NEW ANT COLONY OPTIMISATION ALGORITHMS FOR HIERARCHICAL CLASSIFICATION OF PROTEIN FUNCTIONS

a thesis submitted to
The University of Kent at Canterbury
in the subject of computer science
for the degree
of doctor of philosophy.

By
Fernando Esteban Barril Otero
January 2010

Abstract

Ant colony optimisation (ACO) is a metaheuristic to solve optimisation problems inspired by the foraging behaviour of ant colonies. It has been successfully applied to several types of optimisation problems, such as scheduling and routing, and more recently for the discovery of classification rules. The classification task in data mining aims at predicting the value of a given goal attribute for an example, based on the values of a set of predictor attributes for that example. Since real-world classification problems are generally described by nominal (categorical or discrete) and continuous (real-valued) attributes, classification algorithms are required to be able to cope with both nominal and continuous attributes. Current ACO classification algorithms have been designed with the limitation of discovering rules using nominal attributes describing the data. Furthermore, they also have the limitation of not coping with more complex types of classification problems—e.g., hierarchical multi-label classification problems.

This thesis investigates the extension of ACO classification algorithms to cope with the aforementioned limitations. Firstly, a method is proposed to extend the rule construction process of ACO classification algorithms to cope with continuous attributes directly. Four new ACO classification algorithms are presented, as well as a comparison between them and well-known classification algorithms from the literature. Secondly, an ACO classification algorithm for the hierarchical problem of protein function prediction—which is a major type of bioinformatics problem addressed in this thesis—is presented. Finally, three different approaches to extend ACO classification algorithms to the more complex case of hierarchical multi-label classification are described, elaborating on the ideas of the proposed hierarchical classification ACO algorithm. These algorithms are compare against state-of-the-art decision tree induction algorithms for hierarchical multi-label classification in the context of protein function prediction.

The computational results of experiments with a wide range of data sets—including challenging protein function prediction data sets with very large number of class labels—have shown that the proposed ACO classification algorithms are competitive to well-known classification algorithms from the literature, for both conventional (flat single-label) classification and hierarchical multi-label classification problems. These algorithms address unexplored research areas in the context of ACO classification algorithms to the best of our knowledge, and are therefore original contributions.

Acknowledgements

Firstly, I would like to thank my supervisors Alex Freitas and Colin Johnson for the opportunity of undertaking this work, and for their excellent supervision, support and guidance throughout my PhD.

Furthermore, I owe a big thank you to my family, for always being there. A special thanks to my parents for the opportunities and the support they have given me in all my endeavours. I am also very grateful to Carin Tunåker for all her help, patience, love and support.

Many friends made my time in Canterbury highly enjoyable. I would like to thank Abigail Smith, Beatriz Martinez, Ben Robert, Claire Harris, Cristiano Cantore, Dolores Castaneda, Eva Barbazan, Gisele Pappa, Javier Valbuena, Mariola Esteban and Miguel Leon-Ledesma for all the time that we spent together. My thanks also go to my friends and colleagues from the School of Computing at the University of Kent, in special to Carlos Silla, Lingfang Du, Mudassar Iqbal, Sebastian Marion and Siddhartha Ghosh.

Finally, I would like to gratefully acknowledge the financial support received from the European Union's INTERREG project (Ref. No. 162/025/361) as well as the continuous financial support received from the School of Computing at the University of Kent, during the course of my PhD.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The exponential increase in the data available in biological databases has brought biologists, mathematicians and computer scientists together, working on the creation of computational and statistical techniques and algorithms to support the analysis of biological data—an area which is now referred to as bioinformatics [60, 78]. The aim of this thesis is to present an interdisciplinary research encompassing three different areas, namely data mining, ant colony optimisation and bioinformatics. The thesis investigates the design and implementation of new ant colony optimisation algorithms for the classification task of data mining, in the context of the important bioinformatics problem of protein function prediction.

Data mining is a research area concentrated on designing and employing computational methods to discover (learn) a model (based on a given knowledge representation) from real-world structured data [45, 96]. Since the advances in computer technologies, allowing the storage of virtually any kind of data, have led to an exponential growth of available information, (semi-)automated data analysis techniques and methods have received increased attention. One of the most studied data mining tasks in the literature is the classification task. In essence, the classification task consists of learning a predictive relationship between input values and a desire output. Each example (data instance or record) is described by a set of features (attributes)—referred to as predictor attributes—and a class attribute. Given a set of examples, a classification algorithm aims at creating a model, which represents the relationship between predictor attributes values and class values, and which is able to predict the class of an example based on the values of its predictor attributes.

In the vast majority of the classification problems addressed in the literature, each example is associated with one class value (label) and there are no relationships—i.e. hierarchical relationships—between the different class values. These problems are referred to as *flat single-label* classification problems. However, there are more complex classification problems where class values are organised in a hierarchical structure—defining parent/child relationships between different class values—and examples may be associated with more than one class value at the same time. In the latter case these classification problems are referred to as *hierarchical multi-label* classification problems.

There has been an increasing interest in hierarchical multi-label classification, where early applications have generally been found in text classification [21, 104, 117, 119, 129] and also more recently in the problem of protein function prediction in bioinformatics [8, 13, 24, 89, 127]. The latter is a very active research area, given the large increase in the number of uncharacterised (i.e. with unknown function) proteins available for analysis and the importance of determining their functions in order to improve the current biological knowledge. Proteins are large and complex molecules, assembled from amino acids arranged in a linear sequence using information encoded in genes. They perform most of the functions within a cell and make up the majority of cellular structures.

Determining protein functions is a central goal of bioinformatics and it is crucial for improving biological knowledge, diagnosis and treatment of diseases. Biologists are provided with information about genome sequences, genes and their protein products, but one question still remains—given a protein sequence, what is its function? While biological experiments are the ultimate method to determine the function of proteins, it is not possible to perform a functional assay for every uncharacterised protein. A commonly used approach to predict the function of uncharacterised proteins is to assign a function based on sequence similarity—i.e. if there is a protein with known function similar to the uncharacterised protein, in terms of their amino acid sequences, the function of the former protein is assigned to the uncharacterised protein. It has been shown that the similarity-based approach presents several limitations [51, 52] and it is even considered one of the sources of functional assignment errors found in biological databases [19, 66].

An alternative approach to predict the function of uncharacterised proteins is to induce a classification model from the data about proteins with known functions. In the model-based approach, the protein function prediction problem is cast as a classification problem—proteins correspond to examples to be classified, protein features correspond to predictor attributes and the different functions that a protein can perform correspond to class labels to be predicted—and a model is induced by a classification algorithm. Since it is known that a protein can perform more than one function and protein function definitions are usually organised in a hierarchical structure, the classification problem in this case is an instance of a hierarchical multi-label problem. It is important to emphasise that in the context of protein function prediction, comprehensible classification models—which can be interpreted and validated by the user—are preferred in order to provide useful insights about the correlation of protein features and their functions [51].

Classification problems can be viewed as optimisation problems, where the goal is to find the best model that represents the predictive relationships in the data. A classification problem can be formally specified as: given training data consisting of pairs $\{(e_1, c_1), \ldots, (e_n, c_n)\}$, find a function that maps each example e_i to its correspondent class label c_i, where $1 \leq i \leq n$ and n is the total number of training examples. A wide range of different paradigms of classification algorithms have been used in the literature [45, 48, 85, 133]—e.g, statistical algorithms, neural networks, decision tree induction, evolutionary algorithm and rule induction—and more recently ant colony optimisation [39].

Ant colony optimisation (ACO) is a metaheuristic inspired by the foraging behaviour of ant colonies [37, 38, 39]. Ant colonies, despite the lack of centralised control and the relative simplicity of their individuals' behaviours, are self-organised systems that can accomplish complex tasks by having their individual ants interacting with one another and with their environment. Many ant species, even with limited visual capabilities or entirely blind, are able to find the shortest path between a food source and the nest. While walking from the nest to a food source and vice-versa, ants deposit a chemical substance—called pheromone—on the ground, creating a pheromone trail. The more ants use the same path, the more pheromone is deposited on the trail. Since ants probabilistically choose a path to follow based on its pheromone concentration, the path with higher concentration has a greater chance of being (re-)used—which in general represents the shortest path between the nest and the food source. ACO algorithms simulate the behaviour of real ants using a colony of artificial ants, which cooperate in finding good solutions to optimisation problems. Each artificial ant, representing a simple agent, creates candidate solutions to the problem at hand and communicates indirectly with other artificial ants by means of pheromone. At the same time that ants perform a global search for new solutions, the search is guided to better regions of the search space based on the quality of solutions found so far. The algorithm converges to good solutions as a result of the collaborative interaction amongst the ants.

The motivation for applying ACO algorithms to classification problems is that they perform a robust and global search, using pheromone values as a positive feedback to converge to optimal or near-optimal solutions. Therefore, they are able to cope better with the interaction of attributes than traditional deterministic[1]—and usually greedy[2]—classification algorithms [94].

In the context of the classification task in data mining, ACO algorithms have been successfully applied to different classification problems [50]. Since real-world classification problems are generally described by both nominal (categorical or discrete) and continuous (real-valued) attributes, classification algorithms are required to be able to cope with both nominal and continuous attributes in order to build a classification model. However, most ACO classification algorithms have the limitation of being able to cope with only nominal attributes. A commonly used approach to overcome this limitation is to discretise continuous attributes in a preprocessing step. One potential drawback of this approach is that less information is available to the classification algorithm, since the discretisation procedure creates a fixed number of intervals for each continuous attribute.

In this thesis, novel ACO classification algorithms tailored for hierarchical multi-label classification are proposed to induce comprehensible classification models in the context of protein function prediction. Firstly, an extension to ACO classification algorithms is described in order to cope with continuous attributes directly—i.e. without the need for a discretisation procedure in a preprocessing step. Then, an ACO classification algorithm for the hierarchical single-label classification problem of protein function prediction is described. Finally, different approaches to extend ACO classification algorithms to the more complex case of hierarchical multi-label classification are described, elaborating on the ideas of the proposed hierarchical classification ACO algorithm. All the proposed algorithms are compared to other well-known classification algorithms in the literature by performing experimental evaluation on real-world data sets.

Regardless of the ACO classification algorithms proposed in the literature, extending ACO classification algorithms to cope with continuous attributes directly and to cope with the more complex case of hierarchical multi-label classification

[1]Deterministic algorithms are those that follow a predictable sequence of steps to produce the same result—in this case, the same classification model—given a particular input.

[2]A greedy algorithm makes locally optimal choices at each step in the search for the optimal solution, but it cannot reverse bad choices made at early steps even if previous local choices do not lead to the optimal solution.

are unexplored research areas to the best of our knowledge.

The remainder of this chapter is organised as follows. Section 1.1 summarises the main contributions the thesis and section 1.2 presents its organisation. Finally, Section 1.3 is a list of the publications that have resulted from this research.

1.1 Original Contributions

A summary of the main contributions of this thesis is presented next.

- **A method to cope with continuous attributes in ACO classification algorithms:** While current ACO classification algorithms do not cope with continuous attributes directly, a method to cope with continuous attributes directly is presented, taking full advantage of all continuous attributes' information and thereby not requiring a discretisation procedure in a preprocessing step.

- **ACO classification algorithms for coping with continuous attributes:** New ACO classification algorithms are proposed following the method for handling continuous attributes directly. They are compared with well-known classification algorithms in the literature.

- **An ACO classification algorithm for hierarchical single-label classification problems:** A new ACO classification algorithm for hierarchical classification, which extends the ideas of flat single-label ACO classification algorithms to hierarchical classification problems.

- **ACO classification algorithms for hierarchical multi-label classification problems:** Following the general ideas of the proposed ACO algorithm for hierarchical classification, new ACO algorithms tailored for hierarchical multi-label classification problems are proposed, discussing the design decisions made in each of them.

- **Application of the proposed hierarchical multi-label classification algorithms to protein function prediction problems:** The proposed ACO algorithms for hierarchical multi-label classification problems are empirically evaluated in challenging data sets, comparing the results against state-of-the-art algorithms.

1.2 Structure of the Thesis

The introductory chapter is followed by three background chapters, providing additional information about the research areas related to the thesis.

Chapter 2 introduces the data mining background, focusing on the classification task and common approaches for the discovery of comprehensible classification models. The differences between the conventional (flat single-label) classification problems and the more complex hierarchical multi-label classification problems are explained, providing a description of classification algorithms proposed in the literature.

Chapter 3 introduces the ant colony optimisation (ACO) metaheuristic background. An overview of Ant-Miner—the first ACO classification algorithm—and its variations proposed in the literature is presented, discussing current limitations of ACO classification algorithms.

Chapter 4 introduces the bioinformatics background, focusing on the problem of protein function prediction. An overview of protein databases containing different protein information and protein functional classification schemes publicly available is presented. Furthermore, two approaches commonly applied to protein function prediction are discussed.

Following these background chapters, the research of the thesis is presented in chapters 5 to 8.

Chapter 5 describes the proposed method for handling continuous attributes in ACO classification algorithms. Furthermore, four novel ACO classification algorithms are presented, extending the Ant-Miner algorithm to cope with continuous attributes directly.

Chapter 6 presents the empirical evaluation of the proposed ACO classification algorithms coping with continuous attributes, described in the previous chapter. The proposed algorithms are compare against Ant-Miner and other well-known classification algorithms in terms of predictive accuracy and simplicity (size) of the discovered classification model.

Chapter 7 describes the proposed ACO classification algorithm for hierarchical single-label classification problems. Furthermore, two extensions of the proposed ACO algorithm for hierarchical classification tailored for the more complex case of hierarchical multi-label classification and a baseline hierarchical multi-label ACO algorithm are presented.

Chapter 8 presents the empirical evaluation of the proposed algorithms described in the previous chapter. The proposed algorithms are evaluated in two different sets of experiments involving the prediction of protein functions, and compared to other hierarchical multi-label algorithms proposed in the literature in terms of predictive accuracy and simplicity (size) of the discovered classification model.

Finally, the thesis is concluded in chapter 9. Furthermore, Appendix A provides additional information about the implementation of the proposed algorithms.

Chapter 9 draws conclusions, providing a summary of the contributions and the analysis of the results obtained, and presents future research directions.

Appendix A provides details about the implementation of the proposed algorithms and their availability.

1.3 Publication List

The following list of publications has resulted from the research presented in this thesis, comprising works published and submitted for publication in the scientific literature.

Invited Book Chapter

- F.E.B Otero, M. Segond, A.A. Freitas, C.G. Johnson, D. Robilliard and C. Fonlupt. *An Empirical Evaluation of the Effectiveness of Different Types of Predictor Attributes in Protein Function Prediction.* Foundations of Computational Intelligence: Volume 5—Function Approximation and Classification, pages 339–357. Springer, July 2009.

Peer-Reviewed Conference Papers

- F.E.B. Otero, A.A. Freitas and C.G. Johnson. A hierarchical classification ant colony algorithm for predicting gene ontology terms. In *Proceedings of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio 2009)*, pages 68–79. Lecture Notes in Computer Science 5483, Springer, April 2009.

- F.E.B. Otero, A.A. Freitas and C.G. Johnson. Handling continuous attributes in ant colony classification algorithms. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, pages 225–231. IEEE Press, March 2009.

- F.E.B. Otero, A.A. Freitas and C.G. Johnson. *c*Ant-Miner: an ant colony classification algorithm to cope with continuous attributes. In *Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS 2008)*, pages 48–59. Lecture Notes in Computer Science 5217, Springer, September 2008.

Journal Paper (under review)

- F.E.B Otero, A.A. Freitas and C.G. Johnson. *A Hierarchical Multi-Label Classification Ant Colony Algorithm for Protein Function Prediction.* Submitted to the Memetic Computing journal—special issue on metaheuristics for large-scale data mining.

Chapter 2

Data Mining

Over the past decades, there has been a huge increase in both collection and storage of data. Advances in computer technologies have allowed the storage of virtually any kind of data, from personal choices (e.g. supermarket purchases) to scientific information (e.g. genome sequences). The scale of the stored data has clearly overwhelmed our ability to manually analyse and extract knowledge from it, creating a need for (semi-)automatic techniques and methods to assist the analysis and extraction of useful knowledge, which has led to the emergence of the field of knowledge discovery in databases (KDD) [45, 96].

Data mining is the core step of the broad process of knowledge discovery in databases, responsible for extracting useful patterns and models from data, as defined by Fayyad et al. [45, p. 4]:

> "...the overall *process* of finding and interpreting patterns from data is referred to as the KDD *process*, typically interactive and iterative, involving repeated applications of specific *data mining* methods or algorithms and the interpretation of the patterns generated by these algorithms."

Most data mining algorithms employ concepts and techniques mainly from the machine learning and statistics areas, in order to discover (learn) a model from structured data (data set)—i.e. a set of records (examples) described in terms of fields (attributes), as illustrated in Table 2.1. There are generally two primary 'high level' goals of data mining: *prediction* and *description* [45]. In prediction, the discovered model is used to predict unknown (unseen) values of an attribute of interest, based on the values of other attributes. In description, the discovered

Table 2.1: The data set for the artificial 'weather' problem [97], where each row corresponds to a different 'saturday morning' observation and the 'Class' indicates if whether or not ('P' or 'N', respectively) it is a good day to play tennis.

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	27	high	false	N
2	sunny	25	high	true	N
3	overcast	26	high	false	P
4	rain	17	high	false	P
5	rain	8	normal	false	P
6	rain	10	normal	true	N
7	overcast	9	normal	true	P
8	sunny	18	high	false	N
9	sunny	11	normal	false	P
10	rain	17	normal	false	P
11	sunny	18	normal	true	P
12	overcast	16	high	true	P
13	overcast	29	normal	false	P
14	rain	17	high	true	N

model is used to explain the data—e.g., show the correlation between different attribute values.

There are three important design aspects of a data mining algorithm pertinent to this thesis, as described next.

1. *supervised vs. unsupervised learning strategy*: in supervised learning, the data mining algorithm has access to the value of an attribute of interest—usually referred to as the class attribute—during the training (learning) phase. For example, in the data set presented in Table 2.1, the data mining algorithm would have access to the value of the 'Class' column, as well as the value of the remaining attributes, for each example. On the other hand, in unsupervised learning, the data mining algorithm has no information of a class attribute. This is usually the case when trying to find correlations between different attribute values, rather than trying to predict the value of a particular class attribute.

2. *comprehensible vs. 'black box' knowledge representation*: data mining algorithms may either build a comprehensible (human-readable) model or a model whose inner workings are not interpretable—i.e., it is considered as a 'black box'. It should be noted that in some application domains, the comprehensibility of the model plays an important role [32, 45, 51, 99]—e.g., in medical diagnosis the discovered knowledge needs to be validated/interpreted by medical doctors.

3. *nominal (discrete) vs. continuous attributes*: nominal attributes are those attributes that have a finite number of different values, where the ordering is not relevant (categorical nominal)—e.g., the attribute 'Outlook' in the data set presented on Table 2.1 is a categorical nominal attribute with three different values {sunny, overcast, rain}—or where the ordering is relevant (ordinal nominal)—e.g., the values {'0', '1', '2', '3 or more'} of a nominal attribute representing the number of children in a family. On the other hand, in data mining terminology, continuous attributes are those attributes that have numeric values, either real or integer values, where the ordering is obviously relevant—e.g., the attribute 'Temperature' in the data set presented in Table 2.1 is a continuous attribute with values in the range of 8 and 29. There are data mining algorithms that can only cope with nominal attributes, requiring the discretisation of continuous attributes' values prior to the application of the data mining algorithm, as will be discussed in chapter 5.

In this thesis we focus on the design of data mining algorithms that discover comprehensible classification models, following a supervised learning strategy. Therefore, in the remainder of this chapter we will concentrate on discussing the classification data mining task and algorithms which fall both in the supervised learning and comprehensible knowledge representation categories. A wider discussion of data mining tasks and algorithms can be found in [45, 133].

The remainder of this chapter is organised as follows. Section 2.1 presents an overview of common data mining tasks, while section 2.2 presents an overview of the classification task, which is the task investigated in this thesis. Section 2.3 presents a description of the more specific target task of this thesis, namely the hierarchical classification task. A discussion of evaluation measures in the context of the classification task is presented in section 2.4 and in the context of the hierarchical classification task in section 2.5. Finally, section 2.6 presents the summary of this chapter.

2.1 Common Data Mining Tasks

There are several different tasks derived from the area of data mining, which pose different kinds of problems for data mining algorithms. This section presents a brief overview of the common data mining tasks addressed in the literature. The classification task, one of the most studied data mining tasks and the subject of this thesis, is presented in greater detail in section 2.2. A more complete overview of different data mining tasks can be found in [45, 133].

2.1.1 Regression

The regression task consists of finding a model—in most cases represented as a function—that maps a given input (an example's attributes values) to a numeric prediction, usually involving data sets with continuous attributes. Typical regression applications can be found in forecasting (e.g. predicting the economy growth based on market indicators) and medical diagnosis (e.g. predicting the length of time a patient will live after undergoing a particular type of surgery).

2.1.2 Clustering

The clustering task consists of finding a finite set of categories (clusters) to describe the data. The categories are created based on attributes' values and, as a result, similar examples are grouped together. Thus, a clustering algorithm aims at grouping the examples into categories (clusters) so that the similarity of examples in a cluster is maximised and the similarity of examples from different clusters is minimised.

Clustering is a classical example of unsupervised learning, since there is no predefined class attribute to be predicted. Examples of clustering applications include the grouping of a population of consumers into market segments and image segmentation techniques aiming at dividing an image into distinct regions (e.g. border detection).

2.1.3 Association Rule Learning

The association rule learning task consists of finding rules that represent patterns in the data, by identifying relationships (associations) between attributes. The rules are not limited to find associations between a set of attributes and a class attribute, since there is no class attribute predefined. Therefore, the association rule learning task is an example of unsupervised learning.

For example, consider the data set presented in Table 2.1. Note that as originally defined, this data set has a class attribute ('Class' column), and therefore it would be more naturally handled by a classification algorithm. However, it is possible to apply an association rule learning algorithm to that data set, by either removing the class attribute or ignoring its special status (as a prediction target)—i.e., by considering the 'Class' attribute as any other attribute in the data set. In this case, the following association rule could be discovered:

$$\textit{IF Temperature} < \textit{16 THEN Humidity} = \textit{normal} \;\; ,$$

which is found in 4 examples out of 14. The above rule states if an example has the 'Temperature' attribute value less than 16, then the 'Humidity' attribute value is equal to 'normal'. A classical example of the use association rules is from market basket analysis, where the analysis of supermarket sales data provide insights of which products customers tend to buy together.

2.2 The Conventional (Flat) Classification Task

The classification task consists of finding a model that is able to predict the value of the class attribute of an example based on the values of a set of attributes. Classification is a classical example of supervised learning, since the data mining algorithm has access to the value of the class attribute. The main difference between the classification and regression tasks is that in the classification task, the class attribute to be predicted has a finite number of nominal or discrete values, while in the regression task it has a numeric (continuous) value.

Given the data set presented in Table 2.1, the 'Class' column is called the class attribute (the attribute whose value is to be predicted) and the columns marked with 'Attributes' text are called predictor attributes (the attributes whose values are used to predict the class attribute). A classification problem involves a set of examples, where each example is described by predictor attributes' values and associated with a class label.[1] Thus, the aim of a classification algorithm is to find relationships between predictor and class attributes' values. Note that in order for a classification algorithm to successfully find relationships between predictor and class attributes' values, predictor attributes should represent relevant information for the prediction of the class attribute. For instance, the column 'No.' in the data

[1]The term 'class label' is used to refer to a value of the class attribute.

set presented in Table 2.1 contains irrelevant information for prediction, since there is a unique 'No.' value for each example of the data set, and such unique values cannot be used to make generalised predictions to other examples—therefore the column 'No.' is not used as a predictor attribute.

In general, the classification task involves two phases. In the first phase, the data set being mined is randomly split into training and test sets. Then, a classification model that represents the relationships between predictor and class attributes' values is built by analysing the examples from the training set. Note that the algorithm has access to the information of both predictor and class attributes from the training set. In the second phase, the classification model is used to classify—i.e. predict the value of the class attribute—the examples from the test set. Considering that the classification model was built using only the examples from the training set, the algorithm has no information about the class label of the examples from the test set. The value of the class attribute of a test example is only verified after the classification algorithm predicted its value, in order to evaluate the created classification model. A prediction is considered correct when the predicted value is the same as the actual value of the example; otherwise it is considered incorrect. The more correct predictions on the test set, the better the classification model.

One of the main goals of a classification algorithm is to build a model which maximises the predictive accuracy—i.e. the number of correct predictions divided by the total number of predictions—in the test set, although in some application domains (e.g. credit approval, medical diagnosis and protein function prediction) the comprehensibility of the model plays an important role. For instance, both neural networks and support vector machines (SVMs) are successful methods in term of predictive accuracy when applied to classification, but they produce classification models that are not easily interpretable.[2] Since the focus of this thesis is on the discovery of comprehensible as well as accurate classification models, two types of classification techniques commonly employed to produce comprehensible and accurate classification models are reviewed next.

2.2.1 Decision Tree Induction

Decision trees provide a comprehensible graphical representation of a classification model, where the internal nodes correspond to attribute tests (decision nodes) and

[2] There are techniques for extracting comprehensible models from the ones produced by neural networks [67] and support vector machines [36].

leaf nodes correspond to the predicted class labels. In order to classify an example, the tree is traversed in a top-down fashion from the root node towards a leaf node, moving down the tree by selecting branches according to the outcome of attribute tests represented by internal nodes until a leaf node is reached. At this point, the class label associated with the leaf node is the class label predicted for the example.

A decision tree for the data set presented in Table 2.1 is illustrated in Figure 2.1, adapted from [97]. Internal nodes (including the root node) are represented by attribute names and branches originating from internal nodes correspond to different values of the attribute in a node; each leaf node is represented by a class label. Note that it is not required that an algorithm uses all predictor attributes to build a decision tree, only those that are relevant for the classification problem in hand—e.g. the attribute 'Temperature' was not used in the decision tree illustrated in Figure 2.1. In this example, the first decision is made based on the value of the attribute 'Outlook'. If the value is equal to 'sunny' (left branch), there is a need to test the value of the attribute 'Humidity', which will result in predicting the class label 'N' if the attribute's value equals to 'high' or the class label 'P' if the attribute's value equals to 'normal'. If the value of 'Outlook' is equal to 'overcast' (center branch), a leaf node is reached and the class label predicted is 'P'. If the value is equal to 'rain' (right branch), there is a need to test the value of the attribute 'Windy', which will result in predicting the class label 'N' if the attribute's value equals to 'true' or the class label 'P' if the attribute's value equals to 'false'.

A common approach to create decision trees automatically from data is known as the *divide-and-conquer* approach, which consists of an iterative top-down procedure of selecting the best attribute to label an internal node of the tree. It starts by selecting an attribute to represent the root of the tree. After the selection of the first attribute, a branch for each possible value of the attribute is created and the data set is divided into subsets according to the examples' values of the selected attribute. The selection procedure is then recursively applied to each branch of the node using the corresponding subset of examples—i.e. the subset with examples which have the attribute's value associated with the branch—and it stops for a given branch when all examples from the subset have the same class label (or another stopping criterion is satisfied), creating a leaf node to represent a class label to be predicted. The divide-and-conquer approach represents a greedy

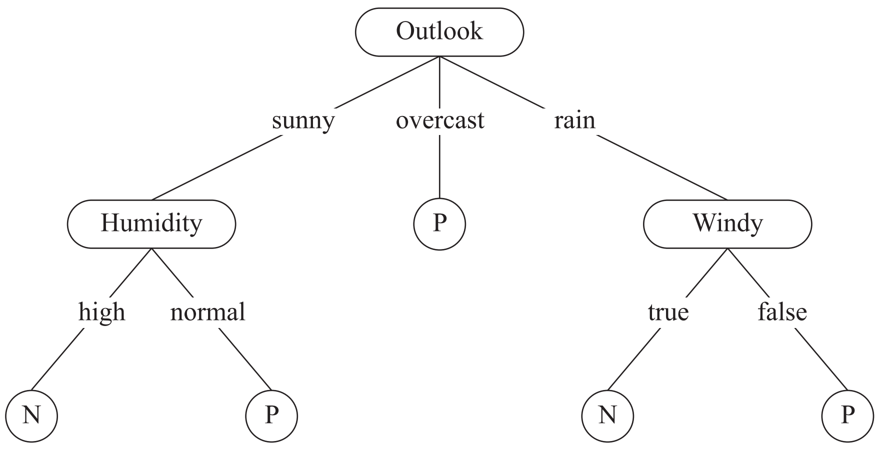


Figure 2.1: Example of a decision tree for the data set presented in Table 2.1, adapted from [97]. Internal nodes (including the root node) are represented by attribute names and branches originating from internal nodes correspond to different values of the attribute in a node; leaf nodes are represented by different class labels. Note that it is not required that an algorithm uses all predictor attributes to build a decision tree, only those that are relevant for the classification problem in hand—e.g. the attribute 'Temperature' was not used in this example.

strategy[3] to create a decision tree, since the selection of an attribute at early iterations cannot be reconsidered at later iterations—i.e. the selection of the best attribute is made locally at each iteration, without taking into consideration its influence over the subsequent iterations.

The main aspect of creating decision trees following the divide-and-conquer approach is how attributes are selected to compose a tree. The selection criterion of attributes varies from algorithm to algorithm. For example, an entropy-based criterion is used in the well-known C4.5 [99], whereas a distance-based criterion is used in Clus [10]. More details of C4.5 and Clus algorithms are provided next.

C4.5

The C4.5 algorithm, probably the most known decision tree induction algorithm, employs an entropy-based criterion in order to select the best attribute to create

[3]A greedy strategy makes locally optimal choices at each step in the search for the optimal solution, but it cannot reverse bad choices made at early steps even if previous local choices do not lead to the optimal solution.

a node. In essence, the entropy measures the (im)purity of a collection of examples relative to their values of the class attribute, where higher entropy values correspond to more uniformly distributed examples, while lower entropy values correspond to more homogeneous examples (more examples associated with the same class label). Hence, at each iteration of the top-down procedure, C4.5's selection criterion favours attributes that minimise the entropy of the generated subsets—i.e., the subsets created according to the examples' value of the selected attribute. C4.5 has been successfully applied to a wide range of classification problems and it is usually used on evaluative comparisons of new classification algorithms. Further details of C4.5 can be found in [99, 100].

Clus

Following the predictive clustering trees (PCT) method [12], wherein a decision tree is viewed as a hierarchy of clusters, Clus induces decision trees in a top-down fashion by selecting attributes based on a distance measure. The basic idea is to map the set of an example's possible class labels onto vectors in an Euclidean space, where each component of the vector indicates the presence/absence of a particular class label. Then, attributes are selected to compose the decision tree based on the variance of the generated subsets (clusters), relative to the examples' class vectors—i.e., it favours the attribute whose test outcome minimises the intra-cluster variance of the generated subsets of examples. An advantage of the distance-based selection criterion employed in Clus, combined with the representation of examples as class vectors into an Euclidean space, is that it can also be applied to more complex classification problems—e.g. hierarchical multi-label classification problems[4]—given a suitable distance measure. Further details of Clus can be found in [10, 13, 127].

2.2.2 Rule Induction

An alternative type of comprehensible representation to decision trees is a set/list of *IF-THEN* classification rules. An *IF-THEN* classification rule is composed by *antecedent* and *consequent* parts: the antecedent part contains attribute-value conditions (terms), which represent tests of particular attributes values, while the consequent contains the class label to be predicted by the rule. It can be graphically represented as

[4]An overview of hierarchical multi-label classification is presented in subsection 2.3.2.

$$IF\ term_1\ AND\ term_2\ AND\ \ldots\ AND\ term_n\ THEN\ class\ label\ ,$$

where the *IF* part represents the antecedent and the *THEN* part represents the consequent. An example that satisfies all the attribute-value conditions (terms) of the rule antecedent is covered by the rule and, consequently, is predicted to have the class label of the rule consequent. The quality of a classification rule is usually calculated based on the number of correctly covered training examples (the training examples covered that have the class label predicted by the rule) relative to the total number of covered training examples. Therefore, the aim of a classification rule is to cover as many training examples as possible, correctly classifying as many training examples as possible at the same time.

Although classification rules have a similar representation to association rules, they differ in two important ways. In association rules, the consequent of a rule can contain any attribute, as well as more than one attribute—i.e. an association rule can predict the value of more than one attribute and there is no predefined class attribute. In contrast, the consequent of a classification rule can only contain the class attribute, since a classification rule predicts the value of a (single) class attribute.

A set of classification rules is illustrated in Figure 2.2, which represents a classification model equivalent to (making the same predictions as) the decision tree presented in Figure 2.1. It is important to emphasise the difference between sets and lists of classification rules. In general, the approaches to create a set or a list of classification rules are very similar. The difference lies in how a set or a list is applied to classify a test example. In a set of classification rules, there is no particular order between rules; therefore, a test example may receive multiple candidate classifications, if it satisfies more than one rule's antecedent.

```
IF Outlook = sunny AND Humidity = high THEN N
IF Outlook = sunny AND Humidity = normal THEN P
IF Outlook = overcast THEN P
IF Outlook = rain AND Windy = true THEN N
IF Outlook = rain AND Windy = false THEN P
```

Figure 2.2: Example of a set of classification rules representing a classification model equivalent to the decision tree presented in Figure 2.1. An example that satisfies the antecedent of a rule (*IF* part) has the class label of the consequent predicted (*THEN* part).

Hence, there is a need for a decision criterion in order to deal with ambiguous cases—e.g., the cases where a test example is covered by rules predicting different class labels. A simple approach is to choose the rule with the highest quality (classification accuracy) measured during the training phase. On the other hand, in a list of rules[5], the order in which rules are organised is relevant when classifying test examples. Given a test example, the prediction of its class label is made by the first rule that covers the example, following the order of the list of rules. Therefore, the example is shown to the rule at the beginning of the list and if the rule does not cover the test example, it is shown to the next rule and so on, until a rule that covers the example is found.

In order to automatically create a list of rules from data, a commonly used approach is to create one rule at a time, removing the training examples covered by the rule, until there are no uncovered training examples. As a result, the problem of creating a list of rules is reduced to a sequence of simpler problems of creating a single rule. This iterative one-rule-at-a-time approach is referred to as *sequential covering*, since it sequentially creates a list of rules that covers the complete training set. Algorithm 2.1 presents the high-level pseudocode of the sequential covering approach to create a list of rules (ordered rules). A similar approach can be employed to create a set of rules (unordered rules), as presented by Witten and Frank [133].

According to Algorithm 2.1, the sequential covering starts with an empty list of rules and a training set that comprises all training examples. Then, it enters an iterative process of creating a rule until there are no uncovered training examples (*while* loop). In general, the rule created at each iteration is the best rule according to a quality measure that can be computed given the training set. The created rule is added to the list of rules and the training examples covered by the rule are removed from the training set. This one-rule-at-a-time iterative process continues until there are no training examples left in the training set, and at the end of this process, the list of rules created is returned as the discovered rules.

There are alternative approaches to create a set/list of rules, as presented by [48, 133]. For example, PART [47] creates a set of rules by using a combination of a sequential covering procedure and decision tree induction; RIPPER [27] employs a global post-processing optimisation step to adjust or replace individual rules in order to increase the accuracy of the set of rules. More details of PART and

[5]There are authors that use the term 'decision list' to denote a list of rules (ordered rules) [103, 133].

Algorithm 2.1: High-level pseudocode of the sequential covering approach to create a list of rules.

```
input  : training examples
output: list of rules
1  begin
2      $training\_set \leftarrow$ all training examples;
3      $rule\_list \leftarrow \emptyset$;
4      while $|training\_set| > 0$ do
5          // creates the best rule given a quality measure
6          $rule \leftarrow CreateRule()$;
7          $rule\_list \leftarrow rule\_list + rule$;
8          // removes the training examples covered by the rule
9          $training\_set \leftarrow training\_set - Covered(rule, training\_set)$;
10     end
11     return $rule\_list$;
12 end
```

RIPPER algorithms are provided next.

PART

Following a sequential covering approach, PART extracts rules from partial decision trees in order to create a set of rules. In essence, to create a single rule at each iteration of the sequential covering approach, a partial decision tree is built using the current training examples—i.e. the training examples that have not been covered by a previous rule—and the path from the leaf node that covers the greatest number of examples towards the root node is used to create a rule. Then, the rule is added to the set of rules and the remainder of the partial tree is discarded, completing an iteration. In order to build a decision tree, PART employs the same C4.5's heuristic to select attributes to create the nodes of the tree. Note that PART's decision tree induction recursively expands the tree by following branches in an increasing order of the entropy value associated with the branch's subset of examples (the subset created according to the examples' value of the selected attribute). The motivation for this bias is that subsets with lower entropy are more likely to produce small subtrees, which in turn produce more general rules. Therefore, the generated (partial) tree can contain branches that are not explored (expanded), since branches corresponding to subsets with higher entropy values may not be expanded. Further details of PART can be found in [47, 133].

RIPPER

Implementing a rule induction procedure with a reduced error pruning strategy [98], RIPPER sequentially creates a set of rules that is subject to a global post-processing step. It starts by designating a fraction of the training examples as the pruning set, which is used to remove terms from rules in order create simpler and more accurate rules. Then, the procedure to create a set of rules can be divided into two steps. In the first step, rules are individually created using a reduced error pruning strategy covering all training examples (excluding the training examples comprising the pruning set). In the second step, a global post-processing step adjusts or replaces rules guided by a performance measure of the modified set of rules achieved in the pruning set. The performance measure takes into account both accuracy and simplicity (size of the rules) of the set of rules. Further details of RIPPER can be found in [27].

2.2.3 Multi-Label Classification

As described in section 2.2, although class attributes in conventional (flat) classification problems—usually referred to as *flat single-label* classification problems—can have two or more different values, each example is associated with at most one class label. On the other hand, in *flat multi-label* classification problems each example can be associated with one or more (different) class labels. Therefore, a multi-label classification algorithm needs to be able to predict two or more class labels for a test example. An example of a multi-label data set is presented in Table 2.2, wherein the class attribute has three different values and the examples are associated with one or more (up to three) different class labels.

A common approach to deal with multi-label classification problems is to treat each class label individually, transforming the multi-label problem into a set of single-label (binary[6]) classification problems. This process is divided in two phases. First, for each class label n ($n = 1, ..., L$, where L is the total number of class labels), a new data set D_n is created out of the original data set using only the information of the class label n. This process will create L data sets, where each data set contains all examples from the original data set. In the n-th data set, each example contains all predictor attributes of the original data set, but just the information of the presence/absence of the n-th class label. In the second

[6]A binary classification problem refers to the problem of classifying examples into two class labels: positive and negative.

Table 2.2: An example of a multi-label data set, where each example is associated with one or more different class labels. In this example, the predictor attributes are omitted and the class attribute has three different values {sports, politics, entertainment}.

Ex.	Class		
	value #1	value #2	value #3
1	sports		entertainment
2	sports	politics	
3	sports	politics	entertainment
4		politics	
5			entertainment
6	sports		
7		politics	entertainment
8			entertainment
9		politics	
10	sports	politics	

phase, a single-label (binary) classification algorithm is applied to each data set to build a classification model for each class label n. Figure 2.3 shows the data sets derived from the multi-label data set presented in Table 2.2, using the approach described above. Transforming a muti-label classification problem into a set of binary classification problems is (very) computationally expensive, given that it requires a run of a single-label classification algorithm for each class label. It also has the limitation of not being able to detect correlations between different class labels—i.e., the presence of a particular class label can increase the chance of the presence of another class label, given that they occur frequently together in the training set.

As an alternative to transforming the multi-label classification problem into a set of single-label (binary) classification problems, some authors have proposed classification algorithms tailored to multi-label classification. For example, Clare and King [25] have presented an extension to the well-known C4.5 decision tree induction algorithm, where each leaf of the decision tree can predict more than one class label at the same time and the entropy formula was modified to cope with multi-label data. Other examples can be found in [106, 136].

Ex.	Class
1	P
2	P
3	P
4	A
5	A
6	P
7	A
8	A
9	A
10	P

(a) sports

Ex.	Class
1	A
2	P
3	P
4	P
5	A
6	A
7	P
8	A
9	P
10	P

(b) politics

Ex.	Class
1	P
2	A
3	P
4	A
5	P
6	A
7	P
8	P
9	A
10	A

(c) entertainment

Figure 2.3: Single-label data sets derived from the multi-label data set presented in Table 2.2: (a) represents the data set for the class label 'sport'; (b) represents the data set for the class label 'politics'; (c) represents the data set for the class label 'entertainment'. In the column 'Class', the values 'P' and 'A' represent the presence or absence of the corresponding class label, respectively.

2.3 The Hierarchical Classification Task

In the vast majority of classification problems addressed in the literature, each example is associated with only one class label and class labels are unrelated—i.e. there are no relationships between class labels. These kind of classification problems are usually referred to as *flat* (non-hierarchical) *single-label* problems and addressed by the (conventional) classification task as described in section 2.2.

However, there are more complex classification problems where the class labels to be predicted are hierarchically structured and, at the same time, examples may be associated with more than one class label at the same time. The former case is referred to as *hierarchical* classification, wherein there are hierarchical relationships (e.g. parent/child relations) between different class labels. The prediction of a particular class label implies that all its ancestor class labels are predicted. Therefore, it is expected that the classification algorithm take into account the dependencies between class labels in order to make predictions consistent with the hierarchy of class labels (class hierarchy). The latter case is referred to as *hierarchical multi-label* classification, wherein the class labels are structured in a hierarchy and examples may be associated with more than one different (non-hierarchically related) class label. Therefore, the classification algorithm must be able to predict more than one class label for each example, while satisfying the dependencies (hierarchical relationships) between class labels.

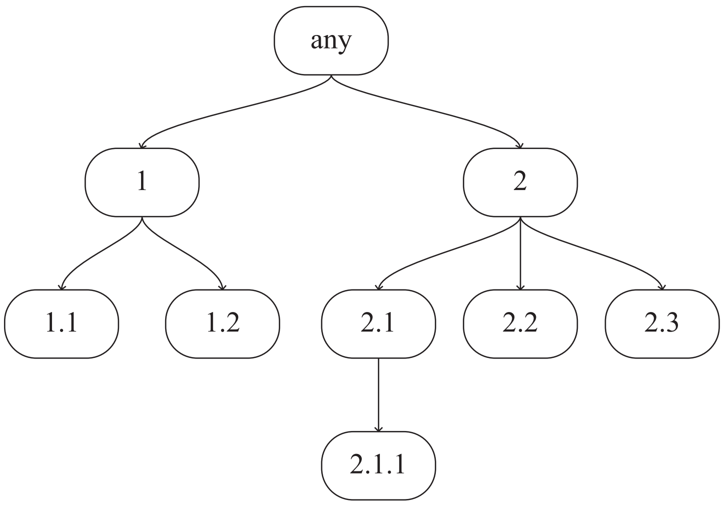
The next subsections present details of hierarchical and hierarchical multi-label classification problems, and discusses different techniques and algorithms that have been proposed in the literature.

2.3.1 Basic Concepts of Hierarchical Classification

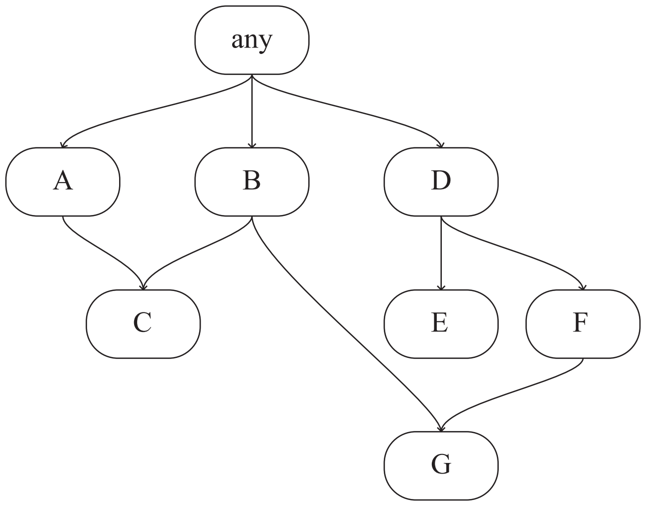
In hierarchical classification problems, the class labels are organised in a hierarchical structure, referred to as the class hierarchy. In general, there are two main types of structures used to represent a class hierarchy: tree and directed acyclic graph (DAG) structures. Figure 2.4 illustrates these structures, in two scenarios: (a) for the case of a tree structure and (b) for the case of a DAG structure. In Figure 2.4, the nodes represent the different class labels, while the edges represent the hierarchical relationship (e.g. parent/child) between class labels—nodes with child nodes are denominated internal nodes and nodes without child nodes are denominated leaf nodes. Given the hierarchical relationships between nodes in a class hierarchy, nodes at deeper levels of the class hierarchy represent more specific (specialised) class labels and nodes at the top of the class hierarchy represent more generic class labels. The root of the hierarchy, representing the case where the class label is unknown since it does not contain any knowledge about the class label of an example, corresponds to the node labelled ‘any’.

As illustrated in Figure 2.4, the main difference between tree and DAG structures lies in the number of parents a node can have—with the exception of the root node, which does not have any parent node. In the case of tree structures, a node has at most one parent label. Therefore, there is a single path from a node towards the root node. On the other hand, in the case of DAG structures, a node can have more than one parent node. Therefore, there are multiple paths from a node towards the root node. For this reason, DAG structures are considered a more challenging case of hierarchical classification.

When compared to flat classification problems, hierarchical classification problems are considered more complex, having different properties that should be taken



(a)



(b)

Figure 2.4: Example of the different types of structures used to represent a class hierarchy: (a) the class labels are organised in a tree structure, with a single parent for each node except the root node; (b) the class labels are organised in a directed acyclic graph (DAG) structure, with potentially multiple parents for each node except the root node. The node ‘any’ corresponds to the root of the class hierarchy and it represents the case where the class label of an example is unknown.

into account when designing hierarchical classification algorithms [49, 117, 118, 129]. Firstly, the number of different class labels in hierarchical classification problems tend to be much greater when compared to flat classification problems, mainly due to the hierarchical class structure.

Secondly, the classification algorithm has to exploit the hierarchical relationship between class labels in order to make predictions that satisfy hierarchical parent-child relationships. For example, considering the tree-structured class hierarchy in Figure 2.4(a), the prediction of class label '2.1' indirectly implies the prediction of parent class label '2', but not the child class label '2.1.1'. Therefore, a hierarchical classification algorithm should be flexible in order to classify examples at different nodes of the class hierarchy. Note that the class hierarchy information is also used when evaluating the predictive accuracy of the hierarchical classification algorithm—i.e., predicting the class label '1.1' when the example belongs to class label '2.2' should be more penalised than predicting class label '2.3' when the example belongs to class label '2.2', since class label '2.2' and '2.3' are more similar (closer) according to the class hierarchy.

Thirdly, it is generally more difficult to discriminate between class labels at the bottom of the hierarchy than class labels at the top of the hierarchy, since the number of examples per node (class label) at the bottom tends to be smaller compared to nodes at the top. Recall that nodes at the bottom of the hierarchy represent more specific class labels, which tend to be more informative than nodes at the top of the hierarchy. Therefore, a hierarchical classification algorithm may have to deal with the trade-off between more reliable predictions (i.e. predictions at higher levels of the class hierarchy) and more informative predictions (i.e. predictions at lower levels of the class hierarchy).

Concerning the level (depth) of the predicted class labels, hierarchical classification problems can be divided into two groups, namely *mandatory leaf-node prediction* and *optional leaf-node prediction* problems [49].[7] In mandatory leaf-node prediction problems, the classification algorithm is require to predict a leaf class label (leaf node of the class hierarchy) for a test example. In optional leaf-node prediction problems, the classification algorithm has the flexibility to decide the class label to be predicted independently of its level (depth) in the class hierarchy. Hence, the classification algorithm—based on a confidence measure (e.g.,

[7]There are authors that use the terminology *virtual category tree* and *virtual directed acyclic category graph* for tree-structured and DAG-structured mandatory leaf-node prediction problems, respectively; *category tree* and *directed acyclic category graph* for tree-structured and DAG-structured optional leaf-node prediction problems, respectively [117, 119].

predictive accuracy)—can either predict a leaf or internal class label as the most specific class label for a given test example. Note that in both types of problems, the classification algorithm is indirectly predicting all the ancestor class labels (internal nodes of the class hierarchy) of the predicted class label.

There are several approaches available for dealing with hierarchical classification problems, varying from transforming the hierarchical problem into a flat one to more elaborate strategies that build a hierarchical classification model [49, 117]. These approaches are discussed next.

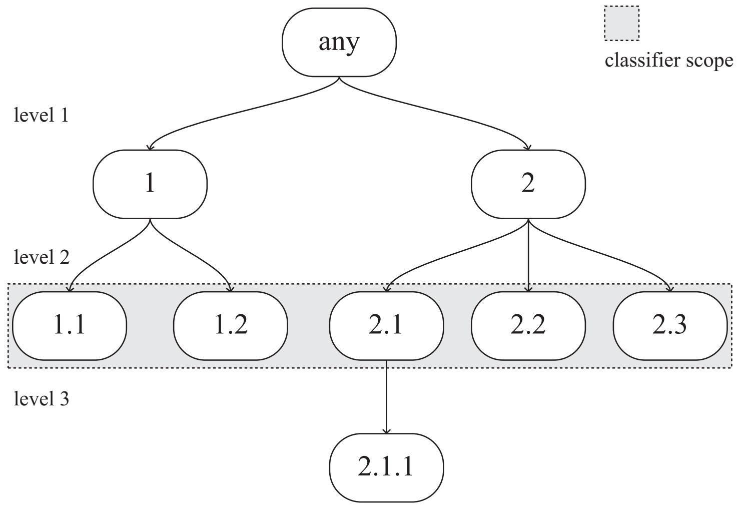
Transforming a Hierarchical Classification Problem into a Flat Classification Problem

The simplest approach to deal with a hierarchical classification problem is to transform the problem into a flat classification problem by selecting one class level[8] or set of class labels (e.g. the set of class labels representing the leaf nodes of the class hierarchy) to represent the class labels to be predicted. After the transformation step, a conventional flat classification algorithm can be applied to solve the new (flat) classification problem.

