rules that cover the example (i.e. the number of class levels) in that path. The formula used to compute each rule weight is given by Equation 6.1 and Equation 6.2.

6.6. The Creation of the Bioinformatics Data Sets

The hierarchical classification methods proposed in the previous section were evaluated in six challenging real-world datasets involving the prediction of protein function. The protein functional classes to be predicted in these data sets are the functional classes of GPCRs (G-Protein-Coupled Receptors) or Enzymes.

The protein functional classes are given unique hierarchical indexes by GPCRDB (Section 2.6.2) in the case of GPCRs and by Enzyme Commission Codes (Section 2.6.3) in the case of enzymes. In the case of GPCRs, examples (proteins) have up to 5 class levels, but only 4 levels are used in the datasets created in this work, as the data in the 5th level is too sparse for training – i.e., in general there are too few examples of each class at the 5th level. In any case, it should be noted that predicting all the first four levels of GPCR's classes is already a challenging task. Indeed, most works on the classification of GPCRs limit the predictions to just the topmost or the two topmost class levels (families and subfamilies but not groups, etc.) [15] [73] [92] [116]. All 4 levels of the Enzyme Commission Codes are used in the created Enzymes data sets.

The data used in our experiments was constructed from data in UniProt (Section 2.6.1) and GPCRDB. UniProt is a well known biological database, containing protein sequence data and a rich annotation about a large number of different kinds of proteins. It also has cross-references for other major biological databases such as Prosite (Section 2.5.1), Prints (Section 2.5.3), Pfam (Section 2.5.2) and Interpro (Section 2.5.4) (see below). It was extensively used in this work as a source of data for creating the data sets used in our experiments. Only the UniProtKB/Swiss-Prot was used as a data source, as it contains a higher quality, manually annotated set of proteins. Unlike Uniprot, GPCRDB is a biological database specialised on GPCR proteins.

We did experiments with four different kinds of predictor attributes, each of them representing a kind of "protein signature", or "motif", namely: FingerPrints from the Prints database, Prosite patterns, Pfam and Interpro entries (see Section 2.5 for descriptions of these different signatures). We created six data sets to evaluate the proposed hierarchical classification methods, three GPCR data sets and three Enzyme data sets. For the GPCR data sets the main predictor attributes were Prints, Prosite and Interpro entries (with a different type of motif used in each of the data sets). In addition, all three GPCR data sets used as predictor attributes the protein's molecular weight and sequence length. For the Enzyme data sets the main predictor attributes were Prosite, Interpro and Pfam entries. Again, all three enzyme data sets used the protein's molecular weight and sequence length as predictors.

Any duplicate examples (proteins) in a data set are removed in a pre-processing step, i.e., before the hierarchical classification algorithm is run, to avoid redundancy. For both GPCR and Enzyme data sets, if there are fewer than 10 examples in any given class in the class tree that class is merged with its parent class. If the parent class is the root node, the entire small class is removed from the data set. This process ensures there is enough training and test data per class to carry out the experiments. (If a class had less than 10 examples, during the 10-fold cross-validation procedure there would be at least one iteration where there would be no example of that class in the test set, an undesirable situation.) Any binary attribute that has a value which occurs in only one example is removed from the corresponding data set, since these binary attributes in general do not have a good predictive power. An initial random sample of 15000 enzymes from the UniProt database was used to generate the enzyme data sets. Less than the original 15000 examples occur in the final data sets because of the duplicate and small class removal process.

After data pre-processing, the final datasets used in the experiments have the numbers of attributes, examples (proteins) and classes per level (expressed as level 1/ level 2/level 3/level 4) indicated in Table 6.5.

	GPCR/ Prints	**GPCR/ Prosite**	**GPCR/ Interpro**	**EC/ Prints**	**EC/ Prosite**	**EC/ Pfam**
#Attributes	283	129	450	382	585	708
#Examples	5422	6261	7461	14038	14048	13995
#Classes	8/46/76/49	9/50/79/49	12/54/82/ 50	6/45/92/ 208	6/42/89/ 187	6/41/96/190

Table 6.5: Main characteristics of the datasets used in the experiments

6.7. Computational Results

This section reports computational results evaluating the methods proposed in Sections 6.2-6.5 in the created datasets described in Section 6.6. Recall that Sections 6.2-6.5 proposed several types of hierarchical classification methods, namely:

(a) Hierarchical Ensemble of Hierarchical Rule Sets (HEHRS) with rule weights computed by Equation 6.1 and Equation 6.2;

(b) HEHRS with rule weights optimized by PSO – two versions of the PSO were proposed, with and without a lower limit of 0 for the rule weights; these two versions are hereafter referred to as LimPSO-HEHRS and PSO-HEHRS, respectively. Both versions of PSO are a "vanilla" PSO [21] with standard parameter settings [35]: W=0.73, $\varphi_1 = \varphi_2 =$

2.05. Note that these parameter settings have been carefully calculated and that the performance of PSO is sensitive to them (as explained in Clerc and Kennedy's work [35]).

(c) The Extended Multiplicative Method adapted for rule-based (rather than probabilistic) classifiers – hereafter called Rule-EMM for short.

(d) The Stacking for HEHRS approach using the rule coverage meta-attributes, using rule induction algorithm RIPPER as the level-1 classifier or Naïve Bayes as the level-1 classifier, referred to as Cov-SHEHRS and Bayes-SHEHRS respectively.

(e) The Stacking for HEHRS approach using the rule weights as meta-attributes, with RIPPER as the level-1 classifier, referred to as Wei-SHEHRS.

(f) The Stacking for HEHRS approach using the rule weight and rule coverage meta-attributes with RIPPER as the level-1 classifier (Cov-Wei-SHEHRS).

These methods are compared against a baseline method, namely the standard top-down approach for hierarchical classification. This approach consists of simply running a rule induction algorithm at each internal (non-leaf) node of the class tree, as described in Section 3.7.2. In the proposed and baseline methods the base rule induction algorithm used in our experiments was the well-known RIPPER algorithm [39].

Throughout the entire set of experiments 10-fold cross validation [164] is used. Since PSO is a stochastic method, the PSO-HEHRS and Lim PSO-HEHRS methods are run 10 times each – with different random seeds used to create the initial population in each run – for each one of the 10 iterations of the cross-validation procedure. As the remainder of the methods are deterministic, they are run just once for each of the 10 cross-validation iterations.

Table 6.6 through Table 6.11 and Table 6.15 through Table 6.20 show the predictive accuracy that the different methods achieved in each data set during 10-fold cross validation. The numbers after the "±" symbol are standard deviations. The first set of tables (Table 6.6 through Table 6.11) report results involving voting schemes for HEHRS and the Extended Multiplicative Method. The second set of tables (Table 6.15 through Table 6.20) report results involving stacking for HEHRS. In all these tables a cell is coloured dark grey if there is a statistically significant win of the method in the corresponding column against the baseline method, according to a two-tailed Student's t-test with significance level of 0.05. The t-test used is WEKA's implementation of Nadeau and Bengio's corrected re-sampled t-test [164]. This more conservative corrected t-test takes into account the ratio of training and test examples in an attempt to limit the number of significant results occurring by chance. A cell is coloured light grey if there is a statistically significant loss when compared to the baseline method. Table 6.12 and Table 6.21 show the cumulative scores – calculated based on the results of the Student's t-test – for each method, at each class level, for all data sets. For each data set, in Table 6.12 and Table 6.21 one is added to the score of each cell if its corresponding method (indicated by the column label) at the corresponding class level (indicated by the row label) significantly beats the baseline approach in that data set. One is deducted from the score in the cell for a loss against the baseline approach in the same manner. The totals in the bottom row of the table are simply the summed results – over all data sets – from each class level for each method.

Table 6.13, Table 6.22 and Table 6.14, Table 6.23 show the un-weighted and weighted – respectively – misclassification costs associated with each experiment. The misclassification cost is computed by finding the shortest path in the class tree from the predicted class node to the actual class node.

In the case of the weighted misclassification cost this path is then weighted (the weights of the edges of the path are added), with edges between the root node and the first class level given a weight of 0.26, the edges between the first a second class level given a weight of 0.13, between the second and third a weight of 0.07 and between the third and fourth class levels a weight of 0.04. The reason for this weighting is to assign a higher cost to more general misclassifications. These general errors are more serious than the finer grained errors at lower levels of the class tree, as if a general error is made, no information about the true class of an example is gained. The particular values assigned to the edges will affect the overall result to a large extent, making the scoring system more sensitive to the misclassifications at, or before the most heavily weighted edge. Also, the differences between the weighting at each level must also be taken into account, with less difference making the weighted misclassification cost behave more like the un-weighted misclassification cost (discussed in the next paragraph). It is important that the analysis of the results take into account the way each edge is weighted.

In the case of the un-weighted misclassification score, each edge is assigned a weight of one. The final misclassification score is normalised by dividing the total number of edges (or total weight) in the path between the predicted and the actual class nodes in the class tree by the total worst possible score for all examples. The latter can be found by finding the weight (or number of edges) from the actual class to any leaf node via the root node, and taking the largest weight (or number of edges) as the worst possible misclassification score.

The accumulative student's t-test score at the bottom of Table 6.13, Table 6.22 and Table 6.14, Table 6.23 shows the number of times the corresponding method is significantly better (+1) or worse (-1) than the baseline approach across all the experiments. The misclassification costs shown in Table 6.13, Table 6.22 and Table 6.14, Table 6.23 are useful as (unlike the accuracy rates) they take into account the hierarchical structure of the classes, and so they provide a way to quickly assess the performance of a hierarchical approach. They can also be tailored to concentrate on general or fine grained errors using weighting.

6.7.1. Voting Schemes for HEHRS and Extended Multiplicative Method Results

Let us first analyze the voting for HEHRS results with respect to accuracy rate (shown in detail in Table 6.6 through Table 6.11 and summarised in Table 6.12). As can be observed in Table 6.12 the pure HEHRS – without rule weights optimized by PSO – achieved a disappointing performance: it obtained an overall score of –7, overall, significantly losing 7 times (according to the student's t-test results) against the baseline approach. Observing both Table 6.12 and the more detailed results per dataset in Table 6.6 through Table 6.11, one can see that, in all the 6 datasets, HEHRS obtains results significantly worse than the baseline method's results in the first two (shallower) class levels. On the other hand, in all the 6 datasets HEHRS obtains results significantly better than the baseline method's results in the fourth (deepest) class level. The consistency of these results is interesting, considering that the 6 datasets contain very different numbers of attributes and examples, as well as different kinds of biological motifs as predictor attributes – as indicated in Table 6.5.