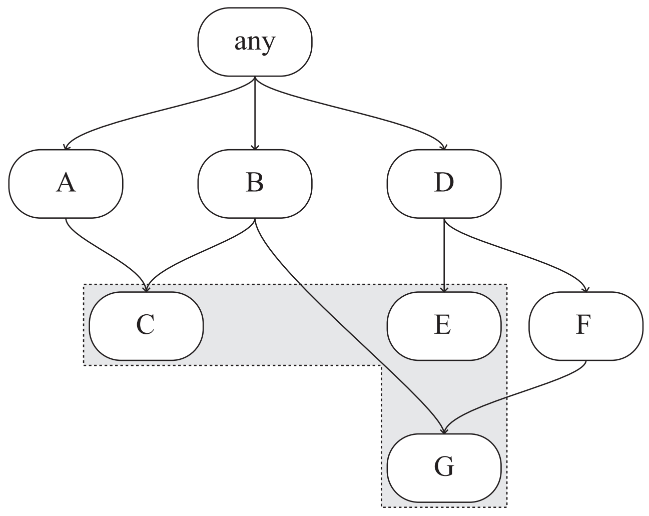
Figure 2.5 presents two examples of how a hierarchical classification problem can be transformed into a flat classification problem. In the case of the tree-structured class hierarchy in Figure 2.5(a), the hierarchical classification problem could be transformed into the problem of predicting the class labels of the second level—i.e. the set of class labels {1.1, 1.2, 2.1, 2.2, 2.3}, represented by the shaded area in Figure 2.5(a). Note that the set of examples associated with class label '2.1' comprises examples associated with both class labels '2.1' and '2.1.1', since all examples associated with class label '2.1.1' are also associated with class labels '2.1' according to the class hierarchy. In the case of the DAG-structured class hierarchy in Figure 2.5(b), the hierarchical classification problem could be transformed into the problem of predicting the class labels that represent leaf nodes of the hierarchy—i.e. the set of class labels {C, E, D}, represented by the shaded area in Figure 2.5(b).

As illustrated in Figure 2.5, transforming the hierarchical classification problem into a flat classification problem avoids the complexity of dealing with a class

[8]The level notion in DAG structures is somewhat ambiguous, since a node (representing a class label) has potentially multiple parents and, therefore, it may be considered in multiple levels—e.g. class label 'G' in Figure 2.5(b) can be either considered in the second level, since it is a child of class label 'B', or in the third level, since it is a child of class label 'F'.



(a)



(b)

Figure 2.5: An example of how hierarchical classification problems can be transformed into flat classification problems: in (a) the hierarchical classification problem could be transformed into the problem of predicting the class labels of the second level of the hierarchy; in (b) the hierarchical classification problem could be transformed into the problem of predicting the class labels that represent leaf nodes of the hierarchy. The shaded area represents the scope of the classifier—i.e. the class labels predicted by the classification model.

hierarchy. However, the class hierarchy information is lost after the transformation and each class label is regarded as a distinct value to be predicted. The choice of the set of class labels or the level whose class labels are to be predicted is a difficult one. Depending on the selected level or set of class labels of the class hierarchy, this approach presents a trade-off between predicting more generic class labels (top of the hierarchy) or more specific class labels (bottom of the hierarchy). Moreover, a potentially large number of class labels has to be discriminated in a single run of a flat classification algorithm, since the number of class labels tend to be larger at deeper levels of the class hierarchy. Examples of this approach can be found in [69, 131].

Transforming a Hierarchical Classification Problem into a Set of Binary Classification Problems

A more elaborated approach, less extreme than transforming the hierarchical classification problem into a single flat classification problem, is to transform the hierarchical classification problem into a set of binary (also flat) classification problems, one per class label of the class hierarchy. Then, for each binary classification problem, a flat classification algorithm is applied in order to predict the presence/absence (positive/negative value) of a particular class label. This approach is similar to the approach which transforms a multi-label classification problem into a set of binary classification problems described in subsection 2.2.3 and, consequently, shares the same limitations—as will be discussed shortly.

Given the tree-structured class hierarchy in Figure 2.4(a), the hierarchical classification problem could be transformed into a set of binary classification problems by training a flat classification algorithm for each class label, which predicts the presence/absence of the corresponding class label. In this case, the class hierarchy is used to define the set of positive/negative training examples used by each classification algorithm. For example, assuming that each example is associated with at most one class label per level, the positive training examples for the class label '2.1' comprise the examples associated with class labels '2.1' and '2.1.1', since examples associated with class label '2.1.1' are indirectly associated with class label '2.1' according to the class hierarchy. Consequently, the negative training examples for the class label '2.1' comprise the examples that are not associated with class labels '2.1' and '2.1.1'—i.e. the examples associated with class labels '1', '1.1', '1.2', '2' (as the most specific class label), '2.2' and '2.3'. It should be noted that not all examples associated with class label '2' are considered as negative examples, only those that have the class label ‘2’ as the most specific known class label. On the other hand, examples associated with class labels ’2.1’ and ‘2.1.1’ and, consequentially, with class label ‘2’ are considered positive examples, since their most specific known class label is ‘2.1’ and ‘2.1.1’, respectively. After building a classification model for each class label, the predictions of each (individually trained) classification model are combined in order to classify a test example. In this way, a test example can be assigned to more than one class label at any level of the class hierarchy.

As discussed in [13], predicting each class label individually has several disadvantages. Firstly, it is slow, since a classifier needs to be trained n times (where n is the number of class labels in the class hierarchy). Secondly, some class labels could potentially have few positive examples in contrast to a much greater number of negative examples, particularly class labels at deeper levels of the hierarchy. Many classifiers have problems with imbalanced class distributions [68]. Thirdly, individual predictions can lead to inconsistent hierarchical predictions, since parent-child relationships between class labels are not imposed automatically during the training. However, more elaborate approaches can correct the individual predictions in order to satisfy hierarchical relationships—e.g., a Bayesian network is used to correct the inconsistent predictions of a set of independent SVM classifiers in [8]. Lastly, the discovered knowledge identifies relationships between predictor attributes and each class label individually, rather than relationships between predictor attributes and the class hierarchy as a whole, which could give more insight about the data.

Top-Down Approach

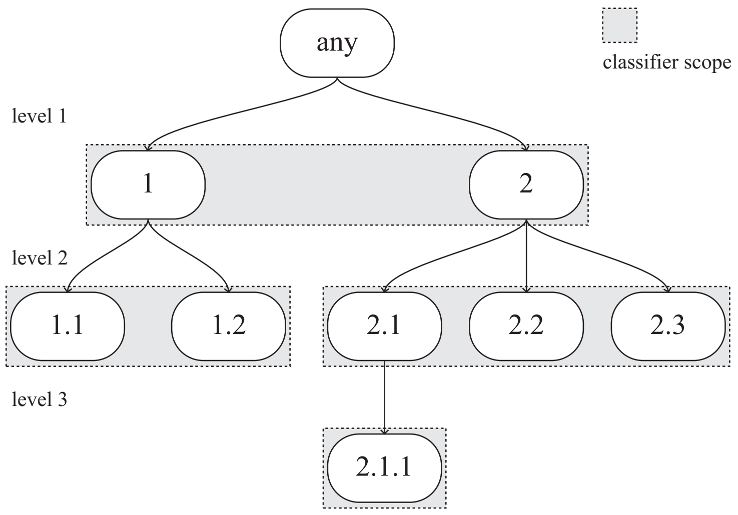
Following a similar divide-and-conquer approach used to induce decision trees, the top-down approach for hierarchical classification combines the output of several flat classification algorithms in order to make hierarchically consistent predictions. In general, each set of class labels at a given level is associated with a classification algorithm, producing a classification model that predicts the presence/absence of the corresponding class labels of the class hierarchy. Figure 2.6 illustrates two examples of the top-down approach applied to a tree-structured class hierarchy. In Figure 2.6(a), a flat classification algorithm(s) is(are) used to produce a classification model(s) for each set of sibling (with a common parent node) nodes at each level of the class hierarchy; while in Figure 2.6(b), flat (binary) classification algorithms are used to produce a classification model for each class label of the class hierarchy.

As illustrated in Figure 2.6, the set of classification models are naturally organised in a hierarchical structure, referred to as the classifier hierarchy, where the term classifier is a synonym for classification model. The process of classifying a test example follows the classifier hierarchy in a top-down fashion, where higher levels guide the classification towards lower levels. In essence, the new example is first presented to the classifier(s) at the first level. Then, the next classifier(s) is(are) chosen based on the class label predicted by the previous classifier(s), and so on, until the test example reaches a leaf classifier or it cannot be classified into any of the lower-level class labels—if the target problem is an optional leaf-node prediction problem.

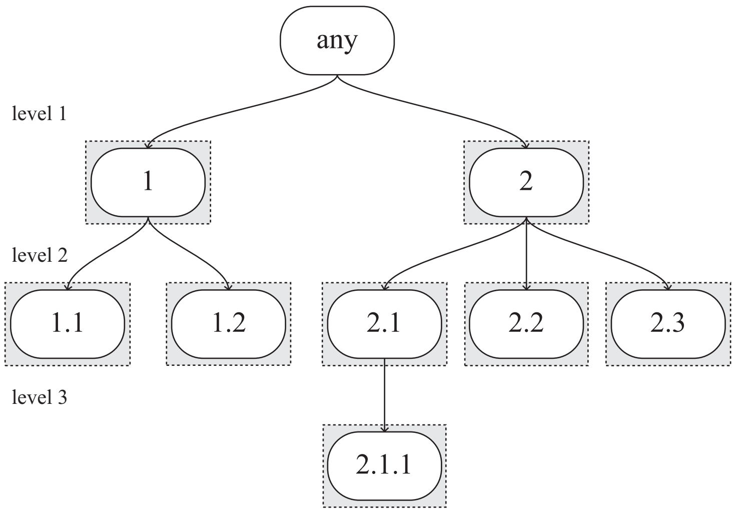
To illustrate this top-down classification procedure, consider the class hierarchy (and the corresponding classifier hierarchy) illustrated in Figure 2.6(b). In this case, a test example would be first classified by the binary classifiers '1' and '2'. Assuming that the test example was assigned to class label '2' and not to class label '1', it would be then passed to classifiers '2.1', '2.2' and '2.3'. Note that there is no need to show the test example to classifier '1.1', since it was already rejected (i.e. it was classified as a negative example) by classifier '1'. If both classifiers '2.1' and '2.3' reject the test example, while classifier '2.2' accepts (i.e. predicts the class label '2.2'), the classification of the test example is completed and the final set of predicted class labels would be {2, 2.2}.

A variation of the top-down approach is presented in [62], where a hybrid particle swarm optimisation/ant colony optimisation (PSO/ACO) algorithm is used to select, out of a set of predefined candidate classification algorithms, the best (most accurate) classification algorithm to be used at each node of the class hierarchy in order to build the classification hierarchy. This selective top-down approach is based on previous work presented in [107], where the selection of the best algorithm at each node is done in a greedy fashion, rather than using the PSO/ACO algorithm.

Although the predictions in a top-down approach are consistent with the class hierarchy, this approach shares the same limitation of requiring multiple (individual) runs of classification algorithms of the previous approach that transforms the hierarchical classification problem into a set of binary classification problems. However, in some cases the number of classifiers to be built is given by the number of internal nodes, rather than the total number of nodes (internal and leaf



(a)



(b)

Figure 2.6: Examples of the top-down approach applied to a tree-structured class hierarchy: in (a) flat classification algorithm(s) is(are) used to produce classification model(s) for each level of the class hierarchy; in (b) flat (binary) classification algorithms are used to produce a classification model for each class label of the class hierarchy. The shaded area represents the scope of the classifier—i.e. the class labels predicted by the classification model. A classification model is required for the class label '2.1.1' only in optional leaf-node prediction problems; in the case of mandatory leaf-node prediction problems, the prediction of class label '2.1' implies the prediction of class label '2.1.1'.

nodes) of the class hierarchy. In this case this approach potentially saves a considerable time, compared to building a classifier for each node (class label of the class hierarchy). In any case, this approach presents two additional drawbacks, as follows.

Firstly, misclassifications at higher levels are propagated to lower levels, given that when a test example is (wrongly) rejected by a given parent classifier, it is not shown to its child classifiers. Therefore in the conventional usage of this approach, it is not possible to recover from a misclassification at deeper levels of the classifier hierarchy. This problem is known as the blocking problem [119, 120] and there are strategies to reduce the blocking effect, as discussed in [120].

Secondly, the top-down classification procedure, as described above, naturally fits a tree-structured class hierarchy, where each node of the class hierarchy has at most a single parent node. Hence, if the parent classifier rejects the test example, a child classifier does not receive the test example. When dealing with DAG-structured class hierarchies, where each node has one or more parent nodes, the top-down procedure requires a modification (extension) to cope with the multiple parents property of a DAG structure. For example, consider the DAG-structure class hierarchy in Figure 2.4(b), where a classifier at the node 'C' would have two parent classifiers 'A' and 'B'. Supposing that classifier 'A' accepts a test example and classifier 'B' rejects it, it is not clear whether the test example should be passed to the (child) classifier 'C' or not. Thus, the top-down classification procedure needs to incorporate a 'conflict resolution' strategy to be able to cope with DAG-structured hierarchies. An example of a top-down approach extended to DAG-structured hierarchies can be found in [127], where each classifier outputs a probability instead of a binary (accept/reject) value and a pessimistic bias of selecting the lowest probability value across all parent nodes is used as an upper limit for the class label probability to be assigned by its child classifiers, in order to produce consistent predictions—i.e. if any of the parents nodes rejects the test example by assigning a probability equal to zero, the child node would also assign a probability equal to zero.

Big Bang Approach

The hierarchical classification approaches described so far rely on existing flat classification algorithms and employ a strategy to combine their output (predictions) in a hierarchically meaningful manner. In order to mitigate the aforementioned limitations of previous approaches, the big bang approach aims at building a classification model using a single run of a hierarchical classification algorithm.

While implementations of hierarchical classification algorithms following the big bang approach are intuitively more complex, since all the class labels of the class hierarchy are taken into account at once, producing a single classification model that is capable of predicting class labels at any level of the class hierarchy has several advantages, as highlighted by Blockeel et al. [10]:

> "...learning a single model for all classes has the advantage that the total size of the predictive theory is typically smaller, and dependencies between different classes w.r.t. membership can be taken into account and may even be explicitated. Advantages of learning a single model for multiple related prediction tasks have been reported several times in the literature (see e.g. [11] for decision trees, [7, 20] for neural networks, [129] for text classification)."

In addition, the previously-mentioned blocking problem of the top-down approach is avoided since a single classification model is employed during the classification of a test example, predicting all class labels of a test example at once while satisfying hierarchical parent/child relationships.

An example of a hierarchical classification algorithm following the big bang approach can be found in [26], where an extension of the well-known C4.5 decision tree algorithm is proposed to deal with hierarchical class labels. The basic idea of their hierarchical C4.5 extension is the modification of the entropy formula—used to decide which attribute is chosen to compose the decision tree—to take into account the class hierarchy, assigning lower entropy values to class labels at higher levels of the hierarchy and higher entropy values to class labels at lower levels of the hierarchy. Also, the leaf nodes of the decision tree were extended to predict a vector of Boolean values, indicating the presence/absence of a particular class label. The hierarchical C4.5 extension was applied to tree-structured class hierarchies [26] and further to DAG-structured class hierarchies [24].

Another example is the CLUS decision tree induction algorithm [10], briefly described in Subsection 2.2.1, where the hierarchical (multi-label[9]) variant of the CLUS classification algorithm following a big bang approach is dubbed CLUS-HMC. Using a weighted Euclidean distance measure, which considers similarities at higher levels of the class hierarchy more important than at lower levels, CLUS-HMC was applied to tree-structured [10, 13] and DAG-structured [127] class

[9]An overview of hierarchical multi-label classification is presented in Subsection 2.3.2.

hierarchies.

As a further example of a hierarchical (multi-label) classification algorithm following the big bang approach, Rousu et al. [104] proposed a kernel-based algorithm, representing the classification model as a variant of the Maximum Margin Markov Network framework [122, 124], and it was applied to tree-structured class hierarchies.

2.3.2 Hierarchical Multi-Label Classification

Hierarchical multi-label classification problems combine the characteristics (and challenges) of both hierarchical and multi-label classification problems: (1) class labels are organised in a hierarchical structure (e.g. a tree or DAG structure); (2) examples may be associated with more than one (non-hierarchically related) class labels. For example, considering the DAG-structured class hierarchy in Figure 2.4(b), an example in a hierarchical multi-label classification problem can be associated with the set of class labels {C, E}—where class labels 'C' and 'E' are not hierarchically related. Therefore, the set of class labels of an example may represent multiple paths in the class hierarchy.

Intuitively, hierarchical multi-label classification presents a more complex problem for a data mining algorithm. Vens et al. [127] have presented the state-of-the-art in this area, which includes several of the previously-mentioned classification algorithms [8, 13, 26, 104], emphasising that hierarchical multi-label classification problems with DAG-structured class hierarchies have not been thoroughly studied in the literature. They proposed three decision tree inductions algorithms for the hierarchical multi-label classification problem capable of coping with both tree-structured and DAG-structured class hierarchies, namely Clus-HMC, Clus-HSC and Clus-SC. These algorithms are used in the evaluation of the hierarchical multi-label classification algorithms proposed in this thesis.

Kiritchenko et al. [73] presented an alternative approach, where the hierarchical classification problem is cast as a multi-label problem by expanding the set of class labels of an example with all their ancestor class labels. As a result, the hierarchical data set is transformed into a multi-label data set. Subsequently, a multi-label classification algorithm—the AdaBoost.MH [106]—is applied to the (new) multi-label data set. Although this approach requires a single run of a classification algorithm, there is still a need for a post-processing step to resolve inconsistencies in the class labels predicted, since the parent/child relationships are not automatically imposed by the multi-label predictions.

2.4 Evaluation Measures for Classification

As discussed in section 2.2, after the classification model is built by a classification algorithm, it is evaluated on the test (hold-out) set. There are many evaluation measures for flat single-label classification [133]. The most widely used evaluation measures in flat single-label classification are the predictive accuracy and the complementary error rate, given by

$$\begin{aligned} Accuracy &= \frac{\textit{number of correct predictions in the test set}}{\textit{total number of examples in the test set}}\,, \\ ErrorRate &= 1 - Accuracy\,. \end{aligned} \tag{2.1}$$

According to Equation (2.1), the higher the number of the correct predictions (out of the total number of prediction), the higher predictive accuracy and, consequently, the lower is the error rate.

Other common evaluation measures, particularly for binary classification problems, are the information retrieval concepts of precision and recall. In the case of binary classification problems, precision corresponds to the number of correct positive predictions divided by the total number of positive predictions; recall corresponds to the number of correct positive predictions divided by the total number of positive examples. These evaluation measures can be summarised by a contingency table (also known as confusion matrix), as presented in Figure 2.7. The precision is given by

$$Precision = \frac{TP}{TP + FP}\,, \tag{2.2}$$

where TP is the number correct predictions (the number of positive examples correctly predicted as positive) and FP is the number of negative examples wrongly predicted as positive. The recall is given by

$$Recall = \frac{TP}{TP + FN}\,, \tag{2.3}$$

where FN is the number of positive examples wrongly predicted as negative. The TN value of the confusion matrix in Figure 2.7, which is not involved in the calculation of precision and recall, is the number of negative examples correctly predicted as negative.

		actual class label	
		+	−
predicted class label	+	true positives (TP)	false positives (FP)
	−	false negatives (FN)	true negatives (TN)

Figure 2.7: The contingency table, also known as confusion matrix, for binary (flat single-label) classification problems.

2.5 Evaluation Measures for Hierarchical Classification

In the case of hierarchical and multi-label classification there is no general consensus regarding which measure is more appropriate. Several researchers evaluate hierarchical and multi-label algorithms based on flat single-label measures, such as predictive accuracy or precision/recall values, computed individually for each class label (e.g. [9, 26]) or each level (e.g. [62, 104]) of the class hierarchy. Exploiting the hierarchical relationships between class labels in a class hierarchy, Sun et al. [117, 119] propose two measures tailored for hierarchical classification problems derived from the similarity and distance between class labels, respectively. The basic idea of the similarity measure is to define pairwise class labels similarity values, either manually or computed automatically, in order to compute the similarity between the predicted set of class labels and the actual set of class labels for each test example. Intuitively, the prediction of a set of class labels that is more similar to the actual set of class labels of a test example is considered better than predicting a completely unrelated set of class labels. The distance measure uses a rather simpler idea, which consists of calculating the distance between predicted and actual class labels based on the number of edges in the class hierarchy between them—the shorter the distance (number of edges), the closer the class labels. A distance-based evaluation measure is also used in [10], where weights are used on the edges to give more importance to edges at higher levels of the class hierarchy—i.e. edges at higher levels are associated with higher weights than edges at lower levels of the class hierarchy.

A more detailed discussion about evaluation measures can be found in [125] for multi-label classification and in [29, 49, 117, 119] for hierarchical classification. A wider discussion on evaluation measures used in different kinds of classification problems—e.g. flat, multi-label and hierarchical—can be found in [114]. Subsections 2.5.1 and 2.5.2 discuss the hierarchical and multi-label evaluation measures used throughout the thesis.

2.5.1 Hierarchical Measures of Precision, Recall and F-measure

Kiritchenko et al. [73] propose a hierarchical classification evaluation measure that discriminates errors by both distance and depth in a class hierarchy, while taking into account partially correct predictions. The basic idea is to extend the notion of (flat) precision and recall in such a way that the set of predicted class labels and the set of actual (true) class labels of a test example are taken into account. Firstly, the hierarchical precision (hP) and hierarchical recall (hR) are computed as

$$hP = \frac{\sum_{i=1}^{n} |A_i \cap P_i|}{\sum_{i=1}^{n} |P_i|} \qquad hR = \frac{\sum_{i=1}^{n} |A_i \cap P_i|}{\sum_{i=1}^{n} |A_i|} \quad , \tag{2.4}$$

where A_i is the set of actual (true) class labels and P_i is the set of predicted class labels of the i-th test example, respectively, and n is the total number of test examples. It should be noted that both A_i and P_i comprise the complete set of actual and predicted class labels of the i-th test example according to the class hierarchy—i.e. the most specific class labels and all their ancestor class labels (except the root class label, since all examples trivially belong to the root class label by default). The hierarchical precision represents the ratio of the number of correct class labels predictions over the total number of class labels predicted across all test examples. The hierarchical recall represents the ratio of the number of correct class labels predictions over the total number of class labels that should have been predicted across all test examples.

Then, the hierarchical precision and hierarchical recall values can be combined into a hierarchical F-measure (hF), which is computed as

$$hF = \frac{2 \cdot hP \cdot hR}{hP + hR} \quad . \tag{2.5}$$

Considering the DAG-structured class hierarchy in Figure 2.4(b), the characteristics of the evaluation measures presented in Equations (2.4) and (2.5) can be summarised as:

1. Partially correct predictions are considered, e.g., misclassifying a test example into class label ‘E’ when it belongs to class label ‘F’ is less penalised than misclassifying it into class label ‘C’, since both ‘E’ and ‘F’ share the same parent class label ‘D’, while class label ‘C’ is not hierarchically related to ‘F’.

2. Errors at higher levels of the class hierarchy are more penalised than errors at lower levels of the class hierarchy, e.g., misclassifying a test example into class label ‘E’ when it belongs to class label ‘F’ is less penalised than misclassifying a test example into class label ‘A’ when it belongs to class label ‘B’. In the former case, the parent class label ‘D’ is correctly predicted, while in the latter no class label is correctly predicted.

2.5.2 Precision-Recall Curves

The output of a hierarchical and multi-label classification model is not necessarily a set of class labels for a particular test example—e.g. in the hierarchical multi-label Clus variations proposed by Vens et al. [127], the output is represented by a vector of class labels probabilities, wherein each component of the vector corresponds to the probability of predicting a particular class label. In such cases, there is a need to select a classification threshold in order to determine whether a class label is predicted or not. If the i-th component value is above a specified classification threshold, then the i-th class label is predicted; otherwise it is not predicted. Instead of arbitrarily selecting a threshold value—or a set of threshold values—to evaluate the (probabilistic) classification model, one can employ a threshold-independent evaluation measure. As discussed in [127], the main motivation for employing an evaluation measure independently from a classification threshold is that different contexts may require different threshold settings.

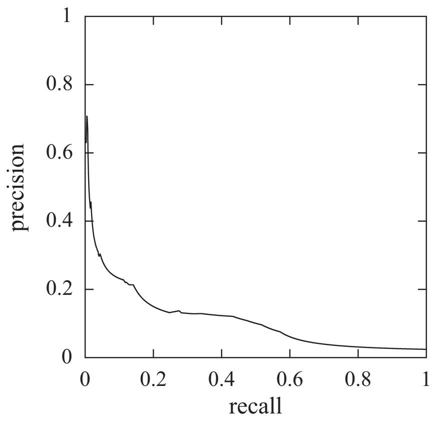
Precision-Recall (PR) curves have been frequently used in information retrieval [82, 101], and more recently in the context of hierarchical multi-label classification [127]. A PR curve plots a precision value against its recall value. The precision value corresponds to the number of correct predictions divided by the total number of predictions; the recall value corresponds to the number of correct predictions divided by the total number of positive examples—i.e., examples belonging to the predicted class label. One of the advantages of using PR curves as a performance measure is the ability to cope with highly skewed data sets (i.e., data sets with a much larger amount of negative examples in contrast to a smaller amount of positive examples) given that the number of negative examples is not used to calculate precision and recall values. This is an important characteristic concerning hierarchical classification since, as previously mentioned, class labels at the lower levels of the class hierarchy usually have much fewer (positive) examples. Therefore, it is more important to measure how well a classification model predicts the presence of a particular class label (positive examples) rather than its absence (negative examples).

A PR curve, illustrated in Figure 2.8, is defined by a set of points, where each point corresponds to a pair of precision and recall values for a particular classification threshold. Given a classification threshold t, decreasing the value of t from 1.0 to 0.0, different pairs of precision and recall values are obtained. With higher classification threshold values, fewer class labels are predicted (lower recall value) while the proportion of correct predictions tends to be greater (higher precision value). As the classification threshold is decreased, more class labels are predicted (higher recall values) while the proportion of correct predictions tends to decrease (lower precision values). Hence, the goal in PR curves is to be on the upper-right-corner, which corresponds to high precision and recall values.

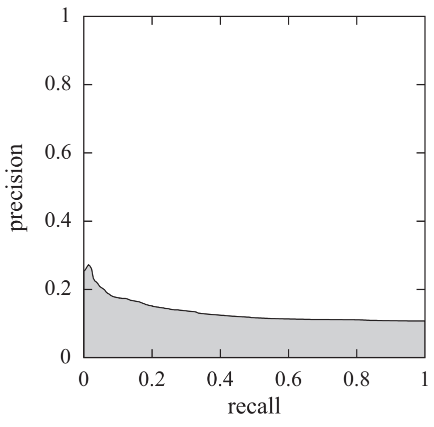
An approach to compute the points of a PR curve, and thus calculate the area under the curve, is described by Vens et al. [127]. This approach consists of creating an overall PR curve by micro-averaging precision and recall values across all class labels for a range of classification thresholds. The averaged precision ($\overline{Prec}$) and recall ($\overline{Rec}$) values for a classification threshold t is given by

$$\overline{Prec_t} = \frac{\sum_i TP_{t,i}}{\sum_i TP_{t,i} + \sum_i FP_{t,i}} \qquad \overline{Rec_t} = \frac{\sum_i TP_{t,i}}{\sum_i TP_{t,i} + \sum_i FN_{t,i}}, \tag{2.6}$$

where i ranges over all class labels (excluding the root label, since it is present in all examples), t ranges over all different probability values found in the vector of



(a)



(b)

Figure 2.8: Examples of precision-recall curves: in (a) a PR curve showing that higher precision values are generally associated with lower recall values; (b) a PR curve illustrating the shaded area of the curve, which corresponds to the area under the averaged PR curve measure, denoted as AU($\overline{\text{PRC}}$).

class labels probabilities, and $TP_{t,i}$, $FP_{t,i}$ and $FN_{t,i}$ are the number of true positives, false positives and false negatives for the i-th class label at the classification threshold t, respectively. The value of $\overline{Prec_t}$ corresponds to the number of correct class labels predictions divided by the number of class labels predicted across all class labels for the given classification threshold t—i.e., $\overline{Prec_t}$ is the proportion of predicted class labels that are correct. The value of $\overline{Rec_t}$ corresponds to the number of correct class labels predictions divided by the total number of class labels should have been predicted across all class labels for the given classification threshold t—i.e., $\overline{Rec_t}$ is the proportion of the available class labels that are correctly predicted. A pair of $\overline{Prec_t}$ and $\overline{Rec_t}$ values corresponds to a point of the PR curve.

Note that points of the PR curve must be interpolated in order to approximate the area under the curve, as discussed in [33]. After determining the points of the PR curve, the area under the PR curve can be approximated by calculating the trapezoidal areas created between each point. Finally, the evaluation measure is defined as the area under the averaged PR curve, denoted as AU($\overline{\text{PRC}}$).

2.6 Summary

An overview of the data mining area, discussing the common data mining tasks focusing on the classification task, has been presented in this chapter. It also described two main approaches for the discovery of comprehensible classification models in the context of flat single-label classification, providing examples of classification algorithms from the literature.

Furthermore, an overview of more complex classification problems—namely hierarchical, multi-label and hierarchical multi-label classification problems—has also been presented, together with a discussion of different approaches and algorithms employed in each case. Finally, this chapter presented an overview of the evaluation measures employed in the different classification problems, and the ones used to evaluate the classification algorithms proposed in this thesis.

Chapter 3

Ant Colony Optimisation

Ant colonies, despite the lack of centralised control and the relative simplicity of their individuals' behaviours, are self-organised systems which can accomplish complex tasks by having their individual ants interacting with one another and with their environment. The intelligent behaviour of the colony emerges from the indirect communication between the ants mediated by small modifications of the environment.

Inspired by the behaviour of natural ant colonies, Dorigo et al. [37, 38, 39] have defined an artificial ant colony metaheuristic that can be applied to solve optimisation problems, called ant colony optimisation (ACO). The main idea for the definition of ACO came from the fact that many ant species, even with limited visual capabilities or entirely blind, are able to find the shortest path between a food source and the nest. It was discovered that most of the communication amongst individual ants is based on the use of a chemical, called pheromone, that is dropped on the ground. As ants walk from a food source to the nest, pheromone is deposited on the ground, creating in this way a pheromone trail on the path used. Shorter paths will be traversed faster and, by consequence, will have stronger pheromone concentration than longer paths over a given period of time. The more pheromone a path contains, the more attractive it becomes to be followed by other ants. Hence, as time goes by, more and more ants will prefer the shorter path, which will have more and more pheromone. In the end, (almost) all ants will be following a single path, which usually will represent the shortest path between the food source and the nest. Figure 3.1 illustrates the iterative collective process of finding the shortest path between a food source and the nest.

ACO algorithms have been successfully applied to several types of optimisation problems [39, 40], such as scheduling and routing, and in the context of discovering

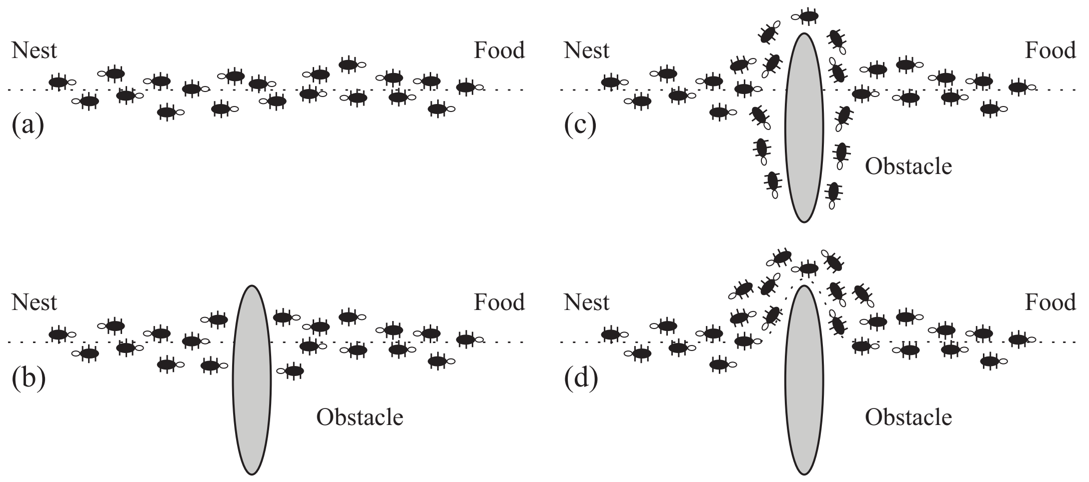


Figure 3.1: Ants are able to find the shorter path between a food source and the nest: in (a) ants in a pheromone trail between nest and food; (b) an obstacle interrupts the trail; (c) ants find two paths to go around the obstacle; (d) a new pheromone trail is formed along the shorter path. Adapted from (http://iridia.ulb.ac.be/∼mdorigo/ACO/).

classification rules in data mining [50]. In this chapter, an overview of the ACO metaheuristic is presented, followed by a description of the Ant-Miner algorithm—the first implementation of an ACO algorithm for the classification task of data mining [93, 94]—and its variations proposed in the literature.

The remainder of this chapter is organised as follows. Section 3.1 presents an overview of the ant colony optimisation (ACO) metaheuristic. The Ant-Miner classification algorithm is described in section 3.2. Section 3.3 presents Ant-Miner variations that have been proposed in the literature. Finally, section 3.4 presents the summary of this chapter.

3.1 The ACO Metaheuristic

Ant colony optimisation (ACO) algorithms simulate the behaviour of real ants using a colony of artificial ants, which cooperate in finding good solutions to optimisation problems. Each artificial ant, representing a simple agent, creates candidate solutions to the problem at hand and communicates indirectly with other artificial ants by means of pheromone. At the same time that ants perform a global search for new solutions, the search is guided to better regions of the search space based on the quality of solutions found so far. The algorithm converges to good solutions as a result of the collaborative interaction amongst the ants—an ant probabilistically chooses a path to follow based on heuristic information and the amount of pheromone deposited by previous ants. The interactive process of building candidate solutions and updating pheromone values allows an ACO algorithm to converge to optimal or near-optimal solutions. Algorithm 3.1 presents a high-level pseudocode of a basic ACO algorithm, comprising four main procedures:

1. *Initialise*: this procedure sets the parameters of the algorithm, and initialises the amount of pheromone and heuristic information associated with vertices or edges of the construction graph.

2. *ConstructAntsSolutions*: this procedure incrementally builds candidate solutions by creating paths on the problem's construction graph, simulating the movement of an artificial ant. Ants traverse the problem's construction graph by applying a stochastic decision policy based on the use of (problem-dependent) heuristic information and pheromone.

3. *ApplyLocalSearch*: this (optional) procedure is used to further refine a solution created by an ant. In general, local search operators/algorithms introduce small modifications to a solution in order to explore neighbour solutions. Dorigo and Stützle [39] have shown that for a wide range of optimisation problems, the use of local search operators/algorithms can boost the performance of ACO algorithms. A local search procedure is one example of problem specific or centralised (optional) daemon actions [39]—i.e. actions that cannot be performed by individual ants.

4. *UpdatePheromones*: this procedure updates the pheromone associated with the components—vertices or edges—of the problem's construction graph by either increasing the amount of pheromone, as ants deposit pheromone on the path used to build their candidate solutions, or decreasing the amount of pheromone, due to the simulation of pheromone evaporation. The quality of a candidate solution influences on how much pheromone will be deposited on the path that represents the candidate solution.

According to Algorithm 3.1, the iterative process of building and evaluating solutions, and updating pheromone is repeated until a termination condition is

Algorithm 3.1: High-level pseudocode of a basic ACO algorithm.

```
begin
    Initialise();
    while termination condition not met do
        ConstructAntSolutions();
        ApplyLocalSearch(); // optional
        UpdatePheromones();
    end
    return best solution;
end
```

met. In general, the termination condition is either a maximum number of iterations of the algorithm (maximum number of iterations of the *while* loop) or a stagnation test, which verifies if the solutions created by the algorithm cannot be further improved.

In the following subsections we discuss different design aspects of ACO algorithms, namely the problem representation explored by ants, the decision policy used to construct solutions and pheromone updating strategies. A more detailed discussion can be found in [39].

3.1.1 Problem Representation

In order to allow ants to incrementally create candidate solutions, the problem is mapped to a graph representation G, which consists of a set of vertices $V = \{v_1, v_2, \ldots, v_{N_v}\}$ (where N_v is the number of vertices) connected by a set of edges $E = \{e_{12}, e_{13}, \ldots, e_{ij}\}$ (where $i, j \leq N_v$ and $i \neq j$). This graph representation $G = (V, E)$ is called construction graph and it represents the problem search space—the vertices represent components of the solution to the problem and the edges represent the connection between two different vertices.

In general, the construction graph consists of a fully connected graph—i.e. there is an edge e_{ij} connecting every different pair of vertices v_i and v_j. Therefore, an ant located at vertex v_i could move to any vertex v_j. Depending of the problem considered, the construction graph may not be fully connected in order to implement constraints to guarantee that only feasible solutions are built—e.g. to prevent that an ant located at vertex v_i move to vertex v_j, the construction graph would not contain an edge between vertices v_i and v_j. Figure 3.2 presents examples of fully connected and not fully connected construction graphs.

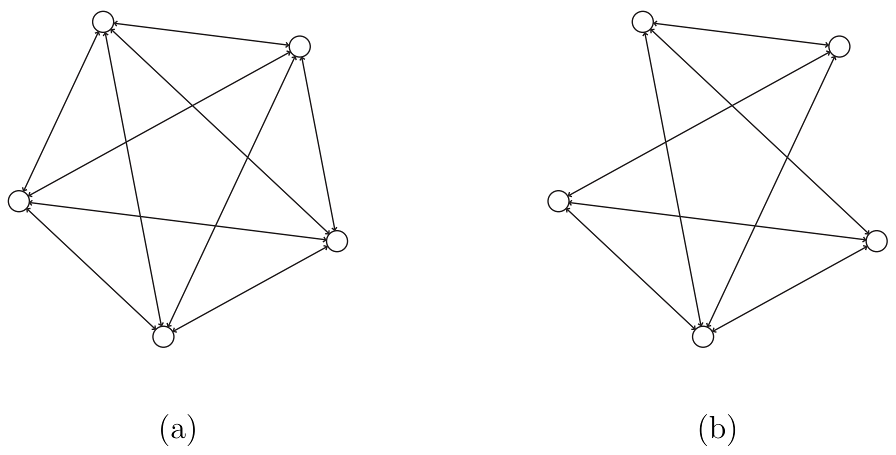


Figure 3.2: Example of graph representations used in ACO: in (a) a fully connected graph, where every pair of vertices is connected by an edge; (b) a not fully connected graph, where two vertices may not be connected by an edge.

3.1.2 Building Solutions

As aforementioned, candidate solutions are created by simulating the movement of artificial ants on the construction graph. Each ant incrementally creates a candidate solution by moving through neighbour vertices of the construction graph G. Hence, a candidate solution is represented by the list of visited vertices, which corresponds to a path in the construction graph. The vertices to be visited are chosen in a stochastic decision process, where the probability of choosing a particular neighbour vertex depends on both the problem dependent heuristic information η and the amount of pheromone τ associated with the neighbour vertex (η_j and τ_j, respectively) or the edge leading to the neighbour vertex (η_{ij} and τ_{ij}, respectively). The heuristic information represents a priori information—usually fixed during the execution of the algorithm—about the quality of the solution's component represented by a vertex or edge, while the pheromone varies—as a function of the algorithm iteration t—according to how frequent (the more frequent, the higher the pheromone) the vertex or edge has been used in previous candidate solutions. In the following equations involving pheromone values, the reference to the iteration t is omitted—e.g., $\tau_{ij}(t)$ is simply represented as τ_{ij}.

For example, an intuitive decision rule to select the next vertex to visit, which combines both the heuristic information and the amount of pheromone associated with vertices, is to make the decision based on the vertices' probabilities. Given an ant currently located at vertex v_i, the probability of selecting a neighbour vertex v_j is given by

$$P_{ij} = \frac{[\tau_j]^\alpha \cdot [\eta_j]^\beta}{\sum\limits_{j=1}^{|\mathcal{F}_i|}([\tau_j]^\alpha \cdot [\eta_j]^\beta)} \ , \qquad \forall\ j \in \mathcal{F}_i \ , \tag{3.1}$$

where τ_j and η_j are the pheromone value and heuristic information associated with the j-th vertex, respectively, $\mathcal{F}_i$ is the feasible neighbourhood of the ant located at vertex v_i (the set of vertices that the ant can visit from vertex v_i), α and β are (user-defined) parameters used to control the influence of the pheromone and heuristic information, respectively. According to Equation (3.1), the probability of choosing a particular neighbour vertex is higher for vertices associated with greater amount of pheromone and heuristic information, and subsequently increases in line with increases of the amount of pheromone. Equation (3.1) can be straightforwardly adapted to use the pheromone value and heuristic information associated with edges by replacing τ_j and η_j with τ_{ij} and η_{ij}, respectively.

3.1.3 Pheromone Trails

After all the ants have finished building the candidate solutions of an iteration, the updating of pheromone trails in the construction graph is usually accomplished in two steps, namely reinforcement and evaporation. The reinforcement step consists of increasing the amount of pheromone of every vertex (or edge, in the case that pheromone is associated with edges of the construction graph) used in a candidate solution and it is usually only applied to the best candidate solution—according to a problem dependent quality measure Q—of the current iteration. In general, the pheromone increment is proportional to the quality of the candidate solution, which in turn increases the probability that vertices or edges used in the candidate solution will be used again by different ants. Assuming that pheromone values are associated with vertices of the construction graph, a simple reinforcement rule is given by

$$\tau_i = \tau_i + \Delta Q(CS) \ , \qquad \forall\ i \in CS \ , \tag{3.2}$$

where $\Delta Q(CS)$ is the amount of pheromone proportional to the quality of the candidate solution CS to be deposited and τ_i is the pheromone value associated with the i-th vertex of the candidate solution CS.

The evaporation step consists of lowering the pheromone value of every vertex or edge—simulating the natural phenomenon of pheromone evaporation—in order to avoid a quick convergence of all ants toward a suboptimal solution. Assuming that pheromone values are associated with vertices, a simple evaporation rule is given by

$$\tau_i = (1 - \rho) \cdot \tau_i \ , \qquad \forall \ i \in G \ , \tag{3.3}$$

where $\rho \in (0, 1]$ is a parameter representing the evaporation factor, τ_i is the pheromone value associated with the i-th vertex of the construction graph G.

3.2 ACO applied to Classification: Ant-Miner

The first application of ACO for the classification task in data mining was reported by Parpinelli et al. [93, 94], where an ACO algorithm—called Ant-Miner—is proposed for the discovery of classification rules. Ant-Miner aims at extracting *IF-THEN* classification rules of the form *IF term*$_1$ *AND term*$_2$ *AND ... AND term*$_n$ *THEN class label* from a data set. Each term in the rule is a triple (*attribute*, *operator*, *value*), where *operator* represents a relational operator and *value* represents a value of the domain of *attribute*—e.g. (*gender*, =, *male*). The *IF* part corresponds to the rule's antecedent and the *THEN* part corresponds to the rule's consequent, which represents the class label to be predicted by the rule. An example that satisfies the rule's antecedent will be assigned the class label predicted by the rule.

Since Ant-Miner can only cope with nominal (categorical or discrete) attributes, the only valid relational operator is '=' (equality operator). If the data set contains continuous attributes, they are required to undergo a discretisation method in a preprocessing step prior to running Ant-Miner.[1]

A high-level pseudocode of Ant-Miner is presented in Algorithm 3.2 [94]. In summary, Ant-Miner works as follows. It starts with an empty list of rules and iteratively (*while* loop) adds one rule at a time to that list while the number of uncovered training examples is greater than a user-specified maximum value (*max_uncovered_examples* parameter)—in a sequential covering fashion. At each

[1]A discretisation method aims at converting continuous attributes into nominal (discrete) attributes by creating interval boundaries. A more detailed discussion of the discretisation method used in Ant-Miner is presented in chapter 5.

Algorithm 3.2: High-level pseudocode of the Ant-Miner algorithm [94].

```
input  : training examples
output: discovered list of rules
begin
    $training\_set \leftarrow$ all training examples;
    $rule\_list \leftarrow \emptyset$;
    while $|training\_set| > max\_uncovered\_examples$ do
        $\tau \leftarrow$ initial value of pheromone;
        $\eta \leftarrow$ initial value of heuristic information;
        $rule_{best} \leftarrow \emptyset$;
        $t \leftarrow 1$;
        repeat
            $rule_{current} \leftarrow \emptyset$;
            for $i \leftarrow 1$ to $colony\_size$ do
                $rule_i \leftarrow CreateRule()$;
                $calculate\_consequent(rule_i)$;
                $Prune(rule_i)$;
                if $Q(rule_i) > Q(rule_{current})$ then
                    $rule_{current} \leftarrow rule_i$;
                end
            end
            $UpdatePheromones(rule_{current}, \tau)$;
            if $Q(rule_{current}) > Q(rule_{best})$ then
                $rule_{best} \leftarrow rule_{current}$;
            end
            $t \leftarrow t + 1$;
        until $t \geq max\_number\_iterations$ OR $RuleConvergence()$ ;
        $rule\_list \leftarrow rule\_list + rule_{best}$;
        $training\_set \leftarrow training\_set - Covered(rule_{best}, training\_set)$;
    end
    return $rule\_list$;
end
```

iteration, a rule is created by an ACO procedure (*repeat-until* loop). In order to create a rule, ants probabilistically select terms to be added to their current partial rule based on the values of the amount of pheromone (τ) and a problem-dependent heuristic information (η) associated with vertices (terms) of the construction graph. Pheromone values and heuristic information are associated with each possible term—i.e. each possible triple (*attribute*, *operator*, *value*). As usual in ACO, the vertices' heuristic information—based on an information theoretical measure of the predictive power of the term—are fixed, while pheromone values are iteratively updated based on the quality of the rules built by ants.

Ants keep adding a term to their partial rule until any term added to their rule's antecedent would make their rule cover less training examples than a user-defined minimum value in order to avoid too specific and unreliable rules, or until all attributes have already been used. The latter rule construction stopping criterion is necessary because an attribute can only occur once in the antecedent of a rule, in order to avoid inconsistencies such as '$gender = male$ AND $gender = female$'. Once the rule construction process has finished, the rule created by an ant is pruned to remove irrelevant terms from the rule antecedent. Then, the consequent of a rule is chosen to be the class label most frequent amongst the set of training examples covered by the rule in question. Finally, pheromone trails are updated using the best rule—based on a quality measure Q—of the current iteration and the best-so-far rule across all iterations is stored/updated.

The process of constructing a rule is repeated until a user-specified number of iterations has been reached (*max_number_iterations* parameter), or the best rule of the current iteration is exactly the same as the best rule constructed by a predefined number of previous iterations, which works as a rule convergence test. The best rule found along this iterative process is added to the list of rules and the covered training examples (training examples that satisfy the antecedent of the best rule) are removed from the training set.

3.2.1 Construction Graph

Given a set of nominal attributes $\mathcal{X} = \{x_1, \ldots, x_n\}$, where the domain of each nominal attribute x_i is a set of values $\mathcal{V}_i = \{v_{i1}, \ldots, v_{id_i}\}$ (where d_i equals to the number of values in the domain of attribute x_i), the Ant-Miner's construction graph consists of an almost fully connected graph. For each pair of nominal attribute x_i and value v_{ij} (where x_i is the i-th nominal attribute and v_{ij} is the

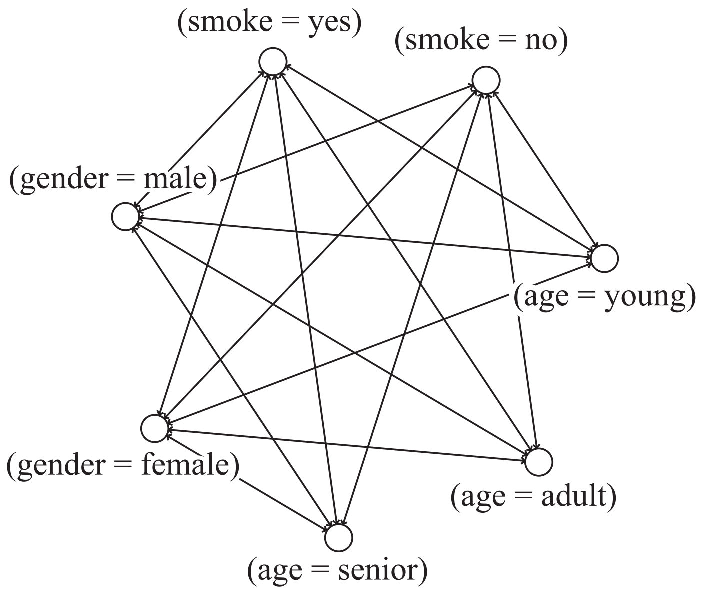


Figure 3.3: The construction graph of Ant-Miner, given a data set containing three nominal attributes, namely '*age*', '*gender*' and '*smoke*'. This example assumes that the '*age*' attribute was originally a continuous attribute, which was discretised into three different intervals $\{young,\ adult,\ senior\}$ in a preprocessing step.

j-th value belonging to the domain of x_i), a vertex representing the term $x_i = v_{ij}$ is added to the construction graph. Then, vertices are connected by edges to every other vertex of the construction graph, with the restriction that there are no edges between vertices referring to the same attribute to avoid inconsistent terms such as '*gender* = *male* AND *gender* = *female*' being included in the same rule. Figure 3.3 illustrates the construction graph of Ant-Miner, given a data set containing three nominal attributes—namely '*gender*', '*smoke*' and an originally continuous attribute '*age*' converted into a nominal attribute.

3.2.2 Rule Construction

In Ant-Miner, ants create rules by probabilistically selecting terms (vertices) of the construction graph. Each ant starts with an empty rule—i.e. a rule with an empty antecedent—and iteratively selects terms to add to its partial rule based on their values of the amount of pheromone τ and a problem-dependent heuristic information η.

The probability of selecting a particular term $x_i = v_{ij}$ at each iteration of the rule construction process is given by

$$P_{x_i=v_{ij}} = \frac{\tau_{x_i=v_{ij}} \cdot \eta_{x_i=v_{ij}}}{\sum_{i=1}^{n}[not\_used(x_i) \cdot \sum_{j=1}^{d_i}(\tau_{x_i=v_{ij}} \cdot \eta_{x_i=v_{ij}})]} \,, \tag{3.4}$$

where $not\_used(x_i)$ is 1 if the attribute x_i has not yet been used in the partial rule and 0 otherwise[2], $\tau_{x_i=v_{ij}}$ and $\eta_{x_i=v_{ij}}$ are the amount of pheromone and heuristic information associated with a neighbour term $x_i = v_{ij}$, respectively, n is the total number of attributes and d_i is the total number of values in the domain of attribute x_i. The selection of a term $x_i = v_{ij}$ is restricted to (1) those terms that do not contain an attribute that is already used in the current partial rule, and (2) those terms whose addition would not make the rule cover less than a user-defined minimum number of training examples. The former restriction prevents the creation of invalid rules, e.g. '*IF gender* = *male AND* ... *AND gender* = *female THEN N*', while the latter restriction prevents the creation of rules covering a small number of training examples, which would typically be a case of overfitting.

This iterative process of selecting a term $x_i = v_{ij}$ to be added to the current partial rule—with probability proportional to the value $P_{x_i=v_{ij}}$ given by Equation (3.4)—is repeated until one of the following stop conditions is met:

1. all attributes have been used in the rule's antecedent, so there are no more terms available;

2. any term to be added to the current partial rule would make it cover less than a user-defined minimum number of training examples.

Once an ant finishes the creation of a rule, the consequent of the rule is set to be the majority class label amongst the examples covered (the examples that satisfy the antecedent of the rule) by the rule—e.g., if the rule covers 20 examples, 5 examples with class label 'N' (negative) and 15 examples with class label 'P' (positive), the consequent of the rule would be set to class label 'P'; ties are broken at random.

[2]The purpose of the $not\_used(x_i)$ verification is to avoid the influence of terms that cannot be used in the rule antecedent, since the attribute x_i has been already used, when normalising the probability of neighbour vertices.

3.2.3 Heuristic Information

The heuristic information associated with every vertex of the construction graph corresponds to an estimate of the quality of the term represented by the vertex, with respect to its capacity of improving the predictive accuracy of a rule. It is based on information theory [30], more precisely, it involves a measure of entropy associated with each term. The entropy of each term $x_i = v_{ij}$ is computed as

$$\mathit{entropy}(x_i = v_{ij}; S) = \sum_{c=1}^{k} [-p(c \,|\, S_{x_i=v_{ij}}) \cdot \log_2 p(c \,|\, S_{x_i=v_{ij}})] \,, \tag{3.5}$$

where $p(c \,|\, S_{x_i=v_{ij}})$ is the empirical probability of observing class label c conditional on having observed $x_i = v_{ij}$ (attribute x_i having the value v_{ij}) in the set of training examples S and k is the total number of class labels. Note that the entropy is a measure of the (im)purity in a collection of examples, hence higher entropy values correspond to more uniformly distributed examples (examples associated with different class labels) and smaller predictive power for the term represented by the vertex in question; lower entropy values correspond to more homogeneous examples (more examples associated with the same class label) and higher predictive power for the term represented by the vertex in question.

Since the entropy value of a term varies in the range $0 \leq \mathit{entropy}(x_i = v_{ij}; S) \leq \log_2 k$ and lower entropy values are preferred over higher entropy values, the heuristic information of the term $x_i = v_{ij}$ corresponds to the normalised entropy value, given by

$$\eta_{x_i=v_{ij}} = \frac{\log_2 k - \mathit{entropy}(x_i = v_{ij}; S)}{\sum\limits_{i=1}^{n} (\mathit{not\_used}(x_i) \cdot \sum\limits_{j=1}^{d_i} [\log_2 k - \mathit{entropy}(x_i = v_{ij}; S)])} \,, \tag{3.6}$$

where $\mathit{not\_used}(x_i)$ is 1 if the attribute x_i has not yet been used in the partial rule and 0 otherwise, n is the total number of attributes and d_i is the total number of values in the domain of attribute x_i. According to Equation (3.6), the lower the entropy of term $x_i = v_{ij}$, the higher its heuristic information, which in turn increases its probability of being selected by an ant. The heuristic information of every term is computed in an initialisation step—rather than every time the heuristic information is used to compute the probability of selecting a vertex—to save computational time, since the heuristic information of a term is fixed during the rule construction process.

3.2.4 Rule Evaluation

After the creation of a rule and the assignment of its consequent, a quality measurement is associated with the rule. The quality of a rule is used to determine the best rule of an iteration, the best-so-far rule across all iterations and the amount of pheromone to be deposited in the trail represented by the rule.

The quality function Q used in Ant-Miner, which corresponds to the measure of *sensitivity* $\cdot$ *specificity*, is given by

$$Q = \frac{TP}{TP + FN} \cdot \frac{TN}{FP + TN} \ , \tag{3.7}$$

where TP is the number of examples covered by the rule that have the class label predicted by the rule, FN is the number of examples not covered by the rule that have the class label predicted by the rule, TN is the number of examples not covered by the rule that do not have the class label predicted by the rule and FP is the number of examples covered by the rule that do not have the class predicted by the rule. According to Equation (3.7), the quality value of a rule is in the range of $0 \leq Q \leq 1$, and the higher the value the better is the quality.

3.2.5 Rule Pruning

The rules created by ants undergo a pruning procedure, which aims at improving the rule quality and generalisation behaviour by removing irrelevant terms from their antecedent. The rule pruning is an iterative procedure consisting of removing one term—the term that leads to the higher improvement of the rule in terms of quality—at a time until only one term is left in the antecedent of the rule or the removal of any of the remaining terms leads to a decrease in the quality of the rule. In ACO terminology, Ant-Miner's rule pruning procedure characterises a local search operator. Algorithm 3.3 presents a high-level pseudocode of Ant-Miner's rule pruning procedure.

Let *rule* be the rule undergoing the pruning, which is considered the best rule ($rule_{best}$) at the beginning of the pruning procedure. At each iteration of the *for* loop in Algorithm 3.3, a candidate rule $rule_i$ is created by removing the i-th term of the antecedent of best rule found so far $rule_{best}$. Since the removal of a term can lead the rule to cover a greater number of examples, the consequent of the rule is set to the majority class label of the (extended) set of covered examples, as described in Subsection 3.2.2. If the quality of $rule_i$ is higher than the quality of the best rule of the current iteration ($rule_{current}$), $rule_i$ substitutes

Algorithm 3.3: High-level pseudocode of Ant-Miner's rule pruning procedure.

```
input  : rule to be pruned
output: the pruned rule
1  begin
2      $rule_{best} \leftarrow rule$;
3      repeat
4          $rule_{current} \leftarrow rule_{best}$;
5          for $i \leftarrow 1$ to $|rule_{best}.antecedent|$ do
6              $rule_i \leftarrow RemoveTerm(i, rule_{best}.antecedent)$;
7              if $Q(rule_i) > Q(rule_{current})$ then
8                  $rule_{current} \leftarrow rule_i$;
9              end
10         end
11         if $Q(rule_{current}) > Q(rule_{best})$ then
12             $rule_{best} \leftarrow rule_{current}$;
13         end
14     until $Q(rule_{current}) < Q(rule_{best})$ OR $|rule_{best}.antecedent| = 1$ ;
15     return $rule_{best}$;
16 end
```

$rule_{current}$. After evaluating the individual removal of every term of the antecedent of $rule_{best}$, which comprises the evaluation of $|rule_{best}.antecedent|$ rules (where $|rule_{best}.antecedent|$ corresponds to the number of terms in the antecedent of the rule), if the quality of the best rule of the current iteration ($rule_{current}$) is higher than the best rule found so far ($rule_{best}$), $rule_{current}$ substitutes $rule_{best}$, completing an iteration of the pruning procedure (*repeat* loop). This procedure is repeated until $rule_{best}$ has just one term left on its antecedent or the best rule of the current iteration does not improve the quality over the best so far rule—i.e. $Q(rule_{current}) < Q(rule_{best})$.

3.2.6 Pheromone Trails

Pheromone Initialisation

At the beginning of each iteration of the *while* loop in Algorithm 3.2—i.e. the beginning of an iteration of Ant-Miner's sequential covering approach—the initial pheromone associated with every vertex is set to be inversely proportional to the number of vertices of the construction graph, given by

$$\tau_{x_i=v_{ij}} = \frac{1}{|V|} \ , \tag{3.8}$$

where $|V|$ is the total number of vertices of the construction graph. According to Equation (3.8), all vertices of the construction graph have the same amount of pheromone at the beginning of the search for a rule.

Pheromone Reinforcement

In order to reinforce the trail representing the best rule of the current iteration of the *repeat* loop in Algorithm 3.2 (denoted as $rule_{current}$), the pheromone of each term occurring in the antecedent of the rule is increased based on the quality of the rule. Given the best rule of the current iteration, the pheromone increment for used terms is given by

$$\tau_{x_i=v_{ij}} = \tau_{x_i=v_{ij}} + [\tau_{x_i=v_{ij}} \cdot Q(rule_{current})] \ , \qquad \forall\ x_i = v_{ij} \in rule_{current}.antecedent \ , \tag{3.9}$$

where $rule_{current}$ is the best rule of the current iteration and $Q(rule_{current})$ corresponds to its quality. Recall that the quality value of a rule is within the range $0 \leq Q(rule_{current}) \leq 1$, and therefore the pheromone increment given by Equation (3.9) to each vertex representing a term in the antecedent of the rule is proportional to the quality of the rule and the current amount of pheromone of the term $x_i = v_{ij}$ in question.

Pheromone Evaporation

The pheromone evaporation in Ant-Miner is achieved by normalising the pheromone of each vertex of the construction graph—i.e. dividing the pheromone of each vertex by the sum of pheromone over all vertices of the construction graph. Hence, the pheromone of vertices that were not reinforced are implicitly lowered as a result of the normalisation. This normalisation is given by

$$\tau_{x_i=v_{ij}} = \frac{\tau_{x_i=v_{ij}}}{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{d_i} \tau_{x_i=v_{ij}}} \ , \qquad \forall\ x_i = v_{ij} \ , \tag{3.10}$$

where n is the total number of attributes and d_i is the total number of values in the domain of attribute x_i. Note that the normalisation does not affect the increment of pheromone of used vertices due to the application of Equation (3.9)—i.e. the pheromone of used vertices increases relatively to decreases of the pheromone of unused vertices.

3.2.7 Classifying New Examples

After the creation of the list of rules is completed (end of Algorithm 3.2), a new (unseen test) example can be classified by applying the rules in a sequential order—i.e. in the order that they were created. Following the sequential order of the rules, the first rule that covers the new example (the first rule for which the new example satisfies the antecedent of the rule) classifies the new example into the class label specified by the rule's consequent.

In the case where no rule covers the new example, a default rule (a rule with an empty antecedent) predicting the majority class label of all uncovered training examples is used to classify the new example. Since the default rule has an empty antecedent, it covers any new example and therefore it is used as the last resource to classify new examples.

3.3 Ant-Miner Extensions

Following the introduction of the Ant-Miner classification algorithm, several variations were proposed in the literature [50]. They involve different rule pruning and pheromone update procedures, new rule quality measures and heuristic information, and the application to more complex classification problems—e.g., discovering fuzzy classification rules and discovering rules for multi-label classification problems—amongst others.

Chan and Freitas [22] have proposed a new rule pruning procedure for Ant-Miner. They have observed that the original Ant-Miner's rule pruning procedure computational time increases significantly with a large increase in the number of attributes, which affects the scalability of the algorithm when dealing with larger (in terms of number of attributes) data sets. To mitigate this limitation, it was proposed a hybrid rule pruning procedure. The basic idea is to select a subset of the terms in the antecedent of a rule, based on the information gain [30, 133] of the terms, and subsequently apply the original Ant-Miner's rule pruning procedure in the reduced set of terms. Experiments with the hybrid pruning procedure have led to the discovery of simpler (shorter) rules and improved the computational time in datasets with a large number of attributes, though in some cases at the cost of decreasing the predictive accuracy when compared to the original Ant-Miner's rule pruning procedure.

Galea and Shen [53] presented an ACO approach for the induction of classification fuzzy rules, named FRANTIC-SRL (Fuzzy Rules from ANT-Inspired Computation – Simultaneous Rule Learning). FRANTIC-SRL runs several ACO algorithm instances in parallel, where each instance generates rules for a particular class label—i.e. the class label that a rule predicts is fixed in advance for each ACO instance. By having multiple ACO instances, separate construction graphs, pheromone matrices and heuristic information are maintained for each class label. This approach differs from Ant-Miner, where different ants in the same colony can predict different class labels and there is a single pheromone matrix updated by the ACO algorithm, containing the amount of pheromone for terms considering their ability in discriminating amongst all class labels as a whole.

Swaminathan [121] proposed an extension to Ant-Miner, which enables interval conditions (terms using the '$<$' and '$>$' relational operators) in the antecedent of rules. While it still uses a discretisation method to define discrete intervals for continuous attributes in a preprocessing step, the continuous values are not replaced in the data set—i.e. although the algorithm can handle continuous values, the discrete intervals of a continuous attribute are fixed during its execution. For each discrete interval of a continuous attribute, a vertex (e.g. $age < 21$) is added to the construction graph and the amount of pheromone associated to the vertex is calculated using a mixed kernel probability density function. Experiments comparing the proposed extension and the original Ant-Miner have shown that overall both algorithms achieved similar results in terms of predictive accuracy. This suggests that by having discrete intervals fixed during the execution of the algorithm, the ability of handling continuous values cannot be fully explored and consequently, it does not provide a clear advantage over replacing the continuous values for discrete intervals as in the original Ant-Miner. Another drawback of the proposed extension is that the complexity of the construction graph can highly increase when applied to data sets with a large number of continuous attributes, each of them containing a large number of discrete intervals, since for each continuous attribute's discrete interval, a vertex is added to the construction graph.

Martens et al. [83] have introduced a new ant colony classification algorithm, named AntMiner+, based on Ant-Miner. It differs from the original Ant-Miner implementation in several aspects. Firstly, the complexity of the construction graph is reduced, in term of the number of edges connecting vertices, by defining it as a direct acyclic graph (DAG). For each nominal attribute x_i, a vertex group consisting of vertices representing the terms $x_i = v_{ij}$ (where v_{ij} is the j-th value in the domain of the attribute x_i), in addition to a dummy vertex which represents the absence of attribute x_i, is created. Then, vertices groups are sequentially connected—i.e., the vertex group of attribute x_i is fully connected to the vertex group of attribute x_j (where $i < j$)—to form a DAG, where their order is not relevant.

Secondly, it makes a distinction between nominal attributes with categorical and ordered values. Categorical nominal attributes are those that have unordered nominal values (e.g. the gender attribute has unordered values 'male' and 'female'), while ordinal nominal attributes are those nominal attributes whose values are ordered (e.g. '0', '1', '2', '3' and '4 or more', which may be the domain of an attribute that represents the number of children in a family). Instead of creating a vertex group for an ordinal nominal attribute x_i, AntMiner+ creates two groups of vertices that represent the interval of values to be chosen by the ants. The first group represents the lower bound of the interval and takes $x_i \geq v_{ij}$ form, and the second group represents the upper bound of the interval and takes $x_i \leq v_{ij}$ form (where v_{ij} is the j-th value in the domain of the attribute x_i). Additionally, a dummy vertex representing the absence of a lower or upper bound interval is added to each of the vertices groups.

Thirdly, the class label to be predicted and weight parameters (α and β) used to control the influence of the pheromone and heuristic information, respectively, are incorporated in the construction graph as vertices. Therefore, before an ant starts selecting vertices (terms) to create a rule, it first selects the class label and weight parameters using the same stochastic decision process—based on heuristic information and pheromone—applied to vertices representing terms. Since the class label to be predicted by the rule currently being built is known before an ant starts creating its antecedent, AntMiner+ also employs a class-specific heuristic information.

Lastly, it employs different pheromone initialisation and update procedures based on the $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) [115, 116]. In essence, in the $\mathcal{MAX}$-$\mathcal{MIN}$ ant system, the amount of pheromone is limited to an interval $[\tau_{min}, \tau_{max}]$ in order to avoid a quick convergence to a local-optimal solution. The initial pheromone is set at τ_{max} for every vertex (or edge, in the case that pheromone is associated with edges) of the construction graph, and it is gradually decreased towards τ_{min} for those vertices (or edges) that are not used in the candidate solutions due to the simulation of pheromone evaporation.

Experiments comparing AntMiner+ and Ant-Miner in terms of predictive accuracy have shown that AntMiner+ performs significantly better than Ant-Miner. Although AntMiner+ has the same limitation of only being able to cope with nominal attributes as the original Ant-Miner, the authors have empirically demonstrated that the incorporation of interval conditions (for ordinal nominal attributes) was relevant for increasing the predictive accuracy of the algorithm.

Chan and Freitas [23] proposed an Ant-Miner extension, named MuLAM (Multi-Label Ant-Miner), for the multi-label classification task. In essence, MuLAM differs from the original Ant-Miner in three aspects. Firstly, a classification rule can predict one or more class labels, as in multi-label classification problems an example can be associated to more than one class label simultaneously. After creating the antecedent of a rule, the class labels to be predicted by the rule are chosen based on a confidence measure—i.e. for each class label, if the confidence is higher than a certain threshold, the class label is added to the consequent of the rule. During the MuLAM's rule pruning procedure, instead of allowing the consequent to be modified during pruning as in Ant-Miner, the consequent of the rule is kept fixed. Secondly, at each iteration, each ant creates a set of rules instead of a single rule as in the original Ant-Miner. The set of rules contains rules that predict all the different class labels. Thirdly, it uses a pheromone matrix for each class label and pheromone updates only occur on the matrix of class labels that occur in the consequent of a rule predicting that class label. The amount of pheromone deposited on the terms used in a rule is based on the quality of the rule, as in Ant-Miner.

Although the result of experiments comparing MuLAM and Ant-Miner[3] have shown no significant differences in terms of predictive accuracy, MuLAM has the advantage of discovering rules that can predict more than one class label using the same rule antecedent. Hence, the correlations between difference class labels can be made explicit.

Despite the Ant-Miner variations proposed in the literature, to the best of our knowledge, incorporating continuous attributes into Ant-Miner's rule construction process 'on-the-fly'—instead of requiring a discretisation method in a

[3]Since Ant-Miner is a single-label classification algorithm, multiple runs are required to individually discover rules for each class label.

preprocessing step—and extending Ant-Miner to cope with hierarchical classification problems are areas of research that have not yet been explored. We believe that extending Ant-Miner to cope with continuous attributes 'on-the-fly' would enhance its predictive accuracy given that the use of a discretisation method in a preprocessing step can lead to loss of predictive power, since less information is available to the classification algorithm. An extension of Ant-Miner, which has been successfully applied to flat single-label classification problems, to cope with hierarchical classification scenarios would enable its application to more complex types of classification problems, e.g. protein function prediction using hierarchical functional classification schemes[4]—which is a major type of bioinformatics problem addressed in this thesis.

It should be noted that while ACO was originally proposed to solve combinatorial (discrete) optimisation problems, where each variable has a finite number of values, adaptations of ACO to continuous and mixed (with both continuous and discrete variables) optimisation problems have been proposed in the literature [40]. For example, Socha and Dorigo [113] have presented an extension of ACO applied to continuous optimisation of numeric functions; Blum and Socha [14] applied an extended ACO to the problem of optimising the (numeric) weights of neural networks; Hong et al. [64] proposed an extended ACO to optimise (numeric) parameters in support vector regression (SVR); further examples can be found in [112, 126]. However, we emphasize that the domain of application of these ACO extensions to continuous optimisation problems proposed in the literature is very different from the problem of discovering classification rules in data mining. Therefore, these approaches cannot be directly applied to discover classification rules from data sets with nominal and continuous attributes.

3.4 Summary

This chapter presented an overview of the ant colony optimisation (ACO) metaheuristic—a metaheuristic inspired by the behaviour of natural ant colonies. The basic idea of ACO is the use a colony of artificial ants (simple agents) to create solutions to optimisation problems in a collaborative fashion, based on indirect communication by means of pheromone.

Furthermore, it presented a description of Ant-Miner, the first implementation of an ACO algorithm for the classification task in data mining, followed by a

[4]An overview of the problem of protein function prediction is presented in section 4.4.

discussion of Ant-Miner variations proposed in the literature. Regardless of the proposed variations, there are two additional unexplored research areas: (1) an extension of Ant-Miner to cope with continuous attributes 'on-the-fly'—during the rule construction process—without the need for the use of a discretisation method in a preprocessing step; (2) an extension of Ant-Miner to cope with hierarchical classification problems. New extensions of Ant-Miner to deal with these two types of problem are presented in the following chapters.

Chapter 4

Bioinformatics

Bioinformatics became a very popular research area after the fully sequenced genomes of numerous organisms enabled biologists to map, sequence and analyse individual genes and their protein products. It comprises various different lines of study, e.g. the identification of regions of similarity in sequences of DNA, RNA or protein, the prediction of the three-dimensional structure of a protein, the identification of interactions between proteins and the prediction of proteins functions.

In essence, bioinformatics refers to the research area that combines computational and statistical methods to manage and analyse biological data, as defined by Higgs and Attwood [60, p. 6]:

> "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."

The recent exponential increase in the number of proteins being identified and sequenced using high-throughput experimental approaches has lead to a growth in the number of uncharacterised (with unknown function) proteins. Determining protein functions is a central goal of bioinformatics, and it is crucial to improve biological knowledge, diagnosis and treatment of diseases. While biological experiments are the ultimate methods to determine the function of proteins, it is not possible to perform a functional assay for every uncharacterised protein. This is due to time and financial constraints, together with the complex nature of these experiments. Hence, a need for using computational methods to assist the annotation of large amounts of protein data appeared. In particular, this presents a significant opportunity to apply data mining techniques to analyse and extract knowledge from biological databases.

Proteins are large and complex molecules, assembled from amino acids arranged in a linear sequence using information encoded in genes. Proteins perform most of the functions within a cell—e.g., almost all biological processes, including metabolism, need enzymes to catalyse chemical reactions in order to occur; transport proteins are involved in the movement of small molecules through membranes. Since proteins do almost all the work in a cell and even make up the majority of cellular structures, understanding the roles of proteins is the key to understanding how the whole cell operates.

Biological databases accumulate vast amounts of protein data, from protein sequences to three-dimensional structures. To facilitate both collaboration and standardisation across different sources, biological databases employ controlled vocabularies (ontologies) to annotate protein sequences and features. Ontologies such as the Enzyme Commission (EC) [130], Gene Ontology [4] and FunCat [105] are organised in a hierarchical structure, allowing the annotation of proteins in terms of their functions at different levels of detail. As a result, the problem of predicting protein functions can be cast as a hierarchical classification problem, where different protein features—e.g., biochemical properties of the amino acids in the protein sequence—are used as predictor attributes and the different functions that a protein can perform are used as class labels to be predicted.

This chapter describes the problem of predicting protein functions, discussing how the protein information available in biological databases can be used to define the target problem as a hierarchical classification problem, followed by a review of current approaches for protein function prediction. Additionally, it presents an overview of the main biological databases and protein functional classification schemes.

The remainder of this chapter is organised as follows. Section 4.1 presents a biological background about proteins. An overview of the main biological databases containing protein information and protein functional classification schemes is presented in sections 4.2 and 4.3, respectively. In section 4.4, the problem of predicting protein functions and current approaches found in the literature are discussed. Finally, section 4.5 presents the summary of this chapter.

4.1 Biological Background

The genetic information of living organisms is stored in DNA (deoxyribonucleic acid) molecules. DNA is a long molecule composed by a sequence of deoxyribonucleic bases, which are linked together by a backbone composed by deoxyribose sugar and phosphate groups. There are four possible nucleotide bases that are found in DNA: adenine (A), guanine (G), cytosine (C) and thymine (T). The DNA molecule structure is a double stranded helix, which is dependent on pairing between the nucleotide bases—adenine is able to pair with thymine, while guanine is able to pair with cytosine. Consequently, the strands in the double helix complement each other in an anti-parallel fashion.

Segments of nucleotide sequences—corresponding to particular genes—in DNA molecules specify amino acid sequences in protein molecules, defining a relationship between DNA and protein molecules. This relationship is part of the *central dogma* of molecular biology [2]. The central dogma states that the information flows from DNA to RNA (ribonucleic acid) to protein. In summary, this is a two-step process, as illustrated in Figure 4.1. In the first step, called transcription, the genetic information stored in a segment of the DNA is used to create a mRNA (*messenger* RNA) molecule. RNA molecules are single-stranded molecules composed by a sequence of four-bases similarly to DNA molecules, with the exception that uracil (U) replaces thymine (T), and the bases are linked together by a backbone composed by ribose sugar.

In the second step, called translation, the mRNA molecule is used as a template in the synthesis of a protein molecule. A series of three nucleotides in the mRNA corresponds to a codon, which in turn corresponds to either a specific amino acid or a signal site to indicate the start/stop of the translation process. There are twenty different amino acids and sixty-four possible codons (four different bases arranged in triples), therefore several codons correspond to the same amino acid. The translation is a complex process carried out by the ribosome, which itself is composed by RNA and proteins molecules, representing a multimolecular 'machine'. For more details about this process refer to [2, 60].

4.1.1 Proteins

Proteins are the building blocks from which every cell in an organism is built [2]. A protein molecule is assembled from a long sequence of amino acids using information encoded in genes (segments of the DNA sequence). Each protein

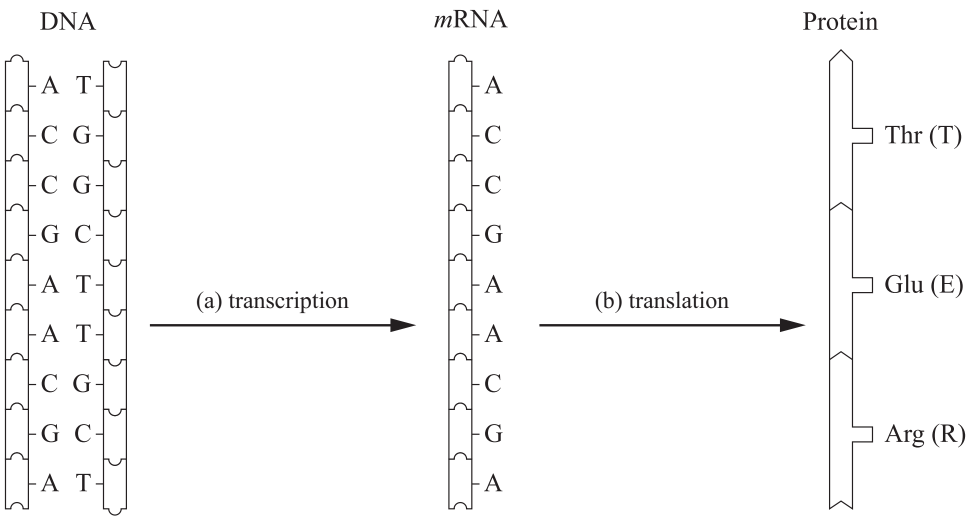


Figure 4.1: The central dogma's information flow: from DNA to RNA to protein: (a) in the transcription step, the genetic information stored in a segment of the DNA is used to create a mRNA (*messenger* RNA) molecule; (b) in the translation step, the mRNA molecule is used as a template in the synthesis of a protein molecule.

has its own unique sequence of amino acids, which is specified by the nucleotide sequence of the gene encoding the protein. There are twenty different types of amino acids—each with different biochemical properties—that can be found in a protein sequence. An amino acid is composed by a central carbon (C)—called the α carbon—attached to a hydrogen (H), an amino group (NH_2), a carboxyl group (COOH) and a variable side chain (R). There are twenty distinct side chains, resulting in the twenty different types of amino acids. The amino acids are linked together by a peptide bond between their amino and carboxyl groups, constituting the protein's backbone—illustrated in Figure 4.2. In general, proteins are 200-400 amino acids long.

The amino acid sequence of a protein is also known as the protein's *primary structure*. It determines the protein's three-dimensional structure (the shape of the protein) and function. Subsequently bondings between the amino and carboxyl groups from different amino acids allow the linear sequence to fold into structures known as alpha helices and beta sheets. Alpha helices (α-helices) are formed when the backbone twists into right-handed helices. Beta sheets (β-sheets) are formed when the backbone folds back on itself in either a parallel or anti-parallel fashion. These structures constitute the protein's *secondary structure*.

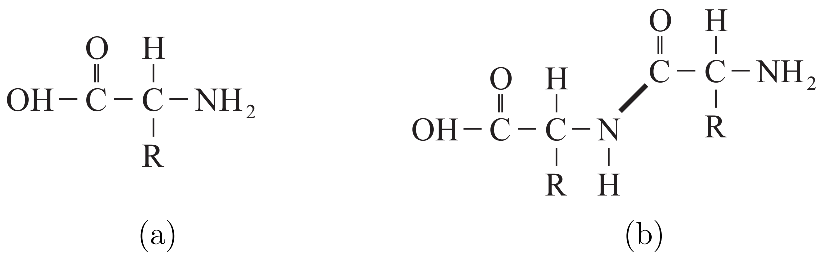


Figure 4.2: In (a) basic amino acid structure; (b) the peptide bond between two amino acids (thick line). Amino acids are linked together by a peptide bond between their amino and carboxyl groups, constituting the protein's backbone. This process is repeated many times for polypeptide proteins.

The three-dimensional shape of the whole protein is known as the protein's *tertiary structure*, which is defined by the spatial relationship between the secondary structures. The three-dimensional shape of a protein is crucial for its function, hence discovering its tertiary structure can provide important information about how the protein performs its function. Proteins are also capable of assembly into complex structures, known as the protein's *quaternary structure*, as a result of interaction between them. There are some proteins that can only be functional when associated in protein complexes [79]. Figure 4.3 illustrates the four levels of organisation in the structure of a protein.

The complexity of determining the different levels of protein structures increases from primary towards quaternary structures. For instance, the primary structure can be determined by simply translating the DNA sequence of the gene that specifies the protein to an amino acid sequence, while the tertiary structure can be determined using complex X-ray crystallography experiments. Consequently, many more proteins sequences (primary structures) are known than proteins three-dimensional structures (tertiary structures). The process from which a protein in one-dimensional state (primary structure) turns to a three-dimensional state (tertiary structure) is called *folding*. While folding occurs spontaneously within a cell, its inherent details are not known. For many proteins, their ability to fold into a proper three-dimensional structure is essential to their function—e.g., antibodies through their surface's binding sites are able to attach to viruses or bacteria in order to destroy them.

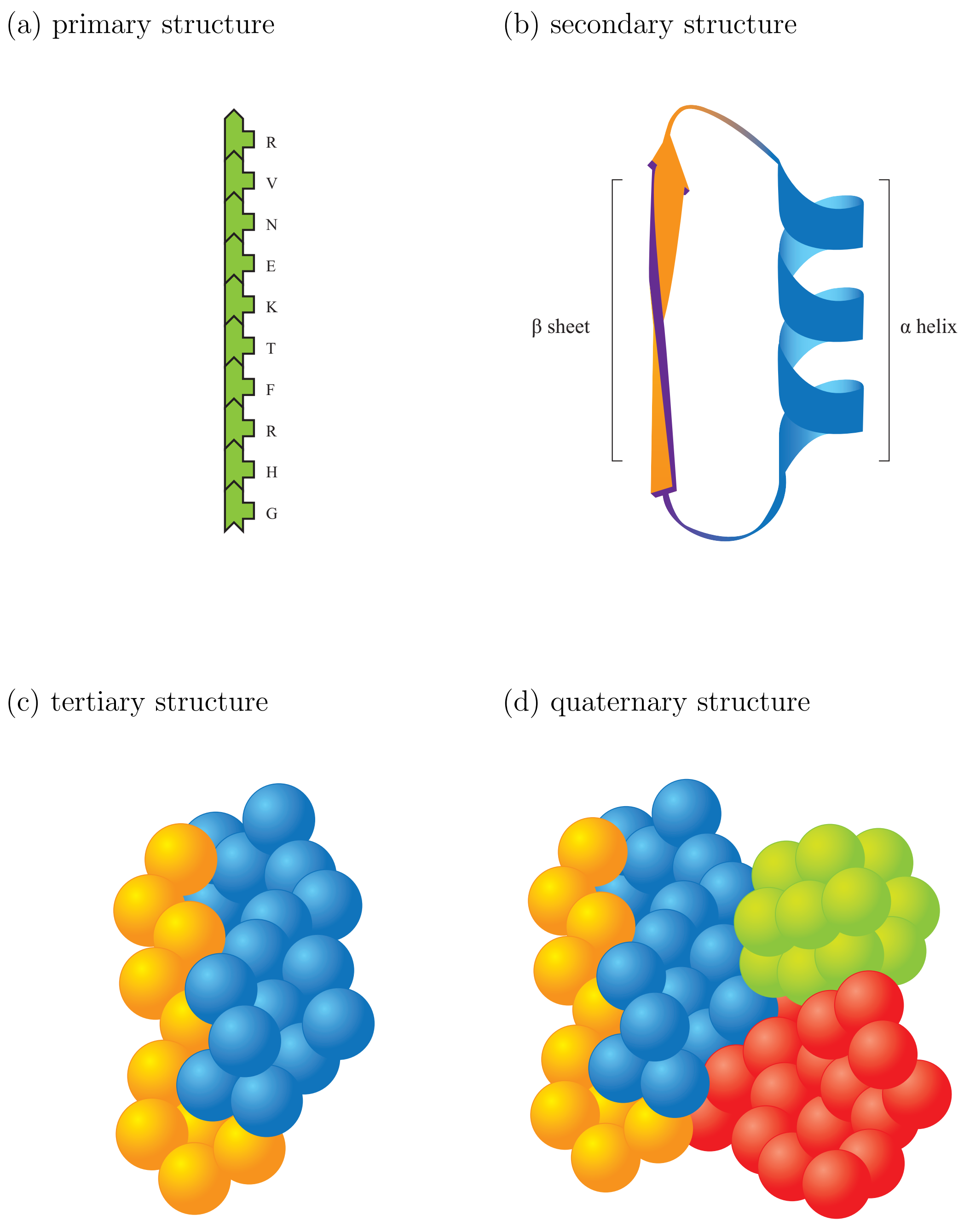


Figure 4.3: The four levels of organisation in the structure of a protein: (a) the sequence of amino acids is known as the *primary structure*; (b) common folding patterns β-sheet and α-helix constitute the *secondary structure*; (c) the full three-dimensional structure is known as the *tertiary structure*; (d) the assembly of complex structures by joining multiple polypeptide chains together is known as the *quaternary structure*.

4.2 Protein Databases

Protein information is widely available in biological databases. Some databases are dedicated to a particular aspect, such as structural information or protein interaction data, while others provide broad information with links (cross-references) to specialised databases. Most databases provide a wide range of tools and online search interfaces to assist the analysis of their data. Table 4.1 presents a summary of biological databases publicly available online.

In general, database entries comprise experimental results combined with annotations. Annotations provide valuable information about a protein, including simple information derived from proteins' primary structures (e.g. molecular weight and amino acid sequence patterns), nature of the experiment and organism where the protein is found. They can be determined either by computational methods or manually, where the latter is more preferable for its reliability. For example, UniProt (Universal Protein Resource) Knowledgebase of protein sequences contains two sections: Swiss-Prot and TrEMBL. Swiss-Prot is the richest annotated protein sequence section, containing manually annotated/curated entries, with extensive database cross-references and literature citations. The TrEMBL section contains computationally analysed records that await full manual annotation. As expected, there are much more entries in the TrEMBL section than in the Swiss-Prot section, since manual annotation requires more time and resources than automated computational annotation.

Another commonly used source of protein annotation information are motif databases. Motifs are preserved segments of amino acid sequences, which usually represent a protein family, domain or a functional site. PROSITE [65], PRINTS [6], Pfam [46] and InterPro [86] are examples of databases that contain a collection of protein motifs. Biological literature databases, such as MEDLINE (Medical Analysis and Retrieval System Online), are a valuable resource and textual analysis of these databases is an area of growing interest [102]. More specialised databases contain information about protein interaction data, protein secondary structures, gene expression data, amongst others.

This section presents a brief overview of a subset of protein databases, namely the UniProt Knowledgebase (UniProtKB) sequence database, the InterPro protein family database and the IntAct protein interaction database. These databases are the source of data for experiments presented in chapter 8. A more complete overview of different protein databases can be found in [60].

Table 4.1: Summary of biological databases publicly available online, mainly containing protein information.

UniProt (http://www.ebi.ac.uk/uniprot/) automatically (TrEMBL) and manually (Swiss-Prot) annotated/curated protein sequences
IntAct (http://www.ebi.ac.uk/intact/) protein interaction data
CATH (http://www.cathdb.info/) hierarchical domain classification of protein structures
PROSITE (http://www.expasy.org/prosite/) protein domains, families and functional sites
PubMed (http://www.ncbi.nlm.nih.gov/pubmed/) biomedical literature
Pfam (http://pfam.sanger.ac.uk/) protein domains and families
InterPro (http://www.ebi.ac.uk/interpro/) protein families, domains and sequence patterns
DIP (http://dip.doe-mbi.ucla.edu/) protein interaction data
MEDLINE (http://medline.cos.com/) biomedical literature
TIGRFAMs (http://www.tigr.org/TIGRFAMs/) protein families
PRINTS (http://www.bioinf.manchester.ac.uk/dbbrowser/PRINTS/) protein families
ArrayExpress (http://www.ebi.ac.uk/arrayexpress/) gene expression data

4.2.1 UniProt Knowledgebase

The UniProt Knowledgebase (UniProtKB) [28] is maintained by the UniProt Consortium, a collaboration between the European Bioinformatics Institute (EBI), the Swiss Institute of Bioinformatics (SIB) and the Protein Information Resource (PIR). UniProtKB is a comprehensive database for protein sequence and annotation information, and one of the most widely used biological databases. Every entry (record) in UniProtKB is extensively annotated, comprising the amino acid sequence, protein name, organism(s) where the protein is found, bibliographic references (e.g., references to experimental or computational reports) and database cross-references to specialised databases (e.g., links to motifs definitions present in the protein), amongst others. The UniProtKB is divided into two sections: (a) a high-quality manually-curated Swiss-Prot section, referred to as UniProtKB/Swiss-Prot; (b) a computationally analysed (i.e., with automatic classification/annotations) TrEMBL section, referred to as UniProtKB/TrEMBL.

The UniProtKB/Swiss-Prot section contains only entries that are manually curated, which guarantees high-quality annotations and non-redundant protein sequence records. The manual curation process involves the analysis of experimental data/results from the literature and predicted data derived from computational methods by an expert or a panel of experts. The curation process is time-consuming and requires specific expert knowledge.

In order to cope with increasing number of protein sequences without compromising the quality of the annotations of UniProtKB/Swiss-Prot, protein sequences with computationally generated annotations are stored in the UniProtKB/TrEMBL section. The UniProtKB/TrEMBL section shares the same format as the UniProtKB/Swiss-Prot section, and entries in the UniProtKB/TrEMBL section are eventually promoted to the UniProtKB/Swiss-Prot section after undergoing manual curation.

4.2.2 InterPro

The InterPro [86] is a database of protein families, domains, regions and functional sites. InterPro integrates the information of multiple source databases, which employ a range of different methodologies in order to find similarities (signatures/patterns) in protein sequences. These databases use the common principle of multiple sequence alignments in order to find similar regions—segments of amino acid sequences with no or few differences—that potentially represent/characterise a protein family, domain or functional site.

For example, the PROSITE [65] database uses regular expressions to detect patterns in sequences; PRINTS [6] database uses fingerprints—a group of conserved motifs derived from sequence alignments—to characterise protein families; Pfam [46] database uses multiple sequence alignments and hidden Markov models to represent protein families. The information of the different source databases are manually merged into InterPro entries, e.g. PROSITE patterns and PRINTS fingerprints describing the same protein family or domain are grouped into a single InterPro entry, with an unique identification number and links (cross-references) to the correspondent PROSITE and PRINTS entries.

4.2.3 IntAct

The IntAct [59] is a database of protein interactions. In short, entries in the IntAct database describe experiments that demonstrate the interaction between proteins, documenting the interaction detection method used to identify the interaction. For each interactor protein, additional information can be present, e.g. its UniProtKB accession number, the region of the protein's sequence used in the interaction. In special cases, entries describing experiments that demonstrate that an interaction does not occur are allowed, which represent negative cases—i.e. absence of interaction.

4.3 Protein Functional Classification Schemes

As discussed in section 4.2, there are several different biological databases storing protein information. Since they were created independently and for different purposes (e.g., some databases are specific to a particular organism), each of the databases employs a particular annotation scheme. In order to facilitate the integration of data amongst different databases, many annotation schemes—particularly for the annotation of protein functions—employ a controlled vocabulary (ontology). More complex schemes are hierarchically structured, allowing protein annotations at different levels, depending on the depth of knowledge about the protein in question.

For example, the Enzyme Commission (EC) nomenclature [130] is a scheme to classify enzymes based on the type of chemical reactions that they catalyse. Enzymes are proteins that are used to increase the rate of (i.e., catalyse) chemical reactions that occur within a cell. The EC classification scheme organises the types of chemical reactions catalysed by enzymes in a hierarchical tree-structured manner. The GPCRDB [1] is a database of G protein-coupled receptors (GPCRs) proteins, which are transmembrane proteins involved in signalling. In GPCRDB, proteins are organised in a hierarchical structure of GPCRs families based on sequence alignments [87].

This section presents an overview of two protein functional annotation schemes, namely Gene Ontology and FunCat. These annotation schemes describe a wide range of protein functions—unlike EC and GPCRDB, which are annotation schemes specific for enzymes and GPCRs related functions, respectively—and they are used as the class hierarchy on the hierarchical classification experiments concerning protein function prediction presented in chapter 8.

4.3.1 Gene Ontology

The Gene Ontology (GO) Consortium [4] has developed ontologies to classify proteins in terms of three different domains: molecular function, biological process and cellular component. The ontologies are defined by a hierarchy of terms (categories), where each term has a unique numerical identifier and a textual description, arranged in a DAG-like structure. In DAG-structured hierarchies, terms can have more than one parent (with the exception of the root term), as opposed to just one parent in tree-structured hierarchies.

Within the GO, the ontology is divided into three different domains, each of which consists of a vocabulary for a particular type of biological knowledge. The Molecular Function (MF) domain describes activities performed at the molecular level, generally accomplished by individual proteins. Examples of molecular functions defined are the general concept ‘transporter activity’ and its specialisation ‘ion transporter activity’, where the latter is represented as a child of ‘transporter activity’. The Biological Process (BP) domain describes activities accomplished by a series of events or molecular functions. Examples of activities defined are high-level processes ‘immune response’ and ‘reproductive process’. Finally, the Cellular Component (CC) domain describes locations, at the levels of subcellular structures and macromolecules complexes. In general, proteins are located in or are a subcomponent of a particular cellular component. Examples of locations are ‘plasma membrane’ and ‘golgi transport complex’.

In the GO hierarchy, parent-child relationships are governed by the *true path* *rule*. The true path rule states that the path from a child term towards top-level terms must always be true. In other words, if a protein is annotated with a term A, it automatically inherits the annotation of all ancestor terms of A. For example, a protein annotated with 'ion transporter activity' will inherit the annotation 'transporter activity', since 'ion transporter activity' is a specialisation of 'transporter activity'. Figure 4.4 illustrates a subset of the Gene Ontology ion channel hierarchy.

4.3.2 FunCat

The Functional Catalogue (FunCat) [105] is a hierarchical tree-structured functional classification scheme. It can be used to classify a wide range of protein function in diverse organisms. Given its tree-structured controlled vocabulary, comprising 27 main functional categories (top of the hierarchy) and up to 6 levels of specialisation, the levels of FunCat categories are separated by dots, e.g. '20.03.01.01 *ion channels*' is an example of a most specific category under the category branches '20 *cellular transport, transport facilities and transport routes*' (which represents a top level category), '20.03 *transport facilities*' and '20.03.01 *channel/pore class transport*'. Table 4.2 illustrates the main (top-level) FunCat categories.

The structure definition of FunCat presents a trade-off between keeping it descriptive and compact, at the same time. Therefore, FunCat categories focus on describing functional processes (more general functions), not molecular functions on the atomic level, facilitating the navigation throughout categories. When a more granular and specific functional classification is required, cross-references to additional classification schemes—such as the Enzyme Commission and Gene Ontology—and/or free text descriptions are added to correspondent FunCat categories. The FunCat category '20.03.01.01 *ion channels*' illustrates the differences in granularity between FunCat and Gene Ontology—while it represents a most specific category in FunCat, the equivalent (mapped) GO term 'GO:0005216' has several descendant terms, representing more specialised ion-channel activities. The difference in granularity also is reflected in the size—number of nodes—of the hierarchy. While the FunCat (version 2.1 released on January 9th, 2007) includes in total 1.362 categories, the Gene Ontology (released November 22th, 2009) includes in total 28.625 (non-obsolete) terms.

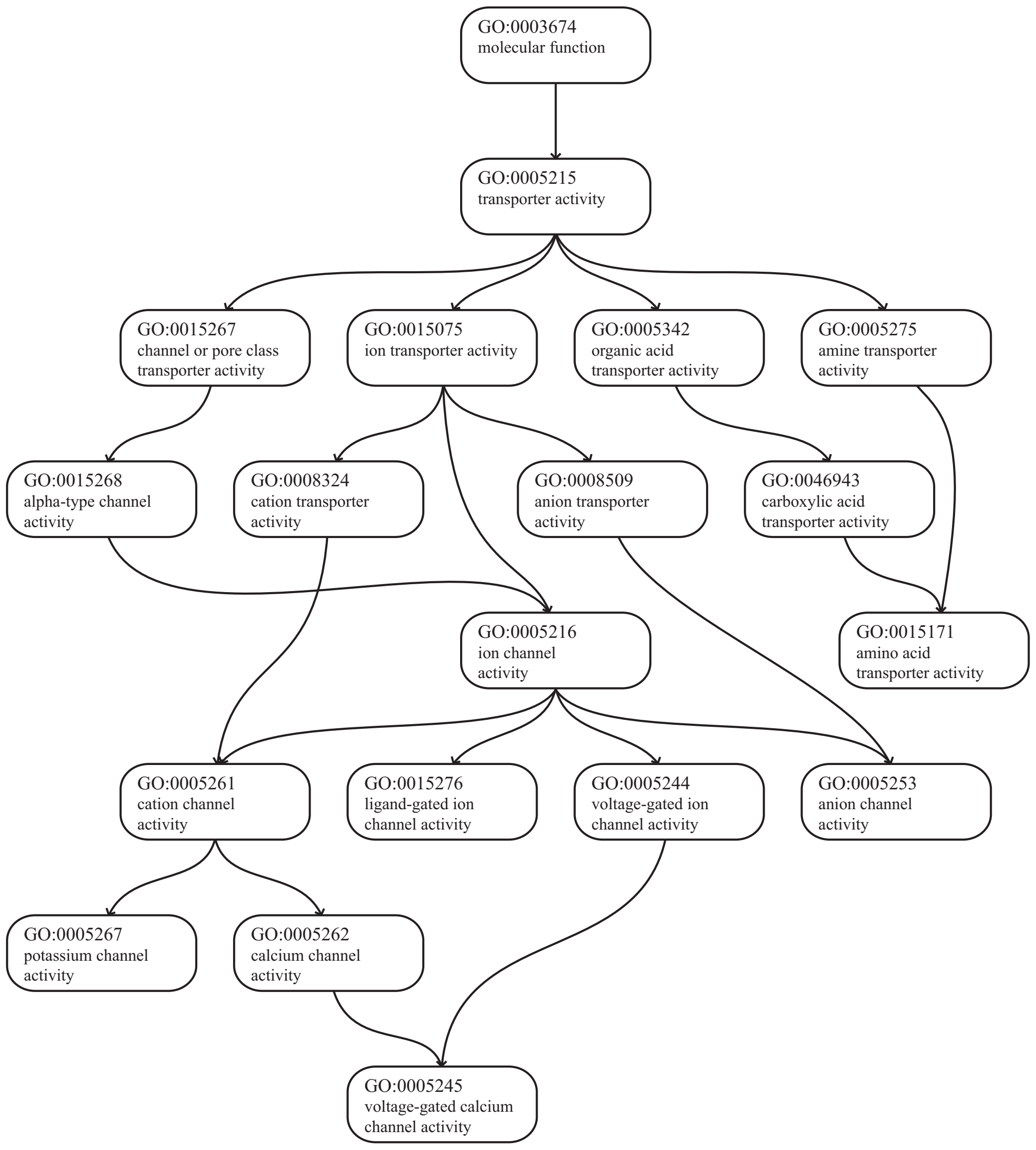

GO:0003674
molecular function
GO:0005215
transporter activity
GO:0015267
channel or pore class transporter activity
GO:0015075
ion transporter activity
GO:0005342
organic acid transporter activity
GO:0005275
amine transporter activity
GO:0015268
alpha-type channel activity
GO:0008324
cation transporter activity
GO:0008509
anion transporter activity
GO:0046943
carboxylic acid transporter activity
GO:0005216
ion channel activity
GO:0015171
amino acid transporter activity
GO:0005261
cation channel activity
GO:0015276
ligand-gated ion channel activity
GO:0005244
voltage-gated ion channel activity
GO:0005253
anion channel activity
GO:0005267
potassium channel activity
GO:0005262
calcium channel activity
GO:0005245
voltage-gated calcium channel activity

Figure 4.4: Subset of the Gene Ontology (GO) ion channel hierarchy. The ion channel activities are part of the 'molecular function' ontology within the GO.