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	91.0±0.65	91.5±0.8	91.3±0.83	90.6±0.41	91.2±0.74
2	65.1±1.25	82.0±1.09	81.7±1.06	77.9±0.46	80.3±1.12
3	37.5±0.84	56.1±1.43	56.1±1.2	55.1±0.95	53.5±1.5
4	44.0±3.49	83.1±3.03	83.0±2.78	82.1±2.33	78.3±2.53

Table 6.6: Predictive accuracy (%) with Prints attributes and GPCR classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	90.2±0.69	91.0±0.71	91.1±0.76	89.7±0.3	90.3±0.71
2	68.5±0.79	83.3±0.97	82.9±0.82	79.1±0.47	81.1±0.74
3	36.4±1.03	55.2±1.33	55.4±1.15	54.6±1.16	52.8±0.87
4	46.0±2.86	86.9±1.78	86.6±1.81	86.5±2.23	82.4±2.65

Table 6.7: Predictive accuracy (%) with InterPro attributes and GPCR classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	87.4±0.88	87.8±0.62	87.5±1.0	86.3±1.36	87.6±0.92
2	49.8±1.18	63.5±1.77	62.9±1.91	61.5±1.79	63.9±1.43
3	18.1±0.59	32.2±1.74	32.3±1.91	29.5±1.62	29.3±1.56
4	12.8±2.39	45.5±3.18	45.5±3.93	36.5±2.46	35.4±1.84

Table 6.8: Predictive accuracy (%) with Prosite attributes and GPCR classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	48.9±2.41	97.8±0.34	97.8±0.41	96.7±0.35	97.4±0.28
2	33.5±2.61	95.0±0.47	95.2±0.67	93.3±0.29	94.6±0.46
3	32.8±1.03	94.1±0.34	94.3±0.65	90.1±0.97	93.8±0.54
4	29.7±0.91	93.4±0.69	93.7±0.79	93.3±0.75	92.8±0.87

Table 6.9: Predictive accuracy (%) with Prints attributes and Enzyme classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	37.0±0.24	98.0±0.2	98.0±0.32	92.3±1.01	95.8±1.84
2	23.3±0.8	96.2±0.43	96.3±0.37	88.7±1.07	94.0±2.04
3	23.5±0.74	94.9±0.5	94.9±0.45	87.6±1.01	92.6±2.26
4	23.5±0.75	96.0±0.48	96.1±0.33	95.1±0.89	94.5±1.19

Table 6.10: Predictive accuracy (%) with Pfam attributes and Enzyme classes

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
1	40.7±0.4	98.7±0.3	98.7±0.24	96.6±0.48	98.5±0.24
2	28.1±0.42	97.4±0.45	97.3±0.41	94.1±0.27	97.1±0.42
3	26.2±0.44	96.2±0.39	96.0±0.34	92.4±0.45	95.9±0.19
4	23.3±0.44	95.2±0.34	95.3±0.41	95.2±0.42	95.0±0.42

Table 6.11: Predictive accuracy (%) with Prosite attributes and Enzyme classes

Overall Scores Against Baseline – The best possible score for each cell in the first 4 rows is 6 (number of data sets)

Class level	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS
1	-3	3	2	-4
2	-6	4	3	-6
3	-6	4	4	-1
4	-6	4	5	4
Totals	**-21**	**15**	**14**	**-7**

Table 6.12: Summation of the number of statistically significant results according to the Student's t-test, when comparing the proposed approaches to the baseline

The poor performance of HEHRS is likely due to its bias towards deeper classes. As it predicts a class based upon the addition of rule weights, classes that are deeper will have more nodes and so more weights when compared to shallower ones. This explanation is supported by the differences seen between the GPCR and Enzyme data sets. In the GPCR data sets the number of examples in each class is quite unbalanced, with one class having a large portion of the examples, this is even more so the case at lower levels. This is an advantage for HEHRS at the lower levels (3 and 4) because it tends to try and classify more examples as the deeper class, which also happens to be one of the largest. The classes are more balanced in the enzyme data set but again the bias towards deeper classes still reaps rewards in the fourth level.

One method of dealing with this bias would be to average the rule weights rather than adding them. However, it is likely that this would cause the opposite problem in HEHRS – a bias towards shallower classes. This is because, in general, rules at deeper class levels tend to have lower qualities, due to the higher number of classes and lower number of examples per class. Hence, the averaging process would favour the classes with fewer descendants, giving fewer and higher weights. Investigating the effect of this averaging process empirically could be a topic for future research. By contrast, this thesis proposed a more sophisticated solution to the above problems, consisting of adaptively adjusting the rule weights with a PSO algorithm, which produced good results – as will be discussed later.

The Rule-EMM method achieved by far the worst results, significantly losing to the baseline method in 21 out of 24 cases. This very bad performance is most likely due to the way in which a decision list is generated by the rule induction algorithm and its interaction with the EMM approach. The Rule-EMM method is reliant on not choosing only the best matching rule (as in RIPPER), but all rules that match the test example at all (at each class level) in a rule list. This is the trade off needed when attempting to find all possible paths to class leaf nodes. The trade off does not seem to pay off with the current rule induction algorithm, RIPPER. It is possible that if the rules produced by the rule induction algorithm were unordered the misclassifications would become less of a problem, since unordered rules tend to be more modular than ordered rules. Investigating this hypothesis is an interesting topic for future research.

In general the best performing methods in terms of predictive accuracy are LimPSO-HEHRS and PSO-HEHRS, with the version of PSO *without* a lower limit on the rule weights (PSO-HEHRS) beating the PSO version with a lower limit (LimPSO-HEHRS) by only one test. Both methods obtained a good performance, with an overall score of 15 or 14, respectively – the maximum possible score is 24 (4 class levels times 6 datasets).

Data Set	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
GPCR Prints	28.3±0.78	18.72±0.62	18.88±0.69	20.76±0.33	19.86±0.61
GPCR Interpro	25.53±0.5	17.13±0.51	17.19±0.5	19.21±0.31	18.44±0.36
GPCR Prosite	37.62±0.66	30.8±0.74	31.21±0.79	32.83±1.13	31.43±0.94
Enzyme Prints	60.5±1.74	4.78±0.39	4.67±0.55	6.74±0.4	5.2±0.42
Enzyme Pfam	69.45±0.51	3.68±0.29	3.65±0.26	9.68±0.91	5.73±1.83
Enzyme Prosite	66.59±0.41	2.99±0.29	3.07±0.22	5.68±0.32	3.21±0.23
Accumulative t-test Score against Baseline	-6	5	3	-6	

Table 6.13: The Un-weighted Misclassification cost, comparing each proposed approach against the baseline

Data Set	Rule-EMM	PSO-HEHRS	LimPSO-HEHRS	HEHRS	Baseline
GPCR Prints	22.24±0.65	15.1±0.65	15.31±0.75	17.01±0.3	15.98±0.66
GPCR Interpro	20.62±0.53	14.3±0.63	14.35±0.65	16.38±0.23	15.51±0.53
GPCR Prosite	30.31±0.77	24.91±0.7	25.33±0.95	26.87±1.29	25.25±0.96
Enzyme Prints	57.74±2.16	3.73±0.36	3.66±0.49	5.4±0.27	4.16±0.32
Enzyme Pfam	68.23±0.43	3.04±0.25	3.0±0.27	9.33±0.96	5.21±1.89
Enzyme Prosite	64.62±0.38	2.2±0.31	2.27±0.23	4.9±0.37	2.42±0.24
Accumulative t-test Score against Baseline	-6	4	3	-6	

Table 6.14: The Weighted Misclassification cost, comparing each proposed approach against the baseline

These conclusions derived from the analysis of accuracy rates are also reflected in general in the misclassification costs (Table 6.13 and Table 6.14), with HEHRS and Rule-EMM getting the same overall negative score against the baseline and the two versions of the PSO getting an overall positive score against the baseline. Also, when the finer grained misclassifications are weighted more evenly (as with the un-weighted misclassification costs) the difference between LimPSO-HEHRS and PSO-HEHRS becomes more apparent, with PSO-HEHRS outperforming LimPSO-HEHRS significantly in 2 out of 6 tests.

Using the PSO to optimise rule weights has the disadvantage that a PSO run is computationally expensive. On a machine with a P4 3.0 GHz CPU it takes about five hours to optimise the weights for the rules generated from a single 10 times 10-fold cross validation run (depending on the number of rules). Also HEHRS itself requires more computational time as many more rule sets must be induced using larger training sets (when compared to the baseline approach). On the same machine the models for a single run of the baseline approach are induced within ten minutes, whereas the HEHRS models take up to one hour on the larger datasets. These models do not vary between approaches and so can be cached, increasing efficiency when comparing multiple approaches. However, note that maximising classification accuracy is usually considered more important than minimizing the processing time taken by a classification algorithm. This is particularly the case in real-world scenarios like the bioinformatics problems addressed in this work, where the time taken by a run of the PSO algorithm is a very small fraction of the time that was spent in preparing our datasets for data mining purposes (about 4 months). This scenario is also often found in other data mining applications, where most of the time taken by the entire knowledge discovery process is spent preparing data [91].

6.7.2. Stacking for HEHRS Results

As can be seen in tables Table 6.15 through Table 6.20 – and summarised in Table 6.21 – the rule based (Cov-SHEHRS, Wei-SHEHRS and Cov-Wei-SHEHRS) stacking approaches (SHERS) do not perform particularly well when compared to the baseline approach. The method that purely uses binary attributes (Cov-SHEHRS) performs the best out of these three methods, with the purely continuous version doing the least well (Wei-SHEHRS) – nearly always losing to the baseline approach. The Bayes-SHEHRS method (which uses a naïve Bayesian classifier as the level-1 classifier) performs better, significantly beating the baseline approach in 8 out of 24 cases (Table 6.21).