Table 4.2: Main (top-level) FunCat categories. All categories, with the exception of 98 and 99, represent a root category of a tree-structured hierarchy.

01 Metabolism
02 Energy
04 Storage protein
10 Cell cycle and DNA processing
11 Transcription
12 Protein synthesis
14 Protein fate (folding, modification, destination)
16 Protein with binding function or cofactor requirement (structural or catalytic)
18 Regulation of metabolism and protein function
20 Cellular transport, transport facilities and transport routes
30 Cellular communication/signal transduction mechanism
32 Cell rescue, defence and virulence
34 Interaction with the environment
36 Systemic interaction with the environment
38 Transposable elements, viral and plasmid proteins
40 Cell fate
41 Development (systemic)
42 Biogenesis of cellular components
43 Cell type differentiation
45 Tissue differentiation
47 Organ differentiation
70 Subcellular localization
73 Cell type localization
75 Tissue localization
77 Organ localization
98 Classification not yet clear-cut
99 Unclassified proteins

4.4 Protein Function Prediction

As aforementioned, the exponential increase in the number of proteins being identified and sequenced using high-throughput experimental approaches has lead to a growth in the number of uncharacterised proteins (proteins for which the function is unknown). Since the rate at which sequencing methods are producing data is far outperforming the rate at which biological methods can determine protein functions, there is an increasing interest in automated protein function prediction methods.

A commonly used approach is to assign a function by sequence similarity, which considers that two or more similar proteins are homologous—i.e. they have evolved from a common ancestor—and therefore its believed that they perform the same function. In essence, the main idea of this approach is to rely on a similarity search in a protein database of known sequences and functions. The similarity between proteins is usually measured in terms of similarities of their amino acid sequence (primary structure). Given a (new) protein of unknown function, the process of predicting the protein function can be divided in two steps. In the first step, a similarity search is performed in order to find the most similar protein in the database of protein with known functions. In the second step, if the similarity between the proteins is higher than a given threshold, the function of the selected protein is predicted as the function of the new protein.

In order to determine the similarity between amino acid sequences, it is first required to find the 'best'[1] alignment between the sequences. For example, consider the pairwise alignment of two (hypothetical) amino acid sequences:

```
  .      .                        .
ELV-LVLLRVDENLLSEQTRGAFETRGAFETRVSQGPSFKERFHASCEI-----LH
ELLRLVLLRADENLL---TRGAFETRGAFETRVSEGPSFKERFHASCEIGSGRKLH
```

Since the sequences can have different lengths and the regions of similarity can occur at different segments (e.g., start, middle or end) of the sequences, gaps (represented by '-' symbols) have been inserted in both sequences. These gaps can either represent that there has been an insertion or a deletion of amino acids during the evolutionary process. The columns marked with a dot ('.' symbol) at the top represent positions in the sequences where the amino acid is different,

[1]As noted by [60], the 'best' alignment is the one that it is believed to represent what has occurred during the evolutionary process.

possibly due to a mutation in the DNA sequence encoding the protein. In essence, a scoring system which takes into account the number of matching amino acid pairs—and potentially a penalty measure for insertion of gaps—is required in order to find the 'best' alignment between different sequences.

The Smith-Waterman exhaustive algorithm [111] and the heuristic FASTA [80] and BLAST (Basic Local Alignment Search Tool) [3] algorithms are examples of algorithms used to perform similarity searches based on sequence alignments in protein databases. The output of these algorithms is usually represented as a list of similar proteins associated with an alignment score. For example, given an input sequence, each similar protein returned by a BLAST search is associated with an E-value (Expectation-value), where the smaller the E-value, the more significant the score—i.e. the more similar the protein is in relation to the input sequence. More details of sequence alignment algorithms can be found in [60].

The approach of prediction of protein functions based on sequence similarities (similarity-based approach) has several limitations, as discussed by [51, 52]. Firstly, it has been shown that proteins with very different sequences may perform the same (or similar) functions, or proteins with very similar sequences may perform different functions [57, 123, 132]. Secondly, the regions of high similarities between the proteins might occur in regions which are not determinants of the function, and therefore, are not relevant to be considered to predict the function of unknown proteins. Thirdly, this approach is not suitable to make predictions in the case where similar proteins cannot be found—i.e. the similarity between the proteins in the database with known functions and a given protein of unknown function is below a given threshold. Lastly, it cannot provide insights about the relationship between the biochemical properties of amino acids—or others protein features, such as the presence/absence of an InterPro pattern—and the protein functions, which would allow biologists to validate the functional prediction and potentially improve their biological knowledge about protein functions.

In order to overcome the aforementioned limitations, an alternative approach based on the induction of a classification model (model-based approach) has been proposed in the literature. In essence, the model-based approach can be divided into two steps. In the first step, a classification model is induced from data (e.g. a protein data set with known functions) using a classification algorithm. The induced classification model represents the relationship between the set of protein features and the set of protein functions to be predicted. In the second step, the classification model is then used to predict the function of new proteins (with unknown functions), based on their features.

In the classification task terminology, discussed in chapter 2, the set of protein features are referred to as predictor attributes, the set of protein functions are referred to as class labels, the protein data set with known functions is referred to as the training set and the protein data set with unknown functions is referred to as the test set. Since it is known that a protein can perform more than one function and functional classification schemes can be hierarchically structured (as discussed in subsection 4.3), the problem of predicting protein function is usually an instance of a hierarchical multi-label classification problem.

By using a classification model to predict the function of a new protein, there is no need to perform a similarity comparison with all the proteins of the database, as in the sequence similarity-based approach. In addition, the function prediction is not limited to finding a similar protein in terms of amino acid composition (protein's primary structure) and different protein features can be used to assist the prediction of the function—e.g., biochemical properties of amino acids, presence/absence of InterPro patterns and secondary structure elements.

One important characteristic of the model-based approach is that, when the classification model is expressed in a comprehensible manner (e.g., as a decision tree or a set/list of rules), the relationships between the protein features and functions represented by the classification model can be interpreted. There are several motivations for producing a comprehensible classification model in the context of protein function prediction [51]: (1) by being able to interpret the classification model, biologists can validate and understand the computational predictions, improving their confidence in the model. The usefulness of building a comprehensible classification model has also been emphasised by Vens et al. [127]; (2) the interpretation of the classification model can lead to new insights about the relationships between protein features and functions, improving the current biological knowledge about protein functions; (3) the analysis of the computational predictions can help to detect errors in the model or in the data, e.g., proteins that are misclassified can be analysed in order to determine if the error is in the classification model or in the protein annotation.

Examples of algorithms following a model-based approach applied to protein function prediction comprise the previously-mentioned works from Rousu et al. [104], Barutcuoglu et al. [8] and Holden and Freitas [62] in subsection 2.3.1, and Kiritchenko et al. [73] in subsection 2.3.2. The hierarchical C4.5 extensions proposed by Clare and King [25, 26] and Clare et al. [24], and the Clus algorithm proposed by Blockeel et al. [10, 13] and its variations by Vens et al. [127], briefly described in subsection 2.3.2, are examples of model-based algorithms that produce a comprehensible classification model in the form of a decision tree. More examples of algorithms can be found in [16, 49, 52, 137].

4.4.1 Protein Features as Predictor Attributes

In this subsection, protein features that commonly have been used as predictor attributes to create a classification model to predict protein functions are discussed. They are grouped into different types of attributes according to how the protein feature is obtained and for each type of attribute, a definition and examples of their usage in the literature are presented.

Sequence Attributes

Sequence attributes are those that can be derived/calculated directly from a protein's primary structure. In most cases, they represent biochemical properties of amino acids that compose the protein's sequence.

Two of the simplest attributes that can be calculated from a protein's primary structure are the sequence length and molecular weight. The sequence length consist of the number of amino acids in the protein's polypeptide chain. The molecular weight consist of the sum of the weights of the amino acids that compounds the polypeptide chain. For example, Pappa et al. [92] have generated rules predicting protein function using the sequence length and molecular weight as predictor attributes, amongst other attributes. In the results obtained with a dataset composed by 445 attributes, the sequence length attribute was used—together with other attributes—in 11 out of 21 rules, suggesting the relevance of this attribute.

The amino acid hydrophobicity value is another example of sequence attributes. It is a measure which refers to the tendency of amino acids to repel water or not to be completely dissolved in water. Amino acids that can be dissolved in water, by forming hydrogen bounds with water, are denominated hydrophilic and have a lower hydrophobicity value. Amino acids that do not dissolve in water, by forming few or no hydrogen bonds, are denominated hydrophobic and have a greater hydrophobicity value. There are different hydrophobicity scales (e.g. Kyte and Doolittle [76], Engelman et al. [42]), which are similar but not identical. The hydrophobicity value has been used mainly in two forms: as a sequence of individual hydrophobicity values (one value for each amino acid) or as a single average value for the entire protein sequence. Weinert and Lopes [131] have used a normalised hydrophobicity value for protein classification using neural networks. The input for the system was the protein sequence encoded using the Kyte and Doolittle hydrophobicity scale, with the values normalised between 0.05 and 1.00 in intervals of 0.05.

Another source of attributes is the Amino Acid Index Database (AAindex) [71]. The AAindex contains a collection of different physical, chemical and biological properties of amino acids. The current release (9.1, August 2006) contains 544 indexes, where each index is a set of 20 numerical values (one value per amino acid). Garian [55] used 401 indexes found in the AAindex database to predict the presence or absence of quaternary structure properties in the protein's primary structure.

Predicted Attributes

Predicted attributes are those that are predicted by a computational method, usually using a protein's primary structure. As they represent a prediction, their measure tend to be less reliable than the measure of sequence attributes.

The most common predicted attribute used is derived from protein's secondary structure predictions, predicted using the sequence of amino acids that compose the protein's primary structure. The importance of this attribute comes from the fact that the protein's structure is closely related to its function. Therefore, it seems natural to use this information to create models that predict protein functions. Recall that secondary structure of a protein can be composed by alpha-helices (H) and beta-strands (E) structures, typically joined by coils (C) (or loops) structures. The predicted structure is usually presented as a sequence of secondary structure symbols (H, E or C) and a confidence value, where each position has the symbol predicted for the amino acid in the corresponding position of the input sequence. Jensen et al. [69] have developed a method for protein function prediction called ProtFun. They have used the predicted secondary structure as one of the 14 features used as predictor attributes. In the results obtained with ProtFun, the predicted secondary structure was the most important feature for distinguishing enzymes from non-enzymes. In [70], they have extended the ProtFun feature set to 16 attributes and applied the method to predict terms of the Gene Ontology classification scheme. They found that the secondary structure predictions are useful for predicting '*stress response*' and '*immune response*' functions. Clare and King [26] have also generated classification rules using predicted secondary structure data.

Other examples of predicted attributes are post-translational modifications (PTMs), which are chemical modifications of a protein that occur after its translation. For example, a PTM can extend the function of a protein by binding functional groups (e.g. phosphate) to it or changing the amino acids composition (e.g. citrullination). A PTM can be characterised by sequence motifs or by complex patterns of amino acid combinations. The prediction usually is presented as a Boolean value, indicating the presence or absence of a particular PTM in a given sequence of amino acids. Jensen et al. [69] have also used predicted PTM features with their ProtFun method.

Attributes Derived from Biological Experiments

This kind of attributes comprises attributes that are derived from biological experiments, which in general are expensive and time-consuming. For these reasons, this kind of information is only available to a restricted set of genomes. They can be roughly divided into gene expression data, protein interaction data, phenotypic data and text mining of the biological literature.

Gene expression data is generated by high-throughput experiments using DNA microarrays. A DNA microarray is a slide onto which DNA sequences are attached at fixed locations (spots). There can be thousands of spots on an array. The DNA sequences on each spot are probes that react with a sample input sequences. The goal of microarray experiments is to measure gene expression levels in a set of particular times and conditions. The output of each experiment usually is a matrix, where each position represents the intensity (numerical value) of the reaction between the probe and the sample in a particular condition at a particular time. The values of gene expression data are used directly, as numerical values, or discretised into categories. For instance, Lægreid et al. [77] used temporal gene expression data to generate classification rules. Numerical gene expression data was discretised into templates ('increase', 'decrease' and 'constant') over time intervals.

Many proteins interact with one another to perform their function—e.g, by assembling multiprotein complexes or metabolic pathways. It is believed that if one can establish an interaction between a protein of known function and a protein of unknown function, the protein interaction information can be used to predict the function of the protein of unknown function. There are several databases that contains protein interaction data—e.g., MIPS [84], DIP [134] and IntAct [59]. Usually, protein interaction data is used as Boolean attributes, indicating if a given pair of proteins interact or not. Deng et al. [35] have developed a method to predict proteins functions using protein interaction data extracted from MIPS.

Another source of attributes is the data from phenotypic growth experiments, which consist of measuring the difference in the phenotype between mutated clone genes and the original form under different growth conditions (growth media). Usually, the measure is expressed by a discrete scale (from 0 to 5). Clare and King [25, 26] have used the phenotypic data from the TRIPLES [75], EUROFAN [88] and MIPS databases.

Textual analysis of the genomics literature, combining text mining and natural language processing strategies, is an area of growing interest [15, 102]. There are several molecular biology journals which contain articles reporting experimental results. Usually, the publication process requires manual reviews, making them a reliable source of information. Xie et al. [135] have used text information extracted from MEDLINE (http://www.ncbi.nlm.nih.gov/) articles to support the prediction of Gene Ontology functional terms.

Pattern Attributes

Pattern attributes are those derived from sequence alignments (which detect similarities in the sequence of proteins) and preserved amino acid sequences (motifs). They can be divided into single sequence and multiple sequence patterns, depending on the number of sequences that are used to calculate them.

The single sequence patterns are those that were derived from only one sequence, as the name suggests. In most cases, they are represented by a Boolean value indicating if a particular motif is present or absent. A motif usually corresponds to a protein family, domain or an activation site. PROSITE [65], PRINTS [6] and Pfam [46] are examples of databases that contain a collection of protein families and domains. Pappa et al. [92] have used Boolean-encoded attributes representing the presence/absence of PROSITE patterns.

The multiple sequence patterns are those that were derived using two or more sequences. In most cases, they represent phylogenetic data and similarity values (e.g. from BLAST alignments), usually used to group (cluster) proteins with similar sequences. Pellegrini et al. [95] have used phylogenetic profiles to predict the function of uncharacterised proteins. For each protein, a phylogenetic profile is created to describe the presence or absence of homologs. The profile is represented by a bit-string of length n, where n corresponds to the number of genomes. If a homolog is present in the genome i ($i = 1, ..., n$), a unit value is found in the i-th position; a zero-value indicate the absence of the homolog. Proteins are then clustered based on the similarity of their profiles and the functions of uncharacterised proteins are predicted by similar proteins within the cluster.

4.5 Summary

This chapter presented an overview of the bioinformatics area—reviewing biological concepts related to proteins—focusing on the problem of predicting protein functions. It discussed protein databases containing different protein information and protein functional classification schemes publicly available.

Furthermore, it discussed two different approaches commonly applied to protein function prediction, namely the similarity-based and model-based approaches. The latter approach, which consists of inducing a classification model in order to make predictions, is the approach followed by the algorithms for protein function prediction proposed in this thesis.

Chapter 5

Handling Continuous Attributes in Ant Colony Classification Algorithms

Although real-world classification problems are often described by both nominal (with a finite number of categorical or discrete values) and continuous (real-valued) attributes, current ant colony optimisation (ACO) classification algorithms have the limitation of being able to cope only with nominal attributes in their rule construction process. In order to overcome this limitation, a commonly used approach is to discretise continuous attributes in a preprocessing step. In essence, a discretisation method aims at converting continuous attributes into nominal (discrete) attributes by creating interval boundaries—e.g., a continuous attribute *age* might be discretised into '0–14', '15–24', '25–64' and '65+' discrete intervals.

There are numerous discretisation methods for handling continuous attributes available in the literature [41, 81]. In short, most discretisation methods employ a scoring function in order to evaluate candidate discretisation (cut) points for a continuous attribute. A discretisation point is represented by a threshold value, which is normally a value in the domain of the attribute undergoing the discretisation method. The best (or the set of best) discretisation point(s) is(are) selected based on an evaluation measure and the continuous attribute values are then replaced by the discrete intervals. These methods can be grouped according to different discretisation strategies. Methods that make use of the examples' class label information are referred to as *supervised*, while *unsupervised* methods do not use the class label information (*supervised* vs. *unsupervised* methods). *Global* methods use the entire example space (training set) to define discrete intervals while *local* methods use a subset of example space (*global* vs. *local* methods). One can also categorise discretisation methods as *static*, if they are applied in a data preprocessing phase before the classification algorithm is run, or as *dynamic*, if they are applied while a classifier is being built (*static* vs. *dynamic* methods). For a more detailed overview of different kinds of discretisation strategies and methods refer to [41, 81].

As aforementioned, the current version of Ant-Miner does not cope with continuous attributes directly—i.e. it requires continuous attributes to be discretised in a preprocessing step. In the experiments reported in [93, 94], the discretisation method C4.5-Disc [74] was applied in a data preprocessing phase prior to running Ant-Miner. In essence, the C4.5-Disc discretisation method consists of using the well-known C4.5 [99] decision tree induction algorithm to create discrete intervals for each continuous attribute individually. For each continuous attribute, the C4.5 algorithm is applied to a reduced training set which only contains the attribute to be discretised and the class attribute. After the decision tree that contains binary splits referring only to the single attribute being discretised is built, each path of the tree from a leaf node to the root node corresponds to a discrete interval. For further details, refer to [74]. The C4.5-Disc discretisation method would be categorised as *supervised*, *global* and *static* based on the criteria described above.

There are two drawbacks associated with not coping with continuous attributes directly during a run of the classification algorithm. Firstly, there is a need for a discretisation procedure in a preprocessing step. Secondly, less information is available to the classification algorithm since the discretisation procedure creates a fixed number of discrete intervals for each continuous attribute—i.e. discrete intervals have a coarser granularity, which can have a negative impact on the accuracy of the discovered knowledge.

This chapter proposes an extension to Ant-Miner, named *c*Ant-Miner (Ant-Miner coping with continuous attributes), which incorporates an entropy-based discretisation procedure in order to cope with continuous attributes during the rule construction process. *c*Ant-Miner has the ability to create discrete intervals for continuous attributes 'on-the-fly'—during the rule construction process—taking advantage of all continuous attributes information, rather than requiring that a discretisation method be used in a preprocessing step. Despite the Ant-Miner variations proposed in the literature, as reviewed in [50] and discussed in section 3.3, extending Ant-Miner to discretise continuous attributes 'on-the-fly' is a research topic that has not yet been explored by other authors to the best of our knowledge. Intuitively, coping with continuous attributes 'on-the-fly' would enhance the classification algorithm's predictive accuracy given that the use of a discretisation method in a preprocessing step can lead to a loss of predictive power, since less information is available to the classification algorithm.

The remainder of this chapter is organised as follows. Section 5.1 presents the *c*Ant-Miner algorithm, the first ACO classification algorithm able to cope with continuous attributes without requiring a discretisation method in a preprocessing step. In section 5.2, a new dynamic discretisation procedure is presented to allow a more flexible representation of continuous attributes' intervals in *c*Ant-Miner. Section 5.3 explores the problem of attribute interaction, which originates from the way that continuous attributes are handled in *c*Ant-Miner, in order to implement an improved pheromone updating procedure. Section 5.4 presents an algorithm consisting in the combination of both extensions presented in sections 5.2 and 5.3. Finally, section 5.5 presents the summary of this chapter.

5.1 Ant-Miner Coping with Continuous Attributes

Recall that Ant-Miner aims at discovering *IF-THEN* classification rules of the form *IF* $term_1$ *AND* $term_2$ *AND* ... *AND* $term_n$ *THEN class label* from a training set. Each term in the rule is a triple (*attribute*, *operator*, *value*), where *operator* represents a relational operator and *value* represents a value of the domain of *attribute*. As Ant-Miner only works with nominal (categorical or discrete) attributes, the only valid relational operator is '$=$' (equality operator).

While the proposed *c*Ant-Miner shares the same underlying procedure of Ant-Miner, as described in Algorithm 3.2 in section 3.2, *c*Ant-Miner extends Ant-Miner in several ways in order to overcome Ant-Miner's limitation of only coping with nominal attributes. Firstly, it includes vertices to represent continuous attributes in the construction graph. Secondly, in order to compute the heuristic information for continuous attributes, it incorporates a dynamic entropy-based discretisation procedure. Thirdly, when a continuous attribute vertex is selected by an ant to be added to its current partial rule, a relational operator '$<$' (less-than operator) or '$\geq$' (greater-than-or-equal-to operator) and a threshold value (cut point) are computed using a similar procedure as for the heuristic information. Therefore, antecedent of rules in *c*Ant-Miner can represent attribute-value conditions in the form $attribute_c < value$ or $attribute_c \geq value$ (where $attribute_c$ is a continuous attribute and *value* is a value in the domain of $attribute_c$). Lastly, the pheromone updating procedure has been extended to cope with continuous attribute vertices.

The technical details of the extensions incorporated in *c*Ant-Miner are presented in the following subsections.

5.1.1 Construction Graph

The original Ant-Miner's construction graph consists of an almost fully connected graph—as described in subsection 3.2.1—in which for each nominal attribute x_i and value v_{ij} (where x_i is the i-th nominal attribute and v_{ij} is the j-th value belonging to the domain of x_i), a vertex is added to the construction graph representing a term $x_i = v_{ij}$.

*c*Ant-Miner extends Ant-Miner's construction graph to cope with continuous attributes as follows—illustrated in Figure 5.1(b). For each continuous attribute y_i, a vertex is added to the graph representing the term y_i. Then, the newly created vertex y_i is connected to all previous vertices of the construction graph. It should be noted that at this point the continuous attributes' values have not been discretised. The discretisation occurs when an ant selects a vertex that represents a continuous attribute to be added to its current partial rule, as described in subsection 5.1.3.

5.1.2 Heuristic Information

Recall that the heuristic information associated with each vertex in Ant-Miner involves a measure of entropy. In the case of nominal attributes, where every vertex has the form $x_i = v_{ij}$, the entropy for the attribute-value pair in the original Ant-Miner is computed as

$$entropy(x_i = v_{ij}; S) = \sum_{c=1}^{k} [-p(c \mid S_{x_i=v_{ij}}) \cdot \log_2 p(c \mid S_{x_i=v_{ij}})] \ , \qquad (5.1)$$

where $p(c \mid S_{x_i=v_{ij}})$ is the empirical probability of observing class label c conditional on having observed $x_i = v_{ij}$ in the set of training examples S and k is the total number of class labels.[1]

[1]Equation (5.1) is equal to Equation (3.5) in subsection 3.2.3 and it is repeated here for the convenience of the reader.

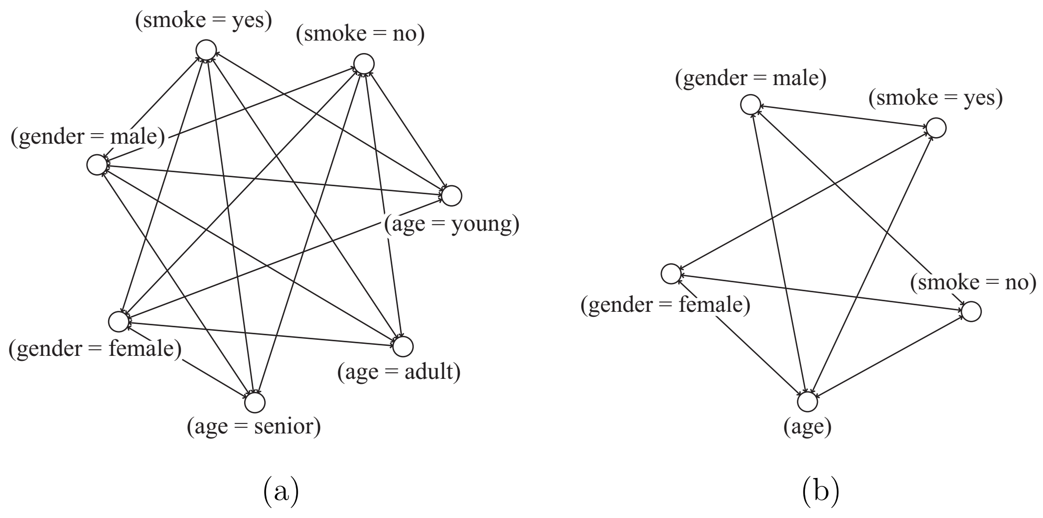


Figure 5.1: The construction graphs of Ant-Miner and *c*Ant-Miner, given a training set containing one continuous attribute ('*age*') and two nominal attributes ('*gender*' and '*smoke*'). In (a) Ant-Miner's construction graph, considering that the continuous attribute '*age*' was discretised into three different discrete intervals in a preprocessing step; (b) *c*Ant-Miner's construction graph, which includes a vertex for the continuous attribute '*age*'.

However, the Equation (5.1) cannot be straightforwardly applied to compute the entropy of vertices representing continuous attributes since these vertices do not represent an attribute-value pair condition as nominal attributes. In order to compute the entropy of a continuous attribute vertex y_i, a threshold value v is selected to dynamically partition the continuous attribute y_i into two intervals: $y_i < v$ and $y_i \geq v$. The best threshold value is the value v that minimises the entropy of the partition, given by

$$\begin{aligned} \mathit{entropy}(y_i, v; S) = & \frac{|S_{y_i<v}|}{|S|} \cdot \mathit{entropy}(y_i < v; S) \\ & + \frac{|S_{y_i \geq v}|}{|S|} \cdot \mathit{entropy}(y_i \geq v; S) \ , \end{aligned} \tag{5.2}$$

where $|S_{y_i<v}|$ is the total number of examples in the interval $y_i < v$ (partition of training examples where the attribute y_i has a value less than v), $|S_{y_i \geq v}|$ is the total number of examples in the interval $y_i \geq v$ (partition of training examples where the attribute y_i has a value greater than or equal to v), $|S|$ is the total number of training examples, $entropy(y_i < v; S)$ and $entropy(y_i \geq v; S)$ are computed according to Equation (5.1). Note that it is not necessary to examine all values in the domain of the continuous attribute y_i in order to find the best threshold value, as discussed by Fayyad and Irani [43]. Fayyad and Irani proved that the threshold value v for a continuous attribute y_i that minimises the entropy of the partition $entropy(y_i, v; S)$ must always be a value between two examples associated with different class labels, given that the set of examples is sorted according to the values of the continuous attribute y_i. Therefore, given a list of distinct ordered values $v_1, v_2, \ldots, v_W$ of the continuous attribute y_i relative to the set of examples S, the list of candidate threshold values comprises the average value of each pair of adjacent values v_w and v_{w+1}, computed as $(v_w + v_{w+1}) / 2$, wherein the examples with value v_w are associated with a different class label than the examples with value v_{w+1}. In addition, examples from the set of training examples S with missing values for the continuous attribute y_i, if present, are not taken into account in the threshold selection.

After the selection of the best threshold value v_{best}, the entropy of the continuous attribute vertex y_i relative to a set of training examples S corresponds to the minimum entropy value of the two generated intervals and it is defined as

$$entropy(y_i; S) = min[entropy(y_i < v_{best}; S), entropy(y_i \geq v_{best}; S)] \ . \tag{5.3}$$

The lowest entropy value is selected since it corresponds to the value associated with the 'purest' interval (the partition with more examples associated with the same class label) and it represents the expected predictive power (quality) of the continuous attribute vertex y_i (when vertex y_i is added to the rule). It should be noted that the entropy of every vertex y_i—i.e. every vertex representing a continuous attribute—is fixed during the rule construction process as the entropy value of every $x_i = v_{ij}$—i.e. every vertex representing an attribute-value pair of a nominal attribute. Therefore, the entropy of all y_i and $x_i = v_{ij}$ vertices are computed in an initialisation step to save computational time, rather than every time the heuristic information is used to compute the probability of selecting a vertex.

Concerning the computational complexity, the process of finding a threshold value can be divided into two steps. First, the continuous attribute values are sorted in order to facilitate the computation of the number of examples belonging to each candidate interval. The time complexity of the sorting step is $O(n \cdot \log n)$ based on the use of a quicksort algorithm [61], where n is the number of examples under consideration. Second, the evaluation of candidate threshold values has the complexity of $O(n)$, where in this case n represents the number of candidate values to be evaluated—which in the worst case scenario is equal to the number of examples under consideration, assuming that each example has a different value. In general, the number of candidate threshold values is smaller than the number of examples under consideration. Additionally, there are two factors that improve the efficiency of the evaluation of candidate threshold values: (1) as aforementioned, not all values in the domain of a continuous attribute have to be evaluated, only those that form boundaries between class labels; (2) the number of examples under consideration, and consequently the potential number of candidate threshold values, tends to decrease in relation to the number of terms in the antecedent of a rule.

The heuristic information of Ant-Miner is then extended to cope with both nominal and continuous attributes vertices (terms) and it is given by

$$\eta_T = \frac{\log_2 k - \mathit{entropy}(T;S)}{\sum\limits_{i=1}^{n}[\log_2 k - \mathit{entropy}(T_i;S)]} \quad , \tag{5.4}$$

where T is a nominal (i.e., $x_i = v_{ij}$) or continuous (i.e., y_i) attribute term, k is the total number of class labels, T_i is the nominal or continuous attribute term represented by the i-th vertex and n is the total number of vertices of the construction graph. The value of $\mathit{entropy}(T;S)$ is calculated according to Equation (5.1), if T corresponds to a nominal attribute term, and according to Equation (5.3), if T corresponds to a continuous attribute term. The same procedure is used to calculate the value of $\mathit{entropy}(T_i;S)$.

5.1.3 Rule Construction

Following a similar approach of extending the Ant-Miner's heuristic information, the probability of selecting a particular vertex representing a nominal or continuous attribute term T at each iteration of the rule construction process in *c*Ant-Miner is given by

$$P_T = \frac{\tau_T \cdot \eta_T}{\sum\limits_{i=1}^{|\mathcal{F}_T|}(\tau_{T_i} \cdot \eta_{T_i})} \quad , \qquad \forall\ T_i \in \mathcal{F}_T \ , \tag{5.5}$$

where T is a nominal or continuous attribute term (vertex), τ_T and η_T are the amount of pheromone and heuristic information associated with term T, respectively, and T_i is a nominal or continuous attribute term in the feasible neighbourhood $\mathcal{F}_T$ of term T. As in Ant-Miner, the feasible neighbourhood $\mathcal{F}_T$ comprises all terms except (1) those that contain an attribute that is already used in the current partial rule, and (2) those whose addition would make the rule cover less than a user-defined minimum number of training examples.

As every term in the antecedent of a rule must be a triple (*attribute*, *operator*, *value*), when an ant chooses a vertex that represents a continuous attribute y_i—i.e. a vertex y_i—to add to its current partial rule, a relational operator and a threshold value are selected as follows. Firstly, the best threshold value for the continuous attribute y_i is selected as described in subsection 5.1.2, subject to one restriction: only the examples covered by the current partial rule are considered in the evaluation of threshold values. Hence, the selection of a threshold value is influenced by the terms occurring in the current partial rule. This is what makes the proposed discretisation procedure a *local* and *dynamic* one, so that the choice of a threshold value is tailored to the current candidate rule. The only exception to this restriction is when the current partial rule is empty. In this case, all training examples are used in the evaluation of threshold values, as given by Equation (5.2).

Then, after selecting the best threshold value v_{best}, a relational operator is selected based on the entropy values of the two intervals generated, given by

$$operator = \begin{cases} < & if \quad entropy(y_i < v_{best}; S) < entropy(y_i \geq v_{best}; S) \\ \geq & if \quad entropy(y_i < v_{best}; S) > entropy(y_i \geq v_{best}; S) \end{cases} . \quad (5.6)$$

According to Equation (5.6), if the interval $y_i < v_{best}$ has a lower entropy, then the operator '<' (less-than operator) is selected; if the interval $y_i \geq v_{best}$ has a lower entropy, then the operator '≥' (greater-than-or-equal-to operator) is selected; ties are broken at random. As a result of Equation (5.6), the operator selection has a bias of selecting the more 'pure' interval, given that lower entropy values are favoured over higher entropy values.

Once the threshold value and the relational operator are selected, a term in the form of a triple (y_i, *operator*, v_{best}) is added to the ant's current partial rule (e.g. *age* $\geq$ 25) and the rule continues to undergo the Ant-Miner's rule construction process.

5.1.4 Pheromone Updating

In the original Ant-Miner, every nominal attribute vertex $x_i = v_{ij}$ has an associated pheromone value which undergoes the pheromone updating (reinforcement and evaporation) process. In summary, the pheromone updating process works as follows. The pheromone associated with each $x_i = v_{ij}$ occurring in the rule created by an ant is increased in proportion to the quality of the rule in question. The pheromone associated with each $x_i = v_{ij}$ that does not occur in the rule is decreased, simulating the pheromone evaporation effect observed in real ant colonies.

In the proposed *c*Ant-Miner, the original Ant-Miner's pheromone updating process has been extended to cope with y_i vertices—i.e., a vertex that represents a continuous attribute y_i. Since the pheromone value is associated with a continuous attribute y_i and not the triple (y_i, *operator*, v_{best}) that is added to the current partial rule, as described in subsection 5.1.3, the operator and the threshold value v_{best} are discarded in the updating process. In other words, there is a single entry for each continuous attribute y_i in the pheromone matrix, in contrast to multiple entries for nominal attributes, which have an entry for every $x_i = v_{ij}$ vertex—where x_i is the i-th nominal attribute and v_{ij} is the j-th value belonging to the domain of x_i.

It should be noted that pheromone is still used to indicate the quality of continuous attributes, but the actual choice of the threshold for each continuous attribute is dynamically customised to each rule being constructed by the ants. This effectively incorporates task-dependent knowledge—i.e., knowledge about the classification task of data mining—into the algorithm, which tends to increase its effectiveness.

5.2 Minimum Description Length-based Discretisation

As discussed in section 5.1, *c*Ant-Miner employs a discretisation procedure into its rule construction process that creates a single binary split in order to define discrete intervals for continuous attributes. Since an attribute can only appear once in the antecedent of a rule, discrete intervals cannot be further refined—i.e.

only intervals representing $y_i < v$ and $y_i \geq v$ conditions can be created (where v is a value in the domain of the continuous attribute y_i).

Fayyad and Irani [44] presented a minimal description length-based (MDL-based) approach where multiple discrete intervals can be extracted by applying a binary discretisation procedure recursively—selecting the best threshold value at each iteration—and using the minimal description length principle as a stopping criterion to determine whether more threshold values should be introduced. The motivation for multiple interval discretisation lies in the fact that the most effective value range may be an internal interval (e.g. $18 \leq age < 21$), which cannot be easily generated by a binary-interval-at-a-time discretisation procedure. The MDL-based approach generally leads to coarse intervals in cases where the examples are homogeneously distributed (distributed across a few different class labels, with many examples having similar values of the continuous attribute associated with the same class label) and to fine intervals in cases of more uniform distributions.

Following Fayyad and Irani, this section presents a new discretisation procedure that incorporates a MDL-based decision criterion to decide whether or not to split a given interval further in *c*Ant-Miner. The basic idea is to apply *c*Ant-Miner's entropy-based discretisation procedure recursively, relying on the MDL criterion to accept or reject a threshold value. In this way, instead of generating only intervals in the form $y_i < v$ and $y_i \geq v$, internal intervals in the form $v_{lower} \leq y_i < v_{upper}$ can be created (where v, v_{lower} and v_{upper} are values in the domain of the continuous attribute y_i).

The MDL-based discretisation procedure is divided into two steps. In the first step, the best threshold value v for a continuous attribute vertex y_i is selected as in the original *c*Ant-Miner—according to Equation (5.2). In the second step, the MDL decision criterion for accepting or rejecting the threshold value v is computed as

$$Gain(y_i, v; S) > \frac{log_2(|S| - 1)}{|S|} + \frac{\Delta(y_i, v; S)}{|S|}, \tag{5.7}$$

$$\begin{aligned} Gain(y_i, v; S) = {} & entropy(S) \\ & - \frac{|S_{y_i<v}|}{|S|} \cdot entropy(y_i < v; S) \\ & - \frac{|S_{y_i \geq v}|}{|S|} \cdot entropy(y_i \geq v; S)\,, \end{aligned} \tag{5.8}$$

$$\begin{aligned} \Delta(y_i, v; S) = log_2(3^k - 2) - [& k \cdot entropy(S) \\ & - k_{y_i<v} \cdot entropy(y_i < v; S) \\ & - k_{y_i \geq v} \cdot entropy(y_i \geq v; S)]\,, \end{aligned} \tag{5.9}$$

where k, $k_{y_i<v}$ and $k_{y_i \geq v}$ are the number of different class labels in set of training examples S, $S_{y_i<v}$ and $S_{y_i \geq v}$, respectively. If the MDL criterion defined in Equation (5.7) is satisfied, the threshold value v for the continuous attribute y_i is accepted; otherwise it is rejected. Note that the entropy values of $entropy(y_i < v; S)$ and $entropy(y_i \geq v; S)$ involved in the evaluation of the threshold value v against the MDL criterion are already computed by the first step (threshold value selection), reducing the computational time required to compute the MDL criterion. The entropy value of $entropy(S)$ is computed according to Equation (5.1), taking into account all training examples of S. Finally, if the threshold value v is accepted, the discretisation procedure is repeated recursively for each partition $S_{y_i<v}$ and $S_{y_i \geq v}$. Figure 5.2 illustrates this process, given a hypothetical continuous attribute '*age*' with values in the range of 0 to 100.

At the end of the MDL discretisation procedure, potentially multiple threshold values could have been created. In order to select the best threshold value(s), the list of threshold values is sorted and the entropy value for each discrete interval is calculated. Then, the interval with the lowest entropy value is selected (based on the fact that lower entropy values represent more 'pure' partitions where most of the examples are associated with the same class label). If an internal interval is selected (an interval between two threshold values), a term in the form $v_z \leq y_i < v_{z+1}$ is generated; otherwise, a term in the form $y_i < v_z$ or $y_i \geq v_z$ is generated (where z is the z-th threshold value); ties are broken at random. Figure 5.3 illustrates the intervals that could have been created by selecting two threshold values for a continuous attribute *age*.

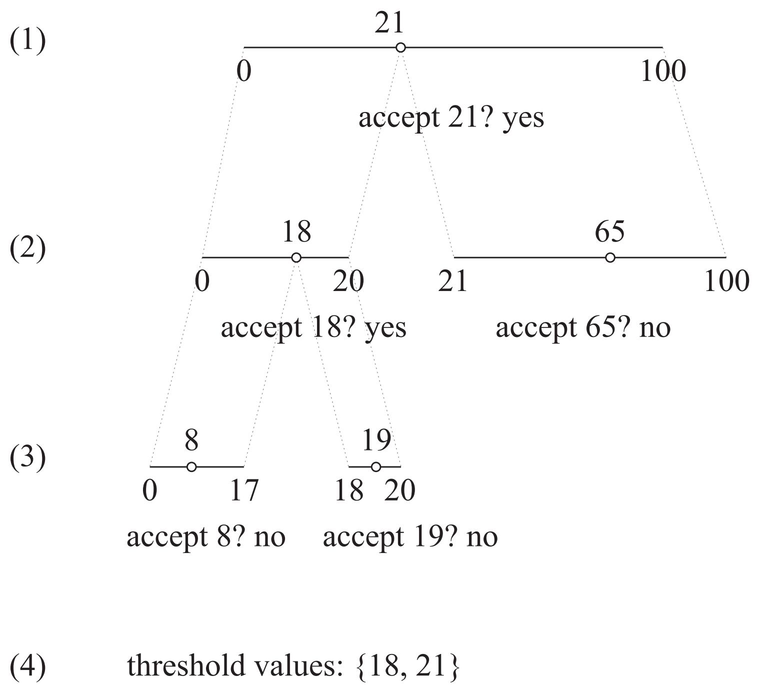


Figure 5.2: Illustration of the recursive nature of the MDL-based discretisation procedure: (1) a binary discretisation procedure is applied to select the best threshold value for a hypothetical continuous attribute '*age*' with values in the range of 0 to 100. Given that the value 21 satisfies the MDL criterion, the binary discretisation procedure is recursively applied to the intervals 0-20 and 21-100; (2) Another two threshold values are selected, one for each of the candidate intervals. The value 18 satisfies the MDL criterion and the discretisation procedure is further applied to the intervals 0-17 and 18-20. The value 65 does not satisfies the MDL criterion and therefore it is rejected; (3) Both selected values 8 and 19 do not satisfy the MDL criterion and they are rejected. Since no more intervals are available, the MDL-based discretisation procedure stops; (4) The result of the MDL-based discretisation procedure comprises the threshold values that satisfied the MDL criterion—in this example, the values 18 and 21.

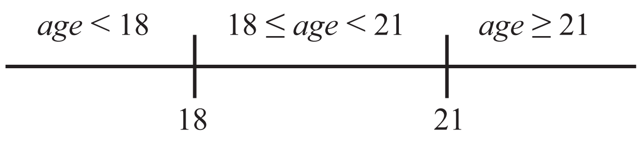


Figure 5.3: Illustration of discrete intervals that could have been created by selecting two threshold values for a continuous attribute *age*. At the end of the MDL discretisation procedure, the interval associated with the lowest entropy value is selected.

A similar approach is used to calculate the entropy value—based on the proposed MDL discretisation—involved in the heuristic information of continuous attributes. Since more than two intervals can be created, Equation (5.3) is extended in order to take into account internal intervals and it is defined as

$$\begin{aligned} entropy(y_i; S) = min[&entropy(y_i < v_1; S), \\ &entropy(v_z \leq y_i < v_{z+1}; S), \\ &\ldots, \\ &entropy(y_i \geq v_Z; S)] \ , \quad \forall\, 1 \leq z < Z, \end{aligned} \tag{5.10}$$

where z corresponds to the z-th threshold value, Z is the total number of threshold values and S is the set of training examples. This extension has the same bias towards the 'purest' interval and can also be computed in an initialisation step, as the one employed by *c*Ant-Miner.

The MDL-based discretisation procedure described in this section was incorporated into *c*Ant-Miner, creating a new *c*Ant-Miner variation dubbed *c*Ant-Miner-MDL (*c*Ant-Miner with MDL-based discretisation).

5.3 Encoding Attribute Interaction as Pheromone Levels: Associating Pheromones with Edges

In the original version of Ant-Miner, pheromone values are associated with vertices in the construction graph, where each vertex represents a term $x_i = v_{ij}$—e.g., *gender* = *male*. Hence, both ants with rule antecedents '*gender* = *male* AND *smoke* = *no*' and '*smoke* = *no* AND *gender* = *male*', for example, will deposit the same amount of pheromone on vertices '*gender* = *male*' and '*smoke* = *no*'.

The original version of *c*Ant-Miner employs a similar[2] pheromone update process, even though the order of terms (vertices) in the antecedent of a rule could affect the selection of threshold values for continuous attributes. Note that this attribute interaction dependency is not observed in the original Ant-Miner, since it only supports nominal attribute vertices which are represented by *attribute* = *value* pairs and their order is irrelevant. For instance, a partial rule with a term '*gender* = *male*' followed by a term '*smoke* = *no*' has no difference in comparison with a partial rule with a term '*smoke* = *no*' followed by a term '*gender* = *male*'. On the other hand, a partial rule with a term '*smoke* = *no*' followed by a term representing the continuous attribute *age* is different in comparison with a partial rule with a term representing a continuous attribute *age* followed by a term '*smoke* = *no*'. In the former case, only examples covered by the partial rule '*smoke* = *no*' are used to compute a threshold value for the continuous attribute *age* (e.g. $age < 18$). In the latter case, all examples of the training set are used to compute a threshold value for the continuous attribute *age* (e.g. $age \geq 65$) since the continuous attribute *age* is the first attribute to be added to the rule. Although the discretisation procedure is deterministic, there is no guarantee that the same values will be chosen in these two different cases since the examples used to compute the threshold values are different. This might affect the way that ants explore the construction graph, since the pheromone values do not accurately reflect paths explored by previous ants.

A straightforward implementation that takes into account the order of terms in a rule is to associate pheromone with the edges instead of the vertices of the construction graph. For instance, when updating pheromone values for the rule '*smoke* = *no* AND $age < 18$', instead of depositing the pheromone on the vertices '*smoke* = *no*' and '*age*', the pheromone is deposited on the edge that connects the vertex '*smoke* = *no*' to the vertex '*age*'. Consequently, when updating pheromone values for the rule '$age \geq 65$ AND *smoke* = *no*', the pheromone is deposited on the edge that connects the vertex '*age*' to the vertex '*smoke* = *no*'. Note that even though the construction graph is conceptually defined as an almost fully connected (bidirectional edges) graph, as described in subsection 5.1.1, in practice there is

[2]The overall reinforcement and evaporation procedures are the same, with the extension for coping with continuous attributes.

one edge for each direction between each pair of vertices. To be able to associate pheromone values to the first vertex of a rule, a dummy vertex '*start*' is added and unidirectionally connected to all vertices in the construction graph. This vertex represents the starting point for creating paths and its purpose is to associate pheromone values with the edge leading to the first vertex of an ant's path—i.e. the first term in the antecedent of a rule. Figure 5.4 illustrates *c*Ant-Miner's construction graph with a dummy '*start*' vertex.

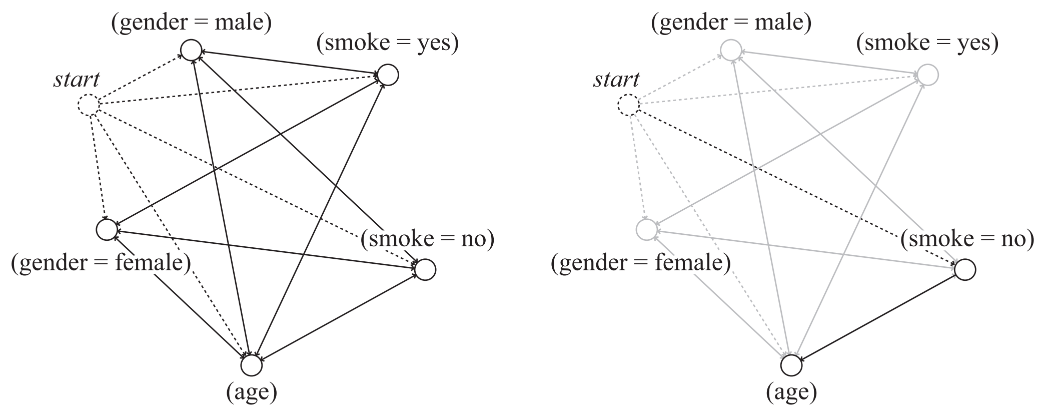
In order to calculate the probability of selecting a vertex (term) T_j to be added to the current partial rule, the pheromone value associated with the edge between the last vertex of the rule—or the dummy '*start*' vertex when the rule is empty—and the candidate vertex is used in combination with the heuristic information associated with the candidate vertex in the same manner as in Ant-Miner and in the original version of *c*Ant-Miner, given by

$$P_{T_j} = \frac{\tau_{edge_{ij}} \cdot \eta_{T_j}}{\sum\limits_{j=1}^{|\mathcal{F}_{T_i}|} \left(\tau_{edge_{ij}} \cdot \eta_{T_j}\right)} \quad , \quad \forall\ T_j \in \mathcal{F}_{T_i} \ , \tag{5.11}$$

where $\tau_{edge_{ij}}$ is the pheromone value associated with the edge that connects the last vertex (term) of the current partial rule (T_i)—or the dummy '*start*' vertex when the rule is empty—to the j-th candidate vertex (T_j), η_{T_j} is the problem-dependent heuristic information of the candidate j-th vertex (T_j) and $\mathcal{F}_{T_i}$ is the feasible neighbourhood of an ant located at the i-th vertex (T_i). The feasible neighbourhood $\mathcal{F}_{T_i}$ comprises all T_j terms except (1) those that contain an attribute that is already used in the current partial rule, and (2) those whose addition would make the rule cover less than a user-defined minimum number of training examples.

The main advantage of this approach can be seen when dealing with continuous attributes. Since the dynamic discretisation procedure of continuous attributes is deterministic and based on the currently covered examples, as discussed previously, using pheromone values that take into account the order in which ants select vertices to compose candidate rules indirectly preserves the information about the effectiveness of threshold values of continuous attributes—i.e. threshold values are preserved without requiring the storage of the actual values.

The idea of associating pheromone values with the edges of the construction graph has been previously explored in AntMiner+ by Martens et al. [83], although it was used with a very different purpose. In AntMiner+, pheromone values



(a) (b)

Figure 5.4: The construction graph of *c*Ant-Miner when pheromone values are associated with edges. In (a) *c*Ant-Miner's construction graph with a dummy '*start*' vertex; (b) a directed path ('*start*' $\rightarrow$ '*smoke* $=$ *no*' $\rightarrow$ '*age*') is highlighted.

associated with edges were used to define the construction graph as a direct acyclic graph (DAG) in order to reduce the complexity of the graph, in comparison to the construction graph employed by Ant-Miner.