Class Level	Bayes-SHEHRS	Cov-SHEHRS	Wei-SHEHRS	Cov-Wei-SHEHRS	Baseline
1	90.5±1.13	91.1±0.9	89.7±1.03	90.5±0.99	91.2±0.74
2	80.5±1.04	78.0±1.1	45.9±1.37	44.3±1.38	80.3±1.12
3	54.9±1.42	53.5±1.12	39.9±1.33	45.2±1.1	53.5±1.5
4	80.3±1.5	80.1±2.66	71.1±2.47	77.7±2.89	78.2±2.53

Table 6.15: Predictive accuracy (%) with Prints attributes and GPCR classes

Class Level	Bayes-SHEHRS	Cov-SHEHRS	Wei-SHEHRS	Cov-Wei-SHEHRS	Baseline
1	90.1±0.84	87.0±11.15	86.8±1.2	90.2±1.41	90.2±0.71
2	80.6±1.1	71.6±11.48	51.4±11.54	57.1±13.6	81.1±0.74
3	54.0±1.07	49.3±5.9	43.3±3.01	46.6±3.23	52.8±0.87
4	83.9±2.17	80.6±2.36	78.1±2.88	83.2±3.52	82.4±2.65

Table 6.16: Predictive accuracy (%) with InterPro attributes and GPCR classes

Class Level	Bayes-SHEHRS	Cov-SHEHRS	Wei-SHEHRS	Cov-Wei-SHEHRS	Baseline
1	87.4±1.4	85.7±2.86	86.3±1.4	86.3±1.39	87.6±0.92
2	64.4±1.41	61.7±2.02	52.4±8.67	45.5±5.93	63.9±1.43
3	30.1±1.72	29.5±1.35	27.8±2.29	26.2±0.97	29.3±1.56
4	36.4±3.35	37.2±1.26	38.6±6.54	41.7±4.01	35.3±1.84

Table 6.17: Predictive accuracy (%) with Prosite attributes and GPCR classes

Class Level	Bayes- SHEHRS	Cov- SHEHRS	Wei- SHEHRS	Cov-Wei- SHEHRS	Baseline
1	97.5±0.28	96.0±0.38	95.4±0.43	95.9±1.05	97.3±0.28
2	94.9±0.57	93.4±0.6	92.2±0.52	92.8±1.11	94.5±0.46
3	93.5±0.8	91.1±0.83	89.4±0.76	90.8±1.17	93.7±0.54
4	92.6±0.82	90.0±0.76	75.6±1.04	91.5±0.76	92.8±0.87

Table 6.18: Predictive accuracy (%) with Prints attributes and Enzyme classes

Class Level	Bayes- SHEHRS	Cov- SHEHRS	Wei- SHEHRS	Cov-Wei- SHEHRS	Baseline
1	97.9±0.33	97.1±0.75	95.3±2.55	96.4±2.06	95.7±1.84
2	96.4±0.49	95.3±0.87	91.3±1.83	92.9±1.66	94.0±2.04
3	94.7±0.7	92.8±0.97	87.4±1.82	90.7±1.76	92.6±2.26
4	96.2±0.46	94.0±0.61	83.6±0.89	94.7±0.5	94.4±1.19

Table 6.19: Predictive accuracy (%) with Pfam attributes and Enzyme classes

Class Level	Bayes- SHEHRS	Cov- SHEHRS	Wei- SHEHRS	Cov-Wei- SHEHRS	Baseline
1	98.6±0.26	98.6±0.35	97.2±0.79	97.7±0.86	98.5±0.24
2	97.1±0.32	97.0±0.52	93.2±0.73	96.0±0.82	97.1±0.42
3	95.8±0.29	95.8±0.29	89.6±0.71	94.5±0.87	95.9±0.19
4	95.0±0.49	94.9±0.48	80.4±0.57	95.1±0.5	94.9±0.42

Table 6.20: Predictive accuracy (%) with Prosite attributes and Enzyme classes

Overall Scores Against Baseline – The best possible score for each cell in the first 4 rows is 6 (number of data sets)

Class Level	Bayes- SHEHRS	Cov- SHEHRS	Wei- SHEHRS	Cov-Wei- SHEHRS
1	1	0	-5	-2
2	1	-3	-5	-5
3	3	-1	-6	-5
4	3	0	-5	0
Totals	**8**	**-4**	**-21**	**-12**

Table 6.21: Summation of the number of statistically significant results, according to the Student's t-test, when comparing each proposed approach to the Baseline

It is initially somewhat surprising that the rule based stacking approaches perform so badly, especially that the Wei-SHEHRS approach performs significantly worse than the considerably more simple HEHRS approach (HEHRS uses weighted majority voting whereas Wei-SHEHRS uses classification rules – as was discussed in the latter part of section 6.4). This is due to the generality and flexibility of the voting scheme and the reliance of Wei-SHEHRS on specific cases found in the training phase that may not occur in the testing phase (see Table 6.4 for an example of such a case). This problem may be amplified by using RIPPER (a general purpose classification algorithm) on a very specific type of problem – finding relationships between the continuous meta-attributes. Rules may be found that make little sense but fit the training data well, e.g., including a term that puts an upper limit on the cumulative score meta-attribute for a particular class.

Furthermore, the performance of the Cov-Wei-SHEHRS and Cov-SHEHRS approaches is most likely due to over fitting and the level-1 algorithm being unaware of the hierarchy involved in HEHRS. This hierarchy is important as higher level rules are usually more reliable than lower level ones within the ensemble. The algorithm is unaware of this during the rule pruning stages, to its detriment, i.e., it is just as likely to prune a term involving a higher level rule as a lower level rule. Obviously if there was sufficient and extensive enough training data for the level-1 classifiers this would not be a problem, but this is always the case with any induction algorithm. The impact of pruning can be seen by the relatively good performance of Bayes-SHEHRS when compared to Cov-SHEHRS. The methods are very similar, except that the Bayes method is influenced by all level-0 rules (due to the nature of the Bayes classification algorithm) whereas the Cov approach is only influenced by the level-0 rules included in the meta-rules. This makes the Cov approach less flexible in the testing phase and more prone to overfitting the training data.

Data Set	**Bayes-SHEHRS**	**Cov-SHEHRS**	**Wei-SHEHRS**	**Cov-Wei-SHEHRS**	**Baseline**
GPCR Prints	19.36±0.61	20.39±0.59	33.36±0.84	31.86±0.97	19.86±0.61
GPCR Interpro	18.21±0.49	21.78±5.6	30.29±3.64	26.16±4.32	18.44±0.36
GPCR Prosite	30.91±1.04	32.55±1.51	35.33±3.33	37.76±1.82	31.43±0.94
Enzyme Prints	5.04±0.49	6.81±0.46	10.93±0.47	6.98±0.89	5.2±0.42
Enzyme Pfam	3.67±0.39	4.87±0.74	10.0±1.8	6.3±1.51	5.73±1.83
Enzyme Prosite	3.19±0.24	3.22±0.31	9.04±0.54	4.13±0.67	3.21±0.23
Accumulative t-test Score against Baseline	3	-1	-6	-5	

Table 6.22: The Un-weighted Misclassification cost, comparing each proposed approach against the baseline

Data Set	**Bayes-SHEHRS**	**Cov-SHEHRS**	**Wei-SHEHRS**	**Cov-Wei-SHEHRS**	**Baseline**
GPCR Prints	16.0±0.85	16.57±0.65	27.68±0.85	26.81±0.92	15.98±0.66
GPCR Interpro	15.54±0.58	19.59±8.59	26.6±3.2	22.29±3.83	15.51±0.53
GPCR Prosite	24.98±1.24	26.85±2.1	29.24±3.08	31.22±1.51	25.25±0.96
Enzyme Prints	3.97±0.4	5.61±0.41	7.55±0.4	5.88±1.0	4.16±0.32
Enzyme Pfam	3.05±0.36	4.05±0.76	7.51±2.14	5.39±1.74	5.21±1.89
Enzyme Prosite	2.38±0.24	2.41±0.34	5.89±0.66	3.34±0.76	2.42±0.24
Accumulative t-test Score against Baseline	1	-2	-6	-5	

Table 6.23: The Weighted Misclassification cost, comparing each proposed approach against the baseline

The un-weighted misclassification cost (Table 6.22) reveals that the Bayes-SHEHRS approach improves predictive accuracy significantly in 3 out of 6 cases. The fact that the scores differ between the weighted and un-weighted misclassification costs (Table 6.23 and Table 6.22 respectively) indicate that the Bayes-SHEHRS and Cov-SHEHRS approaches perform better at lower levels in the class hierarchy. This can be surmised as the weighted misclassification cost measure weights the misclassifications at higher levels more heavily than the lower ones, whereas the un-weighted cost weights all misclassification the same. The fact that there appears to be less difference between the two approaches in the weighted misclassification cost table when compared to the un-weighted misclassification cost table means that most of the difference is in the lower levels of the class hierarchy. This is not entirely surprising as misclassifications that are averted are bound to accumulate towards the lower levels of the class tree due to the nature of TDDC.

6.8. Summary

This work proposed new hierarchical classification methods that use characteristics of hierarchical class data (where the classes are arranged in a tree structure) to try to improve predictive accuracy, with respect to a standard top-down hierarchical classification method. More precisely, four main types of hierarchical classification methods were proposed, namely:

(a) HEHRS (Hierarchical Ensemble of Hierarchical Rule Sets), a method based on exploiting the hierarchical nature of the data to create different training sets to be given as input to a bagging-like ensemble method;

(b) Two versions of a Particle Swarm Optimisation (PSO) method for optimising the rule weights used by HEHRS to classify test examples;

(c) Rule-EMM, the rule-based version of the Extended Multiplicative Method, which tries to reduce the problem of misclassifications at shallower class levels leading to misclassifications at deeper class levels in the standard top-down approach for hierarchical classification; and

(d) Four stacking approaches: Two using rule coverage meta-features, with either RIPPER or Naïve Bayes as the level-1 classifier; Two using either rule weight meta-features, or a combination of rule weight meta-features and rule coverage meta-features (with RIPPER as the level-l classifier).

Out of these four types of methods, the pure HEHRS method, Rule-EMM and RIPPER-based stacking approaches produced disappoint results, in general significantly worse than the standard top-down approach. However, the development of a PSO algorithm to optimise rule weights for HEHRS was very effective, leading to a hierarchical classification system that obtained, overall, predictive accuracies significantly better than the accuracies obtained by the standard top-down approach. The same can be said for the results from Naïve Bayes-based stacking approach, which significantly increased performance beyond that of the baseline. These results were to a large extent consistent across 6 different bioinformatics datasets involving the hierarchical classification of protein functions, a set of challenging real-world bioinformatics problems with large numbers of predictor attributes and large numbers of classes to be predicted.

Chapter 7. Conclusions

7.1. Contributions

This thesis has proposed several new methods which aim to improve the predictive accuracy of hierarchical classification. It has focused on the prediction of protein functions, more specifically the function of two types of protein: G-protein coupled receptors and enzymes. Being able to predict the functions of these proteins automatically and accurately is a major task in bioinformatics and has important applications in our ability to create new drugs. For instance, being able to identify newly discovered or potential proteins that have a particular function relating to a disease allows biologists to design drugs that target them.

The hierarchical nature of the new protein data sets used in our experiments have posed difficult challenges, mainly due to the large number of classes and low number of records per class (at the deeper levels of the class hierarchy, containing more specialised classes). These challenges require new approaches tailored to the problem of hierarchical classification.

In this context, we have created new swarm intelligence techniques for the hierarchical classification problems identified in this work. Swarm intelligence algorithms have proved successful for data mining applications in the past and this thesis has demonstrated that they continue to be successful (one of the goals of the EPSRC-funded XPS project [169] was to assess how applicable Particle Swarm Optimisation could be to data mining problems).

The main original contributions of this thesis are:

- ***A Particle Swarm Optimisation/Ant Colony Optimisation Classifier Selector Algorithm.***

  The proposed PSO/ACO-CS (Section 5.2) algorithm is a new hierarchical classifier selector technique. This is based on the idea that different classifiers will be optimal at each "divide" (classifier node) in the top-down divide-and-conquer (TDDC) tree. Secker et al. [140] proposed a greedy approach to take advantage of this fact and to try and boost classification accuracy. We discussed the sub-optimality of this approach (Section 3.7.2.1) and proposed a swarm intelligence based method to try to find more optimal combinations of classifiers in a global search fashion.

  In terms of classification accuracy PSO/ACO-CS always at least equalled the best performing individual classification algorithm and often beat them in our experiments. Furthermore, PSO/ACO-CS affected a relatively small but significant improvement when compared to Secker et al's approach.

- ***A Particle Swarm Optimisation/Ant Colony Optimisation Misclassification Recovery algorithm.***

  The proposed PSO/ACO-RO (Section 5.3) algorithm is a misclassification recovery optimisation technique which uses a swarm intelligence method to try to mitigate a major drawback of the TDDC approach. This drawback, known as "blocking" in the literature, occurs when in the standard TDDC tree a classifier at a higher level in the tree misclassifies an example. Due to the top-down nature of the classifier tree, the misclassifications at a higher level can never be correctly classified at a lower level in the hierarchy. PSO/ACO-RO attempts to improve the situation by allowing classifiers to redirect examples (during the testing phase) to classifier nodes that are not necessarily one of their child classifier nodes. In this fashion each classifier can attempt to correct any errors that a parent classifier node has made. The swarm intelligence algorithm is used to decide which nodes benefit from this type of recovery and which do not (i.e., which classifier nodes should have misclassification recovery "turned on") in a global search manner.

PSO/ACO-RO and PSO/ACO-CS were combined to create PSO/ACO-CS-RO (Section 5.3.2). This approach was compared against the baseline classifiers and a modified greedy method based on Secker et al's approach (Greedy-PSO/ACO-RO). Overall PSO/ACO-CS-RO was the most effective approach, almost always outperforming all of the baseline classifiers. It was also shown that using PSO/ACO-RO increased the performance of both the greedy and PSO/ACO-CS techniques to a degree.

- ***The Hierarchical Ensembles of Hierarchical Rule Sets Method.***

The proposed HEHRS (Chapter 6) ensemble technique attempts to boost the accuracy of individual classifier nodes within the TDDC tree. It does this by constructing a hierarchical ensemble of rule sets in place of each classifier node within the TDDC tree. This can loosely be seen as a type of bagging (a well known ensemble method), with multiple classifiers derived from the same original data set but in different "forms", in this case the different "forms" are obtained by modifying the classes to be predicted in a way which follows the native class hierarchy. We explored several methods to combine the predictions made by each constituent rule set within each ensemble, including a PSO-based approach which optimises the rule weights in a global fashion.

Overall our experiments showed that the PSO-based technique performed significantly best overall, with the Bayesian technique (based on the naïve Bayes algorithm) also being quite effective. Both of these approaches significantly beat the baseline approach (a standard TDDC classifier using one type of algorithm).

The secondary original contributions of this thesis are:

- ***A Particle Swarm Optimisation/Ant Colony Optimisation Rule Induction algorithm.***

  The proposed PSO/ACO-RI (Chapter 4) is a new proof-of-concept "flat" classification rule induction algorithm. This algorithm is based on the hybrid Particle Swarm Optimisation/Ant Colony Optimisation (PSO/ACO) method proposed in this thesis. The PSO/ACO-RI algorithm copes directly with both nominal/categorical and numeric data.

  Our experiments showed that, in general, PSO/ACO-RI generated simpler rule sets when compared to the well established PART algorithm (based on the industry standard C4.5Rules algorithm). Furthermore, PSO/ACO-RI achieved this without reducing predictive accuracy.

- ***Protein Function Data Sets***

  We have created 7 new hierarchical protein function data sets to evaluate our proposed algorithms. These consist of 4 GPCR data sets and 3 enzyme data sets. These data sets are available to other researchers on request.

One of the main objectives of this thesis was to propose new methods to deal with hierarchical classification problems more effectively. Another main aim was to see if the previously successful swarm intelligence based paradigm would be effective in the type of challenging data mining problem explored in this thesis (namely hierarchical classification in particular applied to protein function). Taking the extensive experimental results reported in this thesis as a whole we believe that a firm conclusion can be made – yes, swarm intelligence is definitely effective and applicable to this type of problem. We hope that future researchers will take this conclusion on board and develop novel, interesting and more sophisticated approaches based on this very versatile paradigm.

7.2. Future Work

Although we have attempted to explore and evaluate as many potentially-effective techniques as possible for hierarchical classification, there remain a large number of potential avenues for future research.

The most obvious direction for future research is applying the techniques proposed in this thesis to more data sets. Different types of proteins and different protein function classification schemes could be used. Also, it should be possible to apply the techniques to text classification where there are often hierarchies present – although the methods proposed in this work might need modification for this specific (and quite different) type of problem in order for them to be effective. In any case, in principle the swarm intelligence based hierarchical methods proposed in this thesis are generic enough to be applied to any hierarchical classification problem in any application domain. This is true as long as the data is pre-processed in a format suitable for the proposed algorithms.

Another obvious direction for future research would be to extend the approaches to deal with data sets where the class nodes form a directed acyclic graph, rather than a tree as addressed in this thesis. This is a particularly interesting and challenging direction for research and it would be very interesting to discover whether the types of approach proposed in this thesis would be applicable to this type of problem. The approaches may also be adapted to deal with hierarchical multi-label problems, where an example can belong to more than one class node at any given class level.

A further interesting avenue for future research would be attempting to use swarm intelligence to create a "big bang" type hierarchical classification approach where the entire classifier – predicting potentially any of the classes in the class hierarchy – is constructed at once. The fact that PSO/ACO algorithm was shown to (in general) create simpler rule sets when compared to J48 and the fact that hierarchical C4.5 [33] (a "big bang" approach) is such a promising algorithm leads us to believe that this could be a very fruitful research direction.

Another potentially very interesting avenue for research would be to apply feature selection methods [41] (which uses PSO) [89] [90] [141] [142] to the datasets before they are processed by the approaches discussed in this thesis. An investigation into how this feature selection interacts with the proposed approaches would also be very interesting, also possibly using feature selection during the run of the proposed algorithms.

It would also be interesting to discover whether PSO/ACO would be effective on other combinatorial optimisation problems such as the travelling salesman problem (we present some very preliminary results in Appendix A for PSO/ACO on binary combinatorial optimisation problems). We believe that due to the flexibility of PSO/ACO – being able to select the topology and the possible advantages discrete recombination [22] (where influence is taken from all neighbouring particles not just the best one in a single interaction) has shown with standard PSO – this would be a promising area. Also, it may be interesting to use an optimiser to optimise the settings/topology, etc, for PSO/ACO for specific types of application. However, the advantage of such parameter optimisation would have to be balanced against the disadvantage of a much longer overall processing time.

It may also be possible to improve PSO/ACO-RI. At present PSO/ACO-RI is partly greedy in the sense that it builds each rule with the aim of optimising that rule's quality individually, without directly taking into account the interaction with other rules. A less greedy, but possibly more computationally expensive way to approach the problem would be to associate a particle with an entire rule set and then to consider the quality of the entire rule set when evaluating a particle. This is known as the "Pittsburgh approach" [61] in the evolutionary algorithm literature, and it could be an interesting research direction. Also, the part of the rule containing nominal attributes is always discovered first and separately from the part containing continuous attributes, it could be advantageous to use a more "co-evolved" [61] approach.

For classifier selection this work only compares the proposed PSO/ACO-CS algorithm with Secker et al's greedy selective approach, so one direction for future research is to compare the PSO/ACO-CS algorithm with another population-based meta-heuristics for optimisation, e.g. evolutionary algorithms. As the focus of this thesis was on swarm intelligence and not on the benefits of other potential algorithms, such a comparison would be an interesting topic of future research.

There are several potential avenues for future research with the ensemble-based approach proposed in Chapter 6. Since optimising the rule weights used by HEHRS with the PSO method proved to be very effective, perhaps the rule weights used by Rule-EMM could be optimised in just as an effective way. In addition, it would be interesting to investigate the performance of Rule-EMM when the base rule induction algorithm used to discover classification rules produces an unordered rule set, rather than an ordered rule list.

References

[1] A. Abraham, C. Grosan, V. Ramos. Swarm Intelligence in Data Mining. *Studies in Computational Intelligence*, Vol. 34. Springer. 2006.

[2] B. Alberts, D. Bray, K. Hopkin, A. Johnson, J. Lewis, M. Raff, K. Roberts. P Walter. *Essential Cell Biology second edition*, Garland Science. 2004.

[3] S. F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool. *J. Mol. Biol*, Vol. 215, No. 3. pp. 403-410. 1990.

[4] T. K. Attwood. The PRINTS database: a resource for identification of protein families. *Brief Bioinform*. pp. 252-63. 2002.

[5] T. K. Attwood, P. Bradley, D. R. Flower, A. Gaulton, N. Maudling, A. Mitchell, G. Moulton, A. Nordle, K. Paine, P. Taylor, A. Uddin, C. Zygouri. PRINTS and its automatic supplement, prePRINTS. *Nucleic Acids Research*, 31(1). pp. 400-402. 2003.

[6] R. Beckers, S. Goss, J. L. Deneubourg & J. M. Pasteels. Colony size, communication and ant foraging strategy. *Psyche*, 96. pp. 239-256. 1989.

[7] A. Bateman, L. Coin, R. Durbin, R. D. Finn, V. Hollich, S. Griffiths-Jones, A. Khanna, M. Marshall, S. Moxon, E. L. L. Sonnhammer, D. J. Studholme, C. Yeats, S. R. Eddy. The Pfam protein families database. *Nucleic Acids Research*, 32 (Database-Issue). pp. 138-141. 2004.

[8] Z. Barutcuoglu, R. Schapire, O. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7). pp. 830-836. 2006.

[9] R. Battiti, A. Colla. Democracy in Neural Nets: Voting Schemes for Accuracy. *Neural Networks*, Vol. 7. pp. 691-707. 1994.

[10] E. Bauer, R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36 (1/2). pp. 105-139. 1999.