As discussed in subsection 3.2.5, after an ant completes the rule construction process, the created rule undergoes a pruning procedure which aims at removing irrelevant terms that might have been included in the rule. The original pruning procedure employed by Ant-Miner and *c*Ant-Miner consists of removing one term at a time while this procedure improves the quality of the rule. Therefore, at each iteration of the pruning procedure, n (where n is the number of terms in the current rule) candidate rules with $n-1$ terms are evaluated and the rule with highest quality is selected for further pruning. This procedure is repeated until no term can be removed which improves the current rule quality or the current rule has only one term left. Clearly, the original pruning procedure does not take into account the order in which terms appear in a rule. For instance, a rule '*gender* $=$ *male* AND *age* ≥ 25 AND *smoke* $=$ *yes*' can be pruned to '*age* ≥ 25 AND *smoke* $=$ *yes*', which will update the pheromone values considering '*age*' as the first vertex of the rule followed by the vertex '*smoke* $=$ *yes*'. This is not consistent with how the threshold of the attribute '*age*' was calculated and there is no guarantee that, if an ant selects the vertex '*age*' as the first term of its rule, it will have the same threshold value—'25' in this case.

Algorithm 5.1: Threshold-aware rule pruning procedure pseudocode.

```
   input  : rule to be pruned
   output : the pruned rule
 1 begin
 2     $rule_{best} \leftarrow rule$;
 3     repeat
 4         $rule_{current} \leftarrow rule_{best}.antecedent - last\_term(rule_{best}.antecedent)$;
 5         $calculate\_consequent(rule_{current})$;
 6         if $Q(rule_{current}) \geq Q(rule_{best})$ then
 7             $rule_{best} \leftarrow rule_{current}$;
 8         end
 9     until $Q(rule_{current}) < Q(rule_{best})$ OR $|rule_{best}.antecedent| = 1$ ;
10     return $rule_{best}$;
11 end
```

To avoid such inconsistencies, a new pruning procedure—dubbed threshold-aware pruning—sensible to the order of attribute terms (vertices) is proposed. Since the order of terms in a rule is consistent with continuous attributes' threshold values, a simple implementation of a threshold-aware pruning is to remove the last term that was added to the rule in order to simplify the rule. The removal process is repeated until the rule quality decreases when the last term is removed or the rule has only one term left. Note that, in this procedure, continuous attributes' threshold values do not have to be re-calculated, since terms are removed in the reverse order that they were added to the rule. Also, only one candidate rule has to be evaluated at each iteration of the pruning procedure, resulting in a more efficient (faster) pruning procedure when compared to the original one employed by Ant-Miner and *c*Ant-Miner. Algorithm 5.1 presents the high-level pseudocode of the threshold-aware rule pruning procedure.

Let *rule* be the rule undergoing the pruning, which is considered the best rule at the beginning of the pruning procedure. At each iteration of the repeat loop in Algorithm 5.1, a candidate rule $rule_{current}$ is created by removing the last term of the antecedent of the current best $rule_{best}$. The consequent of $rule_{current}$ is (re-)computed using the same procedure as Ant-Miner and the original *c*Ant-Miner, which corresponds to assigning the majority class label of the examples covered by the rule to its consequent. Then, if the quality measure $Q(rule_{current})$ is higher than or equal to the current best rule quality $Q(rule_{best})$, $rule_{current}$ substitutes $rule_{best}$, completing an iteration of the pruning procedure. This procedure is repeated until $rule_{best}$ has just one term left on its antecedent or a candidate rule $rule_{current}$ does not improve the quality over $rule_{best}$—i.e., $Q(rule_{best}) > Q(rule_{current})$. According to Algorithm 5.1, the threshold-aware rule pruning procedure has a bias towards simpler (with fewer terms in the antecedent) rules, since if both $rule_{current}$ and $rule_{best}$—where the antecedent of $rule_{current}$ corresponds to the antecedent of $rule_{best}$ after the removal of its last term—have the same quality measure value, $rule_{current}$ substitutes $rule_{best}$ as the best rule found so far. The motivation for this bias is that simpler rules are easier to interpret and they usually cover a greater number of training examples, representing more generic knowledge.

The approach to preserve the order of terms using pheromone values associated with edges described in this section, together with the extension to the construction graph and the threshold-aware pruning procedure, were incorporated into *c*Ant-Miner, creating a new *c*Ant-Miner variation dubbed *c*Ant-Miner2 (*c*Ant-Miner using pheromones on the edges).

5.4 Combining Pheromone Associated with Edges and Minimum Description Length-based Discretisation

The proposed MDL-based discretisation method—described in section 5.2—was combined with the improved pheromone updating procedure—described in section 5.3, in which pheromone values are associated with edges of the construction graph—in order to create a new *c*Ant-Miner variation, dubbed *c*Ant-Miner2-MDL (*c*Ant-Miner2 with MDL-based discretisation). The motivation of combining both MDL-based discretisation and pheromone values associated with edges is to take advantage of the more flexible representation of discrete intervals—particularly the ability of being able to produce discrete intervals in the form $v_{lower} \leq y_i < v_{upper}$—and, at the same time, be able to preserve the information about the effectiveness of threshold values.

5.5 Summary

This chapter presented a method to extend ACO classification algorithms, proposing a new ACO classification algorithm named *c*Ant-Miner (Ant-Miner coping with continuous attribute), which is able to cope with continuous attributes directly during the rule construction process—i.e. without requiring a discretisation method in a preprocessing step. *c*Ant-Miner extended Ant-Miner in several ways. Firstly, *c*Ant-Miner includes vertices in the construction graph to represent continuous attributes. Secondly, it incorporates a dynamic entropy-based discretisation procedure in order to compute the heuristic information for continuous attributes. Thirdly, it incorporates continuous attributes conditions (terms) using a relational operator '$<$' (less-than operator) or '$\geq$' (greater-than-or-equal-to operator) in the antecedent of rules. Finally, it employs an extended pheromone update procedure to cope with continuous attribute vertices.

In addition, this chapter presented two new extensions concerning the handling of continuous attributes in ACO classification algorithms. Following the ideas of *c*Ant-Miner, a new discretisation procedure based on the MDL principle was incorporated in the rule construction process, allowing the creation of discrete intervals using lower and upper bound values—i.e., $v_{lower} \leq attribute < v_{upper}$. This led to a *c*Ant-Miner variation, dubbed *c*Ant-Miner-MDL (*c*Ant-Miner with MDL-based discretisation). Moreover, it was proposed to deposit the pheromone on edges instead of vertices of the construction graph in order to deal with the problem of attribute interaction introduced by the way that the rule construction process copes with continuous attributes in *c*Ant-Miner. Subsequently, this led to another *c*Ant-Miner variation, dubbed *c*Ant-Miner2 (*c*Ant-Miner using pheromone on the edges). Finally, both extensions were combined into a new *c*Ant-Miner variation, dubbed *c*Ant-Miner2-MDL (*c*Ant-Miner2 with MDL-based discretisation).

Chapter 6

Computational Results for Ant-Miner Coping with Continuous Attributes

In this chapter, the proposed *c*Ant-Miner (Ant-Miner Coping with Continuous Attributes) variations—described in chapter 5—are assessed empirically against the original Ant-Miner and three well-known classification algorithms from the Weka's workbench [133], namely J48 (Weka's C4.5 [99] implementation), JRip (Weka's RIPPER [27] implementation) and PART [47]. All classification algorithms are applied to eighteen publicly available data sets from the UCI Machine Learning repository [5], most of them involving continuous attributes either alone or in combination with nominal attributes, belonging to a wide range of domains.

The aim of the experiments is to firstly assess how well *c*Ant-Miner variations perform against the original Ant-Miner, with respect to predictive accuracy and simplicity (size) of the discovered classification model, given that *c*Ant-Miner is able to cope with continuous attributes directly—i.e. during its rule construction procedure—rather than requiring a discretisation method in a preprocessing step as Ant-Miner. Secondly, to assess how *c*Ant-Miner variations are compared against the well-known rule induction algorithms JRip and PART, and the decision tree induction algorithm J48, with respect to predictive accuracy and simplicity of the discovered classification model. Although J48 is a decision tree induction algorithm, it is included in the set of experiments since C4.5 (J48 reimplements the well-known C4.5) is considered a standard classification algorithm in the literature and a decision tree can be easily converted into a set of rules [99]. A summary of the different classification algorithms used in the experiments is presented in

Table 6.1: Summary of the classification algorithms used in the experiments. Ant-Miner is described in chapter 3; *c*Ant-Miner variations are described in chapter 5; J48, JRip and PART are implemented in Weka workbench [133] and briefly described in chapter 2.

Classifier	Description
Ant-Miner	rule induction algorithm
*c*Ant-Miner	Ant-Miner coping with continuous attributes
*c*Ant-Miner-MDL	*c*Ant-Miner with MDL-based discretisation
*c*Ant-Miner2	*c*Ant-Miner using pheromones on the edges
*c*Ant-Miner2-MDL	*c*Ant-Miner2 with MDL-based discretisation
J48	Weka's implementation of C4.5
JRip	Weka's implementation of RIPPER
PART	rule induction algorithm based on C4.5

Table 6.1.

The remainder of this chapter is organised as follows. Section 6.1 presents the details of the data sets used in the evaluation. Section 6.2 describes the user-defined parameters of each classification algorithm used in the experiments. The empirical evaluation results are presented in section 6.3, and finally, section 6.4 summarises the computational results presented in this chapter.

6.1 Data Sets

The main characteristics of the data sets used in the experiments are summarised in Table 6.2. In Table 6.2, the first column of the table gives the data set abbreviation, the second gives the dataset name, the third and forth columns give the number of nominal and continuous attributes respectively, the fifth column gives the number of class labels, the sixth column gives the number of examples in the original data set and the seventh column gives the number of examples in the discrete data set (after the discretisation of continuous attributes and removal of duplicated examples). Note that all data sets, except 'blc' and 'vot', involve continuous attributes either alone or in combination with nominal attributes.

Since Ant-Miner does not cope with continuous attributes directly, the data

Table 6.2: Summary of the data sets used in the experiments. The first column of the table gives the data set abbreviation, the second gives the dataset name, the third and forth columns give the number of nominal and continuous attributes respectively, the fifth column gives the number of class labels, the sixth column gives the number of examples in the original dataset and the seventh column gives the number of examples in the discrete dataset (after the discretisation of continuous attributes and removal of duplicated examples).

Abbr.	Data Set	Attributes		Classes	Examples	
		Nom.	Cont.		Orig.	Disc.
anneal	annealing	29	9	6	896	813
auto	automobile	10	15	7	205	201
bcl	breast cancer (ljubljana)	9	0	2	286	–
bcw	breast cancer (wisconsin)	0	30	2	569	546
cylinder	cylinder bands	16	19	2	540	529
glass	glass	0	9	7	213	200
heart-c	heart (cleveland)	6	7	5	303	281
heart-h	heart (hungarian)	6	7	5	294	275
hep	hepatitis	13	6	2	155	154
horse	horse colic	15	7	2	365	365
ionos	ionosphere	0	34	2	350	345
credit-a	statlog credit (australian)	8	6	2	690	684
credit-g	statlog credit (german)	13	7	2	1000	999
s-heart	statlog heart	6	7	2	270	268
seg	statlog segmentation	0	19	7	2269	2244
park	parkinsons	0	22	2	195	181
vot	voting-records	16	0	2	435	–
wine	wine	0	13	3	178	174

sets containing continuous attributes were discretised in a preprocessing step as follows. Since a ten-fold cross-validation procedure is used in the experiments, each partition was separately discretised by the C4.5-Disc discretisation method[1] [74] using the remaining nine partitions to create discrete intervals, which were then used to discretise the unseen partition (hold-out test set). This separation is necessary because, if the entire data set is discretised before creating the cross-validation partitions, the discretisation method would have had access to the test data. This would have compromised the reliability of the experiments. Moreover, the duplicated examples (examples with the same values for all attributes) were also removed from the resulting discrete data set to avoid the possibility that a test set contains an example that is the same as a training example—i.e. all examples from the hold-out partition (test set) which are duplicated in the remaining nine partitions (training set) are removed, so the hold-out partition contains only unique test examples that do not occur in the training set.

6.2 Experimental Setup

Recall that *c*Ant-Miner variations share the same underlying procedure of Ant-Miner, therefore the same set of user-defined parameter values are used amongst Ant-Miner and *c*Ant-Miner variations in all data sets, which are also considered a standard in the literature [94]; no attempt was made to tune either parameter value for individual data sets. A summary of the used-defined parameters and their correspondent values is shown in Table 6.3.

The only paramenter that had its value empirically determined for the set of experiments was the *colony_size* (number of ants per iteration). It is expected that higher values of the *colony_size* parameter are associated with higher predictive accuracies, since more candidates rules are evaluated during an iteration, at the cost of a higher computational time. However, there is generally a maximum value from which increases result in higher computational times without improvements in predictive accuracy and, in some cases, could lead to a decrease in predictive accuracy due to overfitting[2] the training set. We have tested five different colony size values {1, 10, 30, 60, 100}. Figure 6.1 illustrates the influence of these values

[1]A short description of C4.5-Disc discretisation method is presented in chapter 5. For further details refer to [74].

[2]Overfitting occurs when a classification model is built too tailored to the training set, resulting in a model that does not generalise well and, consequently, it has a lower predictive accuracy in the test set (the set of unseen examples).

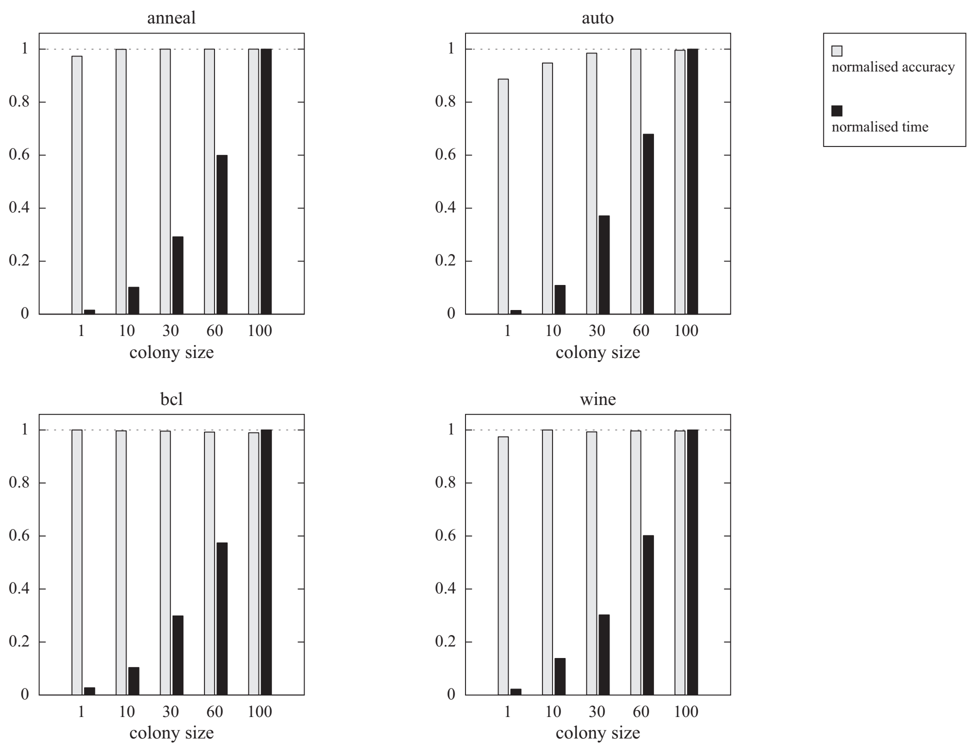
Table 6.3: Summary of the user-defined parameter values used in Ant-Miner and *c*Ant-Miner variations in all data sets.

Parameter	Description	Value
max_uncovered_examples	maximum number of uncovered examples	10
max_number_iterations	maximum number of iterations	1500
rule_convergence	number of iterations used to test the rule convergence	10
min_examples_per_rule	minimum number of covered examples per rule	10
colony_size	number of ants per iteration	60

on both predictive accuracy and execution time in Ant-Miner using the 'annel' and 'bcl' data sets—Figure 6.1(a)—and in *c*Ant-Miner using the 'auto' and 'wine' data sets—Figure 6.1(b). Overall, both 1 and 10 colony size values show slightly lower accuracy values when compared to 30, 60 and 100 values—with the exception of the data set 'bcl', where the increase of the colony size led to a decrease of the accuracy. Increasing the colony size value from 30 to 100 shows small gains in terms of accuracy, while the increase in the execution time is significant. The colony size value 60 seems to be a reasonable trade-off between accuracy and execution time and it is therefore the value used in our experiments.

For J48, JRip and PART algorithms, the Weka's workbench implementations were used with the default parameters.

The experiments were conducted using a ten-fold cross-validation procedure for each data set. A ten-fold cross-validation procedure consists of dividing the data set into ten partitions of examples, wherein each partition has a similar number of examples and class distribution. For each partition, the classification algorithm is run using the remaining nine partitions as the training set and its performance is evaluated using the unseen (hold-out) partition. For stochastic classification algorithms—i.e. Ant-Miner and *c*Ant-Miner variations—the algorithm is run fifteen times using a different random seed to initialise the search for each partition of the cross-validation. In the case of the deterministic algorithms—i.e. J48, JRip and PART—each of them is run just once for each partition of the cross-validation.



(a) (b)

Figure 6.1: Influence of colony size values {1, 10, 30, 60, 100} on both predictive accuracy and execution time. In (a) Ant-Miner using the ‘anneal’ and ‘bcl’ data sets; (b) *c*Ant-Miner using the ‘auto’ and ‘wine’ data sets.

6.3 Results

The results of the experiments concerning predictive accuracy are shown in Table 6.4 for Ant-Miner, *c*Ant-Miner, *c*Ant-Miner-MDL and *c*Ant-Miner2, and in Table 6.5 for *c*Ant-Miner2-MDL, J48, JRip and PART. Each entry in these tables shows the average value of the accuracy obtained via the ten-fold cross-validation procedure followed by the standard deviation; the last row indicates the rank-based score—the higher the score, the better the ranking—according to the non-parametric Friedman test [34, 54], which is also illustrated in Figure 6.2.

In terms of predictive accuracy, overall *c*Ant-Miner and its variations outperform Ant-Miner, with *c*Ant-Miner2-MDL being the most accurate amongst them. When compared to J48, JRip and PART, *c*Ant-Miner variations show competitive results, with both *c*Ant-Miner2 and *c*Ant-Miner2-MDL achieving high rank-scores within the most accurate algorithms as illustrated in Figure 6.2, according to the

Table 6.4: Predictive accuracy (*mean* ± *standard deviation*) obtained with the ten-fold cross-validation procedure in the eighteen data sets by Ant-Miner, *c*Ant-Miner, *c*Ant-Miner-MDL and *c*Ant-Miner2, respectively. The last row of the table indicates the rank-based score—the higher the score, the better the ranking—according to the non-parametric Friedman test [34, 54].

Data Set	Ant-Miner	*c*Ant-Miner	*c*Ant-Miner-MDL	*c*Ant-Miner2
anneal	94.88 ± 3.03	89.32 ± 1.04	88.96 ± 1.14	96.01 ± 0.52
auto	54.19 ± 2.74	67.52 ± 2.57	65.78 ± 2.61	64.08 ± 2.01
bcl	73.67 ± 2.87	73.69 ± 2.84	73.69 ± 2.84	73.47 ± 2.71
bcw	89.81 ± 1.61	93.26 ± 0.65	93.30 ± 0.70	94.55 ± 0.81
cylinder	68.55 ± 1.98	71.30 ± 0.76	70.66 ± 0.98	69.93 ± 1.39
glass	54.09 ± 5.74	67.44 ± 3.08	66.70 ± 1.71	69.46 ± 2.30
heart-c	30.59 ± 4.30	55.99 ± 1.42	56.11 ± 1.58	56.68 ± 1.13
heart-h	38.55 ± 3.56	62.91 ± 1.60	63.54 ± 1.84	63.76 ± 1.24
hep	74.71 ± 3.17	76.84 ± 3.13	77.96 ± 3.26	79.41 ± 2.57
horse	80.83 ± 2.04	80.45 ± 2.58	80.75 ± 2.55	78.44 ± 2.61
ionos	90.15 ± 1.73	87.08 ± 1.49	83.99 ± 1.09	87.35 ± 1.29
credit-a	84.49 ± 1.28	85.30 ± 0.93	86.03 ± 0.74	84.85 ± 1.05
credit-g	69.11 ± 1.70	70.66 ± 1.00	70.72 ± 1.05	71.41 ± 0.87
s-heart	73.12 ± 3.44	77.88 ± 2.23	78.82 ± 2.11	78.62 ± 2.08
seg	77.67 ± 0.79	93.72 ± 0.38	90.29 ± 0.39	93.53 ± 0.36
park	63.14 ± 9.98	87.40 ± 1.83	88.16 ± 1.53	86.92 ± 1.44
vot	93.69 ± 1.35	93.69 ± 1.35	93.69 ± 1.35	93.52 ± 1.41
wine	82.47 ± 3.63	91.38 ± 1.72	91.58 ± 1.55	92.17 ± 1.75
score	2.33	4.03	4.22	4.67

Table 6.5: Predictive accuracy (*mean* ± *standard deviation*) obtained with the ten-fold cross-validation procedure in the eighteen data sets by *c*Ant-Miner2-MDL, J48, JRip and PART, respectively. The last row of the table indicates the rank-based score—the higher the score, the better the ranking—according to the non-parametric Friedman test [34, 54].

Data Set	*c*Ant-Miner2-MDL	J48	JRip	PART
anneal	96.22 ± 0.63	92.46 ± 1.14	94.87 ± 0.59	94.63 ± 0.68
auto	66.58 ± 2.12	81.36 ± 2.50	68.69 ± 2.45	76.64 ± 3.08
bcl	73.13 ± 2.71	72.86 ± 2.28	69.44 ± 2.15	68.32 ± 1.85
bcw	94.40 ± 0.62	94.91 ± 0.71	94.20 ± 0.98	95.08 ± 1.00
cylinder	70.78 ± 1.58	74.50 ± 2.09	64.29 ± 2.29	74.51 ± 1.72
glass	69.21 ± 1.77	68.50 ± 1.78	66.54 ± 2.94	65.62 ± 3.01
heart-c	56.39 ± 1.19	51.20 ± 1.65	54.48 ± 1.59	53.52 ± 2.37
heart-h	63.98 ± 1.47	66.73 ± 2.99	63.72 ± 0.80	64.64 ± 3.16
hep	78.34 ± 2.63	80.67 ± 2.52	78.13 ± 2.66	83.25 ± 3.47
horse	79.07 ± 2.42	84.75 ± 1.49	83.54 ± 1.87	82.39 ± 2.10
ionos	86.78 ± 1.35	90.24 ± 1.23	90.24 ± 1.23	90.23 ± 1.44
credit-a	85.14 ± 0.94	85.80 ± 1.01	85.51 ± 1.46	84.35 ± 1.08
credit-g	71.38 ± 0.90	69.60 ± 1.58	72.20 ± 1.46	70.40 ± 1.60
s-heart	79.53 ± 2.20	75.56 ± 3.13	78.52 ± 2.33	75.93 ± 1.93
seg	90.82 ± 1.24	96.59 ± 0.44	94.58 ± 0.51	95.61 ± 0.32
park	88.67 ± 1.34	88.16 ± 1.55	88.76 ± 2.37	86.18 ± 2.02
vot	93.54 ± 1.40	94.48 ± 1.22	93.66 ± 1.38	94.51 ± 1.15
wine	90.85 ± 2.03	93.30 ± 2.45	92.19 ± 2.22	92.75 ± 1.44
score	4.83	5.89	4.97	5.06

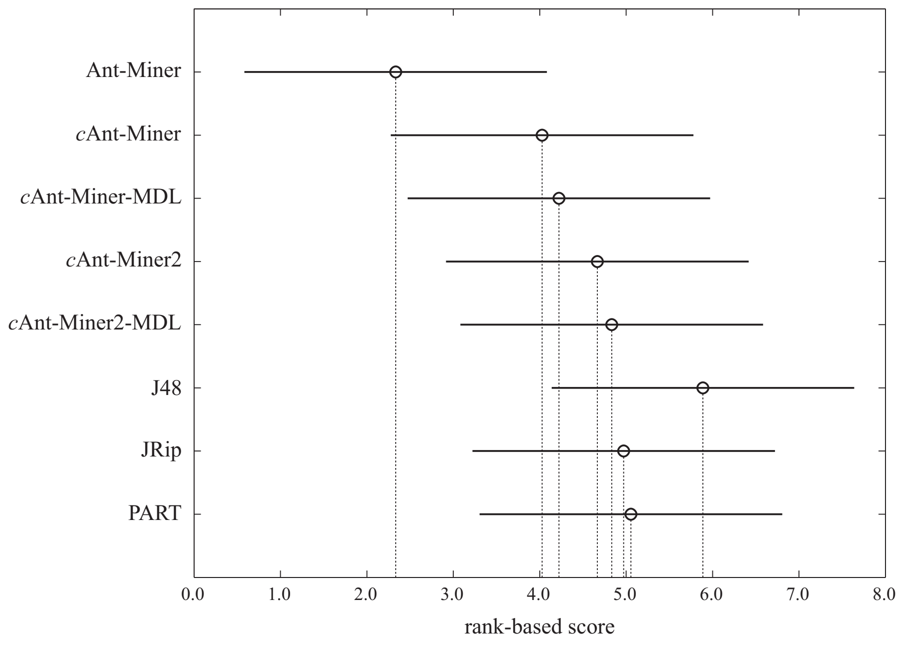


Figure 6.2: Comparison of the predictive accuracy achieved by the classification algorithms used in our experiments across all data sets, according to the non-parametric Friedman test with a Scheffé's post-hoc test at the 0.01 significant level [34, 54]. Two rank-based scores—the higher the score, the better the ranking—are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap.

non-parametric Friedman test with a Scheffé's post-hoc test at the 0.01 significant level [34, 54]. It should be noted that the algorithm with highest predictive accuracy overall, J48, is actually a decision tree induction algorithm, as mentioned earlier. Taking into account only the rule induction algorithms, *c*Ant-Miner2-MDL achieved roughly the same level of predictive accuracy as JRip and PART.

The results of the experiments concerning the size of the discovered model—measured as the number of rules—are shown in Table 6.6 for Ant-Miner, *c*Ant-Miner, *c*Ant-Miner-MDL and *c*Ant-Miner2, and in Table 6.7 for *c*Ant-Miner2-MDL, J48, JRip and PART. For the special case of the decision tree induction algorithm J48, the model size is defined by the number of leaf nodes in the tree, since each path from the root node to a leaf node can be viewed as a rule. Each entry in those tables shows the average model size obtained via the ten-fold cross-validation procedure followed by the standard deviation; the last row indicates

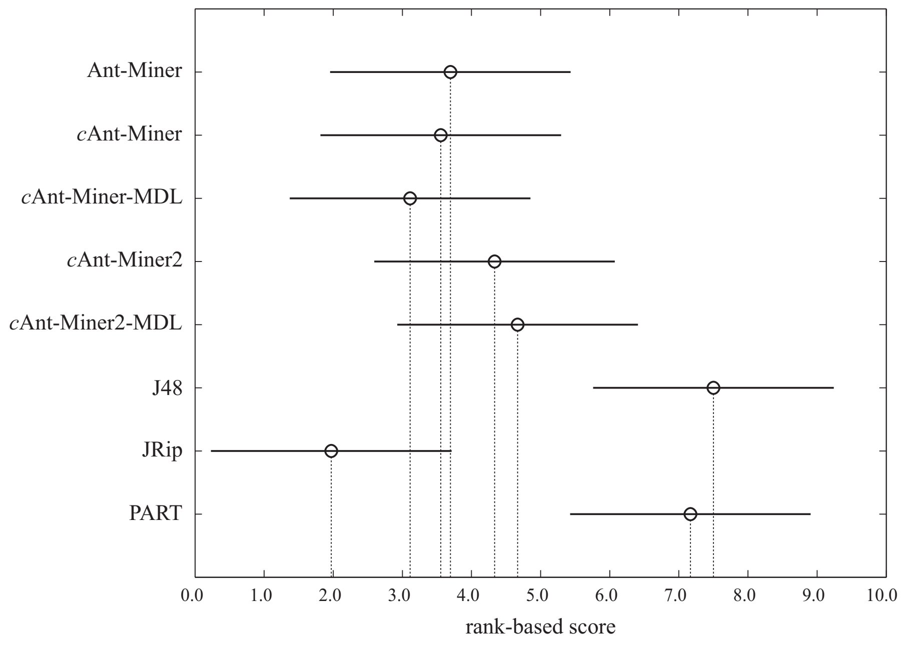


Figure 6.3: Comparison of the size of the classification model discovered by the algorithms used in our experiments across all data sets, according to the non-parametric Friedman test with a Scheffé's post-hoc test at the 0.01 significant level [34, 54]. Two rank-based scores—the lower the score, the better the ranking—are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap.

the rank-based score—in this case, the lower the score, the better the ranking, since smaller models are preferred—according to the non-parametric Friedman test [34, 54], which is also illustrated in Figure 6.3.

In terms of simplicity (size) of the discovered model, all *c*Ant-Miner variations and Ant-Miner perform equally overall, with the discovered model of *c*Ant-Miner-MDL being the simplest (smallest) amongst them. The size of the models obtained by *c*Ant-Miner, *c*Ant-Miner-MDL and JRip is significantly simpler than the models obtained by J48 and PART, according to the non-parametric Friedman test with a Scheffé's post-hoc test at the 0.01 significant level [34, 54] and illustrated in Figure 6.3. There are no statistically significant differences in the size of the model obtained by *c*Ant-Miner2 and *c*Ant-Miner2-MDL when compared to the remaining algorithms used in our experiments.

In order to evaluate the influence of each individual Ant-Miner's extensions

Table 6.6: Model size (*mean* ± *standard deviation*) obtained with the ten-fold cross-validation procedure in the eighteen data sets by Ant-Miner, *c*Ant-Miner, *c*Ant-Miner-MDL and *c*Ant-Miner2, respectively. The last row of the table indicates the rank-based score—the lower the score, the better the ranking, since smaller models are preferred—according to the non-parametric Friedman test [34, 54].

Data Set	Ant-Miner	*c*Ant-Miner	*c*Ant-Miner-MDL	*c*Ant-Miner2
anneal	9.00 ± 0.00	9.51 ± 0.11	9.97 ± 0.13	8.98 ± 0.12
auto	8.40 ± 0.18	8.19 ± 0.09	8.43 ± 0.08	8.94 ± 0.16
bcl	5.99 ± 0.12	6.01 ± 0.13	5.97 ± 0.12	5.83 ± 0.06
bcw	5.42 ± 0.10	5.03 ± 0.04	4.95 ± 0.09	5.19 ± 0.05
cylinder	6.44 ± 0.19	6.54 ± 0.10	6.55 ± 0.07	7.24 ± 0.04
glass	8.59 ± 0.34	8.22 ± 0.12	8.05 ± 0.12	8.66 ± 0.11
heart-c	7.53 ± 0.32	8.59 ± 0.12	8.57 ± 0.11	9.05 ± 0.07
heart-h	6.19 ± 0.19	7.08 ± 0.15	6.99 ± 0.14	7.47 ± 0.13
hep	4.90 ± 0.10	4.91 ± 0.11	4.87 ± 0.08	5.21 ± 0.06
horse	7.13 ± 0.23	7.27 ± 0.21	7.13 ± 0.22	6.61 ± 0.14
ionos	6.14 ± 0.28	5.50 ± 0.08	5.10 ± 0.09	5.66 ± 0.10
credit-a	6.46 ± 0.27	7.07 ± 0.19	7.22 ± 0.20	6.97 ± 0.09
credit-g	8.53 ± 0.17	8.58 ± 0.06	8.50 ± 0.05	8.70 ± 0.10
s-heart	6.16 ± 0.19	6.07 ± 0.04	6.03 ± 0.03	6.11 ± 0.09
seg	16.59 ± 0.69	12.22 ± 0.09	12.40 ± 0.13	13.58 ± 0.11
park	4.79 ± 0.13	4.95 ± 0.04	4.99 ± 0.02	5.01 ± 0.03
vot	4.80 ± 0.13	4.80 ± 0.13	4.80 ± 0.13	4.80 ± 0.13
wine	4.90 ± 0.22	4.01 ± 0.01	4.00 ± 0.00	4.32 ± 0.12
score	3.69	3.56	3.11	4.33

Table 6.7: Model size (*mean* ± *standard deviation*) obtained with the ten-fold cross-validation procedure in the eighteen data sets by *c*Ant-Miner2-MDL, J48, JRip and PART, respectively. The last row of the table indicates the rank-based score—the lower the score, the better the ranking, since smaller models are preferred—according to the non-parametric Friedman test [34, 54].

Data Set	*c*Ant-Miner2-MDL	J48	JRip	PART
anneal	11.20 ± 0.16	44.00 ± 4.18	11.50 ± 0.31	28.70 ± 1.11
auto	9.23 ± 0.07	46.60 ± 2.34	12.10 ± 0.67	19.60 ± 0.73
bcl	5.79 ± 0.09	8.20 ± 2.57	2.90 ± 0.28	18.70 ± 1.46
bcw	5.83 ± 0.09	11.50 ± 0.52	4.70 ± 0.21	7.30 ± 0.34
cylinder	7.29 ± 0.04	68.80 ± 3.52	6.00 ± 0.91	33.30 ± 0.63
glass	8.32 ± 0.19	23.80 ± 0.55	7.60 ± 0.50	16.20 ± 0.36
heart-c	9.05 ± 0.06	44.70 ± 1.61	3.30 ± 0.37	42.30 ± 1.09
heart-h	7.39 ± 0.14	27.60 ± 0.88	3.50 ± 0.45	24.10 ± 0.89
hep	5.17 ± 0.06	9.70 ± 0.56	2.70 ± 0.21	8.40 ± 0.34
horse	6.65 ± 0.08	5.20 ± 0.47	3.70 ± 0.34	9.60 ± 0.45
ionos	6.41 ± 0.15	13.40 ± 0.70	5.90 ± 0.61	7.50 ± 0.52
credit-a	6.97 ± 0.15	19.80 ± 2.22	4.10 ± 0.64	31.90 ± 2.93
credit-g	8.61 ± 0.09	83.80 ± 6.59	3.90 ± 0.41	68.70 ± 1.99
s-heart	6.14 ± 0.08	20.80 ± 1.58	4.30 ± 0.40	18.20 ± 0.81
seg	18.00 ± 0.15	41.40 ± 0.87	17.20 ± 0.83	27.90 ± 0.92
park	4.92 ± 0.05	10.70 ± 0.45	3.90 ± 0.23	7.00 ± 0.33
vot	4.80 ± 0.13	5.70 ± 0.15	2.70 ± 0.26	6.60 ± 0.37
wine	4.20 ± 0.13	5.20 ± 0.13	4.00 ± 0.15	4.60 ± 0.16
score	4.67	7.50	1.97	7.17

Table 6.8: Summary of the pairwise comparisons in terms of predictive accuracy amongst Ant-Miner and *c*Ant-Miner variations conducted in order to evaluate the influence of each individual Ant-Miner's extensions proposed in chapter 5. For each row, the '$\oplus$' ('$\ominus$') symbol indicates that the first algorithm performs better (worse) than the second algorithm, followed by the sum of positive/negative ranks (Score column) and the corresponding *p*-value, according to the Wilcoxon signed rank test. The significant differences at the 0.01 level are shown in bold.

dynamic vs. static discretisation	Score	p
*c*Ant-Miner vs. Ant-Miner	$\oplus$ **134.0/19.0**	$\mathbf{7.00 \cdot 10^{-4}}$
binary vs. MDL-based discretisation	Score	p
*c*Ant-Miner vs. *c*Ant-Miner-MDL	$\oplus$ 69.0/67.0	$9.79 \cdot 10^{-1}$
*c*Ant-Miner2 vs. *c*Ant-Miner2-MDL	$\ominus$ 84.5/86.5	$9.83 \cdot 10^{-1}$
vertex vs. edge pheromone updating	Score	p
*c*Ant-Miner vs. *c*Ant-Miner2	$\ominus$ 61.0/110.0	$2.96 \cdot 10^{-1}$
*c*Ant-Miner-MDL vs. *c*Ant-Miner2-MDL	$\ominus$ 49.0/122.0	$1.17 \cdot 10^{-1}$

proposed in chapter 5, we have also conducted pairwise comparisons to show the significance of the increase/decrease in terms of predictive accuracy achieved by the introduction of a particular extension, according to the Wilcoxon signed rank test [34, 54]. The pairwise comparisons amongst Ant-Miner and *c*Ant-Miner variations are summarised in Table 6.8 and they are described next.

1. *dynamic vs. static discretisation*: the introduction of a dynamic discretisation procedure in *c*Ant-Miner's rule construction process has led to a significant increase in predictive accuracy when compared to Ant-Miner using the static C4.5-Disc discretisation method, according to the Wilcoxon signed rank test at the 0.01 significant level, as shown in Table 6.8 (*c*Ant-Miner vs. Ant-Miner row). This result supports our argument that by handling continuous attributes directly during the rule construction process in *c*Ant-Miner enhances the predictive accuracy over Ant-Miner, since more information is available to the classification algorithm and the choice of a discretisation point (threshold value) is tailored to the current rule being constructed

rather than fixed in a preprocessing step.[3] Note that both *c*Ant-Miner and Ant-Miner perform equally on data sets containing only nominal attributes (data sets 'bcl' and 'vot' in Table 6.4), as expected.

2. *binary vs. MDL-based discretisation*: the introduction of a MDL-based discretisation procedure—which allows the creation of discrete intervals with lower and upper threshold values—in *c*Ant-Miner has led to a very small decrease in predictive accuracy, but the differences are not significant, as shown in Table 6.8 (*c*Ant-Miner vs. *c*Ant-Miner-MDL row); in the case of *c*Ant-Miner2, it has led to a very small increase, but the differences are not significant either, as shown in Table 6.8 (*c*Ant-Miner2 vs. *c*Ant-Miner2-MDL row). Therefore, we conclude that there is no evidence to support the selection of the MDL-based discretisation over the binary discretisation, since both discretisation procedures perform equally in terms of prediction accuracy. Figure 6.4 illustrates the difference in how continuous attributes conditions are represented in the antecedent of rules discovered by Ant-Miner, *c*Ant-Miner and *c*Ant-Miner-MDL. Note that *c*Ant-Miner2 shares the same discretisation procedure of *c*Ant-Miner, therefore also the same representation of continuous attributes conditions in the antecedent of rules—this is also the case between *c*Ant-Miner2-MDL and *c*Ant-Miner-MDL.

3. *vertex vs. edge pheromone updating*: the introduction of an improved pheromone updating procedure, which deposits pheromone on the edges instead of vertices of the construction graph, in *c*Ant-Miner2 has led to an increase in predictive accuracy that, although it is not significant according to the Wilcoxon signed rank test at the 0.01 significant level, it is noticeable when compared to *c*Ant-Miner, as shown in Table 6.8 (*c*Ant-Miner vs. *c*Ant-Miner2 row); this is also the case when comparing *c*Ant-Miner2-MDL and *c*Ant-Miner-MDL, where the former shows an increase in predictive accuracy that is not significant but noticeable, as shown in Table 6.8 (*c*Ant-Miner-MDL vs. *c*Ant-Miner2-MDL row). These results support the argument that preserving the order of terms in a rule by depositing pheromone on the edges of the construction graph improves the predictive accuracy, since the pheromone values on the edge better reflect paths explored by previous

[3] It should be noted that while both Ant-Miner using the static C4.5-Disc discretisation method and *c*Ant-Miner's dynamic discretisation procedure consist on an entropy-based discretisation of continuous attributes, the use of a different discretisation method/procedure can produce a different result.

ants—i.e. the pheromone values indirectly preserve the information about the effectiveness of threshold values of continuous attributes.

6.4 Summary

The results presented in this chapter show that the dynamic discretisation procedure of continuous attributes incorporated in *c*Ant-Miner (and in its variations) has led to significant improvements in terms of predictive accuracy when compared to Ant-Miner. At the same time, no significant differences are observed in the size of the discovered model.

Furthermore, the incorporation of a MDL-based discretisation procedure in *c*Ant-Miner-MDL and *c*Ant-Miner2-MDL achieved similar results in terms of predictive accuracy as the binary discretisation employed in *c*Ant-Miner and *c*Ant-Miner2, respectively. The improved pheromone updating process incorporated in *c*Ant-Miner2 and *c*Ant-Miner2-MDL, which deposits pheromone on the edges of the construction graph to preserve the order of terms in a rule and indirectly the information about the effectiveness of threshold values of continuous attributes, achieved higher predictive accuracies than the one employed in *c*Ant-Miner and *c*Ant-Miner-MDL, that although not statistically significant higher, they are relatively large increases.

Comparisons with well-known rule induction, namely JRip and PART, and decision tree induction, namely J48, classification algorithms have shown that *c*Ant-Miner and its variations are competitive both in terms of predictive accuracy and size of the discovered model.

(a) Ant-Miner

```
IF family = TN THEN 5
IF hardness = 1 THEN U
IF steel = A THEN 3
IF surface-quality = E THEN 3
IF condition = S AND steel = R THEN 2
IF width = 0 AND strength = 0 THEN 3
IF shape = COIL AND width = 0 THEN 1
IF surface-finish = P THEN 2
IF product-type = C THEN 2
IF <empty> THEN 3
```

(b) *c*Ant-Miner

```
IF family = TN THEN 5
IF hardness >= 75.0 THEN U
IF steel = S AND thick < 1.3 THEN 1
IF steel = A THEN 3
IF surface-quality = E THEN 3
IF condition = S AND steel = R THEN 2
IF strength < 425.0 THEN 3
IF steel = R THEN 3
IF shape = SHEET THEN 2
IF <empty> THEN 2
```

(c) *c*Ant-Miner-MDL

```
IF family = TN THEN 5
IF 0.5005 <= thick < 0.7995000000000001 THEN 3
IF steel = A AND hardness >= 82.5 THEN U
IF steel = S AND shape = COIL AND strength >= 200.0 THEN 1
IF surface-quality = E AND carbon < 5.0 THEN 3
IF surface-quality = G THEN 3
IF steel = R THEN 2
IF carbon >= 27.5 THEN 3
IF condition = S AND steel = A THEN 3
IF len >= 0.5 AND thick < 2.6 THEN 2
IF product-type = C AND steel = V THEN 2
IF <empty> THEN U
```

Figure 6.4: Example of a list of rules discovered for the 'anneal' data set by Ant-Miner, *c*Ant-Miner and *c*Ant-Miner-MDL, respectively. The 'anneal' data set contains steel annealing data, described by 38 predictor attributes and distributed in 6 class labels. This example illustrates the differences in how continuous attributes ('carbon', 'hardness', 'len', 'strength', 'thick' and 'width' in this example) are handled in: (a) Ant-Miner: continuous attributes are discretised in a preprocessing step; (b) *c*Ant-Miner: a binary entropy-based discretisation procedure is used to dynamically create discrete intervals during the rule construction process; (c) *c*Ant-Miner-MDL: a MDL-based discretisation procedure is used (instead of a binary one) to allow the creation of intervals with lower and upper threshold values.

Chapter 7

Hierarchical and Multi-Label Ant Colony Classification Algorithms

Many classification schemes for defining protein functions—such as FunCat [105] and Gene Ontology [4], presented in section 4.3—are organised in a hierarchical structure. As discussed in section 2.3, from a data mining perspective, hierarchical classification presents a more challenging problem than conventional flat classification. Much work on hierarchical classification of protein functions has been focused on training a classifier for each function (class label) independently, using the hierarchy to determine positive and negative examples associated with each classifier [9, 70, 77]. Predicting each class label individually has several disadvantages, as discussed in subsection 2.3.1.

This chapter presents novel ant colony optimisation (ACO) classification algorithms tailored for the hierarchical and multi-label classification problem of predicting protein functions. Although the design of these algorithms took into consideration the hierarchical problem of predicting protein functions, they can be applied to hierarchical classification problems from different domains. It should be noted that all other ant colony algorithms for classification proposed in the literature have been applied to flat classification problems [50]; therefore the ACO algorithms proposed in this chapter are the first ACO algorithms for hierarchical and hierarchical multi-label classification—to the best of our knowledge.

The remainder of this chapter is organised as follows. Section 7.1 presents the hAnt-Miner (hierarchical classification Ant-Miner) algorithm. Section 7.2 discusses the limitation of the proposed hAnt-Miner algorithm when applied to hierarchical multi-label classification problems and presents an extended approach, named hmAnt-Miner. In section 7.3, the hmAnt-Miner$_{PB}$ algorithm is presented, which employs a further extended approach—inspired by the Pittsburgh approach in evolutionary algorithms—to discover a list of hierarchical multi-label classification rules. Section 7.4 presents a baseline method, named *c*Ant-Miner$_{HM}$, in which the hierarchical multi-label problem is divided into a set of binary classification problems and a flat classification algorithm is applied to discover rules for each class label independently. Finally, section 7.5 presents a summary of the algorithms proposed in this chapter.

7.1 Hierarchical Classification Ant-Miner

The target problem of the proposed hierarchical classification Ant-Miner (*h*Ant-Miner) algorithm is the discovery of hierarchical classification rules in the form *IF antecedent THEN consequent*. The antecedent of a rule is composed by a conjunction of attribute-value conditions based on predictor attribute values (e.g. *length* > 25 AND *IPR00023* = *yes*) while the consequent of a rule is composed by a set of class labels in potentially different levels of the class hierarchy (e.g. GO:0005216, GO:0005244—where GO:0005244 is a subclass of GO:0005216). *IF-THEN* classification rules have the advantage of being intuitively comprehensible to users. *h*Ant-Miner divides the rule construction process into two different ant colonies, one colony for creating rule antecedents and one colony for creating rule consequents, and the two colonies work in a cooperative fashion.

In order to discover a list of classification rules, a sequential covering approach is employed to cover all (or almost all) training examples. Algorithm 7.1 presents a high-level pseudocode of the sequential covering procedure employed in *h*Ant-Miner. The procedure starts with an empty list of rules (*while* loop) and adds a new rule to the list while the number of uncovered training examples is greater than a user-specified maximum value (*max_uncovered_examples* parameter). At each iteration, a rule is created by an ACO procedure (*repeat-until* loop). Given that a rule is represented by paths in two different construction graphs, antecedent and consequent, two separate colonies are involved in the rule construction process. Ants in the antecedent colony create paths on the antecedent construction graph while ants in the consequent colony create paths on the consequent construction graph. In order to create a rule, an ant from the antecedent colony is paired with an ant from the consequent colony (the first ant from the antecedent colony is paired with the first ant from the consequent colony, and so forth), so that the construction of a rule is synchronised between the two ant colonies. Therefore,

Algorithm 7.1: High-level pseudocode of the sequential covering procedure employed in hAnt-Miner.

```
input  : training examples
output: discovered list of rules
begin
    $training\_set \leftarrow$ all training examples;
    $rule\_list \leftarrow \emptyset$;
    while $|training\_set| > max\_uncovered\_examples$ do
        $rule_{best} \leftarrow \emptyset$;
        $i \leftarrow 1$;
        repeat
            $rule_{current} \leftarrow \emptyset$;
            for $j \leftarrow 1$ to $colony\_size$ do
                // use separate ant colonies for antecedent
                // and consequent construction
                $rule_j \leftarrow CreateAntecedent() + CreateConsequent()$;
                $Prune(rule_j)$;
                if $Q(rule_j) > Q(rule_{current})$ then
                    $rule_{current} \leftarrow rule_j$;
                end
                $j \leftarrow j + 1$;
            end
            $UpdatePheromones(rule_{current})$;
            if $Q(rule_{current}) > Q(rule_{best})$ then
                $rule_{best} \leftarrow rule_{current}$;
            end
            $i \leftarrow i + 1$;
        until $i \geq max\_number\_iterations$ OR $RuleConvergence()$ ;
        $rule\_list \leftarrow rule\_list + rule_{best}$;
        $training\_set \leftarrow training\_set - Covered(rule_{best}, training\_set)$;
    end
    return $rule\_list$;
end
```

it is a requirement that both colonies have the same number of ants (*colony_size* parameter). The antecedent and consequent paths are created by probabilistically choosing a vertex to be added to the current path based on the values of the amount of pheromone (τ) associated with edges and problem-dependent heuristic information (η) associated with vertices of the construction graph in question (i.e., antecedent or consequent construction graph). There is a restriction that the antecedent of the rule must cover at least a user-defined minimum number of examples (*min_covered_examples* parameter), to avoid overfitting. Once the rule construction process has finished, the rule constructed by the ants is pruned to remove irrelevant terms (attribute-value conditions) from the rule antecedent and irrelevant class labels from the rule consequent. Then, pheromone values are updated using the best rule (based on a quality measure Q) of the current iteration and the best-so-far rule (across all iterations) is stored/updated. The rule construction process is repeated until a user-specified number of iterations has been reached (*max_iterations* parameter), or the best-so-far rule is exactly the same in a predefined number of previous iterations (*convergence_test* parameter). The best-so-far rule found is added to the list of rules and the covered training examples (examples that satisfy the rule's antecedent conditions) are removed from the training set.

The proposed hAnt-Miner is an extension of the flat classification Ant-Miner [94] in several important ways. Firstly, it uses two separate ant colonies for constructing the antecedent and the consequent of a rule. Secondly, it uses a hierarchical classification rule evaluation measure to guide pheromone updating. Thirdly, it uses a new rule pruning procedure. Lastly, it uses a type of heuristic information adapted for hierarchical classification. The technical details of the extensions incorporated in hAnt-Miner are presented in the following subsections.

7.1.1 Construction Graphs

Antecedent Construction Graph

Given a set of nominal attributes $\mathcal{X} = \{x_1, \ldots, x_n\}$, where the domain of each nominal attribute x_i is a set of values $\mathcal{V}_i = \{v_{i1}, \ldots, v_{id_i}\}$ (where d_i equals to the number of values in the domain of attribute x_i), and a set of continuous attributes $\mathcal{Y} = \{y_1, \ldots, y_m\}$, the antecedent construction graph is defined as follows. For each nominal attribute x_i and value v_{ij} (where v_{ij} is the j-th value belonging to the domain of x_i) a vertex is added to the graph representing the term $x_i = v_{ij}$.