[11] K. P. Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, 26. pp. 156-160. 1994.

[12] F. V. D. Bergh, An Analysis of Particle Swarm Optimizer, PhD thesis, University of Pretoria. 2001.

[13] M. Bernaschi, F. Castiglione, A. Ferranti, C. Gavrila, M. Tinti, G. Cesareni. ProtNet: a tool for stochastic simulations of protein interaction networks dynamics. *BMC Bioinformatics 2007*, 8(Suppl. 1), S4. 2007.

[14] R. A. Berk. Data Mining within a Regression Framework. In Proc. *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*, Oded Maimon and Lior Rokach (eds.), Kluwer Academic Publishers, Forthcoming.

[15] M. Bhasin, G.P. Raghava. GPCRpred: an SVM-based method for prediction of families and subfamilies of G-protein coupled receptors. *Nucleic Acids Res*, 1, 32 (Web Server issue). pp. 383-9. 2004.

[16] Eur. J. Biochem. Nomenclature Committee of the International Union of Biochemistry and Molecular Biology, 264. pp. 610-650. 1999.

[17] Eur. J. Biochem. IUPAC-IUB Joint Commission on Biochemical Nomenclature (JCBN). Nomenclature and Symbolism for Amino Acids and Peptides. *Recommendations*, 138. pp. 9-37. 1984.

[18] H. Blockeel, L. Schietgat, J. Struyf, S. Dzeroski, A. Clare. Decision Trees for Hierarchical Multilabel Classification: A Case Study in Functional Genomics. In Proc. of *PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, LNAI, vol 4213. pp. 18-29. 2006.

[19] H. Blockeel, M. Bruynooghe, S. Dzeroski, J. Ramon, J. Struyf. Hierarchical multi-classification. In Proc. *KDD-2002 Workshop Notes: MRDM 2002 – Workshop on Multi-Relational Data Mining*. pp. 21-35. 2002.

[20] E. Bonabeau, G. Théraulaz. Swarm Smarts. *Scientific American*. pp 73-79. 2000.

[21] D. Bratton, J. Kennedy. Defining a Standard for Particle Swarm Optimization. In Proc. of the *2007 IEEE Swarm Intelligence Symposium*. pp. 120-127. 2007.

[22] D. Bratton, T. Blackwell. A Simplified Recombinant PSO. *Journal of Artificial Evolution and Applications (JAEA), special issue on Particle Swarms: The Second Decade*. Article ID 654184. doi:10.1155/2008/654184. 2008.

[23] L. Breiman. Bagging Predictors. *Machine Learning*, Vol. 24. pp. 123-140. 1996.

[24] G. Brown, J. Wyatt, R. Harris, X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1). pp. 5-20. 2005.

[25] C. Bru, E. Courcelle, S. Carrère, Y. Beausse, S. Dalmar, D. Kahn. The ProDom database of protein domain families: more emphasis on 3D. *Nucleic Acids Res*, 33: D212-D215. 2005.

[26] P. Bucher, A. Bairoch. A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In Proc. *ISMB-94*. pp. 53-61. AAAI/MIT Press. 1994.

[27] A. Bulashevska, R. Eils. Predicting protein subcellular locations using hierarchical ensemble of Bayesian classifiers based on Markov chains. *BMC Bioinformatics*, 7:298. 2006.

[28] A. Chan, A. A. Freitas. A new ant colony algorithm for multi-label classification with applications in bioinformatics. In Proc. *Genetic and Evolutionary Computation Conference (GECCO-2006)*. pp. 27-34. ACM. 2006.

[29] P.K. Chan, S. J. Stolfo. On the Accuracy of Meta-Learning for Scalable Data Mining. *J. Intell. Inf. Syst*, 8(1). pp. 5-28. 1997.

[30] O. Chapelle, B. Schölkopf, A. Zien. Semi-Supervised Learning. *MIT Press*. 2006.

[31] C. Chen. A PSO-Based Method for Extracting Fuzzy Rules Directly from Numerical Data. *Cybernetics and Systems*, 37(7). pp. 707-723. 2006.

[32] A. Clare, R. D. King. Knowledge discovery in multi-label phenotype data. In Proc. *5th European Conference on Principles and Practice of Knowledge Discovery and Data Mining (PKDD-2001)*, LNAI 2168. pp. 42-53. 2001.

[33] A. Clare. Machine learning and data mining for yeast functional genomics. PhD Thesis. University of Wales Aberystwyth. 2003.

[34] P. Clark, R. Boswell. Rule induction with CN2: Some recent improvements. *Machine Learning - EWSL-91*. pp. 151-163. 1991.

[35] M. Clerc, J. Kennedy. The particle swarm-explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6 (1). pp. 58-73. 2002.

[36] M. Clerc. Particle Swarm Optimization. *ISTE Ltd*. London. 2006.

[37] R. Caruana, A. Niculescu-Mizil. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In Proc. of the *6th ACM SIGMOD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*. pp. 69-78. ACM Press. 2004.

[38] G. Cochrane et. al. Priorities for nucleotide trace, sequence and annotation data capture at the Ensembl Trace Archive and the EMBL Nucleotide Sequence Database. DOI 10.1093/nar/gkm1018. *Nucleic Acids Research Journal*. 2007.

[39] W. W. Cohen. Fast effective rule induction. In Proc. of the 12th Int. Conf. on *Machine Learning (ICML-95)*, A. Prieditis and S. Russell, editors. pp. 115–123. Morgan Kaufmann. 1995.

[40] G. F. Cooper. The Computational Complexity of Probabilistic Inference using Bayesian Belief Networks. *Artificial Intelligence*, 42. pp. 393-405. 1990.

[41] E. S. Correa, A. A. Freitas, C. G. Johnson. Particle swarm for attribute selection in Bayesian classification: an application to protein function prediction. *Journal of Artificial Evolution and Applications*. 2008:12 pages. 2008.

[42] E. P. Costa, A. C. Lorena, A. C. P. L. F. Carvalho, A. A. Freitas, N. Holden. Comparing several approaches for hierarchical classification of proteins with decision trees. *Advances in Bioinformatics and Computational Biology (Proc. Second Brazilian Symposium on Bioinformatics, BSB-2007)*, LNBI 4643. pp. 126-137. Springer. 2007.

[43] R. Daly, Q. Shen. Using Ant Colony Optimisation in Learning Bayesian Network Equivalence Classes. In Proc. *2006 UK Workshop on Computational Intelligence*. pp 111-118. ACTA Press. 2006.

[44] M. O. Dayhoff, R. M. Schwartz, B. C. Orcutt, A model of evolutionary change in proteins. In Dayhoff, M. O. [ed] Atlas of protein sequece and structure, supplement 3. *National Biomedical Research Foundation*. pp. 345-352. 1978.

[45] O. Dekel, J. Keshet,Y. Singer. Large margin hierarchical classification. *ICML'04*. pp. 209-216. 2004.

[46] P. Derbeko, R. E. l.-Yaniv, R. Meir. Variance Optimized Bagging. In Proceedings of the *13th European Conference on Machine Learning*. pp. 60-71. 2002.

[47] T. G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4). pp. 97-136. 1997.

[48] Y. Dong, K. Han. Text classification based on data partitioning and parameter varying ensembles. SAC '05: *Proceedings of the 2005 ACM symposium on Applied computing*. pp. 1044-1048. ACM Press. 2005

[49] M. Dorigo, L.M. Gambardella. Ant colonies for the travelling salesman problem. *Biosystems*, 43. pp. 73-81. 1997.

[50] M. Dorigo, T. Stützle. Ant Colony Optimization. *MIT Press*. 2004.

[51] M. Dorigo, K. Socha. An Introduction to Ant Colony Optimization. To appear in T. F. Gonzalez, *Approximation Algorithms and Metaheuristics*. CRC Press. 2008.

[52] S. Dumais, H. Chen. Hierarchical classification of Web content. In Proc. of the *23rd ACM Int. Conf. on Research and Development in Information Retrieval*. pp. 256–263. 2000.

[53] S. Eddy. HMMER User's Guide - Biological sequence analysis using profile hidden Markov models. *http://hmmer.wustl.edu/*. 2003.

[54] A.E. Eiben, J.E. Smith. Introduction to Evolutionary Computing. Springer. *Natural Computing Series*. 2nd edition. 2007.

[55] The ENCODE Project Consortium et al. Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. *Nature*. pp. 799-816. 2007.

[56] R. Eisner, B,.Poulin, D. Szafron, P. Lu, R. Greiner. Improving Protein Function Prediction using the Hierarchical Structure of the Gene Ontology. In Proc. 2005 *IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology*. 2005.

[57] W. J. Ewens, G. R. Grant. Statistical Methods in Bioinformatics: An Introduction, Second Edition. *Springer*. 2005.

[58] I. D. Falco, A. D. Cioppa, E. Tarantino. Facing classification problems with Particle Swarm Optimization. *Appl. Soft Comput*, 7(3). pp. 652-658. 2007.

[59] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth. From data mining to knowledge discovery: an overview. *Advances in Knowledge Discovery and Data Mining*. pp. 1-34. AAAI/MIT. 1996.

[60] D. Fillmore. It's a GPCR World. Modern Drug Discovery, 11(7). pp 24-28. 2004

[61] A. A. Freitas. Data Mining and Knowledge Discovery with Evolutionary Algorithms. *Springer*. August 2002.

[62] A. A. Freitas and A. C. P. L. F. de Carvalho. A Tutorial on Hierarchical Classification with Applications in Bioinformatics. In. D. Taniar (Ed.) *Research and Trends in Data Mining Technologies and Applications*. pp. 175-208. Idea Group. 2007.

[63] A. A. Freitas, R. S. Parpinelli, H. S. Lopes. Ant Colony Algorithms for Data Classification. To appear in: M. Khosrou-Pour (Ed.) *Encyclopedia of Information Science and Technology,* 2nd Ed. Idea Group. 2008.

[64] Y. Freund, R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, Academic Press, Inc., 55. pp. 119-139. 1997.

[65] G. Fung, S. Sandilya, and R. Bharat Rao. Rule extraction from linear support vector machines. In Proc. of the *11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD-05)*. pp. 32–40. 2005.

[66] J. Furnkranz and G. Widmer. Incremental reduced error pruning. In Proc. the *11th Int. Conf. on Machine Learning (ICML-94)*. pp. 70–77. 1994.

[67] M. Galea, Q. Shen. Linguistic Hedges for Ant-Generated Fuzzy rules. In. Proc. *IEEE International Conference on Fuzzy Systems*. pp. 1973–1980. 2006.

[68] M. Galea, Q. Shen. Simultaneous Ant Colony Optimization Algorithms for Learning Linguistic Fuzzy Rules. In. *Swarm Intelligence in Data Mining*. pp. 75–100. Springer. 2006.

[69] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics 25*. pp. 25-29. 2000.

[70] GPCRDB. *http://www.gpcr.org/*. Visited December 2007.

[71] S. Günter, H. Bunke. Evaluation of Classical and Novel Ensemble Methods for Handwritten Word Recognition. In Proc. *10th Int. Workshop on Structural and Syntactic Pattern Recognition (SSPR)*. pp. 583-591. 2004.

[72] S. Günter, H. Bunke. Optimization of Weights in a Multiple Classifier Handwritten Word Recognition System Using a Genetic Algorithm. *ELCVIA*(3). No. 1. pp. 25-44. 2004.

[73] Y. Z. Guo, M. L. Li, K. L. Wang, Z. N. Wen, M. C. Lu, L. X. Liu, L. Jiang, Classifying G protein-coupled receptors and nuclear receptors on the basis of

protein power spectrum from fast Fourier transform. *Amino Acids*, 30(4). pp. 397-402. Epub. 2006.

[74] D. J. Hand. Construction and Assessment of Classification Rules. *Wiley*. 1997.

[75] R. Hassan, B. Cohanim, O. L de Weck, Venter G. A Comparison of Particle Swarm Optimization and the Genetic Algorithm. *AIAA Multidisciplinary Design Optimization Specialist Conference*. 2005.

[76] Y. Huang, J. Cai, L. Ji, Y. Li. Classifying G-protein coupled receptors with bagging classification tree. *Computational Biology and Chemistry*, 28(4). pp. 275-280. 2004.

[77] S. Henikoff, J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci*. pp. 10915-10919. 1992.

[78] P. G. Higgs, T. K. Attwood. Bioinformatics and Molecular Evolution. Blackwell. 2005.

[79] N. Holden, A. A. Freitas. A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In Proc. *2005 IEEE Swarm Intelligence Symposium (SIS-05)*. pp. 100-107. IEEE. 2005.

[80] N. Holden, A. A. Freitas. Hierarchical Classification of G-Protein-Coupled Receptors with a PSO/ACO Algorithm. In Proc. *IEEE Swarm Intelligence Symposium (SIS-06)*. pp. 77-84. IEEE, 2006.

[81] N. Holden, A. A. Freitas. A hybrid PSO/ACO algorithm for classification. In Proc. *Genetic and Evolutionary Computation Conference (GECCO-2007)* Workshop on Particle Swarms: The Second Decade. pp 2745-2750. ACM. 2007.

[82] N. Holden, A. A. Freitas. Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation. *Soft Computing Journal, special issue on Evolutionary and Metaheuristics-based Data Mining*. 14 pages. doi:10.1007/s00500-008-0321-0. Springer. 2008.

[83] N. Holden, A. A. Freitas. Improving the performance of hierarchical classification with swarm intelligence. In Proc. *6th European Conf. on*

Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio-2008). LNCS 4973. pp. 48-60. Springer. 2008.

[84] N. Holden, A. A. Freitas. A Hybrid PSO/ACO Algorithm for Discovering Classification Rules in Data Mining. *Journal of Artificial Evolution and Applications (JAEA), special issue on Particle Swarms: The Second Decade.* Article ID 316145. 11 pages. doi:10.1155/2008/316145. 2008.

[85] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. S. Langendijk-Genevaux, M. Pagni, C. J. A. Sigrist. The PROSITE database. *Nucleic Acids Res.* 34:D227-D230. 2006.

[86] InterPro. *http://www.ebi.ac.uk/InterPro/* .Visited December 2007.

[87] H. Jacobsson. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17. pp. 1223–1263. 2005.

[88] L.J. Jensen, R. Gupta, N. Blom, D. Devos, J. Tamames, C. Kesmir, H. Nielsen, H. H. Staerfeldt, K. Rapacki,C. Workman,C. A. F. Andersen, S. Knudsen, A. Krogh, A. Valencia, S. Brunak. Prediction of human protein function from posttranslational modifications and localization features. *J. Mol. Biol*, 319. pp. 1257-1265. 2002.

[89] R. Jensen, Q. Shen. Fuzzy-rough sets assisted attribute selection. *IEEE Transactions on Fuzzy Systems*, 15(1):73-89. 2007.

[90] R. Jensen, Q. Shen. Computational Intelligence and Feature Selection: Rough and Fuzzy Approaches. *IEEE Press and Wiley & Sons*. 2008.

[91] A. K. Kamrani, E. S. Abouel Nasr. Data-Mining Process Overview. *Springer US*. pp. 89-102. 2008.

[92] R. Karchin, K. Karplus, D. Haussler. Classifying G-protein coupled receptors with support vector machines. *Bioinformatics*, 18(1). pp. 147-59. 2002.

[93] J. Kennedy, R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In Proc. of the *1997 Conference on Systems, Man, and Cybernetics*. pp. 4104-4109. 1997.

[94] J. Kennedy, W. Spears. Matching Algorithms to Problems: An experimental Test of the Particle Swarm and some Genetic Algorithms on the Multimodal Problem Generator. *IEEE International Conference on Evolutionary Computation*. pp. 78-83. 1998.

[95] J. Kennedy and R. C. Eberhart, Y. Shi. *Swarm Intelligence*. San Francisco: Morgan Kaufmann/Academic Press. 2001.

[96] J. Kennedy, R. Mendes. Population structure and particle swarm performance. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*. Honolulu, Hawaii USA. IEEE. 2002.

[97] A. Krogh, M. Bron, I.S. Mian, K. Sjolander, D. Haussler. Hidden Markov models in computational biology: Applications to protein modelling. *J. Mol. Biol*, 235. pp. 1501–1531. 1994.

[98] S. Larson, C. Snow, M. Shirts, V. Pande. Folding@home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. In Grant, R. (Ed.). *Computational Genomics: Theory and Application*. Horizon Press. 2003.

[99] M. Lapinsh, P. Prusis, S. Uhlén, J.E.S Wikberg. Improved approach for proteochemometrics modeling: application to organic compound - amine G protein-coupled receptor interactions. *Bioinformatics*, 21. pp. 4289-4296. 2005.

[100] D. Latek, D. Ekonomiuk, A. Kolinski. Protein structure prediction: combining de novo modeling with sparse experimental data. *J. Comput. Chem*. 28(10). pp. 1668-1676. 2007.

[101] M. H. Li, X. L. Wang, L. Lin, T. Liu. Protein-protein interaction site prediction based on conditional random fields. *Bioinformatics*, 23(5). pp. 597-604. 2007.

[102] J. S Mattick, I. V. Makunin,. Non-coding RNA. *Human Molecular Genetics*, 15. pp. 17-29. 2006.

[103] J. McDowall, InterPro: Exploring a Powerful Protein Diagnostic Tool. *ECCB05*. Tutorial. pp 14. 2005.

[104] E. Mezura-Montes, J. Velázquez-Reyes, C. Coello Coello, A comparative study of differential evolution variants for global optimization. In Proceedings of *Genetic and Evolutionary Computation Conference (GECCO '06)*. ACM, 2006.

[105] A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review*, vol. 63. pp. 81-97. 1956.

[106] T. Mitchell. Machine Learning. *Mc Graw Hill.* 1997.

[107] R. Mouser, S. A. Dunn. Comparing genetic algorithms and particle swarm optimisation for an inverse problem exercise. *Computational Techniques and Applications Conference (CTAC-2004).* pp 89-101. 2005.

[108] S. Mukhopadhyay, A. Mandal. Fuzzy Rule Extraction Using Robust Particle Swarm Optimization. *Advances in Neural Networks - ISNN 2006.* pp. 762-767. 2006.

[109] N. J. Mulder et al. New developments in the InterPro database. *Nucleic Acids Res. 35 (Database Issue)*:D224-D228. 2007.

[110] S. B. Needleman, C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology 48.* pp. 443–53. 1970.

[111] D. J. Newman, S. Hettich, C. L. Blake, C. J. Merz. UCI Repository of machine learning databases. *http://www.ics.uci.edu/~mlearn/MLRepository.html.* CA: University of California, Department of Information and Computer Science. 1998.

[112] D. Nguyen, T. A. Dung, T. H. Cao. Text Classification for DAG Structured Categories. *Advances in Knowledge Discovery and Data Mining*, LNCS. pp. 290-300. 2005.

[113] H. Nunez, C. Angulo, and A. Catala. Rule extraction from support vector machines. In Proc. of the *European Symposium on Artificial Neural Networks (ESANN-02)*, pp. 107-112, 2002.

[114] C. Orengo, D. Jones, J Thornton. Bioinformatics: Genes, Proteins and Computers. *BIOS Scientific Publishers Ltd.* 2003.

[115] A. Papagelis, D. Kalles. Breeding decision trees using evolutionary techniques. *ICML 2001*. 2001.

[116] P. K. Papasaikas, P. G. Bagos, Z. I. Litou, S. J. Hamodrakas. A novel method for GPCR recognition and family classification from sequence alone using signatures derived from profile hidden Markov models. *SAR QSAR Environ Res*, 14(5-6). pp. 413-20. 2003.

[117] G. L. Pappa, A. J. Baines and A. A. Freitas. Predicting post-synaptic activity in proteins with data mining. *Bioinformatics 21* (2). pp. ii19-ii25. 2005.

[118] G. L. Pappa. Automatically Evolving Rule Induction Algorithms with Grammar-Based Genetic Programming. PhD Thesis. University of Kent. 2007.

[119] R. S. Parpinelli, H. S. Lopes and A. A. Freitas. Data Mining with an Ant Colony Optimization Algorithm, *IEEE Trans. on Evolutionary Computation, special issue on Ant Colony algorithms*, 6(4). pp. 321-332. 2002.

[120] J. Parry-Smith, T. K. Attwood: ADSP - a new package for computational sequence analysis. *Computer Applications in the Biosciences* 8(5). pp. 451-459. 1992.

[121] W. R. Pearson, D. J. Lipman: Improved tools for biological sequence comparison, Proc. *Natl. Acad. Sci.* USA, 85. pp. 2444-2448. 1988.

[122] T. Peng, W. Zuo, F. He. Text Classification from Positive and Unlabeled Documents Based on GA. *VECPAR06* (7). 2006.

[123] Pfam. *http://www.sanger.ac.uk/Software/Pfam/.* Visited December 2007.

[124] R. Poli, J. Kennedy, T. Blackwell. Particle Swarm Optimization: An Overview. *Swarm Intelligence*. pp. 33-57. 2007.

[125] A. Porollo, J. Meller. Prediction-based fingerprints of protein-protein interactions. *Proteins,* 66(3). pp. 630-645. 2007.