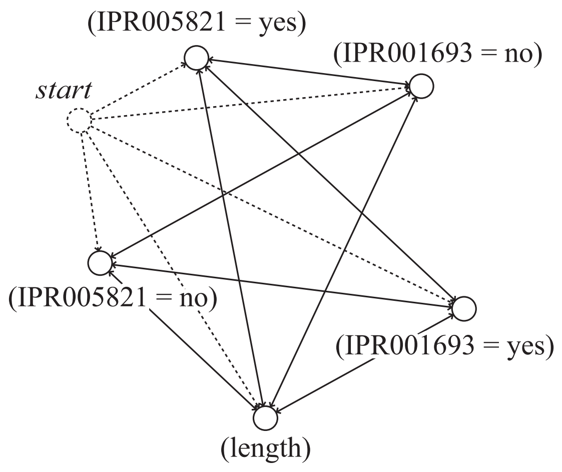


Figure 7.1: Example of an antecedent construction graph in *h*Ant-Miner (‘*IPR-005821*’ and ‘*IPR001693*’ are nominal attributes, and ‘*length*’ is a continuous attribute). The dummy vertex ‘*start*’ is unidirectionally connected to all vertices to allow the association of pheromone values on the edge of the first term of an ant’s path.

For each continuous attribute y_i a vertex is added to the graph representing the continuous attribute y_i. Since continuous attribute vertices do not represent a complete term (attribute-value condition) to be added to a rule, when an ant visits a continuous attribute vertex, a threshold value is selected to create a term using ‘$<$’ or ‘$\geq$’ relational operators (e.g. $y_i < value$). The selection of this value is deterministic and incorporates task-specific (classification-related) knowledge, increasing the effectiveness of the algorithm, as described in subsection 5.1.3 and in [90, 91].

The vertices representing an attribute term (nominal or continuous) are subsequently connected to every other vertex referring to another attribute term, with the restriction that there are no edges between nominal attribute vertices referring to the same attribute to avoid inconsistent terms such as ‘$IPR00023 = yes$’ and ‘$IPR00023 = no$’ being included in the same rule. As a result, attribute term vertices are almost fully connected. In addition, a dummy vertex ‘*start*’ is added and unidirectionally connected to all vertices in the construction graph. This vertex represents the starting point for creating paths—i.e. the 0-th vertex of a path that represents the antecedent of a rule—as described in subsection 5.3.

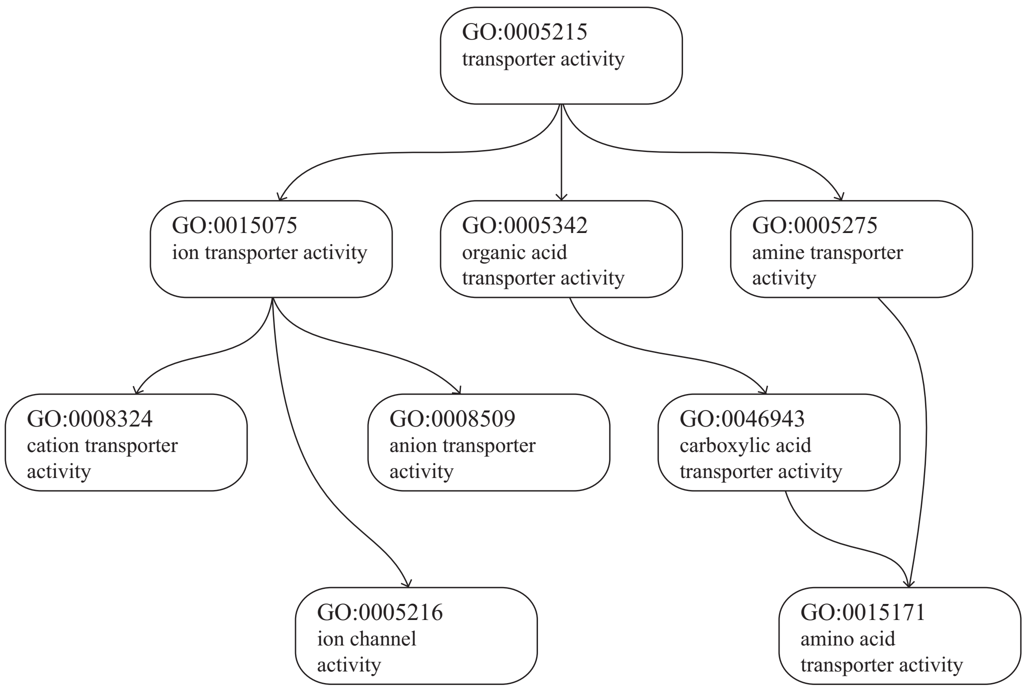

GO:0005215
transporter activity
GO:0015075
ion transporter activity
GO:0005342
organic acid transporter activity
GO:0005275
amine transporter activity
GO:0008324
cation transporter activity
GO:0008509
anion transporter activity
GO:0046943
carboxylic acid transporter activity
GO:0005216
ion channel activity
GO:0015171
amino acid transporter activity

Figure 7.2: Example of a consequent construction graph in *h*Ant-Miner, which is defined by the class hierarchy of the problem at hand. In this example, the class hierarchy is represented by a subset of the Gene Ontology's ion channel hierarchy.

Consequent Construction Graph

Since the class labels are hierarchically structured as a directed acyclic graph (DAG) or tree, this structure can be directly used to represent the consequent construction graph as follows. For each class label $l_i \in \mathcal{L}$, where $\mathcal{L}$ is the hierarchy of class labels, a vertex is added to the graph. Subsequently, for every child vertex l_j of a parent vertex l_i (where l_j, $l_i \in \mathcal{L}$), a directed connection from l_i to l_j is added to the graph. As a result, the consequent construction graph is a DAG, which is exactly the DAG of class labels of the target problem—i.e. the DAG containing all class labels and all parent-child hierarchical relationships in the target problem. Ants traverse the consequent construction graph from the root vertex towards a leaf vertex and a path represents a set of predicted class labels, consistent with the hierarchy (satisfying parent-child relationships).

7.1.2 Rule Construction

Given that a rule is represented by paths in two different construction graphs, antecedent and consequent, two separate colonies are involved in the rule construction process. Ants in the antecedent colony are responsible for creating paths on the antecedent construction graph, representing the antecedent of a rule, while ants in the consequent colony are responsible for creating paths on the consequent construction graph, representing the consequent of a rule. Thus, a rule is composed by two ants' paths.

The rule construction process is synchronized in such a way that a complete rule is created in parallel. In order to create a rule, an ant from the antecedent colony is paired with an ant from the consequent colony. Therefore, it is a requirement that both colonies have the same number of ants. Details of the antecedent construction and the consequent construction procedures are presented next.

Antecedent Construction

The antecedent of a rule is constructed by iteratively selecting vertices, which represent rule terms, to form a path in the antecedent construction graph. An ant starts with an empty antecedent (path) and adds one term (vertex) a time to its current partial antecedent at each step of the antecedent construction procedure. The probability of adding a candidate term T_j is a combination of a problem-dependent heuristic information (η) associated with the candidate term T_j and the amount of pheromone (τ) associated with the edge $edge_{ij}$ that connects the current partial rule's last term T_i to the candidate term T_j, given by

$$P_{T_j} = \frac{\tau_{edge_{ij}} \cdot \eta_{T_j}}{\sum\limits_{j=1}^{|\mathcal{F}_{T_i}|} \left(\tau_{edge_{ij}} \cdot \eta_{T_j}\right)}, \quad \forall\ T_j \in \mathcal{F}_{T_i} \ , \tag{7.1}$$

where:

$\tau_{edge_{ij}}$	is the pheromone value associated with the edge that connects the last term of the current partial antecedent (T_i) to the candidate term (T_j) and indicates the desirability of adding the term T_j after the term T_i to the antecedent of the rule. If the current partial antecedent is empty, the pheromone associated with the edge that connects the dummy '*start*' vertex to the candidate term is used. The pheromone value of the $edge_{ij}$ increases as a direct result of the quality of the

rules constructed by ants wherein T_i is followed by T_j. The higher the pheromone value of $edge_{ij}$, the higher is the probability of an ant located at term T_i to select term T_j to add to its current path. Hence, at later iterations, the best path in the antecedent construction graph will have higher pheromone values associated, increasing the probability of its vertices being chosen.

η_{T_j} is the problem-dependent heuristic information of term T_j. Higher heuristic information increases the probability of selecting term T_j and provides a notion of its quality for the classification problem in hand. The heuristic information of term T_j is fixed over all iterations and it is independent of the current partial rule.

$\mathcal{F}_{T_i}$ is the feasible neighbourhood of an ant located at term (vertex) T_i. It consists of all terms except: (1) those terms that contain an attribute that is already used in the current partial rule and (2) those terms that would make the current partial rule cover less than a user-defined minimum number of examples. The restriction (1) avoids that ants create invalid antecedents such as '*IPR00023* = *yes* AND *IPR00023* = *no*', while restriction (2) avoids that ants create antecedents that cover very few examples, which would typically be a case of overfitting.

The process of adding one term at a time to the current antecedent is repeated until one of the following stop conditions is met:

- all attributes have been used in the antecedent, so there are no more terms available;
- any term to be added would make the antecedent cover less than a user-defined minimum number of examples.

Consequent Construction

The consequent of a rule is represented by a path from the root class vertex towards a leaf class vertex on the consequent construction graph, which corresponds to the hierarchy of class labels. An ant starts with a consequent which consists of only the root class label. At each iteration of the construction process an ant adds to the consequent being constructed one child class label from the parent class label that was last added to its current partial consequent, until a leaf class label is reached. Given a set of child labels $\mathcal{C}_i = \{c_1, \ldots, c_n\}$ of a parent class label l_i, the probability of selecting a child label c_j is given by

$$P_{c_j} = \frac{\tau_{edge_{ij}} \cdot \eta_{c_j}}{\sum_{j=1}^{|\mathcal{C}_i|} (\tau_{edge_{ij}} \cdot \eta_{c_j})}, \quad \forall\ c_j \in \mathcal{C}_i, \tag{7.2}$$

where:

$\tau_{edge_{ij}}$ is the pheromone value associated with the edge that connects the parent label l_i to the child label c_j. The pheromone value of $edge_{ij}$ indicates the desirability of adding the child class label c_j after visiting the parent class label l_i and it increases as a direct result of the quality of the rules constructed by ants wherein l_i is followed by c_j.

η_{c_j} is the problem-dependent heuristic information of the child class label c_j. The heuristic information of c_j is an estimate of the quality of the child class label for the classification problem in hand.

At the end of the consequent construction procedure, the consequent is a complete path from the root class vertex towards a leaf vertex, consistent with the class hierarchy, which can also be viewed as a set of predicted class labels. The consistency of the consequent is guaranteed since an ant adds, at each step of the construction procedure, a class label to the consequent being constructed which is a sub-class of the last class label added. Therefore, class labels are added in a 'top-down' and consistent fashion, avoiding the generation of an inconsistent consequent.

7.1.3 Rule Evaluation

Since the target problem of *h*Ant-Miner is the discovery of hierarchical classification rules, a variation of the hierarchical accuracy measure proposed in [73]—described in subsection 2.5.1—is used to evaluate rules constructed by ants. The measure is a combination of both hierarchical precision and hierarchical recall measures, and it takes into account the fact that an example belongs not only to its most specific class label, but also to all its ancestor class labels according to the class hierarchy—except the root class label, since all examples trivially belong to the root class label by default.

As discussed earlier, the consequent of a rule is represented by a complete path from the root class label vertex to a leaf class label vertex. In DAG structures, multiple paths between a given pair of class labels can exist. Therefore, immediately after an ant finishes building the consequent for a rule r, the set of predicted class labels P_r of rule r is extended with the corresponding ancestor labels $({P_r}')$ as

$$ {P_r}' = P_r \cup \{\cup_{l_i \in P_r} Ancestors(l_i)\} - l_{root} \, , \tag{7.3} $$

where $Ancestors(l_i)$ corresponds to all ancestor class labels of the class label l_i and l_{root} is the root class label of the hierarchy. Then, the hierarchical measures of precision (hP) and recall (hR) are computed as

$$ hP = \frac{\sum_{i \in S_r} \frac{|A_i \cap {P_r}'|}{|{P_r}'|}}{|S_r|} \qquad hR = \frac{\sum_{i \in S_r} \frac{|A_i \cap {P_r}'|}{|A_i|}}{|S_r|} \, , \tag{7.4} $$

where S_r is the set of all examples covered by (satisfying the rule antecedent of) rule r and A_i is the set of actual (true) class labels of the i-th example. The hierarchical precision (hP) is the average number of true class labels that are predicted by rule r divided by the total number of predicted class labels across the examples covered by rule r. The hierarchical recall (hR) is the average number of true class labels that are predicted by rule r across the examples covered by rule r divided by the total number of true class labels which should have been predicted across the examples covered by rule r.

The rule quality measure Q is defined as a combination of the hP and hR measures, equivalent to the hierarchical F-measure. The F-measure, commonly employed as an evaluation measure in information retrieval systems, corresponds to the harmonic mean of precision and recall measures. Given that precision and recall measures in this case correspond to the hierarchical precision and hierarchical recall measures, the hierarchical F-measure is given by

$$ Q = hF = \frac{2 \cdot hP \cdot hR}{hP + hR} \; . \tag{7.5} $$

7.1.4 Rule Pruning

The rule pruning procedure aims at improving the rule quality by removing irrelevant terms that might have been added during the rule construction process and it is applied as soon as the rule construction is completed. Recall that a rule is

Algorithm 7.2: *h*Ant-Miner rule pruning procedure pseudocode.

```
input  : rule to be pruned
output: the pruned rule
1  begin
2      $rule_{best} \leftarrow rule$;
3      $q_{best} \leftarrow Q(rule_{best})$;
4      repeat
5          $antecedent \leftarrow rule_{best}.antecedent - last\_term(rule_{best}.antecedent)$;
6          $rule_i \leftarrow antecedent + rule_{best}.consequent$;
7          $q_i \leftarrow Q(rule_i)$;
8          $consequent_j \leftarrow rule_{best}.consequent$;
9          repeat
10             $consequent_j \leftarrow consequent_j - last\_class(consequent_j)$;
11             $rule_j \leftarrow antecedent + consequent_j$;
12             if $(Q(rule_j) > q_i)$ then
13                 $rule_i \leftarrow rule_j$;
14                 $q_i \leftarrow Q(rule_j)$;
15             end
16         until $|consequent_j| = 1$ ;
17         if $(q_i \geq q_{best})$ then
18             $rule_{best} \leftarrow rule_i$;
19             $q_{best} \leftarrow q_i$;
20         end
21     until $q_i < q_{best}$ OR $|rule_{best}.antecedent| = 1$ ;
22     return $rule_{best}$;
23 end
```

composed by antecedent and consequent parts, which in turn are represented by different ant paths that might contain irrelevant vertices.

A rule undergoes the pruning procedure as follows. At the first step, the quality of the rule is computed using the quality measure Q as given by Equation (7.5). In the second step, the rule is submitted to an iterative removal of the last term added to its antecedent while the quality of the rule is improved. At each iteration, the consequent of a candidate rule is also submitted to an iterative removal of the last added class label in an attempt to improve the generalization behaviour of the candidate rule. Note that, for the purpose of both these iterative removal procedures, the terms and class labels in the antecedent and consequent, respectively, are considered as an ordered list, and therefore terms and class labels are removed in an order inverse to the order in which they were added to the rule.

Algorithm 7.2 describes the rule pruning procedure. Let $rule_r$ be the rule undergoing the pruning procedure and q_r be the quality measure of $rule_r$. At each iteration of the outer repeat loop in Algorithm 7.2, a candidate rule $rule_i$ is created by removing the last term of the antecedent of $rule_r$ and its quality measure q_i is computed. Subsequently, j $(0 < j < |rule_i.consequent|)$ candidate rules are sequentially created by removing the last j class label(s) of the consequent of $rule_i$. This is implemented by the inner repeat loop in Algorithm 7.2. If the quality measure of a $rule_j$ is higher than q_i, $rule_i$ is substituted by $rule_j$. Finally, $rule_i$ substitutes $rule_r$ if $q_r \leq q_i$, completing an iteration of the pruning procedure. This procedure is repeated until $rule_r$ has just one term left on its antecedent or a candidate rule $rule_i$ does not improve the quality over $rule_r$ (i.e. $q_r > q_i$).

7.1.5 Pheromone Trails

Pheromone Initialisation

In order to reinforce paths followed by ants that represent good rules, pheromone values are associated with edges in the antecedent and consequent construction graphs. For each vertex i of both antecedent and consequent construction graphs, the initial amount of pheromone deposited at each edge originating at vertex i is inversely proportional to the number of edges originating at vertex i, computed as

$$\tau_{edge_{ij}} = \frac{1}{|E_i|}\,, \tag{7.6}$$

where E_i is the set of edges originating at vertex i and $edge_{ij}$ is the edge that connects vertex i to its j-th neighbour vertex. As a result of Equation (7.6), the same amount of pheromone is initially associated with every $edge_{ij}$ coming out from vertex i.

Pheromone Reinforcement

The pheromone trails followed by ants are updated based on the quality of the rules that they represent, which in turn guides future ants towards better regions of the search space. Since a rule is composed by antecedent and consequent paths, the pheromone reinforcement procedure is divided into two steps.

In the first step, the trail that represents the antecedent of a rule r is updated. Starting from the dummy 'start' vertex (0-th vertex), the pheromone value of the edge that connects the i-th vertex to the $(i + 1)$-th vertex $(0 \leq i <$

$|rule.antecedent|$) is incremented according to

$$\tau_{edge_{ij}} = \tau_{edge_{ij}} + \tau_{edge_{ij}} \cdot Q(r)\,, \tag{7.7}$$

where i and j are the i-th and j-th vertices of an edge from i to j in the trail being updated ($edge_{ij}$) and $Q(r)$ is the quality measure of rule r given by Equation (7.5). Equation (7.7) is similar to the pheromone reinforcement equation used by the original Ant-Miner algorithm [94], with the difference that pheromone values in hAnt-Miner are associated with edges of the construction graph.

In the second step, the pheromone value of every edge of the consequent of rule r that connects the i-th vertex to the $(i+1)$-th vertex ($0 < i < |rule.consequent|$) is incremented according to Equation (7.7). Note that, before computing the rule quality, the consequent is expanded to include all ancestor class labels of the class labels originally added to the rule's consequent by an ant, since there can be multiple paths between class labels, as detailed in subsection 7.1.3. However, during pheromone updating, only the actual trail that was followed to create the original consequent is updated. This avoids reinforcing trails that did not directly contribute to the consequent construction.

Pheromone Evaporation

This is implemented by normalising the pheromone values of edges of each construction graph G (antecedent and consequent). The normalization procedure indirectly decreases the pheromone of unused edges, since just before normalisation the pheromone of used edges has been increased by Equation (7.7) while the pheromone of unused edges has not been increased. This normalisation is given by

$$\tau_{edge_{ij}} = \frac{\tau_{edge_{ij}}}{\sum\limits_{\tau_{edge_{ij}} \in G} \tau_{edge_{ij}}}\,. \tag{7.8}$$

7.1.6 Heuristic Information

Antecedent Heuristic Information

The heuristic information used in the antecedent construction graph is based on information theory [31], more specifically, it involves a measure of the entropy associated with each term (vertex) of the graph. In the case of nominal attributes, where a term has the form $x_i = v_{ij}$, the entropy for the term is computed as

$$\mathit{entropy}(x_i = v_{ij}; S) = \sum_{k=1}^{|\mathcal{L}|} [-p(l_k \mid S_{x_i=v_{ij}}) \cdot \log_2 p(l_k \mid S_{x_i=v_{ij}})] \ , \qquad (7.9)$$

where $p(l_k \mid S_{x_i=v_{ij}})$ is the empirical probability of observing class label l_k conditional on having observed $x_i = v_{ij}$ (attribute x_i having the specific value v_{ij}) in the set of training examples S and $|\mathcal{L}|$ is the total number of class labels. The entropy is a measure of the (im)purity in a collection of examples, hence higher entropy values correspond to more uniformly distributed examples (examples associated with different class labels) and smaller predictive power for the term represented by the vertex in question. Equation (7.9) is a direct extension of the heuristic information for flat classification of the original Ant-Miner [94] into the problem of hierarchical classification.

In the case of continuous attributes, where a vertex represents just an attribute (and not an attribute-value condition), a threshold value v is chosen to dynamically partition the continuous attribute y_i into two intervals: $y_i < v$ and $y_i \geq v$. *h*Ant-Miner chooses the threshold value v that minimizes the entropy of the partition, given by

$$\begin{aligned} \mathit{entropy}(y_i, v; S) = & \frac{|S_{y_i<v}|}{|S|} \cdot \mathit{entropy}(y_i < v; S) \\ & + \frac{|S_{y_i\geq v}|}{|S|} \cdot \mathit{entropy}(y_i \geq v; S) \, , \end{aligned} \qquad (7.10)$$

where $|S_{y_i<v}|$ is the total number of examples in the partition $y_i < v$ (partition of training examples where the attribute y_i has a value less than v), $|S_{y_i\geq v}|$ is the total number of examples in the partition $y_i \geq v$, $|S|$ is the total number of training examples, and $\mathit{entropy}(y_i < v; S)$ and $\mathit{entropy}(y_i \geq v; S)$ are the entropy values of the terms represented by $y_i < v$ and $y_i \geq v$ as given by Equation (7.9). The list of candidate threshold values is determined using a similar approach as *c*Ant-Miner, detailed in subsection 5.1.2, and examples from the set of training examples S with missing values for the continuous attribute y_i, if present, are not taken into account in the threshold selection. In addition, Equations (7.9) and (7.10) are used in the entropy-based discretisation procedure derived from *c*Ant-Miner, as described in subsection 5.1.3, in order to find the best threshold value and a relational operator ('$<$' or '$\geq$') when an ant chooses a vertex that represents a continuous attribute y_i.

After the selection of the threshold v_{best}, the entropy of the term representing the continuous attribute y_i given a set of training examples S corresponds to the minimum entropy value of the two partitions and it is defined as

$$entropy(y_i; S) = min[entropy(y_i < v_{best}; S), entropy(y_i \geq v_{best}; S)] \ . \tag{7.11}$$

Furthermore, the heuristic information used in hAnt-Miner is straightforwardly extended to hierarchical classification as follows. Since the entropy of the i-th term (nominal or continuous) of the antecedent construction graph varies in the range $0 \leq entropy(T_i; S) \leq \log_2(|\mathcal{L}| - 1)$—where $|\mathcal{L}| - 1$ is the number of class labels in the class hierarchy without considering the root class label and S is the set of training examples—and lower entropy values are preferred over higher values, the heuristic information for the i-th term is computed as

$$\eta_{T_i} = \log_2(|\mathcal{L}| - 1) - entropy(T_i; S)\,, \quad \forall\, T_i \in G_A\,, \tag{7.12}$$

where T_i is the i-th term (vertex) of the antecedent construction graph G_A and S is the set of training examples. The value of $entropy(T_i; S)$ is calculated according to Equation (7.9), if T_i corresponds to a vertex representing a nominal attribute condition (e.g. $x_i = v_{ij}$), or according to Equation (7.11), if T_i corresponds to a vertex representing a continuous attribute (e.g. y_i). Note that Equation (7.12) will give a higher probability of being selected to terms with lower entropy values, which corresponds to terms with higher predictive power.

Consequent Heuristic Information

The heuristic information used in the consequent construction graph is based on the frequency of training examples for each class label of the hierarchy, given by

$$\eta_{l_i} = |S_{l_i}|, \quad \forall\, l_i \in G_C\,, \tag{7.13}$$

where $|S_{l_i}|$ is the number of training examples that belong to class label l_i and G_C is the consequent construction graph. Note that the heuristic information has a bias towards class labels that have a greater number of examples, which therefore will initially favour the discovery of rules with these class labels in the consequent. However, due to the use of a sequential covering procedure, rules predicting less frequent classes will be eventually discovered as well.

7.1.7 Using a Rule List to Classify New Examples

In order to classify a test (unseen) example, rules in the discovered list of rules are applied in a sequential order—i.e. the order in which they were discovered. Therefore, a test example is classified according to the consequent of the first rule that covers the example. More precisely, the example is assigned the class labels predicted by the rule's consequent.

In the situation where no rule in the discovered list of rules covers the test example, a default rule (a rule with an empty antecedent) predicting the set of class labels that occur in all uncovered training examples is used to classify the test example. For example, assume that there are three uncovered examples $e1$, $e2$ and $e3$, belonging to class labels $\{1, 1.2, 1.2.1\}$, $\{1, 1.2, 1.2.2\}$ and $\{1, 1.2, 1.2.1, 1.2.1.3\}$, respectively. The set of class labels occurring in all uncovered examples in this case comprise the set $\{1, 1.2\}$, which would be the set of predicted class labels of the default rule.

7.2 Coping with Multi-Label Data

While analysing hAnt-Miner, we have identified the following limitations. Firstly, the heuristic information used in hAnt-Miner, which involves a measure of entropy, is not very suitable for hierarchical classification—i.e. it does not take into account the hierarchical relationships between class labels. Although hAnt-Miner's entropy measure is calculated throughout all class labels of the class hierarchy (except for the root class label), each class label is evaluated individually without considering parent-child relationships between class labels.

Secondly, the rule quality measure is prone to overfitting. Since only the examples covered by the rule are considered in the rule evaluation, rules with a small coverage are favoured over more generic rules. For example, considering the class label 1.2.1 with 20 examples and two rules that have 1.2.1 as the most specific class label in their consequent: $rule_1$ covering correctly 5 examples out of a total of 5 covered and $rule_2$ covering correctly 19 examples out of a total of 20 covered. In this case, $rule_1$ would have a higher quality, since all the examples covered by the rule are correctly classified, than $rule_2$, which misclassifies one example, though $rule_2$ covers all but one examples belonging to class 1.2.1. One could argue that the rule quality measure of hAnt-Miner could be easily modified to avoid overfitting by evaluating a rule considering all the examples of its most specific class. The drawback of this approach is that it favours rules predicting class labels at the top of the hierarchy, since the numbers of examples per class are greater at top class levels. This could potentially prevent the discovery of rules predicting more specific class labels given that the examples covered by a rule are removed from the training set—indeed, this problem was observed in some preliminary experiments.

Lastly, *h*Ant-Miner does not support multi-label classification since a single path in the consequent construction graph corresponds to the consequent of a rule. In the case of protein function prediction, where it is known that a protein can perform more than one function, this is an important limitation.

This section presents a new hierarchical multi-label ant colony classification algorithm, named *hm*Ant-Miner (hierarchical multi-label classification Ant-Miner), which is aimed at overcoming the aforementioned limitations. In summary, the proposed *hm*Ant-Miner differs from *h*Ant-Miner in the following aspects:

- the consequent of a rule is calculated using a deterministic procedure based on the examples covered by the rule, allowing the creation of rules that can predict more than one class label at the same time, while satisfying hierarchical class label relationships (hierarchical multi-label rules). Therefore, *hm*Ant-Miner uses a single construction graph in order to create a rule—only the antecedent is represented in the construction graph;

- the heuristic information is based on the Euclidean distance, where each example is represented by a vector of class membership values in the Euclidean space. By using a distance measure, instead of entropy as in *h*Ant-Miner, it is possible to take into account the relationship between class labels given that examples belonging to related class labels will be more similar than examples belonging to unrelated class labels. The use of the Euclidean distance was inspired by a similar use in the Clus-HMC algorithm for hierarchical multi-label classification [127], which is based on the paradigm of decision tree induction, rather than rule induction. Note that the Euclidean distance is used as the heuristic information, as well as, in the dynamic discretisation procedure for continuous attributes;

- the rule quality is evaluated using a distance-based measure, which is a more suitable evaluation measure for hierarchical multi-label problems;

- the pruning procedure is not applied to the consequent of a rule. The consequent of a rule is (re-)calculated when its antecedent is modified during pruning, since the set of covered examples might have changed.

7.2.1 Multi-Label Rule Consequent

As presented in subsection 7.1.1, the consequent of a rule in *h*Ant-Miner is represented as a single path in the consequent construction graph, from the root class label towards a leaf class label of the class hierarchy. Although the consequent predicts multiple class labels in a hierarchical structure, it has the limitation of not being able to predict unrelated class labels—i.e. multiple paths in the class hierarchy. One could argue that the consequent could be represented by multiple paths in order to be able to predict unrelated class labels, however it is not clear how to find the optimal combination and number of paths to consider without introducing yet another user-defined parameter.

A sensible approach is to use the information available from the examples covered by the rule—i.e. examples that satisfy the rule antecedent—in order to determine the rule consequent. As a result, the consequent of a rule in *hm*Ant-Miner is calculated using a deterministic procedure. Given the set of examples S_r covered by a rule r, the consequent is a vector of length m—where m is equal to the number of class labels in the class hierarchy. The value for each i-th component of the consequent vector for rule r is given by

$$consequent_{r,i} = \frac{|S_r \ \& \ label_i|}{|S_r|}, \tag{7.14}$$

where $|S_r \ \& \ label_i|$ is the number of examples covered by rule r that belong to the i-th class label of the class hierarchy ($label_i$). In other words, the consequent of a rule is a vector where each i-th component is the proportion of covered examples that belong to the i-th class label.

According to Equation (7.14), each position of the consequent vector is a continuous value between 0.0 and 1.0, rather than a presence/absence value of a particular class label. As a result, the value in the i-th component of the consequent of a rule represents the probability of an example that satisfies its antecedent to be associated with the corresponding i-th class label of the hierarchy. Figure 7.3 illustrates the consequent of a rule in *hm*Ant-Miner. In this example, the predictor attributes in the antecedent of the rule correspond to amino acid ratios from the protein's sequence and the class labels in the consequent of the

```
IF
      aa_rat_pair_a_h >= 0.053
      AND aa_rat_pair_t_c >= 0.1055
      AND aa_rat_pair_c_w < 0.0695
      AND aa_rat_pair_a_e < 0.2960
      AND aa_rat_pair_t_h >= 0.0275
THEN
      GO0000226:0.10,GO0000943:0.50,
      GO0001302:0.10,GO0003674:1.00,
      GO0003676:0.50,GO0003723:0.50,
      GO0003824:0.50,GO0003887:0.50,
      ...
      GO0044464:1.00,GO0045053:0.10,
      GO0045185:0.10,GO0046907:0.20,
      GO0051234:0.20,GO0051235:0.10,
      GO0051649:0.20,GO0051651:0.10
```

Figure 7.3: Example of the consequent of a rule in *hm*Ant-Miner. In this example, the predictor attributes in the antecedent of the rule correspond to amino acid ratios from the protein's sequence and the class labels in the consequent of the rule are represented by Gene Ontology terms—the number following the colon of a class label in the consequent corresponds to the probability of predicting the associated class label. Only a subset of the class labels predicted by the rule are shown.

rule are represented by Gene Ontology terms—the number following the colon of a class label in the consequent corresponds to the probability of predicting the associated class label.

In order to obtain class label predictions from a rule, it is necessary to select a classification threshold. If the value of the i-th component is greater than or equal to the classification threshold, the corresponding i-th class label is predicted. Note that the consequents of the rules fulfil the requirements for the hierarchical multi-label classification task: (1) the classes predicted are consistent with the class hierarchy, since the probability of a parent class label is always equal to or greater than the probability of its children class labels; (2) multiple unrelated (non-hierarchically related) class labels can be predicted according to the examples covered by the rule.

The same deterministic procedure is applied to compute the consequent of a default rule when classifying an unseen example, as described in subsection 7.1.7, with the difference that the uncovered set of examples—i.e., the set of examples which is not covered by any rule—is taken into account in Equation (7.14).

7.2.2 Distance-based Heuristic Information

Recall that the heuristic information in Ant-Miner, *c*Ant-Miner and *h*Ant-Miner involves a measure of entropy. The entropy characterises the homogeneity of a collection of examples related to the class attribute values (labels), giving a notion of (im-)purity of the class values' distribution. The more examples of the same class label the lower the value of entropy will be and the 'purest' is the collection of examples. It should be noted that in all calculations involving entropy the different class labels are independently evaluated—i.e. no relationship between class labels is taken into account. In the case of Ant-Miner and *c*Ant-Miner, which are applied to flat classification problems, the use of the entropy measure does not present a limitation, since there is no relationship between class labels. On the other hand, the same cannot be said for *h*Ant-Miner, which aims at extracting hierarchical classification rules, derived from data where the class labels are organised in a hierarchical structure.

To illustrate the limitation of the entropy measure when used in hierarchical problems, let's consider the following example. Given a tree-structured class hierarchy, where class labels {1, 2, 3} are children of the root class label, class labels {2.1, 2.2} are children of the '2' class label and each class label has 10 examples. Although the entropy is calculated—according to Equation (7.9)—accross all class labels, the hierarchical relationships are not taken into account. Therefore, the entropy of a hypothetical term '*IPR00023* = *yes*' which is present in 10 examples associated with class label '1' and in 10 examples associated with class label '3' would be the same as of a hypothetical term '*IPR00023* = *no*' which is present in 10 examples associated with class label '2' and in 10 examples associated with class label '2.1'. The drawback in this case is that it is known that class labels '2' and '2.1' are more similar than class labels '1' and '3'. Hence, it would be expected/desired that the entropy measure—or an alternative heuristic information—exploit hierarchical relationships in order to better reflect the quality of each term in the case of hierarchical classification problems. Intuitively this becomes even more important when dealing with bigger (in terms of number of class labels and depth) hierarchical structures. It should be noted that several Ant-Miner variations—as discussed in [50]—have used a heuristic information based on the relatively frequency of the class label predicted by the rule (or the majority class label) amongst all the examples that have a particular term, which would also present the aforementioned limitation.

*hm*Ant-Miner employs a distance-based heuristic information, which directly incorporates information from the class hierarchy. More precisely, the heuristic information of a term corresponds to the variance of the set of examples covered by the term (the set of examples that satisfy the condition represented by the term). In order to calculate the variance, the class labels of each example are represented by a numeric vector of length m (where m is the number of class labels of the hierarchy without considering the root class label). The i-th position of the class label vector of an example is equal to 0 or 1 if the correspondent class label is absent or present, respectively. The distance between class label vectors is defined as the weighted Euclidean distance, given by

$$distance(\vec{v}_1, \vec{v}_2) = \sqrt{\sum_{i=1}^{m} [w(l_i) \cdot (\vec{v}_{1,i} - \vec{v}_{2,i})^2]} \ , \tag{7.15}$$

where $w(l_i)$ is the weight associated with the i-th class label, $\vec{v}_{1,i}$ and $\vec{v}_{2,i}$ are the values of the i-th position of the class label vectors $\vec{v}_1$ and $\vec{v}_2$, respectively. Then, the variance of set of examples is defined as the averaged squared distance between each example's class label vector and the set's mean class vector, given by

$$variance(S_T) = \frac{\sum_{j=1}^{|S_T|} distance(\vec{v}_j, \overline{v})^2}{|S_T|} \ , \tag{7.16}$$

where S_T is the set of examples covered by a term T (i.e., the set of examples that satisfies the condition represented by term T) and $\overline{v}$ is the set's mean class label vector. Finally, the heuristic information of a term (vertex) T is given by

$$\eta_T = \frac{variance_{max} - variance(S_T)}{variance_{max}} \ , \tag{7.17}$$

where $variance_{max}$ is defined as the sum of the worst and best variance values observed across all terms in order to assign values greater than zero to the worst terms, which otherwise would avoid them to be selected by an ant. Note that the heuristic information is normalised so the smaller the value of the variance of a term T the greater its heuristic information becomes. This is analogous to the use of the entropy measure in Ant-Miner, where smaller values are preferred over bigger values since they correspond to a more homogeneous partition (where the great majority of examples are associated with the same class label). It should be noted that for continuous attribute terms (vertices), it is required to firstly select a threshold value and a relational operator to form a triple (*attribute*, *operator*, *value*) taking into account all training examples—as detailed in subsection 7.2.3—since continuous attribute terms are only represented by the continuous attribute—i.e. they do not represent a complete attribute-value condition—in the construction graph.

Recall that the distance function in Equation (7.15) requires the definition of a class-specific weight. In Vens et al. [127], where the proposed Clus-HMC algorithm also uses a variance measure based on a weighted Euclidean distance, several weighting schemes have been evaluated in the context of hierarchical multi-label classification. As a result of their findings, the preferred weighting scheme—and the one used in *hm*Ant-Miner—is defined as

$$w(l) = w_0 \cdot \frac{\sum_{i=1}^{|P_l|} w(p_i)}{|P_l|}\,, \tag{7.18}$$

where w_0 is arbitrarily set to 0.75, P_l is the parents class label set of the class label l and $w(p_i)$ is the weight associated with the i-th parent class label of the class label l. In other words, the weight of a class label l is the multiplication of the w_0 weight and the average weight of its parent class labels. For class labels at the top of the hierarchy (children of the root class label), their weights are set to w_0. According to Equation (7.18), class labels appearing higher in the hierarchy will have greater weights than class labels appearing lower in the hierarchy. Therefore, concerning the weighted distance function in Equation (7.15), similarities at higher levels of the hierarchy are more important than similarities at lower levels. Figure 7.4 illustrates the class-specific weight distribution for the class hierarchy presented in Figure 7.2.

7.2.3 Distance-based Discretisation of Continuous Values

As discussed in subsection 7.2.2, the entropy measure is not very suitable for hierarchical multi-label classification problems. Therefore, the entropy-based discretisation procedure employed by *h*Ant-Miner (derived from *c*Ant-Miner) presents the same limitation of evaluating each of the class labels individually, not taking into account their relationships. Consequently, the quality of continuous attributes threshold values are compromised, which can lead to poor discovered rules.

Using the variance measure defined in Equation (7.16), *hm*Ant-Miner employs a distance-based discretisation procedure of continuous attributes values in its rule

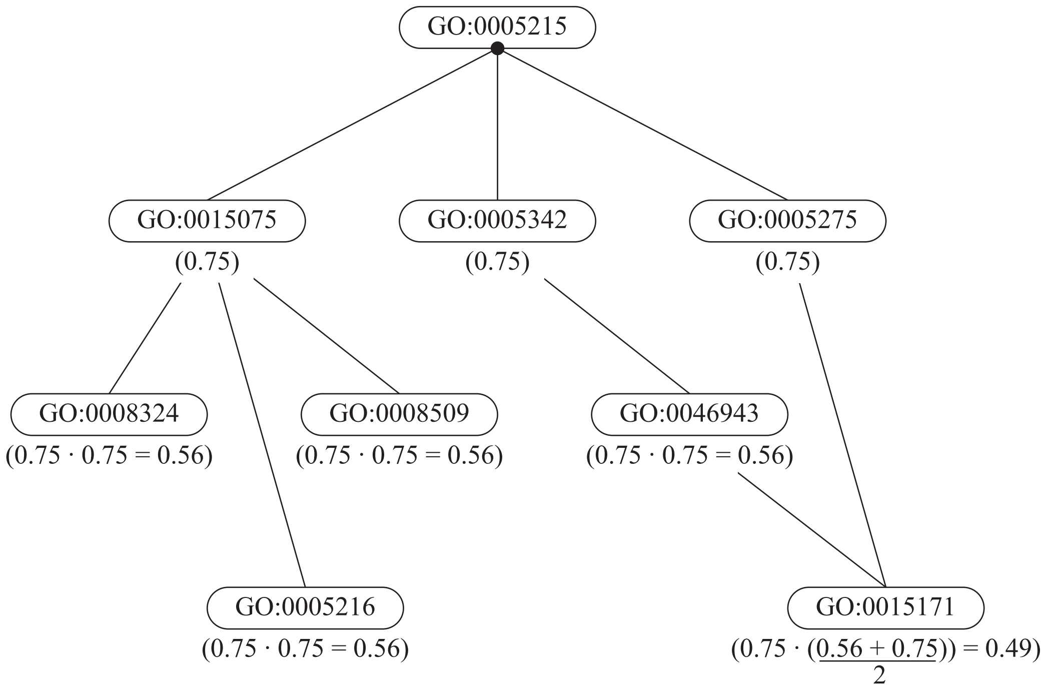


Figure 7.4: Illustration of the class-specific weights—according to Equation (7.18)—for the class hierarchy presented in Figure 7.2. Note that the class label 'GO:0005215' does not have a weight associated, since it represents the root of the class hierarchy.

construction process. Given a continuous attribute y_i, the basic idea is to find a threshold value v (where v is a value in the domain of attribute y_i) that maximises the variance gain of both $y_i < v$ and $y_i \geq v$ generated partitions of examples—i.e. the set of examples which have the value of attribute y_i less than v and the set of examples which have the value of the attribute y_i greater than or equal to v—relative to a set of examples S. The distance-based discretisation procedure, dubbed variance-gain discretisation, is divided into two steps as follows.

Let y_i be a continuous attribute to undergo the discretisation procedure and v a value in the domain of y_i. The best threshold value for attribute y_i is the value v which minimises the variance of the $y_i < v$ and $y_i \geq v$ generated partitions of examples from S, maximising the variance gain relative to S as a result, given by

$$\begin{aligned} variance\_gain(y_i, v; S) = variance(S) &- \frac{|S_{y_i<v}|}{|S|} \cdot variance(S_{y_i<v}) \\ &- \frac{|S_{y_i \geq v}|}{|S|} \cdot variance(S_{y_i \geq v}) \ , \end{aligned} \tag{7.19}$$

where $|S_{y_i<v}|$ is the total number of examples in the partition $y_i < v$ (partition of training examples where the attribute y_i has a value less than v), $|S_{y_i \geq v}|$ is the total number of training examples in the partition $y_i \geq v$ (partition of training examples where the attribute y_i has a value greater than or equal to v) and $|S|$ is the total number of training examples. The values of $variance(S)$, $variance(S_{y_i<v})$ and $variance(S_{y_i \geq v})$ are calculated according to Equation (7.16). The variance gain measure is calculated for all values v, which comprises the average value of each pair of adjacent values v_w and v_{w+1} in the domain of the attribute y_i—computed as $(v_w + v_{w+1})/2$—and the value v with the highest variance gain associated is then selected as the best threshold value. As in *c*Ant-Miner and *h*Ant-Miner, examples from the set of training examples S with missing values for the continuous attribute y_i, if present, are not taken into account in the threshold selection.

Note that the set of training examples S varies according to the context of the rule construction process, that is to say, the set of training examples S is restricted to the set of training examples covered by the current partial rule being constructed. The only exception to this restriction is when the current partial rule is empty, thus all training examples are used on the evaluation of threshold values. As a result of this restriction, the choice of a threshold value during the rule construction process is tailored to the current candidate rule.

After the selection of the best threshold value v_{best}, a relational operator is selected based on the individual variance values of the generated partitions. If the partition of examples $y_i < v_{best}$ has a lower variance, then the operator '$<$' (less-than operator) is selected; if the partition of examples $y_i \geq v_{best}$ has a lower variance, then the operator '$\geq$' (greater-than-or-equal-to operator) is selected; ties are broken at random. As can be noticed, the operator selection has a bias of selecting the more homogeneous partition, given that lower variance values are preferred over higher values. This is analogous to the bias of the entropy-based discretisation of *c*Ant-Miner, where lower entropy values are preferred since they are associated with the 'purest' partition—i.e. the partition with more examples associated with the same class label.

At the end of the discretisation process, a term represented as a triple (y_i, *operator*, v_{best}) is created to be added to the current partial rule (e.g. $y_i < 20$) and the rule continues to undergo the rule construction process.

Concerning the computational time complexity of the entropy-based discretisation used in *h*Ant-Miner and the proposed distance-based discretisation in *hm*Ant-Miner, the process of finding a threshold value can be divided into two steps. First, both discretisation procedures require the sorting of continuous attribute values in order to facilitate the partition of examples. The time complexity of this step is $O(n \cdot \log n)$, where n is the number of training examples under consideration.

In the case of the entropy-based discretisation, the second step involves the evaluation of potentially n candidate threshold values—assuming that each training example have a different value for the continuous attribute undergoing discretisation[1]—over k different class labels. The complexity of this step is $O(n \cdot k)$, and the total complexity of the entropy-based discretisation is $O(n \cdot \log n) + O(n \cdot k)$.

In the case of the distance-based discretisation, the second step involves the calculation of the mean class label vector for each partition of training examples. This calculation has time complexity of $O(n \cdot k)$. Furthermore, it involves the calculation of the distance between each example's class label vector and the partition mean's class label vector in order to determine the variance of the partitions. Since each class label vector has k positions and there are potentially n candidate threshold values, the time complexity of the variance calculation is $O(n \cdot k)$. Given that for each candidate threshold value, the mean class label vectors of the partitions must be recalculated because a partition's example distribution varies according to the threshold value, the total time complexity of the distance-based discretisation is $O(n \cdot \log n) + O([n \cdot k]^2)$.

Intuitively, if both complexity notations are simplified by dropping the common element $O(n \cdot \log n)$, the entropy-based discretisation growing factor is linear in relation to $n \cdot k$, while the distance-based discretisation growing factor is quadratic in relation to $n \cdot k$. Therefore, the distance-based discretisation is more computationally complex than the entropy-based discretisation. It should be noted that the number of training examples n covered by a rule, and consequently the potential number of candidate threshold values, tends to decrease in relation to the number of terms in the antecedent of a rule. Hence, the efficiency of the discretisation procedure is increased at later stages of the rule construction

[1]This represents the worse case scenario for the discretisation procedure, and in general, the number of candidate threshold values is smaller than the number of training examples.

procedure, since fewer candidate threshold values have to be evaluated.

7.2.4 Hierarchical Multi-Label Rule Evaluation

Following a similar approach of using a distance-based measure for the discretisation of continuous values, the variance gain can be applied to compute a rule quality measure. The basic idea to evaluate a rule r using the variance gain measure is to virtually divide the training set S (where S corresponds to the set of all training examples) into two partitions: the set of examples covered by the rule r (S_r) and the set of examples not covered by the rule r ($S_{\neg r}$). Then, the variance gain of rule r relative to S can be computed as

$$\begin{aligned} variance\_gain(r, S) = variance(S) &- \frac{|S_r|}{|S|} \cdot variance(S_r) \\ &- \frac{|S_{\neg r}|}{|S|} \cdot variance(S_{\neg r}) . \end{aligned} \tag{7.20}$$

The motivation of using the variance as a rule quality measure is as follows. Firstly, it can naturally cope with hierarchical multi-label data, taking into account the relationships and similarities between class labels. Secondly, it favours rules that partition the training set into a more homogeneous sets of examples. As a result, rules that cover a more homogeneous set of examples, as well as leaving uncovered a more homogeneous set of examples (which should facilitate the discovery of other rules in the future), are preferred.

7.2.5 Simplified Rule Pruning

Since the consequent of a rule is determined as detailed in subsection 7.2.1, *hm*Ant-Miner does not employ a second colony in order to construct the consequent of rules. Therefore, the rule pruning procedure is simplified as follows. The rule is submitted to a removal process of its antecedent's last term and have its consequent re-calculated, since the set of covered examples could change after the removal of the term. The removal process is repeated until the quality of the rule decreases when its last term is removed or the rule has only one term left in the antecedent. Algorithm 7.3 presents a high-level pseudocode of the rule pruning procedure.

Let $rule_{current}$ be the rule undergoing the pruning, which is considered the

Algorithm 7.3: *hm*Ant-Miner rule pruning procedure pseudocode.

```
input  : rule to be pruned
output: the pruned rule
1  begin
2      $rule_{best} \leftarrow rule_{current}$;
3      $q_{best} \leftarrow Q(rule_{best})$;
4      repeat
5          $rule_i \leftarrow rule_{best}.antecedent - last\_term(rule_{best}.antecedent)$;
6          $calculate\_consequent(rule_i)$;
7          $q_i \leftarrow Q(rule_i)$;
8          if $(q_i \geq q_{best})$ then
9              $rule_{best} \leftarrow rule_i$;
10             $q_{best} \leftarrow q_i$;
11         end
12     until $q_i < q_{best}$ OR $|rule_{best}.antecedent| = 1$ ;
13     return $rule_{best}$;
14 end
```

best rule at the beginning of the pruning procedure. At each iteration of the repeat loop in Algorithm 7.3, a candidate rule $rule_i$ is created by removing the last term of the antecedent of the current best $rule_{best}$ and the consequent of $rule_i$ is computed according to subsection 7.2.1. Then, the quality measure q_i for $rule_i$ is computed. If the quality measure q_i is higher than the current best quality q_{best}, $rule_i$ substitutes $rule_{best}$, completing an iteration of the pruning procedure. This procedure is repeated until $rule_{best}$ has just one term left on its antecedent or a candidate rule $rule_i$ does not improve the quality over $rule_{best}$ (i.e. $q_{best} > q_i$).

7.3 Pittsburgh-based Approach

The algorithms presented in subsections 7.1 and 7.2 can be seen as direct and sophisticated extensions of Ant-Miner for the hierarchical multi-label classification problem. While *h*Ant-Miner and *hm*Ant-Miner employ heuristic information, evaluation measures and pruning procedures tailored for hierarchical—and multi-label in the latter case—problems, they share the same sequential covering strategy of Ant-Miner in order to build a list of rules that covers all training examples. At each iteration of the sequential covering, an ACO procedure is used to create a single rule which covers a set of examples. The examples covered by the rule are removed from the training set and a new rule is then created, until a stopping criterion is reached.

Drawing a comparison concerning rule discovery strategy with the broader area of evolutionary algorithms, the sequential covering strategy employed in Ant-Miner falls into the iterative rule learning (IRL) approach [58, 128]. In the IRL approach, each run of the evolutionary procedure—analogous to the ACO procedure in Ant-Miner case—discovers a single rule (the best rule produced over all iterations) and the procedure is repeated multiple times in order to discover a list of rules. Another two approaches for rule discovery have been used in the evolutionary algorithm literature: the Michigan [17, 18] and the Pittsburgh [109, 110] approaches. In the Michigan approach, each individual corresponds to a rule and a list of rules is represented by the entire population, using some mechanism to ensure that different rules cover different regions of the data space. Hence, a single run of an evolutionary procedure following a Michigan approach discovers a complete list of rules. Similarly, in the Pittsburgh approach, each run of the evolutionary procedure discovers a complete list of rules (the best list of rules produced over all iterations). One of the main differences between IRL/Michigan and Pittsburgh approaches is that in the latter a complete list of rules, which constitutes an individual, is evaluated instead of a single rule, in order to guide the discovery process. As discussed in [48], evaluating the quality of a rule individually, instead of the quality of a list of rules as a whole, has difficulty with the problem of rule interaction—i.e the list of best rules is not necessarily the best list of rules.

The problem of rule interaction in Ant-Miner, and consequently in hAnt-Miner and hmAnt-Miner, can be illustrated as follows. In Ant-Miner's sequential covering strategy, the discovery of a rule can be seen as an independent search problem for the best rule given the current training set. In each iteration of the sequential covering, a rule is constructed by an ACO procedure and the examples covered by the rule are removed from the data set. This iterative process of constructing a rule is repeated until the training set is empty (or almost empty). Although rules are discovered in an one-at-a-time fashion, the outcome of a rule (the examples covered by the rule) affects the rules that can be discovered subsequently since the search space is modified due to the removal of examples covered by previous rules. Therefore, the sequential covering performs a greedy search for a list (sequence) of rules which is not guaranteed to be the best list of rules that covers the training set, since the interaction between them is not taken into account during the search. Additionally, there is no guarantee that the best rule of an iteration will be part of the best list of rules.

As aforementioned, hierarchical multi-label classification problems are more complex in comparison to flat problems, since they usually deal with a greater number of class labels and there are hierarchical relationships between class labels. Therefore, the problem of rule interaction is potentially aggravated when the sequential covering strategy is applied to hierarchical multi-label problems.

In this section, we focus on extending the sequential covering strategy to elaborate a Pittsburgh-based ACO classification algorithm for the discovery of hierarchical multi-label classification rules. The main motivation is to mitigate the problem of rule interaction that potentially affects the performance of the previously proposed methods, namely hAnt-Miner and hmAnt-Miner. In order to cope with the rule interaction problem, we propose building a complete list of rules within the ACO procedure, instead of a single rule. Therefore, at each iteration of the ACO procedure, a complete list of rules is used to guide the search—i.e. the search is guided using the quality of the best list of rules of an iteration. By evaluating a list of rules as a whole, the interaction between rules is taken into account. As a result, a single run of the ACO procedure is needed to discover a list of rules.

Subsection 7.3.1 presents the technical details of the proposed hierarchical multi-label classification algorithm, named hmAnt-Miner$_{PB}$ (hierarchical multi-label classification Ant-Miner based on the Pittsburgh approach), which employs a Pittsburgh-based ACO procedure to discover a list of rules.

7.3.1 Extended Sequential Covering Strategy

The hierarchical classification ACO algorithms in previous sections of this chapter followed the same sequential covering strategy: they start with an empty list of rules and iteratively add one-rule-at-a-time to the list, maximising a specified rule quality measure in order to build a list of rules. Therefore, the list of rules is created in a greedy fashion—i.e. the best rule, which is evaluated independently from other rules, found at each iteration is selected. hmAnt-Miner$_{PB}$ employs a variation of the ACO-based sequential covering strategy with the aim of performing a more global search for the optimal list of rules. It differs from hAnt-Miner and hmAnt-Miner as follows.

Firstly, the ACO search is guided by the quality of a candidate list of rules. Therefore, pheromone values are updated based on the quality of a list of rules, in contrast to the quality of a single rule. Secondly, the heuristic information is updated after a rule is added to a candidate list of rules. In this way, the heuristic information is used to direct the search for different rules in order to build a list of rules since pheromone values are constant within an iteration—i.e. during the creation of a candidate list of rules. On the other hand, hmAnt-Miner$_{PB}$ shares the same rule representation, rule evaluation measures and pruning procedure as hmAnt-Miner presented in section 7.2, and the same antecedent construction graph, pheromone initialisation and update procedures as hAnt-Miner presented in section 7.1.

Algorithm 7.4 presents a high-level pseudocode of hmAnt-Miner$_{PB}$. In summary, hmAnt-Miner$_{PB}$ works as follows. It starts initialising the antecedent construction graph pheromone values and then enters an iterative (*repeat* loop) procedure to create a complete list of rules until a user-specified maximum number of iterations is reached or the quality of the best list of rules is not improved over a user-specified number of iterations, which works as a convergence test. At each i-th iteration, *max_number_lists* lists of rules are created and evaluated (outer *for* loop). In order to create a list of rules, it starts with an empty list of rules and adds one rule at a time until the number of uncovered training examples is lower than or equal to a user-specified maximum value (*while* loop).

At the beginning of the construction process of a list of rules, the training set for the current iteration is initialised with all training examples and the heuristic information of the antecedent construction graph is (re-)calculated. The construction process of a list of rules consists of an iterative process, wherein *colony_size* rules are created by ants at each iteration (inner *for* loop). To create rules, ants start with an empty rule (no terms in its antecedent) and add one term at a time to their rule antecedent. Terms (vertices of the antecedent construction graph) are probabilistically chosen to be added to the current partial rule based on the values of the amount of pheromone (τ) associated with the edge connecting the last term of the rule to the candidate term in question and their problem-dependent heuristic information (η). Recall that, as in hAnt-Miner, an ant begins the creation of a rule starting from the dummy '*start*' vertex.

Ants keep adding a term to their partial rule until any term added to their rule would make it cover less training examples than a user-specified minimum number of covered examples or all attributes are already present in the antecedent of the rule. The first rule construction stopping criterion is used to avoid the creation of very specific rules, which would not generalise well on the test set; the latter criterion is necessary to avoid inconsistencies caused by the selection of terms representing different conditions using the same attribute, such as '$IPR00023 = yes$' and '$IPR00023 = no$'—similar to the original hAnt-Miner algorithm.

Then, the consequent of the rule is calculated according to Equation (7.14). Once the rule construction process has finished, the rule created by an ant undergoes a pruning procedure, according to Algorithm 7.3, in order to remove irrelevant terms from its antecedent. The best rule amongst these constructed rules is added to the current list of rules and the training examples covered by it are removed from the training set. Finally, the heuristic information is re-calculated and the iterative rule construction process continues.

It is important to emphasise that heuristic information is not fixed, while pheromone values are fixed, between iterations of the rule construction process (*while* loop). The rationale behind the re-calculation of the heuristic information is as follows. Since a complete execution of the rule construction process is used to create a list of rules, at each iteration of this process, a rule covering a different set of training examples needs to be created. Given that pheromone values are fixed during the creation of a list of rules in order to guide the search based on the quality of a list of rules instead of a single rule, the heuristic information is re-calculated after the removal of the examples covered by previous rules to facilitate the discovery of different rules—i.e. rules covering different examples of the training set.

Finally, the quality of the best list of rules—measured using precision-recall curves, as described in subsection 2.5.2—of the i-th iteration is used to update the pheromone values, which will then be used in the next iteration. Note that the reference for the best list of rules, based on its quality, found so far is updated if the quality of the list of rules of the i-th iteration is greater than the quality of the current best. At the end of the process, the best list of rules found over all iterations is selected as the discovered list of rules.

As can be seen in Algorithm 7.4, pheromone values are updated according to the quality of the best list of rules of the current iteration (end of the *repeat* loop); heuristic information is initialised at the beginning of the construction process for each candidate list of rules (beginning of the outer *for* loop); heuristic information is updated, while pheromone levels are fixed, after a rule is added to a candidate list of rules (end of the outer *for* loop).

Algorithm 7.4: High-level pseudocode of the Pittsburgh-based ACO procedure employed in *hm*Ant-Miner$_{PB}$.

```
input  : training examples
output : discovered list of rules
begin hmAnt-Miner$_{PB}$
    $rule\_list_{best} \leftarrow \emptyset$;
    $\tau \leftarrow$ initializes pheromones;
    $i \leftarrow 1$;
    repeat
        $rule\_list_i \leftarrow \emptyset$;
        for $j \leftarrow 1$ to $max\_number\_lists$ do
            $tr\_set_j \leftarrow$ all training examples;
            $rule\_list_j \leftarrow \emptyset$;
            $\eta \leftarrow$ initialise heuristic information;
            // creates a complete rule list
            while $|tr\_set_j| > max\_uncovered\_examples$ do
                $rule_{best} \leftarrow \emptyset$;
                for $k \leftarrow 1$ to $colony\_size$ do
                    $rule_k \leftarrow CreateRule()$;
                    $Prune(rule_k)$;
                    if $Q(rule_k) > Q(rule_{best})$ then
                        $rule_{best} \leftarrow rule_k$;
                    end
                    $k \leftarrow k + 1$;
                end
                $rule\_list_j \leftarrow rule\_list_j + rule_{best}$;
                $tr\_set_j \leftarrow tr\_set_j - Covered(rule_{best}, tr\_set_j)$;
                $update\_heuristic(\eta, tr\_set_j)$;
            end
            // updates the best rule list of the $i$-th iteration
            if $Q(rule\_list_j) > Q(rule\_list_i)$ then
                $rule\_list_i \leftarrow rule\_list_j$;
            end
            $j \leftarrow j + 1$;
        end
        if $Q(rule\_list_i) > Q(rule\_list_{best})$ then
            $rule\_list_{best} \leftarrow rule\_list_i$;
        end
        $update\_pheromones(\tau, rule\_list_i)$;
        $i \leftarrow i + 1$;
    until $i \geq max\_number\_iterations$ OR $convergence()$ ;
    return $rule\_list_{best}$;
end
```

7.3.2 Updating Pheromone Values Based on the Rule List Quality

Since the search in hmAnt-Miner$_{PB}$ is guided by the quality of a list of rules, rather than a single rule, the quality of the list of rules is used to increase pheromone values, as follows.

For each rule of the best list of rules of a given iteration, the pheromone value of the edge that connects the i-th vertex to the $(i + 1)$-th vertex $(0 \leq i < |rule.antecedent|)$ in the rule's antecedent—where the 0-th vertex corresponds to the 'dummy' start vertex—is incremented according to

$$\tau_{edge_{ij}} = \tau_{edge_{ij}} + \tau_{edge_{ij}} \cdot Q(rule\_list)\,, \tag{7.21}$$

where i and j are the i-th and j-th vertices of an edge from i to j in the trail (rule antecedent) being updated $(edge_{ij})$, $Q(rule\_list)$ corresponds to the quality of the best list of rules of the current iteration—i.e., the list of rules from which the rule belongs. Equation (7.21) can be seen as an extension of Equation (7.7) employed in hmAnt-Miner, wherein the quality of a list of rules is used to update pheromone values of edges between the vertices used by rules instead of using the quality of a single rule.

7.4 A Baseline Approach for Hierarchical Multi-Label Classification with Ant-Miner: Building One Classifier per Class

After having proposed three different ACO classification algorithms for hierarchical and multi-label classification problems that build a list of rules taking into account all class labels at the same time, this section presents a baseline algorithm which discovers rules for each class label individually, using multiple runs of an ACO algorithm for flat classification. The basic idea is to divide the hierarchical classification problem into a set of binary classification problems by training an adapted cAnt-Miner2-MDL[2] algorithm for each class label and then combine the discovered rules into a hierarchical set of rules. Although several individual classifiers (sets of rules) are built, the final combined set of rules predictions are

[2]We have chosen cAnt-Miner2-MDL based on the results presented in chapter 6, which shows cAnt-Miner2-MDL as the most accurate cAnt-Miner variation overall.

consistent with the class hierarchy.

The motivation for using an adapted *c*Ant-Miner2-MDL is as follows. Each run of *c*Ant-Miner2-MDL deals with a binary classification problem, where the positive examples are those that have the class label associated with the classifier being built, regardless of whether or not that class label is the most specific or a higher level (ancestor) class label associated with the example. In many cases, specially at lower levels of the class hierarchy, the number of positive examples will be much smaller that the number of negative examples. To mitigate the problem of dealing with unbalanced class distributions, each run of *c*Ant-Miner2-MDL is set to discover only rules predicting the positive class—i.e. the presence of the class label associated with the classifier being built. Therefore, the consequent of a rule is fixed during the rule construction. As a result, a class-specific heuristic information is employed in an adapted *c*Ant-Miner2-MDL algorithm in order to facilitate the discovery of rules covering positive examples.

Note that this approach is different from most one-classifier-per-class approaches found in the literature, since in our case each classifier is built only with rules predicting the positive class (presence of a class label). Therefore, unlike most of the literature, rules that predict the absence of a class label (negative class) are not discovered, thus not all training examples from each binary-class data set are covered. A complete classification model is obtained by combining the output (rules) of each classifier, so that the combined, aggregated set of rules is able to make hierarchical multi-label predictions.

The main drawback of the algorithm proposed in this section, named *c*Ant-Miner$_{HM}$ (*c*Ant-Miner for hierarchical multi-label classification), when compared to the previously proposed algorithms is that it requires multiple runs of a classification algorithm (*c*Ant-Miner2-MDL in this case). More precisely, for a class hierarchy with $|\mathcal{L}|$ labels, $|\mathcal{L}| - 1$ executions of the classification algorithm are required, since there is no need to build a classifier for the root class label. Subsection 7.4.1 presents the technical details of the proposed method.

7.4.1 The Baseline Ant Colony Algorithm

In essence, the proposed *c*Ant-Miner$_{HM}$ algorithm divides the hierarchical multi-label problem into several binary classification problems. It uses the class hierarchy to define the set of positive and negative examples for each training set associated with a class label of the class hierarchy. Then, an adapted cAnt-Miner2-MDL algorithm is employed to discover classification rules covering the positive examples of each training set. Subsequently, the individually discovered sets of rules—i.e. one set of rules for each class label of the class hierarchy—are combined into a global set of rules, which is used to make hierarchical multi-label predictions. Algorithm 7.5 presents the high-level pseudocode of cAnt-Miner$_{HM}$.

Given a class hierarchy $\mathcal{L}$, where $|\mathcal{L}|$ denotes the number of class labels in the class hierarchy, and a set of training examples, cAnt-Miner$_{HM}$ works as follows. For each class label $l_i \in \mathcal{L}$, a binary-class training set is created by labelling all examples that have the class label l_i (as its most specific or a higher-level class label) as positives and all examples that do not have the class label l_i as negatives. Then, an adapted cAnt-Miner2-MDL algorithm is applied to discover a set of rules covering the positive examples of the newly created binary-class training set for class label l_i.

The adapted cAnt-Miner2-MDL algorithm (outer *for* loop in Algorithm 7.5) starts with an empty set of rules and iteratively adds one rule at a time to that set while the number of uncovered positive training examples is greater than a user-specified maximum value. At the beginning of each iteration, it initialises the pheromone values and heuristic information. The heuristic information consists of a measure of the frequency of positive examples relative to the training set, as detailed in subsection 7.4.2.

In order to create a rule, a single ant starts with an empty rule (no terms in its antecedent) and adds one term at a time to the rule antecedent. It probabilistically chooses a term to be added to the current partial rule based on the values of the amount of pheromone (τ) associated with edges and heuristic information (η) associated vertices of the construction graph. The ant keeps adding a term to the partial rule until any term added to the antecedent would make the rule cover less training examples than a user-specified threshold, which would make the rule too specific and unreliable, or all attributes have already been used by the ant—similar to the original cAnt-Miner algorithm.

Once this process of rule construction has finished, the rule created by the ant is pruned to remove irrelevant terms from its antecedent. Then, the rule is evaluated based on a quality measure Q and the best rule of the current iteration (the rule with the highest quality amongst all rules created by the ants) is selected to update the pheromone values, and then another iteration of the rule construction process starts. The process of constructing a rule is repeated until a user-specified number

Algorithm 7.5: High-level pseudocode of the *c*Ant-Miner$_{HM}$ baseline algorithm.

```
input  : training examples
output: discovered set of rules
begin cAnt-Miner_{HM}
    $rule\_set \leftarrow \emptyset$;
    $training\_set \leftarrow$ all training examples;
    // discovers rules for each class label independently
    // using an adapted cAnt-Miner2-MDL algorithm
    for $i \leftarrow 1$ to $|\mathcal{L}| - 1$ do
        // creates the binary training set for the $i$-th
        // class label
        $tr\_set_i \leftarrow local\_training(training\_set, i)$;
        $rule\_set_i \leftarrow \emptyset$;
        repeat
            $rule_{best} \leftarrow \emptyset$;
            $initialise(\tau, \eta)$;
            $k \leftarrow 0$;
            // discovers a classification rule
            repeat
                for $j \leftarrow 1$ to $colony\_size$ do
                    $rule_j \leftarrow CreateRule()$;
                    $Prune(rule_j)$;
                    if $Q(rule_j) > Q(rule_{best})$ then
                        $rule_{best} \leftarrow rule_j$;
                    end
                    $j \leftarrow j + 1$;
                end
                $\tau \leftarrow update\_pheromones()$;
            until $k \geq max\_number\_iterations$ OR $rule\,convergence$ ;
            $rule\_set_i \leftarrow rule\_set_i + rule_{best}$;
            $tr\_set_i \leftarrow tr\_set_i - PositiveCoveredCases(rule_{best}, tr\_set_i)$;
        until $PositiveCases(tr\_set_i) \leq max\_uncovered\_examples$ ;
        // adds the rules for the $i$-th class label to the
        // global rule set
        $rule\_set \leftarrow rule\_set + rule\_set_i$;
    end
    return $rule\_set$;
end
```

of iterations has been reached, or the best rule of the current iteration is exactly the same as the best rule created by a predefined number of previous iterations, which works as a rule convergence test.

Recall that the consequent of a rule is fixed—i.e. it always predicts the positive class—since only rules predicting the positive class (the presence of the class label l_i) are created. Therefore, the quality measure Q reflects how well the rule covers the positive examples of the training set. The best rule found along this iterative rule construction process is added to the set of rules and the correctly classified training examples are removed from the training set. An example is considered correctly classified if it satisfies the rule antecedent and has the class label predicted by the rule consequent.

After the creation of the set of rules for class label l_i is completed, the consequents of the rules are set to predict the i-th class label and all its ancestor class labels. Since the rules discovered for the i-th class label always predict the presence of the class label (positive class in a binary classification problem), they predict the class label i and all its ancestor according to the class hierarchy $\mathcal{L}$ due to the 'is-a' class relationship, where an example associated with a class label is automatically associated with all of its ancestor class labels. Finally, the discovered rules are added to a global set of rules and rules for the remaining class labels are created. Since the global set of rules contains rules predicting all class labels of the class hierarchy $\mathcal{L}$, it can be used to make hierarchical multi-label predictions, as described in subsection 7.4.5.

7.4.2 Class-specific Heuristic Information

Recall that the class predicted by a rule is fixed, since only rules predicting the positive class (presence of a class label) are created to mitigate the problem of dealing with a very unbalanced class distribution at lower levels of the class hierarchy. In order to facilitate the discovery of rules covering positive examples, *c*Ant-Miner$_{HM}$ employs a class-specific heuristic information which takes into account the frequency of positive examples in the training set.

Let l_i be the i-th class label of the class hierarchy $\mathcal{L}$ and S_i the binary-class training set for class label l_i. The heuristic information for a nominal attribute term T_N relative to the training set S_i is given by

$$\eta_{T_N} = \frac{|T_N \ \& \ S_i^+|}{|S_i|} , \tag{7.22}$$

where $|T_N \ \& \ S_i^+|$ corresponds to the number of positive examples of the training set S_i in which the nominal term T_N is present—i.e. the number of positive examples satisfying the nominal attribute condition represented by term T (e.g. *IPR00023* = *yes*). As a result, terms that are not present in positive training examples have the heuristic value set to 0 (zero), preventing them to be selected by an ant during the rule construction process.

For continuous attributes terms—wherein the heuristic information is calculated by firstly discretising the continuous attribute in order to find the best threshold value and then selecting the entropy value associated with the best ('purest') partition in *c*Ant-Miner2-MDL, as described in section 5.2—the partition selection is modified to favour the partition associated with the highest frequency of positive examples. Therefore, after the selection of the best threshold value(s) according to the MDL discretisation procedure, the heuristic information of a continuous attribute term T_C associated with a continuous attribute y_i is given by

$$\eta_{T_C} = max\left(\frac{|S^+_{i,y_i<v_1}|}{|S_{i,y_i<v_1}|}, \frac{|S^+_{i,v_z\leq y_i<v_{z+1}}|}{|S_{i,v_z\leq y_i<v_{z+1}}|}, \ldots, \frac{|S^+_{i,y_i\geq v_Z}|}{|S_{i,y_i\geq v_Z}|}\right), \quad \forall\ 1 \leq z < Z, \qquad (7.23)$$

where $|S^+_{i,y_i<v_1}|$, $|S^+_{i,v_z\leq y_i<v_{z+1}}|$ and $|S^+_{i,y_i\geq v_Z}|$ correspond to the number of positive examples on the partition of examples that have the attribute's y_i value less than v_1 ($S_{i,y_i<v_1}$), between v_z (inclusive) and v_{z+1} ($S_{i,v_z\leq y_i<v_{z+1}}$), and greater than or equal to v_Z ($S_{i,y_i\geq v_Z}$), respectively; Z is the total number of threshold values and S_i is the binary-class training set for the i-th class label. As a result of Equation (7.23), the heuristic information of a continuous attribute term T_C corresponds to the frequency value of the discrete partition with the highest number of positive examples relative to its size.

7.4.3 Class-specific Interval Selection for Continuous Attributes

Following a similar approach as the one employed to calculate the heuristic information value of continuous attributes, *c*Ant-Miner$_{HM}$'s interval selection during the discretisation process is adapted to take into account the frequency of positive examples in the intervals. Note that the selection of the best threshold value(s) is still based on the entropy measure, but the discrete interval used to define the attribute-value condition of a continuous attribute term is chosen based on the frequency of positive examples, as follows.

Since potentially multiple threshold values can be created by the MDL discretisation procedure, as described in section 5.2, the list of threshold values is sorted and the frequency of positive examples relative to the total number of examples in the interval of each discrete interval is calculated. Then, the discrete interval with the higher frequency of positive examples is selected, based on the fact that it corresponds to the interval with the greater number of positive examples relative to the interval's size. If an internal interval is selected (an interval between two threshold values), a term in the form $v_z \leq y_i < v_{z+1}$ is generated; otherwise, a term in the form $y_i < v_z$ or $y_i \geq v_z$ is generated (where z is the z-th threshold value); ties are broken at random. As a result, the interval selection employed in *c*Ant-Miner$_{HM}$ is analogous to the one employed in *c*Ant-Miner2-MDL, with the difference that it is based on the frequency of positive examples instead of the entropy measure.

7.4.4 Rule Quality Measure

Since *c*Ant-Miner$_{HM}$ divides the hierarchical multi-label problem into several binary classification problems, rules are evaluated using a flat classification measure. The rule quality measure Q employed in *c*Ant-Miner$_{HM}$ is a combination of both precision (*Prec*) and recall (*Rec*) values, given by

$$\mathit{Prec} = \frac{TP}{TP + FP} \qquad \mathit{Rec} = \frac{TP}{TP + FN} \ , \tag{7.24}$$

where *TP* is the number of positive examples covered by the rule (true positives), *FP* is the number of negative examples covered by the rule (false positives) and *FN* is the number of positive examples not covered by the rule (false negatives). The rule quality measure is then given by

$$Q = \mathit{F\text{-}measure} = \frac{2 \cdot \mathit{Prec} \cdot \mathit{Rec}}{\mathit{Prec} + \mathit{Rec}} \ . \tag{7.25}$$

It should be noted that the number of negative examples not covered by the rule (true negatives) is not taken into account in the rule quality measure. Recall that the consequent of a rule is fixed to predict the positive class (presence of a class label), therefore the number of negative examples not covered by the rule is not relevant to determine its quality.

Algorithm 7.6: High-level pseudocode of the procedure to create the vector of class probabilities for a test example employed in cAnt-Miner$_{HM}$.

```
input  : test example
output: vector of class probabilities
begin
    // the rules that cover the test example
    $rules \leftarrow cover(test\_example, rule\_set)$;
    $probabilities\_vector \leftarrow [1 \ldots |\mathcal{L}|]$;
    // increments the occurrence of each class label found in
    // the consequent of the rules
    foreach $rule$ in $rules$ do
        foreach $label$ in $rule.consequent$ do
            $probabilities\_vector[index(label)]{+}{+}$;
        end
    end
    for $i \leftarrow 1$ to $|\mathcal{L}|$ do
        $probabilities\_vector[i] \leftarrow probabilities\_vector[i]\ /\ |rules|$;
    end
    return $probabilities\_vector$;
end
```

7.4.5 Classifying New Examples

In order to classify a test (unseen) example, a vector of class probabilities is created based on the rules (from the global set of rules) that cover the example. The basic idea is to calculate the probability of predicting a specific class label based on the number of rules covering the example that have the class label in their consequent. The interpretation of the vector of class probabilities is analogous to the one employed by *hm*Ant-Miner, described in subsection 7.2.1.

Algorithm 7.6 presents the high-level pseudocode of the procedure to create the vector of class probabilities for a test example. In essence, the procedure starts by finding the rules from the global set of rules created by cAnt-Miner$_{HM}$ that cover the test example. Then, the vector of class probabilities is initialised to accommodate all class labels of the class hierarchy $\mathcal{L}$. Each position of the vector is initialised to 0 (zero). Moreover, for each class label in the consequent of the rules that cover the test example, its correspondent position on the vector of probabilities is incremented by one for each of those rules. Finally, each position of the vector of probabilities is divided by the number of rules covering the test example. As a result, each position of the class vector represents the probability, defined by the relative frequency of rules that have that class label in their consequent, of predicting the correspondent class label. Note that the vector of class probabilities fulfil the requirements for hierarchical multi-label classification: (1) predictions are consistent with the hierarchical relationships, since the probability of predicting a child class label is guaranteed to be equal to or lower than the probability of predicting each of its ancestral class labels; (2) unrelated class labels—i.e. class labels which are not ancestors/descendants of each other—can be predicted according to the rules that cover the test example.

7.5 Summary

This chapter presented four novel hierarchical and multi-label classification algorithms: a hierarchical classification ACO algorithm, two ACO classification tailored for hierarchical multi-label problems and an adaptation of a flat classification ACO algorithm to the problem of hierarchical multi-label classification.

Section 7.1 presented a hierarchical ACO classification algorithm, named *h*Ant-Miner. *h*Ant-Miner discovers a single global classification model in the form of an ordered list of *IF-THEN* classification rules which can predict class labels at all levels of the class hierarchy, satisfying the parent-child relationships between class labels. *h*Ant-Miner is a major extension of the flat classification Ant-Miner to the hierarchical classification problem, incorporating several important modifications: (1) two separate ant colonies for constructing the antecedent and consequent of a rule; (2) a hierarchical classification rule measure based on the hierarchical precision and recall values; (3) a pruning procedure which removes irrelevant terms from rules' antecedents, as well as, class labels from rules' consequents; (4) a heuristic information straightforwardly adapted to hierarchical classification problems.

In section 7.2, it was presented a hierarchical multi-label ACO classification algorithm, named *hm*Ant-Miner. *hm*Ant-Miner is an extension of *h*Ant-Miner in order to cope with hierarchical multi-label data, differing from the latter in several aspects: (1) the consequent of a rule is deterministically calculated and the algorithm is able to cope with hierarchical multi-label data; (2) the heuristic information is based on a distance measure of a class membership vector of the examples; (3) it employs distance-based rule quality measure and a distance-based discretisation procedure; (4) it has a simplified pruning procedure, since the consequent of a rule is not represented in a construction graph.

Section 7.3 presented a hierarchical multi-label ACO classification algorithm, named hmAnt-Miner$_{PB}$, which is built on top of the ideas of hAnt-Miner and hmAnt-Miner. However, hmAnt-Miner$_{PB}$ employs a different rule discovery approach in order to perform a more effective search for the best list of rules, following a Pittsburgh-based strategy for rule discovery. In this way, the search for rules is guided by the quality of a list of rules in contrast to a single rule.

Finally, section 7.4 presented a baseline hierarchical multi-label ACO classification algorithm, named cAnt-Miner$_{HM}$. It differs from the previously presented methods in several ways. Firstly, the hierarchical multi-label problem is divided into a set of binary classification problems (binary-class data sets). Secondly, an adapted cAnt-Miner2-MDL algorithm is used to discover a set of rules from each binary-class data set predicting the presence of a particular class label. Thirdly, the individually created sets of rules are then combined into a global set of rules which can be used to make hierarchical multi-label predictions. The main drawback of the cAnt-Miner$_{HM}$ algorithm is that several runs of a flat classification algorithm (cAnt-Miner2-MDL in this case) are needed, more specifically, one per class label (except for the root class label).

Chapter 8

Computational Results for Hierarchical and Multi-Label Ant-Miner

In this chapter, the proposed hierarchical—namely *h*Ant-Miner—and hierarchical multi-label—namely *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$—classification algorithms described in chapter 7 are assessed empirically against (1) a baseline approach consisting of training a flat single-label classification algorithm for each class label individually and (2) state-of-the-art decision tree induction algorithms for hierarchical multi-label classification. The data sets in the experiments involve the hierarchical prediction of ion channel protein functions using the Gene Ontology (DAG-structured class hierarchy) functional classification scheme in the former case, and the prediction of protein functions from the yeast (*Saccharomyces cerevisiae*) model organism using both the FunCat (tree-structured class hierarchy) and Gene Ontology functional classification schemes in the latter case.

The assessment of the classification algorithms in terms of predictive accuracy was performed using the evaluation measures for hierarchical classification discussed in section 2.5. Furthermore, in the case of the hierarchical multi-label experiments, the classification algorithms were also assessed in terms of the size of the discovery classification model, since the size is usually considered an important factor that contributes to the comprehensibility of the classification model.

The remainder of this chapter is organised as follows. Section 8.1 presents the details of the experiments concerning *h*Ant-Miner in the context of predicting ion channel protein functions. In section 8.2, the experiments concerning the hierarchical multi-label hmAnt-Miner, hmAnt-Miner$_{PB}$ and cAnt-Miner$_{HM}$ classification algorithms are described. Finally, section 8.3 summarises the computational results presented in this chapter.

8.1 Initial Work on Protein Function Prediction—the *h*Ant-Miner algorithm

This section reports the computational results evaluating the hAnt-Miner (hierarchical classification Ant-Miner) algorithm presented in chapter 7. The experiments in this section consist of our initial work on hierarchical (single-label) classification in the context of protein function prediction. The hAnt-Miner algorithm was compared against a baseline approach, which consists of training a flat classification algorithm for each class label of the class hierarchy individually following a simplified[1] top-down approach.

We have created five data sets involving ion channel proteins associated with corresponding Gene Ontology (GO) terms. Hence, the class hierarchy is represented as a directed acyclic graph (DAG) structure. Since hAnt-Miner is not a hierarchical multi-label algorithm, only proteins associated with a single GO term as its most specific class label are included in the data sets.

The remainder of this section is organised as follows. Subsection 8.1.1 describes the ion channel data sets used in our experiments. Subsection 8.1.2 presents the experimental setup, including the description of the baseline approach algorithm. Finally, the computational results and discussion are presented subsection 8.1.3.

8.1.1 Data Preparation

In order to evaluate the proposed hAnt-Miner algorithm, we have created five data sets involving ion-channel protein functions. Ion channel proteins are present in all living cells and they form a pore across the cell membrane [2]. The function of ion channels is to allow specific inorganic ions (e.g., Na^+, K^+, Ca^{2+}, Cl^-) to cross the cell membrane. They play an essential role in many cell functions, such as in functions related to the nervous system, muscle contraction and T-cell activation.

[1]The baseline approach does not deal with potentially inconsistent predictions resulting from using a top-down approach on directed acyclic graph (DAG) class hierarchies. Instead, it relies on the evaluation measure to penalise for such inconsistencies.

The selection of the protein examples was divided into three steps. In the first step we selected a subset of the Gene Ontology annotation scheme to represent the class hierarchy to be predicted. As we focus on ion channel proteins, all the ancestral and descendant terms of the 'GO:0005216 *ion channel activity*' term were selected. In the second step, we retrieved protein interaction data from the IntAct database (release 15/12/2007). Records with database cross-references to the GO terms selected in the previous step were retrieved. Since many GO terms (class labels) selected in the previous step did not have a reasonable number of proteins associated with them, we discarded GO terms with less than 10 protein examples. In the third step, for each protein example retrieved in the previous step we selected the amino acid sequence and InterPro pattern references from the UniProtKB/Swiss-Prot database (release 12.0), using the database cross-reference to UniProt found in IntAct protein records. At the end of the selection procedure, we ended up with 147 protein examples involving 17 GO terms, which were used to create three different data sets. A summary of the five data sets created for the experiments are presented on Table 8.1.

The first data set (dubbed 'DS1 AA') used the amino acid composition information as predictor attributes, consisting of the percentage of the sequence composition relative to each of the 20 different amino acids. The second data set (dubbed 'DS1 InterPro') used the InterPro pattern information as predictor attributes, consisting of Boolean attributes representing the presence or absence of a particular InterPro pattern. The third data set (dubbed 'DS1 IntAct') used the IntAct protein interaction data as predictor attributes, consisting of Boolean attributes representing the presence or absence of a particular interaction.

Using a similar approach, without the restriction of selecting proteins with protein interaction data available, we increased the number of proteins to 359 examples to create two additional data sets. The first data set (dubbed 'DS2 AA') used the amino acid composition information as predictor attributes. The second data set (dubbed 'DS2 InterPro') used the InterPro pattern information as predictor attributes.

Moreover, two simple predictor attributes derived directly from the proteins primary sequences, namely sequence length and molecular weight, were added all data sets. As a result, the data sets using amino acid composition information have 20 attributes referring to the percentage composition of the 20 amino acids plus the sequence length and molecular weight of the protein, giving a total of 22 predictor attributes; the data sets using InterPro pattern information have

Table 8.1: Summary of the data sets used in the experiments evaluating *h*Ant-Miner. The first column ('Data Set') gives the data set name, the second column ('Size') gives the data set size, the third column ('Attributes') gives the number of attributes and the forth column ('Classes') gives the number of class labels in the class hierarchy.

Data Set	Size	Attributes	Classes
DS1 AA	147	22	17
DS1 InterPro	147	92	17
DS1 IntAct	147	2096	17
DS2 AA	359	22	17
DS2 InterPro	359	155	17

Boolean attributes referring to the presence/absence of a particular InterPro patters plus the sequence length and molecular weight of the protein; similarly, the data sets using IntAct protein interaction data have Boolean attributes referring to the presence/absence of a particular interaction plus the sequence length and molecular weight of the protein.

8.1.2 Experimental Setup

A summary of the user-defined parameters of *h*Ant-Miner, their descriptions and correspondent values are presented in Table 8.2. We have used default values for these parameters, which are also considered a standard in the literature [94]. This makes the comparison with results of the baseline approach fair, since the latter was used with its default parameter values. Parameter optimisation is a difficult optimisation problem by itself, hence the optimisation of the parameters of the hierarchical classification algorithm is a topic left for future research. For this set of experiments, the parameter values 500 for the *max_number_iterations*, 5 for the *min_examples_per_rule* and 20 for the *colony_size* presented a good trade-off between computational time and predictive accuracy.

The baseline approach used to evaluate *h*Ant-Miner consists of training a flat classification algorithm for each class label individually. The J48 classification

Table 8.2: Summary of the user-defined parameter values used by hAnt-Miner for all data sets. No attempt was made to tune either parameter value for individual data sets. The first column ('Parameter') gives the parameter name, the second column ('Description') gives a short description and the third column ('Value') gives the value used in our experiments.

Parameter	Description	Value
max_uncovered_examples	maximum number of uncovered examples	10
max_number_iterations	maximum number of iterations	500
rule_convergence	number of iterations used to test the rule convergence	10
min_examples_per_rule	minimum number of covered examples per rule	5
colony_size	number of ants per iteration	20

algorithm (Weka [133] implementation of the well-known C4.5 decision tree algorithm [99]) was chosen in this approach. In order to determine the set of positive/negative examples for each classification algorithm run associated with a class label—represented by a GO term—of the class hierarchy, the relationships of the class hierarchy were used as follows. For each classification algorithm run predicting if an example belongs or not to the particular class label (GO term), the set of positive examples consists of all examples that belong to the GO term in question (as the most specific GO term or as an ancestor of their most specific GO term); the set of negative examples consists of all the remaining training examples. Recall that each example can only have one GO term as its most specific term, since the data sets represent hierarchical single-label classification problems—as discussed in subsection 8.1.1.

After training the individual classification algorithms, producing a classification model (classifier) for each class label of the class hierarchy, a test example is classified in a top-down fashion. At the beginning, the example is classified only by the child classifiers of the root GO term. For each child classifier, if the classifier predicts the positive class (the example is predicted to belong to the GO term associated with the classifier), then the example is 'pushed downwards' and classified by its children classifiers. This procedure goes on until a classifier does not predict the GO term (the example is predicted as a negative example) or when a leaf classifier is reached. Note that since we are dealing with a DAG-structured class hierarchy, class labels—and consequently classifiers—can have more than one parent. Therefore, the predictions can be inconsistent with the class hierarchy—e.g. one of the parent classifiers of a given classifier might predict the example as negative, while the other might predict the example as positive. Since the aim of this approach is to have a relatively simple baseline to evaluate hAnt-Miner and not to propose a new top-down classification algorithm capable of coping with DAG-structured class hierarchies, we rely on the evaluation measure (predictive accuracy on the test set) to penalise for such inconsistencies.

The experiments were conducted using a ten-fold cross-validation procedure for each data set.[2] In order to make the comparison of the results fair, we have used the same partitions of training set and test set across the ten-fold procedure for both hAnt-Miner and the baseline approach.

8.1.3 Results and Discussion

We compare the performance of hAnt-Miner and J48 in terms of the hierarchical measures of precision, recall and F-measure, defined by Equation (2.4) and Equation (2.5) in Subsection 2.5.1. The results comparing the hAnt-Miner algorithm against the baseline approach (dubbed 'top-down J48') are shown in Table 8.3. Since the experiments were conducted running a ten-fold cross-validation procedure, the values reported in Table 8.3 are average values with standard deviation (*mean* ± *standard deviation*) computed over the ten different iterations. In addition, hAnt-Miner was run fifteen times for each of the cross-validation folds using a different random seed to initialise the search—given that hAnt-Miner is a stochastic algorithm. The result of these experiments have also been presented in [89].

Overall hAnt-Miner achieved better results than the 'top-down J48' baseline approach in our set of experiments. The baseline approach was significantly outperformed—according to a Student's t-test (see Table 8.3)—in two out of five data sets, namely 'DS1 InterPro' and 'DS1 IntAct'. These data sets can be considered 'difficult' based on their small size (147 proteins) and distribution of examples in the GO hierarchy (GO terms at deeper levels of the hierarchy have few examples). Therefore, the poor performance of the baseline could be the result of the problem that there are many more negative examples than positive examples for

[2] A brief description of the ten-fold cross-validation procedure is presented in Section 6.2.

Table 8.3: Hierarchical measures of precision (hR), recall (hR) and F-measure (hF) values (*mean* ± *standard deviation*) obtained with the ten-fold cross-validation procedure in the five data sets. An entry in the 'hF' column is shown in bold if the hierarchical F-measure value obtained by one of the algorithms was significantly greater than the other algorithm—according to a two-tailed Student's t-test with 99% confidence.

		top-down J48	
	hP	hR	hF
DS1 AA	0.73 ± 0.04	0.55 ± 0.03	0.63 ± 0.03
DS1 InterPro	0.69 ± 0.04	0.68 ± 0.05	0.69 ± 0.04
DS1 IntAct	0.69 ± 0.03	0.37 ± 0.04	0.47 ± 0.03
DS2 AA	0.71 ± 0.02	0.61 ± 0.02	0.65 ± 0.02
DS2 InterPro	0.91 ± 0.01	0.84 ± 0.02	**0.87 ± 0.01**
		*h*Ant-Miner	
	hP	hR	hF
DS1 AA	0.56 ± 0.06	0.55 ± 0.06	0.56 ± 0.06
DS1 InterPro	0.82 ± 0.04	0.81 ± 0.04	**0.81 ± 0.04**
DS1 IntAct	0.77 ± 0.04	0.54 ± 0.03	**0.63 ± 0.03**
DS2 AA	0.63 ± 0.02	0.59 ± 0.02	0.61 ± 0.01
DS2 InterPro	0.83 ± 0.01	0.75 ± 0.01	0.79 ± 0.01

each GO term, particularly at deeper level in the GO hierarchy, which shows that *h*Ant-Miner is more robust than the baseline when dealing with unbalanced hierarchical class distributions. This problem was not observed in the experiments concerning the data sets with a greater number of protein examples, were the baseline significantly outperformed *h*Ant-Miner in the 'DS2 InterPro' data set. In the remaining two data sets, namely 'DS1 AA' and 'DS2 AA', there were no significant difference between the results of both algorithms.

The analysis of the results of *h*Ant-Miner suggests that *h*Ant-Miner's rules are good at classifying a small number of examples. Hence, better results of *h*Ant-Miner are observed in the smaller data sets, namely 'DS1 InterPro' and 'DS1 IntAct'. As discussed in Section 7.2, the rule quality measure employed by *h*Ant-Miner has a bias in favour of accurate (high precision) rules with small coverage (low recall). On data sets with a greater number of examples, this could potentially lead to overfitting—which we believe is the case for the underperformance of *h*Ant-Miner in 'DS2 InterPro'.

8.2 Hierarchical Multi-Label Protein Function Prediction in Yeast

This section reports computational results evaluating the hierarchical multi-label algorithms presented in chapter 7, namely *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$. These algorithms were compared against three decision tree inductions algorithms for hierarchical multi-label classification proposed by Vens et al. [127]: Clus-HMC, which involves inducing a single ('big bang') decision tree that predicts all class labels at once; Clus-HSC, which involves inducing decision trees in a top-down fashion; Clus-SC, which involves the induction a decision tree for each class label individually.

We have selected twenty bioinformatics data sets from Vens et al. [127] involving the prediction of protein functions from the yeast (*Saccharomyces cerevisiae*) model organism.[3] The data sets use two different class hierarchy structures: tree structure (FunCat data sets) and directed acyclic graph structure (Gene Ontology data sets). They were used to evaluate the algorithms with respect to predictive accuracy and model size.

The remainder of this section is organised as follows. Subsection 8.2.1 describes the bioinformatics data sets used in the experiments. Subsection 8.2.2 presents the experimental setup, followed by the computational results and discussion in subsection 8.2.3.

8.2.1 Data Sets

A summary of the twenty data sets used in the experiments are presented on Table 8.4. According to Table 8.4, there are ten different data sets describing different aspects of proteins in the yeast genome. The proteins in these data sets are subsequently associated with corresponding FunCat Categories (FunCat data sets) and Gene Ontology terms (Gene Ontology data sets), resulting in a total of twenty data sets. The variation in the number of examples between the FunCat

[3]The data sets are publicly available at:
`http://www.cs.kuleuven.be/~dtai/clus/hmcdatasets/`

and Gene Ontology versions of a data set is due to the fact that some proteins were not associated with Gene Ontology terms, and therefore, were not including in the Gene Ontology version of the data set. A description of the different aspects of proteins used as predictor attributes in each of the data sets is presented next.

- *pheno*: consists of nominal attributes derived from phenotypic growth experiments. Each attribute can have one of the following values: 'n' (no data), 'w' (no phenotypic effect), 's' (sensitive) and 'r' (resistant).
- *seq*: consists mainly of numeric (integer and real-valued) attributes—with the exception of two discrete attributes—derived from the protein's primary structure. These include amino acid ratios (percentage of each amino acid relative to the sequence length), sequence length, molecular weight, hydrophobicity, information calculated using ExPASy's ProtParam tool [56] and chromosome number from the MIPS' chromosome tables [84].
- *cellcycle, church, derisi, eisen, gasch1, gasch2, spo*: consist of numeric (real-valued) attributes derived from microarray experiments involving gene expression levels.
- *expr*: consists of numeric (real-valued) attributes resulting from the concatenation of the microarray data sets 'cellcycle', 'church', 'derisi', 'eisen', 'gasch1', 'gasch2' and 'spo'.

Table 8.5 presents the average number of class labels per example and the average number of class labels in the class hierarchy of both FunCat and Gene Ontology data sets. As can be seen in Table 8.5, Gene Ontology data sets have a much greater average number of class labels, both in terms of class labels in the class hierarchy and class labels per example.

It should be noted that these data sets present a challenging problem for any classification algorithm. The majority of the data sets contain examples with missing values, several class labels—particularly class labels at deeper levels of the class hierarchy—have a few examples associated (positive examples), the number of class labels to be predicted ranges from 456 to 4134 and each example is associated with more than one class label.

8.2.2 Experimental Setup

Throughout the entire set of experiments, we have used the same set of user-defined parameters values for hmAnt-Miner, hmAnt-Miner$_{PB}$ and cAnt-Miner$_{HM}$.

Table 8.4: Summary of the data sets used in our experiments. The first column ('Data Set') gives the data set name, the second column ('Training Size') gives the number of training examples, the third column ('Test Size') gives the number of test examples, the forth column ('Attributes') gives the number of attributes and the fifth column ('Classes') gives the number of class labels in the class hierarchy.

		FunCat		
Data Set	Training Size	Test Size	Attributes	Classes
cellcycle	2476	1281	77	500
church	2474	1281	27	500
derisi	2450	1275	63	500
eisen	1587	837	79	462
expr	2488	1291	551	500
gasch1	2480	1284	173	500
gasch2	2488	1291	52	500
pheno	1009	582	69	456
seq	2580	1339	478	500
spo	2437	1266	80	500
		Gene Ontology		
Data Set	Training Size	Test Size	Attributes	Classes
cellcycle	2473	1278	77	4126
church	2471	1278	27	4126
derisi	2447	1272	63	4120
eisen	1583	835	79	3574
expr	2485	1288	551	4132
gasch1	2477	1281	173	4126
gasch2	2485	1288	52	4132
pheno	1005	581	69	3128
seq	2568	1332	478	4134
spo	2434	1263	80	4120

Table 8.5: The average number of class labels in the hierarchy and the average number of class labels per example of both FunCat and Gene Ontology data sets.

Average number of class labels	FunCat	Gene Ontology
in the class hierarchy	491.8	3971.8
per example	8.8	35.4

A summary of the parameters, their descriptions and correspondent values are presented in Table 8.6. These parameters values are considered a standard in the literature [94]. The *colony_size* parameter was set to 30 instead of 60, which represents a better trade-off between computational time and predictive accuracy in this set of experiments. Additionally, the *hm*Ant-Miner$_{PB}$ specific parameter *max_number_lists* (marked with a symbol '*' in Table 8.6) was set to 5, which represents a sensible value taking into consideration both the computational time and the potential gain in predictive accuracy.

In the experiments conducted by Vens et a. [127], 2/3 of each data set was used for training and the remaining 1/3 for testing. We have used the same training and testing partitions in our experiments in order to make our results comparable. Since *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ are stochastic algorithms, they are run fifteen times using a different random seed to initialise the search for each data set. In the case of the deterministic Clus-HMC, Clus-HSC and Clus-SC algorithms, we report the results from Vens et al. [127].

8.2.3 Results and Discussion

We compared all algorithms in terms of predictive accuracy using a measure derived from precision-recall (PR) curves, more specifically the area under the averaged PR curve—denoted as AU($\overline{\text{PRC}}$)—discussed in Subsection 2.5.2. Table 8.7 presents the predictive accuracy achieved by *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ across all data sets using FunCat and Gene Ontology, respectively. Each entry in Table 8.7 shows the average value of the AU($\overline{\text{PRC}}$) obtained by fifteen runs of the algorithm, followed by the standard deviation (*average* ± *standard deviation*). Table 8.8 presents the predictive accuracy achieved by Clus-HMC, Clus-HSC and Clus-SC across all data sets using FunCat and Gene Ontology, respectively. Each entry in Table 8.8 shows the AU($\overline{\text{PRC}}$) obtained by a single

Table 8.6: Summary of the user-defined parameter values used by *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ for all data sets. The first column ('Parameter') gives the parameter name, the second column ('Description') gives a short description and the third column ('Value') gives the value used in our experiments. The parameter *max_number_lists* (marked with a symbol '*') is only applied to *hm*Ant-Miner$_{PB}$.

Parameter	Description	Value
max_uncovered_examples	maximum number of uncovered examples	10
max_number_iterations	maximum number of iterations	1500
rule_convergence	number of iterations used to test the rule convergence	10
min_examples_per_rule	minimum number of covered examples per rule	10
colony_size	number of ants per iteration	30
*max_number_lists**	maximum number of lists per iteration	5

run of the algorithm, since they are deterministic algorithms. The last row in both Tables 8.7 and 8.8 indicates the rank-based score—the higher the score, the better the ranking—according to the non-parametric Friedman test [34, 54]; the rank-based scores are illustrated in Figure 8.1.

Table 8.9 presents the induced classification model size—measured as the number of rules discovered—achieved by *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ across all data sets using FunCat and Gene Ontology, respectively. Each entry in Table 8.9 shows the average number of rules obtained by fifteen runs of the algorithm, followed by the standard deviation (*average* ± *standard deviation*). Table 8.10 presents the induced classification model size—measured as the total number of leaf nodes in the decision tree(s), since each path from the root node to a leaf node can be viewed as a rule—achieved by Clus-HMC, Clus-HSC and Clus-SC across all data sets using FunCat and Gene Ontology, respectively. Each entry in Table 8.10 shows the classification model size obtained by a single run of the algorithm, since they are deterministic algorithms. Note that for Clus-HSC and Clus-SC, the total number of leaf nodes corresponds to the number of leaf nodes across all induced decision trees, since these algorithms create more than one decision tree. The last row in both Tables 8.9 and 8.10 indicates the rank-based score—in this case the lower the score, the better the ranking, since smaller classification model size is preferred—according to the non-parametric Friedman test [34, 54]; the rank-based scores are illustrated in Figure 8.2.