[126] Prosite. *http://www.expasy.ch/prosite/.* Visited 2007.

[127] K. V. Price, R. M. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization. *Springer*. 2005.

[128] K. V. Price, R. M. Storn. Differential Evolution Java Implementation. *http://www.icsi.berkeley.edu/~storn/code.html*. Visited on July. 2007.

[129] Prints. *http://www.bioinf.manchester.ac.uk/dbbrowser/PRINTS/*. Visited December 2007.

[130] D. Pyle. Data Preparation for Data Mining. *Morgan Kaufmann*. 1999.

[131] R. Quinlan. C4.5: programs for machine learning. *Morgan Kaufmann*. 1993.

[132] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In Proc. of the IEEE. 77 (2). p. 257–286. 1989

[133] R. Ranawana, V. Palade. MVGen - Ensemble Learning for MCS Majority voting with a Genetic Algorithm. Internal Report. *Oxford University Computing Laboratory*. 2005.

[134] T. K. Rasmussen, T. Krink. Improved Hidden Markov Model training for multiple sequence alignment by a particle swarm optimization-evolutionary algorithm hybrid. In: *Biosystems*, 72(1-2). pp 5-17. 2003.

[135] I. Rish. An empirical study of the naive Bayes classifier. IJCAI 2001 *Workshop on Empirical Methods in Artificial Intelligence*. 2001.

[136] B. Rost, G. Yachdav, J. Liu. The PredictProtein Server. *Nucleic Acids Research* 32(Web Server issue). W321-W326. 2004.

[137] J. Rousu, C. Saunders, S. Szedmak, J. Shawe-Taylor. Learning hierarchical multi-category text classification models. In Proc. *ICML '05: Proceedings of the 22nd international conference on Machine learning*. ACM. pp. 744-751. 2005.

[138] M. Sasaki, K. Kita. Rule-based text categorization using hierarchical categories. In Proc. of the *IEEE Int. Conf. on Systems, Man, and Cybernetics*. pp. 2827–2830. 1998.

[139] P. Scordis, D. R. Flower, T. K. Attwood. FingerPRINTScan: intelligent searching of the PRINTS motif database. *Bioinformatics* 15(10). pp. 799-806. 1999.

[140] A. Secker, M. N. Davies, A. A. Freitas, J. Timmis, M. Mendao, D. Flower. An experimental comparison of classification algorithms for the hierarchical prediction of protein function. *Expert Update (the BCS-SGAI Magazine). Vol. 9, No. 3, Special Issue on the 3rd UK KDD Workshop*. pp. 17-22. 2007.

[141] Q. Shen, A. Chouchoulas. A rough-fuzzy approach for generating classification rules. *Pattern Recognition*. 35(11):2425-2438. 2002.

[142] Q. Shen, R. Jensen. Selecting informative features with fuzzy-rough sets and its application for complex systems monitoring. *Pattern Recognition*, 37(7):1351-1363, 2004.

[143] L. Sheneman, J. A. Foster. Evolving Better Multiple Sequence Alignments. *Genetic and Evolutionary Computation. GECCO-2004*. pp. 449-460. 2004.

[144] D. B. Skalak. Prototype Selection for Composite Nearest Neighbour Classifiers. PhD Thesis. University of Massachusetts, Amherst, Massachusetts. 1997.

[145] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences. *J. Mol. Biol.* 147. pp. 195-197. 1981.

[146] T. Sousa, A. Silva, A. Neves. Particle Swarm based Data Mining Algorithms for classification tasks. *Parallel Computing 30*. pp. 767–783. 2004.

[147] C. D. Stefano, A. D. Cioppa, A. Marcelli. Exploiting Reliability for Dynamic Selection of Classifiers by Means of Genetic Algorithms. In Proc. of the *Seventh international Conference on Document Analysis and Recognition - Volume 2*, ICDAR. IEEE Computer Society. 2003.

[148] J. Struyf, S. Dzeroski, H. Blockeel, A. Clare. Hierarchical multi-classification with predictive clustering trees in functional genomics. In Proc. *Progress in Artificial Intelligence: 12th Portugese Conference on Artificial Intelligence, EPIA 2005*, vol 3808. pp. 272-283, 2005.

[149] A. Sun, E.-P. Lim. Hierarchical text classification and evaluation. In Proceedings of the *1st IEEE International Conference on Data Mining*. IEEE Computer Society Press. pp. 521-528. 2001.

[150] A. Sun, E.-P. Lim, W.-K. Ng. Performance measurement framework for hierarchical text classification. *Journal of the American Society for Information Science and Technology* 54(11). pp. 1014-1028. 2003.

[151] A. Sun, E.-P. Lim, W.-K. Ng. Hierarchical text classification methods and their specification. In: A.T.S. Chan, S.C.F. Chan, H.V. Leong, V.T.Y. Ng. (Eds.) *Cooperative Internet Computing*. pp. 236-256. 2003.

[152] A. Sun, E.-P. Lim, W.-K. Ng. Blocking reduction strategies in hierarchical text classification. *IEEE Transactions on Knowledge and Data Engineering* 16(10). pp. 1305-1308. 2004.

[153] A. Tan, D. Gilbert, Y. Deville. Multi-class protein fold classification using a new ensemble machine learning approach. *Genome Informatics*, 14. pp. 206-217. 2003.

[154] J. D. Thompson, D. G. Higgins, T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22. pp. 4673-4680. 1994.

[155] J. D. Thompson, F. Plewniak, O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res*. J13: pp. 2682-2690. 1999.

[156] TrEMBL. *http://www.ebi.ac.uk/swissprot/sptr_stats/full/index.html.* Visited June 2007.

[157] UniProt. *http://www.expasy.UniProt.org/.* Visited December 2007.

[158] G. Valentini, N. Cesa-Bianchi. HCGene: a software tool to support the hierarchical classification of genes. *Bioinformatics* Vol. 24. pp. 729-731. 2008.

[159] R. Vilalta, Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*. 18(2). pp. 77-95. 2002.

[160] J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13(2). pp. 260–269. 1967.

[161] K. Wang, S. Zhou, and Y. He. Hierarchical classification of real life documents. In Proc. of the *1st SIAM Int. Conf. on Data Mining*, Chicago, 2001.

[162] W. Weinert, H. Lopes. Neural networks for protein classification. *Applied Bioinformatics*, 3(1). pp. 41-48. 2004.

[163] S. M. Weiss and C. A. Kulikowski. Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. *Morgan Kaufmann Publishers*. 1991.

[164] I. H. Witten, E. Frank, Data Mining: Practical machine learning tools and techniques, 2nd Edition, *Morgan Kaufmann*, San Francisco. CA. 2005.

[165] D. H. Wolpert. Stacked Generalization. *Neural Networks*. Vol. 5. pp. 241-259. 1992.

[166] C. H. Wu, R. Apweiler, A. Bairoch, D. A. Natale, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, R. Mazumder, C. O'Donovan, N. Redaschi, B. E. Suzek: The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Research* 34(Database-Issue). pp. 187-191. 2006.

[167] L. Yang, Z. Qin. Combining Classifiers with Particle Swarms. *ICNC* (2). pp. 756-763. 2005.

[168] E. M. Zdobnov, R. Apweiler. InterProScan - an integration platform for the signature-recognition methods in InterPro. *Bioinformatics*. 17(9). pp. 847-8. 2001.

[169] XPS Project. *http://xps-swarm.essex.ac.uk/*. Visited July 2008.

Publications from the work in this Thesis

Journal Papers

- N. Holden, A. A. Freitas. A Hybrid PSO/ACO Algorithm for Discovering Classification Rules in Data Mining. *Journal of Artificial Evolution and Applications (JAEA), special issue on Particle Swarms: The Second Decade.* Article ID 316145. 11 pages. doi:10.1155/2008/316145. 2008.
- N. Holden, A. A. Freitas. Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation. *Soft Computing Journal, special issue on Evolutionary and Metaheuristics-based Data Mining.* 14 pages. doi:10.1007/s00500-008-0321-0. Springer. 2008.

Conference Papers

- N. Holden, A. A. Freitas. Improving the performance of hierarchical classification with swarm intelligence. In Proc. *6th European Conf. on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio-2008).* LNCS 4973. pp. 48-60. Springer. 2008. **Won best paper award at this conference.**
- E. P. Costa, A. C. Lorena, A. C. P. L. F. Carvalho, A. A. Freitas, N. Holden. Comparing several approaches for hierarchical classification of proteins with decision trees. In Proc. of the *2007 Brazilian Symposium on Bioinformatics (BSB-2007).* LNBI 4643. pp. 126-137. Springer. 2007.

- N. Holden, A. A. Freitas. Hierarchical Classification of G-Protein-Coupled Receptors with a PSO/ACO Algorithm. In Proc. *IEEE Swarm Intelligence Symposium (SIS-06)*, pp. 77-84. IEEE. 2006.
- R. Poli, W. B. Langdon, P. Marrow, J. Kennedy, M. Clerc, D. Bratton, N. Holden. Communication, Leadership, Publicity and Group Formation in Particle Swarms. In Proc. of *ANTS '06*. LNCS 4150. pp. 132-143. Springer. 2006.
- N. Holden, A. A. Freitas. A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In Proc. *2005 IEEE Swarm Intelligence Symposium*. pp. 100-107. IEEE. 2005.

Workshop Paper

- N. Holden, A. A. Freitas. A hybrid PSO/ACO algorithm for classification. In Proc. of the *GECCO-2007 Workshop on Particle Swarms: The Second Decade*. pp. 2745-2750. ACM Press. 2007.

Appendix A. Comparing the Performance of PSO/ACO and Binary PSO in Benchmark Binary Optimisation Problems

In this appendix we present some preliminary results comparing the performance of the "standard" Binary-PSO (BPSO) [95] (briefly reviewed in Section 3.4) to the PSO/ACO algorithm proposed in Section 4.3 in several simple benchmark functions involving optimisation problems with binary variables (unrelated to the data mining problem). Both algorithms used 100 particles with Von-Neumann topology. Note that Von-Neumann topology does not always lead to the fastest particle convergence. However, our preliminary experiments demonstrated that it provided the most consistent results across both the easier and more difficult problems. Neither algorithm was particularly optimised for these specific problems. For PSO/ACO (as it standard) α was set to 1, and the minimum and maximum pheromone level were set to 0.01. The BPSO algorithm uses the standard update equation:

$$v_{id} = v_{id} + c_1\text{Rand}()(p_{id} - x_{id}) + c_2\ \text{Rand}()\ (p_{gd} - x_{id})$$

Notice that a constriction coefficient is not used as it significantly reduced BPSO's performance. *k* was set to 2 as it was found to be the best overall in initial testing – and a significant improvement over *k = 1*. *c1* and *c2* were set to 2.05 as is usual in the literature. *VMax* was set to 6 as per the original algorithm [93]. Rand() produces a random number between 0 and 1.

The benchmark problems have the following details where x is a candidate binary solution with a number of dimensions (bits) d and the value to maximise is returned by the functions defined by the corresponding pseudocode:

- **OneMax** – The highest score is the bitstring with all 1's. For instance, a 2-bit instance of the problem, the bitstring 01 would have the score 0.5

```
$score = 0, toAdd = 1 \div d$
FOR EACH $x_j$
    IF $x_j = 1$
        $score = score + toAdd$
    END IF
LOOP
RETURN $score$
```

Pseudocode A.1: One Max

- **OneFirst** – The highest scoring bitstring is all 1's but the scoring starts at the left hand side and stops when a 0 is detected. For instance 1011 would have the score 0.25.

```
$score = 0, toAdd = 1 \div d$
FOR EACH $x_j$
    IF $x_j = 1$
        $score = score + toAdd$
    ELSE
        RETURN $score$
    END IF
LOOP
RETURN $score$
```

Pseudocode A.2: One First

- **OneMaxNoisy** – The same as OneMax, but 1% noise is added to the fitness evaluation, swapping a bit 1% of the time.

```
$score = 0, toAdd = 1 \div d$
FOR EACH $x_j$
    IF $x_j = 1$
        // Random() returns a number in the range [0..1] with uniform probability
        //distribution
        IF Random() $\leq 0.99$
        $score = score + toAdd$
        END IF
    END IF
LOOP
RETURN $score$
```

Pseudocode A.3: One Max Noisy

- **AllSame** – The highest scoring bitstring is either all 1's or all 0's, the maximum scoring bitstring is taken. For instance, in a 4-bit problem 0111 and 1000 would score the same (0.75).

```
$scoreA = 0, scoreB = 0, toAdd = 1 \div d$
FOR EACH $x_j$
    IF $x_j = 1$
        $scoreA = scoreA + toAdd$
    ELSE
        $scoreB = scoreB + toAdd$
    END IF
LOOP
RETURN $max(scoreA, scoreB)$
```

Pseudocode A.4: All Same

- **AllSameFirst** – The highest scoring bitstring is either all 1's or all 0's, but scoring starts at the left hand side of the bitstring and stops when the next bit is not the same as the current bit. For instance, 0101 would have the score 0.25, and 0010 would have the score 0.5.

```
$score = 0$, $toAdd = 1 \div d$
FOR EACH $x_j$ FROM $x_1$ TO $x_{d-1}$
    IF $x_j = x_{j+1}$
        $score = score + toAdd$
    ELSE
        RETURN $score$
    END IF
LOOP
RETURN $score$
```

Pseudocode A.5: All Same First

Table A.1 shows the results for both algorithms in the labelled benchmark functions with each function having 200 dimensions (bits). Each cell provides the average number of iterations it took each approach to find the optimal bitstring. Both PSO/ACO and BPSO were limited to a maximum of 2000 iterations for simplicity; so that if the optimum is not reached within 2000 iterations, 2000 is used as score for that run. Note that, since both algorithms use the same population size, the number of iterations required to reach the optimum is a fair measure of their relative cost-effectiveness. Each experiment was repeated 100 times (the average performance is provided in each cell), with standard deviations shown after the "±" symbol and the results of the WEKA two tailed t-tests (significance 1%) indicated by shadings. Light grey indicates a statistically significant win, so in every test PSO/ACO performs significantly better than BPSO.

	BPSO	PSO/ACO
AllSame	76.81±6.69	65.92±8.38
AllSameFirst	1854.38±323.78	706.97±55.41
OneFirst	1057.77±351.91	693.92±51.56
OneMax	64.35±2.47	56.04±2.64
OneMaxNoisy	76.67±4.88	68.68±5.39

Table A.1: Number of iterations to find the maximum valued bitstring in each benchmark problem for PSO/ACO and Binary PSO

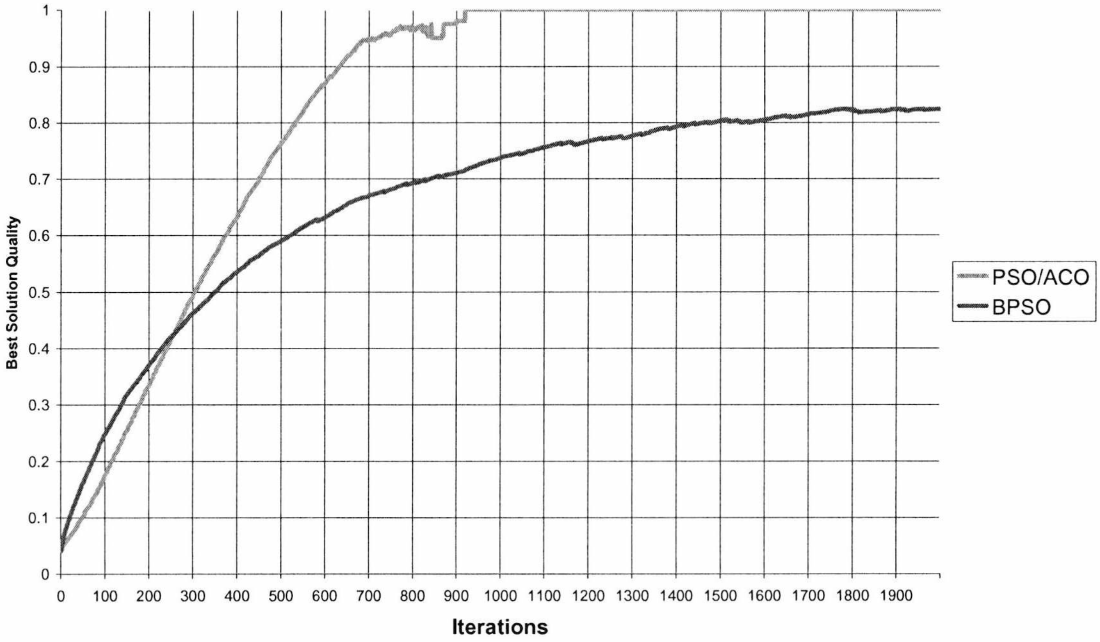


Figure A.1: The average performance of the best particle during 100 runs of the All Same First benchmark function, for both PSO/ACO and BPSO

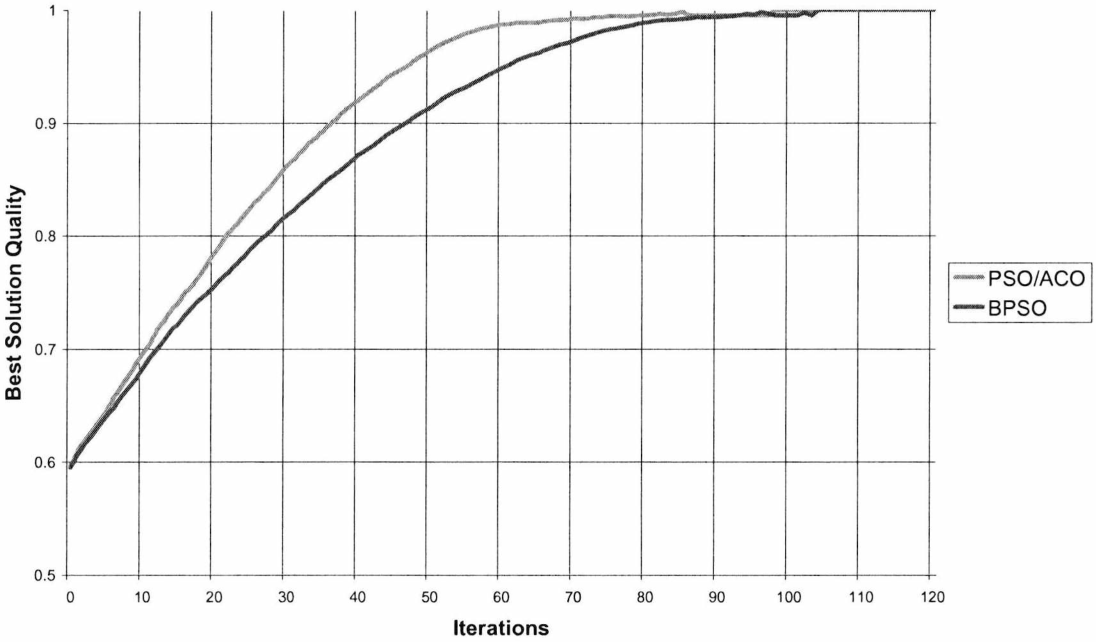


Figure A.2: The average performance of the best particle during 100 runs of the All Same benchmark function, for both PSO/ACO and BPSO

Most of the problems do not take the algorithms many iterations to solve. As expected the trickier "All Same First" and "One First" problems take the most number of iterations to solve. As can be seen by the high standard deviations of the results for BPSO (Table A.1) it was not able to solve the aforementioned problems in a very consistent number of iterations (or indeed within the maximum number of iterations in some cases). Note that, if the maximum number of iterations was exceeded during a run then the number of iterations reported for this run was set to this maximum number.

In Figure A.1 the different performance characteristics of PSO/ACO and BPSO in the "All Same First" function can be seen. BPSO finds better solutions at the beginning of the run, but after about 250 iterations PSO/ACO overtakes BPSO. This may be a sign that BPSO is converging too quickly and getting trapped too easily in local maxima.

The performance of both algorithms in the simpler “All Same” problem is shown in Figure A.2, where it can be seen that the same sort of behaviour is not present in this problem. However, notice that the qualities of the solutions are higher at the start of the “All Same” problem when compared to the “All Same First” problem. It is likely that the type of behaviour seen in both Figure A.1 and Figure A.2 can be attributed to the way in which PSO/ACO uses the fitness of the best solution each particle has found so far to add to the pheromone entries. This means that in the Figure A.1, at the beginning of the run, the solutions found so far are not of high quality and so not much pheromone is added to the entries. However, in Figure A.2 the solutions found are reasonably good at the beginning of the run and so more pheromone is added. This means that convergence is speeded up in the case of the easier “All Same” problem, and slowed down in the “All Same First” problem.

Despite the simplicity of these benchmark functions we believe that they add to the evidence that PSO/ACO is an effective optimiser. Furthermore, the results show that PSO/ACO is more consistent (when compared to BPSO) in the number of iterations it requires to find the maximum quality combination of bits, with it rarely seeming to get trapped in a local optimum (if it did it would cause large standard deviations). This is a useful feature to have in an optimiser for both these benchmark problems and the data mining problems investigated in this thesis.