In terms of predictive accuracy—measured as the AU($\overline{\text{PRC}}$)—overall *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and Clus-HMC algorithms perform significantly better than the Clus-SC algorithm across all data sets; there are no significant differences between the top three *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and Clus-HMC algorithms—as illustrated in Figure 8.1. In this figure, two rank-based scores are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap. The most accurate ant colony classification algorithm is *hm*Ant-Miner, although there are no significant differences between *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$. On the other hand, *c*Ant-Miner$_{HM}$ performs significantly worse than Clus-HMC and relatively (with large differences) worse than *hm*Ant-Miner.

In terms of the size of the discovered classification model, *hm*Ant-Miner and Clus-HMC algorithms perform equally overall, discovering significantly smaller classification models than the *c*Ant-Miner$_{HM}$, Clus-HSC and Clus-SC algorithms across all data sets—as illustrated in Figure 8.2. In this figure, two rank-based scores are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap. Although the classification models discovered by *hm*Ant-Miner$_{PB}$ are slightly greater than the ones discovered by *hm*Ant-Miner and smaller than the ones discovered by *c*Ant-Miner$_{HM}$ and Clus-HSC in term of size, no significant differences are observed when compared to the latter ones. When compared to Clus-SC, *hm*Ant-Miner$_{PB}$'s classification models are significantly smaller in size. As expected, the classification models of *c*Ant-Miner$_{HM}$, Clus-HSC and Clus-SC are much bigger, since these algorithms build many individual classification models in order to make hierarchical multi-label predictions and the size reflects the total size of all classification models.

The results obtained in our experiments can be summarised as follows. The most accurate algorithms are *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and Clus-HMC, and there are no significant differences in terms of predictive accuracy amongst them. A closer look at a sample of the overall precision-recall curves—consisting of examples of curves from six data sets illustrated in Figure 8.3—reveals that the larger differences in precision occur at lower recall values ($recall < 0.2$), where Clus-HMC achieves the highest predictive accuracy in average, and at higher

Table 8.7: The AU($\overline{\text{PRC}}$) value obtained on the test set by *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ across all data sets used in our experiments. The value of each row represents the average value obtained over fifteen runs of the algorithm, followed by the standard deviation (*average* ± *standard deviation*). The last row of the table indicates the rank-based score—the higher the score, the better the ranking—according to the non-parametric Friedman test [34, 54].

	FunCat		
Data Set	*hm*Ant-Miner AU($\overline{\text{PRC}}$)	*hm*Ant-Miner$_{PB}$ AU($\overline{\text{PRC}}$)	*c*Ant-Miner$_{HM}$ AU($\overline{\text{PRC}}$)
cellcycle	0.154 ± 0.001	0.154 ± 0.001	0.135 ± 0.001
church	0.168 ± 0.001	0.158 ± 0.001	0.139 ± 0.001
derisi	0.161 ± 0.002	0.154 ± 0.001	0.122 ± 0.002
eisen	0.180 ± 0.003	0.184 ± 0.002	0.159 ± 0.001
expr	0.175 ± 0.002	0.178 ± 0.002	0.149 ± 0.002
gasch1	0.175 ± 0.003	0.174 ± 0.002	0.154 ± 0.002
gasch2	0.163 ± 0.002	0.156 ± 0.001	0.133 ± 0.000
pheno	0.162 ± 0.001	0.151 ± 0.001	0.130 ± 0.001
seq	0.181 ± 0.002	0.174 ± 0.002	0.139 ± 0.001
spo	0.174 ± 0.002	0.170 ± 0.002	0.130 ± 0.001
	Gene Ontology		
Data Set	*hm*Ant-Miner AU($\overline{\text{PRC}}$)	*hm*Ant-Miner$_{PB}$ AU($\overline{\text{PRC}}$)	*c*Ant-Miner$_{HM}$ AU($\overline{\text{PRC}}$)
cellcycle	0.332 ± 0.002	0.333 ± 0.003	0.295 ± 0.002
church	0.345 ± 0.002	0.336 ± 0.002	0.278 ± 0.001
derisi	0.334 ± 0.003	0.329 ± 0.003	0.279 ± 0.000
eisen	0.376 ± 0.002	0.365 ± 0.005	0.325 ± 0.000
expr	0.351 ± 0.003	0.353 ± 0.002	0.309 ± 0.001
gasch1	0.356 ± 0.002	0.357 ± 0.002	0.309 ± 0.001
gasch2	0.344 ± 0.002	0.337 ± 0.002	0.289 ± 0.001
pheno	0.337 ± 0.001	0.325 ± 0.003	0.255 ± 0.000
seq	0.366 ± 0.003	0.364 ± 0.001	0.310 ± 0.001
spo	0.341 ± 0.003	0.334 ± 0.003	0.281 ± 0.001
score	4.500	3.900	2.350

Table 8.8: The AU($\overline{\text{PRC}}$) value obtained on the test set by Clus-HMC, Clus-HSC and Clus-SC across all data sets used in our experiments. Recall that these algorithms are deterministic and therefore they are run just once for each data set. The last row of the table indicates the rank-based score—the higher the score, the better the ranking—according to the non-parametric Friedman test [34, 54]. The results in this table are taken from Vens et al. [127].

	FunCat		
Data Set	Clus-HMC AU($\overline{\text{PRC}}$)	Clus-HSC AU($\overline{\text{PRC}}$)	Clus-SC AU($\overline{\text{PRC}}$)
cellcycle	0.172	0.111	0.106
church	0.170	0.131	0.128
derisi	0.175	0.094	0.089
eisen	0.204	0.127	0.132
expr	0.210	0.127	0.123
gasch1	0.205	0.106	0.104
gasch2	0.195	0.121	0.119
pheno	0.160	0.152	0.149
seq	0.211	0.091	0.095
spo	0.186	0.103	0.098
	Gene Ontology		
Data Set	Clus-HMC AU($\overline{\text{PRC}}$)	Clus-HSC AU($\overline{\text{PRC}}$)	Clus-SC AU($\overline{\text{PRC}}$)
cellcycle	0.357	0.371	0.252
church	0.348	0.397	0.289
derisi	0.355	0.349	0.218
eisen	0.380	0.365	0.270
expr	0.368	0.341	0.249
gasch1	0.371	0.351	0.239
gasch2	0.365	0.378	0.267
pheno	0.337	0.416	0.316
seq	0.386	0.282	0.197
spo	0.352	0.371	0.213
score	5.675	3.325	1.250

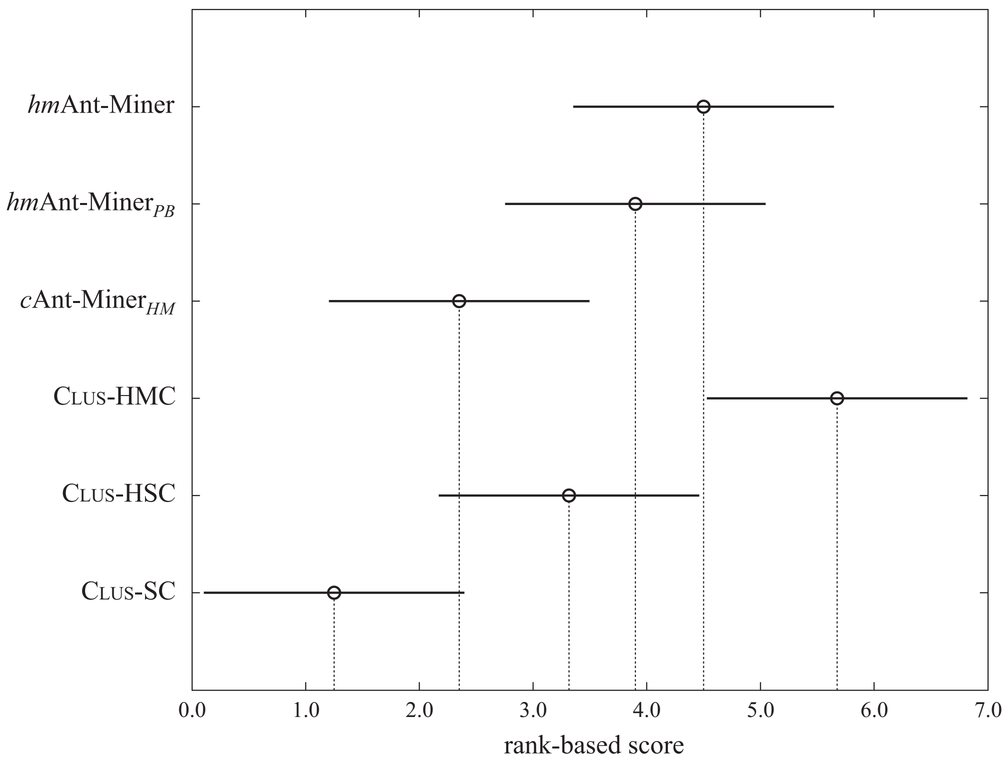


Figure 8.1: Comparison of the predictive accuracy—measured as the area under the averaged PR curve—achieved by the algorithms used in our experiments across all data sets, according to the non-parametric Friedman test with a Scheffé's post-hoc test at the 0.01 significant level [34, 54]. Two rank-based scores—the higher the score, the better the ranking—are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap.

recall values (*recall* > 0.8), where *hm*Ant-Miner achieves the highest predictive accuracy in average. Considering the ant colony classification algorithms, *hm*Ant-Miner achieves the highest predictive accuracy, but the differences in the results of the three algorithms are not statistically significant. In terms of the simplicity (size) of the discovered model, *hm*Ant-Miner is competitive with Clus-HMC and discover statistically significant smaller classification models than *c*Ant-Miner$_{HM}$, Clus-HSC and Clus-SC.

Although the differences are not statistically significant, note that overall *c*Ant-Miner$_{HM}$ overperformed Clus-SC, both in terms of predictive accuracy and size of the total classification model. This is an interesting result, since both algorithms share a similar approach, consisting of transforming the hierarchical multi-label

Table 8.9: The classification model size (number of rules) of *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ across all data sets used in our experiments. The value of each row represents the average model size over fifteen runs of the algorithm, followed by the standard deviation (*average* ± *standard deviation*). The last row of the table indicates the rank-based score—the lower the score, the better the ranking—according to the non-parametric Friedman test [34, 54].

		FunCat	
Data Set	*hm*Ant-Miner Size	*hm*Ant-Miner$_{PB}$ Size	*c*Ant-Miner$_{HM}$ Size
cellcycle	28.667 ± 1.623	32.467 ± 0.951	1934.600 ± 8.970
church	8.200 ± 0.579	32.467 ± 0.951	1934.600 ± 8.970
derisi	19.333 ± 1.661	25.133 ± 0.780	1971.400 ± 6.600
eisen	19.000 ± 0.981	22.200 ± 1.065	1242.600 ± 8.542
expr	30.600 ± 1.466	28.800 ± 0.874	2045.400 ± 17.058
gasch1	24.867 ± 1.701	29.467 ± 0.915	1784.400 ± 4.643
gasch2	32.333 ± 1.517	40.600 ± 1.073	1843.000 ± 3.435
pheno	7.400 ± 0.767	25.000 ± 1.234	1107.400 ± 4.589
seq	20.067 ± 1.152	20.800 ± 0.868	2105.200 ± 11.307
spo	15.800 ± 1.172	22.533 ± 1.199	1749.400 ± 11.750
		Gene Ontology	
Data Set	*hm*Ant-Miner Size	*hm*Ant-Miner$_{PB}$ Size	*c*Ant-Miner$_{HM}$ Size
cellcycle	35.400 ± 1.594	37.600 ± 2.619	7302.500 ± 25.500
church	18.333 ± 1.453	27.600 ± 1.030	8820.500 ± 15.500
derisi	22.533 ± 1.939	32.200 ± 2.634	7136.000 ± 17.000
eisen	18.200 ± 0.823	29.800 ± 0.800	5060.000 ± 0.000
expr	28.600 ± 1.778	30.800 ± 1.068	7431.500 ± 30.500
gasch1	27.933 ± 0.918	31.200 ± 1.393	6854.000 ± 4.000
gasch2	34.200 ± 1.631	40.400 ± 1.030	6976.500 ± 11.500
pheno	7.133 ± 0.792	18.400 ± 2.358	4670.000 ± 9.000
seq	18.067 ± 1.016	20.200 ± 1.319	6659.500 ± 12.500
spo	26.333 ± 2.520	36.400 ± 2.315	6763.000 ± 19.000
score	1.750	2.800	4.050

Table 8.10: The classification model size (number of tree leaves) of Clus-HMC, Clus-HSC and Clus-SC across all data sets used in our experiments. Recall that these algorithms are deterministic and therefore they are run just once for each data set. The last row of the table indicates the rank-based score—the lower the score, the better the ranking—according to the non-parametric Friedman test [34, 54]. The results in this table are taken from Vens et al. [127].

		FunCat	
Data Set	Clus-HMC	Clus-HSC	Clus-SC
	Size	Size	Size
cellcycle	24	4037	9671
church	17	2221	4186
derisi	4	3520	7807
eisen	29	2995	6311
expr	12	4711	10262
gasch1	10	4761	10447
gasch2	26	3756	7850
pheno	8	777	1238
seq	14	4923	10443
spo	6	3623	8527
		Gene Ontology	
Data Set	Clus-HMC	Clus-HSC	Clus-SC
	Size	Size	Size
cellcycle	21	19085	36260
church	7	12368	16049
derisi	10	16693	31175
eisen	37	14384	24844
expr	35	20812	38313
gasch1	30	20070	37838
gasch2	27	18546	34204
pheno	6	5691	6213
seq	15	21703	38969
spo	14	15552	35400
score	1.450	4.950	6.000

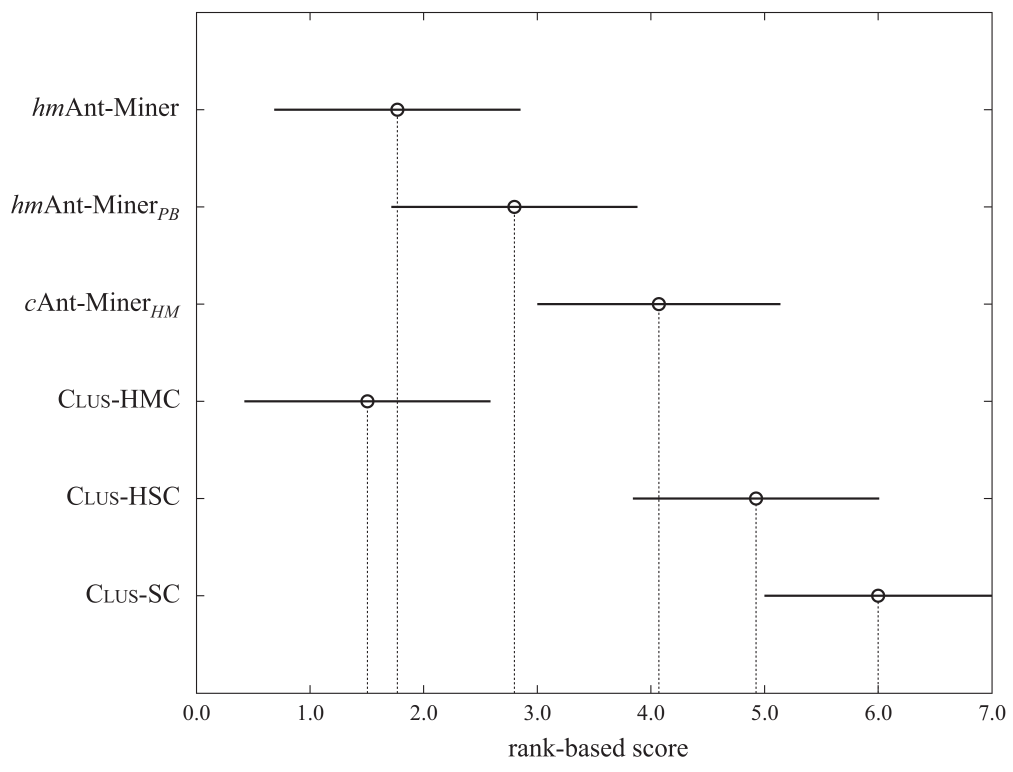


Figure 8.2: Comparison of the size of the classification model discovered by the algorithms used in our experiments across all data sets, according to the non-parametric Friedman test with a Scheffé's post-hoc test at the 0.01 significant level [34, 54]. Two rank-based scores—in this case the lower the score, the better the ranking, since smaller classification model size is preferred—are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap.

classification problem into a set of binary classification problems. The differences on the overall precision-recall curves are illustrated in Figure 8.4, for the same six data sets used in Figure 8.3.

While it was expected that the results achieved by *c*Ant-Miner$_{HM}$, both in terms of predictive accuracy and size of the classification model, would be significantly worse than *hm*Ant-Miner—given that *c*Ant-Miner$_{HM}$ discovers rules for each class label independently—the results achieved by *hm*Ant-Miner$_{PB}$ are disappointing. Recall that *hm*Ant-Miner$_{PB}$ incorporates an extended sequential covering strategy to mitigate the problem of rule interaction that potentially affects *hm*Ant-Miner, where the search of the ACO algorithm is guided by the quality of a list of rules and each iteration of the algorithm builds a complete list of rules instead of a single rule. hmAnt-Miner$_{PB}$ underperformed both in terms of predictive accuracy and size of the classification model when compared to hmAnt-Miner, and although the differences are not significant, a better or at least an equal performance was expected.

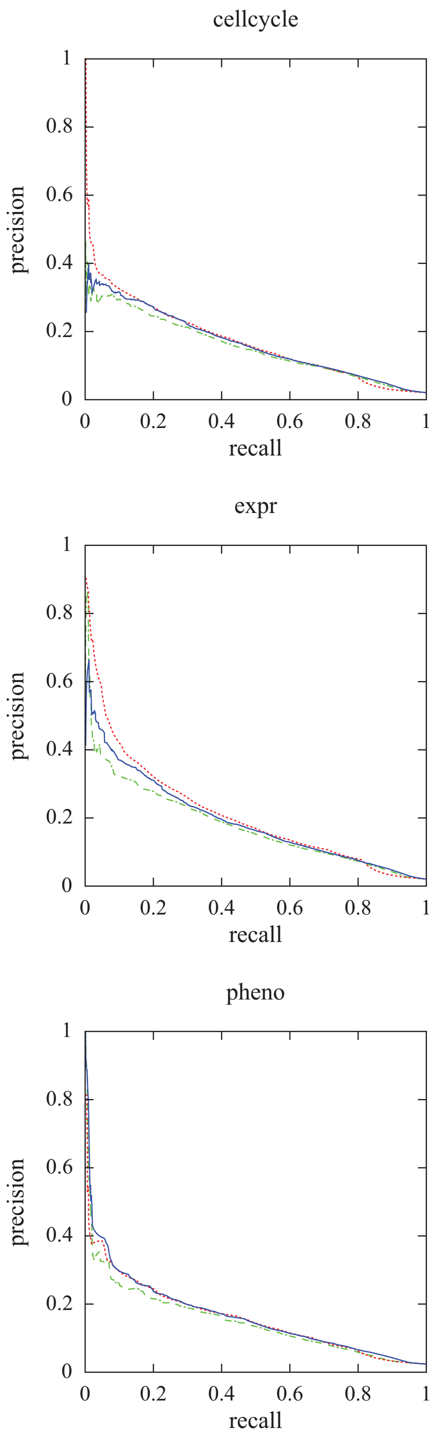
Observing the cases where the predictive accuracy and the size of the classification model of hmAnt-Miner$_{PB}$ differ from the ones of hmAnt-Miner, the poor performance of hmAnt-Miner$_{PB}$ is likely due to overfitting the training data. Overfitting occurs when the discovered classification model is too tailored to the training set, compromising its generalisation ability, which is usually associated with reduced predictive accuracy in the test set. For example, in the 'church' FunCat and Gene Ontology data sets, the size of the classification model of hmAnt-Miner$_{PB}$ is much greater than the one of hmAnt-Miner, specially in the 'church' FunCat data set. At the same time, the predictive accuracy of hmAnt-Miner$_{PB}$ in the test set is lower than the one of hmAnt-Miner. This suggests that the classification model of hmAnt-Miner$_{PB}$—representing the list of rules that achieved the highest predictive accuracy on the training set—is overly complex, consisting of a greater number of rules too tailored to the training set, which do not generalise well in the test set. Similar results are observed in the 'derisi', 'gasch2' and 'pheno' data sets, both FunCat and Gene Ontology.

8.3 Summary

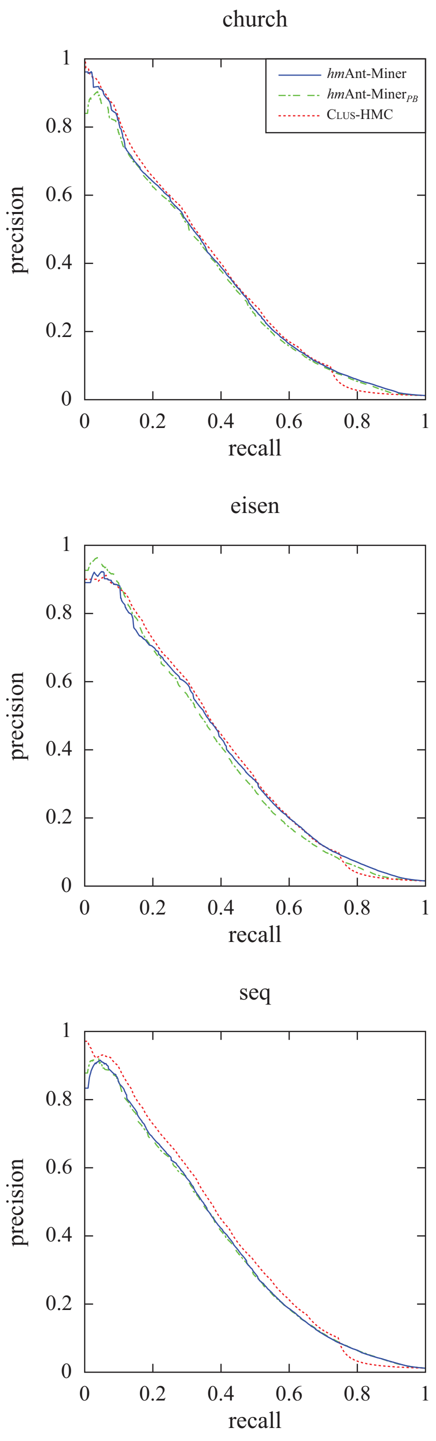
This chapter presented the computational results of the hierarchical and multi-label classification algorithms described in chapter 7 in two sets of experiments. The first set of experiments compared the hAnt-Miner algorithm against a baseline approach in the context of hierarchical prediction of ion channel protein functions using the Gene Ontology functional classification scheme. Although the data sets used in this experiment were relatively small, the results provided useful insights for the improvement of the algorithm—taken into account in the design of the hmAnt-Miner hierarchical multi-label classification algorithm.

The second set of experiments compared hmAnt-Miner, hmAnt-Miner$_{PB}$ and cAnt-Miner$_{HM}$ against state-of-the-art decision tree induction algorithms—namely Clus-HMC, Clus-HSC and Clus-SC—in twenty challenging bioinformatics data sets involving the hierarchical multi-label prediction of protein functions from the yeast (*Saccharomyces cerevisiae*) model organism using both the FunCat (tree-structured class hierarchy) and Gene Ontology (DAG-structured class hierarchy) functional classification schemes. The results have shown that *hm*Ant-Miner and *hm*Ant-Miner$_{PB}$ are competitive both in terms of predictive accuracy and size of the discovered knowledge with Clus-HMC, which corresponds to the highest performing Clus variation.

Overall we regard the results as positive, especially considering that the method of inducing decision trees using predictive clustering trees (PCT)—which is employed by all variations of Clus algorithms—has been evolving for more than one decade, with early applications in [11, 12] and more recently in the context of hierarchical multi-label classification [10, 13, 127]. On the other hand, *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ are the first ACO algorithms tailored for hierarchical multi-label classification—to the best of our knowledge—and even the application of ACO algorithms for flat classification is relatively recent [94].



(a) FunCat



(b) Gene Ontology

Figure 8.3: Overall precision-recall curves of *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and Clus-HMC for: (a) 'cellcycle' 'expr' and 'pheno' FunCat data sets; (b) 'church', 'eisen' and 'seq' Gene Ontology data sets.

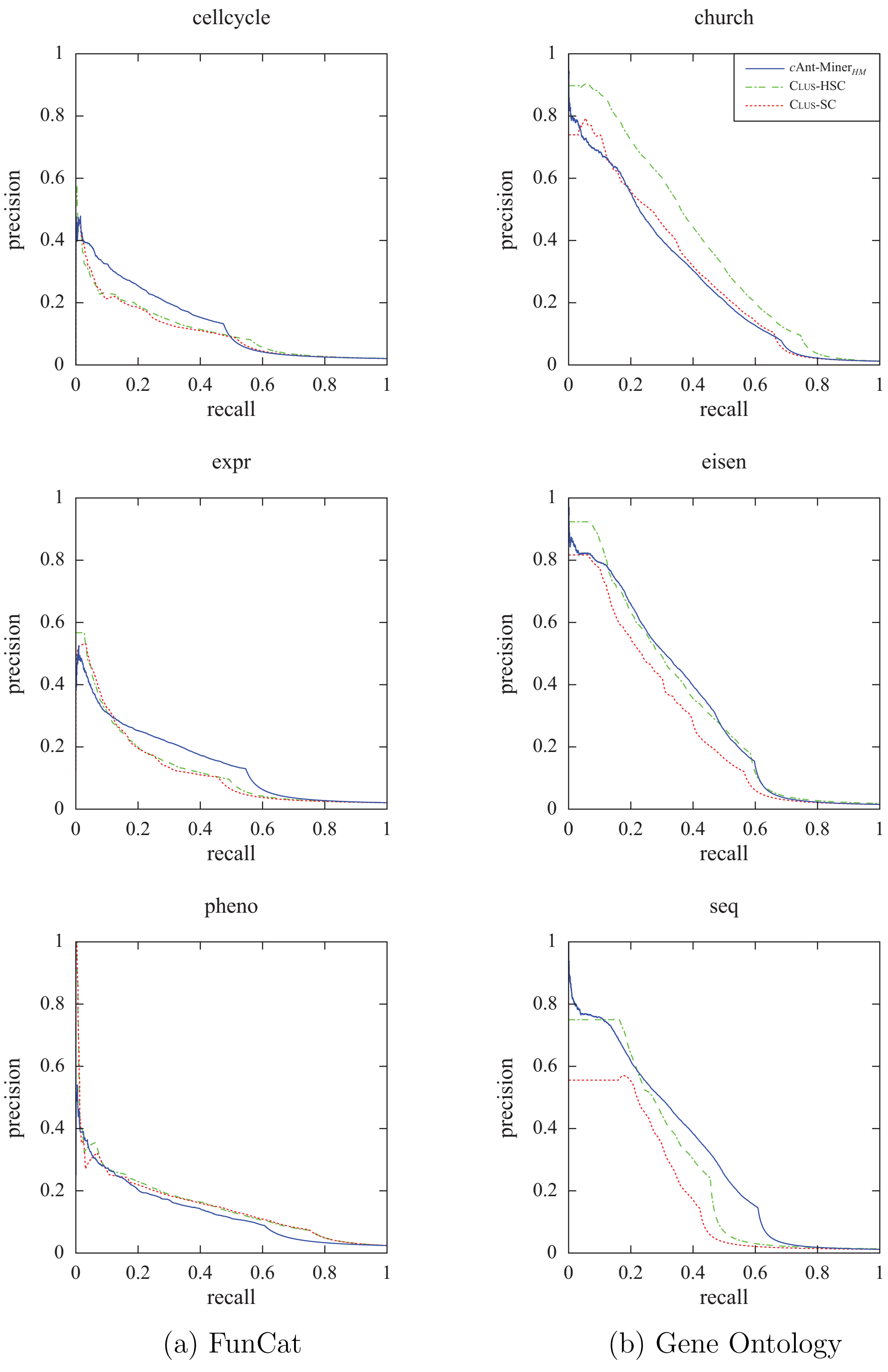


Figure 8.4: Overall precision-recall curves of cAnt-Miner$_{HM}$, Clus-HSC and Clus-SC for: (a) 'cellcycle' 'expr' and 'pheno' FunCat data sets; (b) 'church', 'eisen' and 'seq' Gene Ontology data sets.

Chapter 9

Conclusions and Future Research

The research described in this thesis has presented novel ant colony optimisation (ACO) algorithms in the context of the classification task in data mining. ACO algorithms have been successfully applied to several types of optimisation problems [39, 40]—e.g., scheduling and routing problems—and more recently to the problem of discovering classification rules in data mining [50, 94]. Overcoming current limitations of ACO classification algorithms proposed in the literature, this thesis has focused on two unexplored research areas: (1) extending ACO classification algorithms to cope with continuous attributes directly without the need to discretise them in a preprocessing step; (2) extending ACO classification algorithms to cope with the more complex case of hierarchical multi-label classification.

In the context of hierarchical multi-label classification, this thesis has studied the hierarchical problem of predicting protein functions. Given the large increase in the number of uncharacterised (with unknown function) proteins available for analysis—as a result of advances in high-throughput technologies—determining protein functions is a central goal of bioinformatics and it is crucial to improve current biological knowledge. It is important to emphasise that in this context, comprehensible classification models—i.e. models that can be interpreted and validated by the user—are preferred [51], since the classification model can provide insights about the targeted biological problem.

The ACO classification algorithms proposed in this thesis are aimed at discovering *IF-THEN* classification rules from data, which have the advantage of being a comprehensible knowledge representation. All proposed algorithms were evaluated in experiments comparing them, in terms of predictive accuracy and simplicity (size) of the discovered classification model (list or set of rules), to well-known classification algorithms in the literature.

The remainder of this chapter is organised as follows. Section 9.1 presents a summary of the contributions of this thesis, followed by a discussion of potential avenues for future research in section 9.2.

9.1 Contributions

A summary of the contributions to the ant colony optimisation research area in the context of the classification task of data mining, discussed in chapters 5 to 8, is presented in this section.

A method to extend ACO classification algorithms in order to cope with continuous attributes avoiding the need for a discretisation method in a preprocessing step was presented in chapter 5. The basic idea is to include continuous attributes in the construction graph and employ a dynamic discretisation procedure to create discrete intervals in the rule construction process. A new algorithm named *c*Ant-Miner (Ant-Miner coping with continuous attributes) was presented, incorporating a binary entropy-based discretisation procedure able to create discrete intervals using the '$<$' (less-than) and '$\geq$' (greater-than-or-equal-to) relational operators. Furthermore, a new discretisation procedure employing the MDL principle was proposed, allowing the creation of discrete intervals using lower and upper bound values—e.g., $v_{lower} \leq attribute < v_{upper}$, where *attribute* represents a continuous attribute. It was also proposed that pheromone should be deposited on the edges instead of vertices of the construction graph in order to preserve the order of terms in the antecedent of a rule and consequently preserve the threshold values of continuous attributes. These proposals led to three variations of the *c*Ant-Miner algorithm: *c*Ant-Miner-MDL (*c*Ant-Miner with MDL-based discretisation), *c*Ant-Miner2 (*c*Ant-Miner using pheromones on the edges) and *c*Ant-Miner2-MDL (*c*Ant-Miner2 with MDL-based discretisation).

The motivation for coping with continuous attributes directly—i.e. during the rule construction process—lies in the fact that more information is available to the classification algorithm, since continuous attributes' values have a finer granularity in contrast to a coarser granularity of a fixed number of discrete interval values created by a discretisation method in a preprocessing step. It is expected that the classification algorithm exploits the flexibility of being able to create discrete intervals tailored for the rule under construction, which would be reflected in an increase of the predictive accuracy. Indeed, the results of empirical experiments presented in chapter 6 support this assumption. The experiments show that the dynamic discretisation procedure of continuous attributes incorporated in *c*Ant-Miner and its variations has led to significant improvements in terms of predictive accuracy when compared to Ant-Miner. The *c*Ant-Miner2-MDL achieved the highest predictive accuracy overall, amongst Ant-Miner and *c*Ant-Miner variations. In relation to well-known classification algorithms—namely J48, JRip and Part—the experiments have shown that *c*Ant-Miner and its variations are competitive both in terms of predictive accuracy and simplicity (size) of the discovered classification model (list of rules).

Focusing on hierarchical and hierarchical multi-label classification problems in the context of predicting protein functions, chapter 7 proposed four novel ACO classification algorithms. Section 7.1 presented the *h*Ant-Miner (hierarchical classification Ant-Miner) algorithm, which aims at discovering an ordered list of *IF-THEN* classification rules for hierarchical classification problems. *h*Ant-Miner follows the big bang approach, building a classification model able to predict any class label from the class hierarchy in a single run of the algorithm. In order to cope with hierarchical problems, *h*Ant-Miner employs a second construction graph—dubbed the consequent construction graph—to build the consequent of the rules, as well as hierarchical measures to evaluate the quality of the rules and to compute the heuristic information. The analysis of the computational result comparing *h*Ant-Miner against a baseline algorithm—which consists of training the J48 classification algorithm for each class label of the class hierarchy—has hinted that *h*Ant-Miner's rules are good at classifying a small number of examples, given the bias of the rule quality measure towards accurate (high precision) rules with small coverage (lower recall). This could potentially lead to overfitting on data sets with a greater number of examples.

Section 7.2 discussed the limitations of *h*Ant-Miner in order to propose an extension, named *hm*Ant-Miner (hierarchical multi-label classification Ant-Miner), focusing on coping with hierarchical multi-label data. *hm*Ant-Miner employs heuristic information and rule quality evaluation based on a weighted Euclidean distance, which is a more suitable measure for hierarchical multi-label classification. It also employs a deterministic procedure to compute the consequent of rules, avoiding the need of the second construction graph used to build the consequent of rules by *h*Ant-Miner. Moreover, section 7.3 discussed the problem of rule interaction derived from the sequential covering approach followed by *hm*Ant-Miner, where the discovery of each rule can be seen as an independent search problem for the best rule given the current training examples. Therefore, the sequential covering approach can be seen as a greedy search for a list of rules, which is not guaranteed to be the best list of rules that covers the training set. An extension of the sequential covering to mitigate the problem of rule interaction was proposed, where a complete list of rules is created at each iteration of the algorithm and the quality of the list of rules is used to guide the search. This extension was incorporated in the proposed hmAnt-Miner$_{PB}$ (hierarchical multi-label classification Ant-Miner based on the Pittsburgh approach) algorithm.

A baseline hierarchical multi-label ant colony algorithm, named cAnt-Miner$_{HM}$, was presented in Section 7.4. The cAnt-Miner$_{HM}$ algorithm consists of transforming the hierarchical multi-label classification problem into a set of binary classification problems (binary-class data sets) and applying an adapted version of the flat single-label cAnt-Miner2-MDL classification algorithm in order to predict the presence/absence of a particular class label. The rules discovered by each individual cAnt-Miner2-MDL algorithm are then combined into a global set of rules, which can be used to make hierarchical multi-label predictions. Although the predictions are guaranteed to be consistent with the class hierarchy, the baseline approach has the disadvantages of predicting each class label individually, as discussed in subsection 2.3.1.

It should be noted that, while the proposed algorithms cAnt-Miner and its variations—used to evaluate the method to cope with continuous attributes—incoporated discretisation procedures based on the entropy measure for flat single-label classification problems, the algorithms proposed for hierarchical and hierarchical multi-label have also benefited from being able to cope with continuous attributes directly. More specifically, hAnt-Miner employs a discretisation procedure based on an entropy measure straightforwardly adapted to hierarchical classification problems; hmAnt-Miner and hmAnt-Miner$_{PB}$ employ a distance-based discretisation procedure, based on a weighted Euclidean distance measure; cAnt-Miner$_{HM}$ employs an adapted entropy-based discretisation procedure combined with a class frequency interval selection. This demonstrates the flexibility of the proposed method for handling continuous attributes, where the dynamic discretisation procedure incorporated in the rule construction process can be adapted to suit different types of classification problems.

The computational results concerning the hierarchical and hierarchical multi-label experiments were presented in chapter 8. Section 8.1 described the experiments of hAnt-Miner, involving data sets of ion channel proteins associated with corresponding Gene Ontology terms (DAG-structured class hierarchy). While *h*Ant-Miner experiments were conducted in data sets with a relative small number of training examples and compared against a simple baseline, they contributed to point out the limitations of *h*Ant-Miner, which were taken into account in the design of *hm*Ant-Miner and *hm*Ant-Miner$_{PB}$ algorithms. Sections 8.2 described the more challenging hierarchical multi-label experiments, involving the prediction protein functions from the yeast (*Saccharomyces cerevisiae*) model organism (tree-structured and DAG-structured class hierarchies). In these experiments, the proposed *hm*Ant-Miner, *hm*Ant-Miner$_{PB}$ and *c*Ant-Miner$_{HM}$ algorithms were compared to state-of-the-art decision tree induction algorithms, namely Clus-HMC, Clus-HSC and Clus-SC [127].

The experiments have shown that the hierarchical multi-label classification ant colony algorithms *hm*Ant-Miner and *hm*Ant-Miner$_{PB}$ are competitive both in terms of predictive accuracy and simplicity (size) of the discovered classification model with Clus-HMC and Clus-HSC algorithms—these four algorithms are the most accurate in the set of experiments. The baseline *c*Ant-Miner$_{HM}$ algorithm overperformed the Clus-SC algorithm both in terms of predictive accuracy and simplicity of the discovered classification model. This is interesting, since both algorithms share a similar approach of building a classifier for each class label of the class hierarchy.

The research described in this thesis has presented novel ideas for the handling of continuous attributes during the rule construction process and for extending the application scope of ACO classification algorithms to the more complex case of hierarchical multi-label classification—these are unexplored research areas in the context of ACO classification algorithms to the best of our knowledge, and are therefore our original contributions. It is hoped that the quality of the results obtained will motivate the use of ACO classification algorithms to different hierarchical classification problems and encourage others to improve them.

9.2 Future Research

Following the ideas presented in this thesis, there are several potential directions for future research, covering both conventional (flat single-label) and hierarchical multi-label ACO classification algorithms.

In relation to handling continuous attributes in ACO classification algorithms, a natural direction is to evaluate the incorporation of different discretisation methods. There is a wide range of different discretisation methods available in the literature [41, 81]—e.g., 1R [63] and ChiMerge [72], in addition to the entropy-based methods described in chapter 5—and choosing a suitable discretisation method is highly dependant on the data to be discretised. Therefore, different discretisation methods might help to achieve higher predictive accuracy in different cases. Another aspect to take into account when choosing a discretisation method is its computational time. Given that the dynamic discretisation procedure in the rule construction process is associated with an overhead, more time efficient discretisation methods might be beneficial or even necessary for larger data sets.

Although the results comparing the binary and the minimum description length (MDL) discretisation methods proposed in chapter 5—where the latter is able to produce discrete intervals with lower and upper threshold values—did not show statistically significant differences, it would be interesting to investigate a variation of the MDL-based discretisation method for hierarchical multi-label classification. Note that in order to apply a variation of the MDL criterion to hierarchical multi-label classification problems, the equation of the MDL criterion has to be adapted, given that in many cases the number of class labels—used as an exponent in the equation of the MDL criterion—is in the order of thousands in these problems.

Another interesting direction is to evaluate different pheromone update policies, hinted by the fact that depositing pheromone on the edges instead of vertices of the construction graph led to improvements of predictive accuracy when dealing with data sets comprising continuous attributes. As suggested by Stützle and Hoos [115, 116] in their $\mathcal{MAX}$-$\mathcal{MIN}$ ant system, the use of lower and upper pheromone limits are effective in increasing the exploration of the search space, which is particularly important for large problems. Therefore, the incorporation of a $\mathcal{MAX}$-$\mathcal{MIN}$-based pheromone update policy could prove beneficial for the algorithms proposed in this thesis.

Recall that both *hm*Ant-Miner and *hm*Ant-Miner$_{PB}$ algorithms incorporate a deterministic procedure to compute the consequent of a rule. Different procedures for determining the consequent of a rule and the employment of a pruning procedure on the consequent of rules could potentially improve the predictive accuracy. The rule quality measure plays an important role in the search of the ACO algorithm, since the pheromone is updated based on the quality of the rule. The evaluation of different rule quality measures is also a direction worth further exploration.

After identifying the problem of rule interaction and proposing a Pittsburgh-based extended sequential covering approach in section 7.3, the hmAnt-Miner$_{PB}$ algorithm, no statistically significant differences compared to hmAnt-Miner were observed in the hierarchical multi-label experiments. While the search in hmAnt-Miner$_{PB}$ is guided by the quality of a complete list of rules, the rules are used to update pheromone trails on the construction graph. However, there is no mechanism to maintain the rules and the order that they are created in an iteration. This might affect the convergence of the algorithm to better regions of the search space, which consequently affects the overall performance of the algorithm. Following the ideas presented in section 7.3, it would be interesting to evaluate a more sophisticated Pittsburgh-based approach addressing the aforementioned limitations, for both conventional and hierarchical multi-label classification problems. An approach to mitigate the problem of overfitting observed in the experiments of hmAnt-Miner$_{PB}$ in chapter 8 is also an interesting research direction.

Finally, as another direction for future research, it would be interesting to investigate the discovery of a set of rules (unordered rules) instead of a list of rules (ordered rules). As discussed in subsection 2.2.2, rules in a list of rules must be interpreted in order; therefore, the context of the rule—i.e. the previous rules in the list—must be taken into consideration to understand a rule. On the other hand, a set of rules provides the advantage that each rule can be interpreted independently of the others in a modular fashion, contributing to the comprehensibility of the discovery knowledge. Although the discovery of a set of rules has been previously investigated in the literature [108] in the context of conventional classification problems, the application to hierarchical multi-label problems remains unexplored. Recall that the baseline cAnt-Miner$_{HM}$ algorithm for hierarchical multi-label classification discovers a set of rules; nevertheless, the size of the set of rules—in the order of thousands of rules—undermines the comprehensibility of the knowledge discovered by cAnt-Miner$_{HM}$. Therefore, extensions of hmAnt-Miner and hmAnt-Miner$_{PB}$ algorithms for discovering a set of rules might provide more interesting results.

References

[1] GPCRDB – Information system for G protein-coupled receptors. http://www.gpcr.org/7tm/ (accessed on 15/11/2009).

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *The Molecular Biology of the Cell.* Garland Press, 4th edition, 2002.

[3] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[4] M. Ashburner, C.A. Ball, J.A. Blake, D.Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A. Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G.M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[5] A. Asuncion and D.J. Newman. UCI Machine Learning Repository. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 2007.

[6] T.K. Attwood, A. Mitchell, A. Gaulton, G. Moulton, and L. Tabernero. The prints protein fingerprint database: functional and evolutionary applications. In M. Dunn, L. Jorde, P. Little, and A. Subramaniam, editors, *Encyclopaedia of Genetics, Genomics, Proteomics and Bioinformatics.* John Wiley & Sons, 2006.

[7] B. Bakker and T. Heskes. Task clustering for learning to learn. In *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence*, pages 33–40, 2001.

[8] Zafer Barutcuoglu, Robert E. Schapire, and Olga G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

[9] Ran Bi, Yanhong Zhou, Feng Lu, and Weiqiang Wang. Predicting Gene Ontology functions based on support vector machines and statistical significance estimation. *Neurocomputing*, 70:718–725, 2007.

[10] H. Blockeel, M. Bruynooghe, S. Džeroski, J. Ramon, and J. Struyf. Hierarchical multi-classification. In Sašo Džeroski, Luc De Raedt, and Stefan Wrobel, editors, *Proceedings of the First SIGKDD Workshop on Multi-Relational Data Mining (MRDM 2002)*, pages 21–35. University of Alberta, Edmonton, Canada, 2002.

[11] H. Blockeel, S. Džeroski, and J. Grbović. Simultaneous Prediction of Multiple Chemical Parameters of River Water Quality with TILDE. In *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, pages 32–40. Springer, 1999.

[12] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning*, pages 55–63. ACM, 1998.

[13] H. Blockeel, L. Schietgat, J. Struyf, S. Džeroski, and A. Clare. Decision Trees for Hierarchical Multilabel Classification: A Case Study in Functional Genomics. In *PKDD-2006, LNAI 4213*, pages 18–29, 2006.

[14] C. Blum and K. Socha. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In *CD-ROM Proceedings of Hybrid Intelligent Systems Conference (HIS-2005)*, 2005.

[15] J.R. Bock and D.A. Gough. Predicting protein-protein interactions from primary structure. *Bioinformatics*, 17(5):455–460, 2001.

[16] J.R. Bock and D.A. Gough. In Silico Biological Function Attribution: a different perspective. *BioSilico*, 2(1):30–37, 2004.

[17] L.B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment.* PhD thesis, University of Michigan, 1982.

[18] L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, pages 235–282, 1989.

[19] P. Bork and E.V. Koonin. Predicting functions from protein sequences–where are the bottlenecks? *Nature Genetics*, 18:313–318, 1998.

[20] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[21] N. Cesa-Bianchi, L. Zaniboni, and M. Collins. Incremental algorithms for hierarchical classification. In *Journal of Machine Learning Research*, pages 31–54. MIT Press, 2004.

[22] A. Chan and A.A. Freitas. A new classification-rule pruning procedure for an ant colony algorithm. In *Artificial Evolution (Proceedings of the EA-2005)*, Lecture Notes in Artificial Intelligence 3871, pages 25–36, 2005.

[23] A. Chan and 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)*, pages 27–34, 2006.

[24] A. Clare, A. Karwath, H. Ougham, and R. King. Functional bioinformatics for *Arabidopsis thailana*. *Bioinformatics*, 22(9):1130–1136, 2006.

[25] A. Clare and R. King. Knowledge discovery in multi-label phenotype data. In *ECML/PKDD*, pages 42–53, 2001.

[26] A. Clare and R. King. Predicting gene function in saccharomyces cerevisiae. *Bioinformatics*, 19(2):ii42–ii49, 2003.

[27] W.W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[28] The UniProt Consortium. The universal protein resource (uniprot). *Nucleic Acid Research*, 37:D169–D174, 2009.

[29] E.P. Costa, A.C. Lorena, A.C.P.L.F. Carvalho, and A.A. Freitas. A review of performance evaluation measures for hierarchical classifiers. In *Evaluation Methods for Machine Learning II: papers from the 2007 AAAI Workshop*, pages 1–6, 2007.

[30] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.

[31] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 2nd edition, 2006.

[32] M. Daud and D. Corne. Human Readable Rule Induction In Medical Data Mining: A Survey Of Existing Algorithms. In *Proc. of WSEAS European Computing Conference*, 2007.

[33] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *ICML '06: Proceedings of the 23rd International Conference on Machine learning*, pages 233–240. ACM, 2006.

[34] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research*, 7:1–30, 2006.

[35] M. Deng, F. Sun, and T. Chen. Assessment of the reliability of protein-protein interactions and protein function prediction. In *Eighth Pacific Symposium on Biocomputing*, pages 140–151, 2003.

[36] J. Diederich, editor. *Rule Extraction from Support Vector Machines*, volume 80 of *Studies in Computational Intelligence*. Springer, 2008.

[37] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32, 1999.

[38] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[39] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[40] M. Dorigo and T. Stützle. Ant Colony Optimization: Overview and Recent Advances. Technical Report TR/IRIDIA/2009-013, IRIDIA, Université Libre de Bruxelles, http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2009-013r001.pdf, 2009.

[41] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning: Proceedings of the Twelfth Int. Conference on Artificial Intelligence*, pages 194–202. Morgan Kauffmann, 1995.

[42] D.A. Engelman, T.A. Steitz, and A. Goldman. Identifying non-polar transbilayer helices in amino acid sequences of membrane proteins. *Annual Review of Biophysics and Biophysical Chemistry*, 15:321–353, 1986.

[43] U. Fayyad and K. Irani. On the Handling of Continuous-Valued Attributes in Decision Tree Generation. *Machine Learning*, 8:87–102, 1992.

[44] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Thirteenth International Joint Conference on Artifical Inteligence*, pages 1022–1027. Morgan Kaufmann, 1993.

[45] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smith. From data mining to knowledge discovery: an overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery & Data Mining*, pages 1–34. MIT Press, 1996.

[46] R.D. Finn, J. Tate, J. Mistry, P.C. Coggill, S.J. Sammut, H.R. Hotz, G. Ceric, K. Forslund, S.R. Eddy, E.L.L. Sonnhammer, and A.ateman. The pfam protein families database. *Nucleic Acids Research*, 36:D281–D288, 2008.

[47] E. Frank and I.H. Witten. Generating Accurate Rule Sets Without Global Optimization. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann, 1998.

[48] A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002.

[49] A.A. Freitas and A.C.P.L.F. de Carvalho. A tutorial on hierarchical classification with applications in bioinformatics. In D. Taniar, editor, *Research and Trends in Data Mining Technologies and Applications*, pages 175–208. Idea Group, 2007.

[50] A.A. Freitas, R.S. Parpinelli, and H.S. Lopes. Ant colony algorithms for data classification. In *Encyclopedia of Information Science and Technology*, volume 1, pages 154–159. 2nd edition, 2008.

[51] A.A. Freitas, D.C. Wieser, and R. Apweiler. On the Importance of Comprehensible Classification Models for Protein Function Prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(1):172–182, 2010.

[52] I. Friedberg. Automated protein function prediction—the genomic challenge. *Briefings in Bioinformatics*, 7(3):225–242, 2006.

[53] M. Galea and Q. Shen. Simultaneous ant colony optimization algorithms for learning linguistic fuzzy rules. In A. Agraham, C. Grosan, and V. Ramos, editors, *Swarm Intelligence in Data Mining*, pages 75–99. Springer-Verlag, 2006.

[54] S. García and F. Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Machine Learning Research*, 9:2677–2694, 2008.

[55] R. Garian. Prediction of quaternary structure from primary structure. *Bioinformatics*, 17(6):551–556, 2001.

[56] E. Gasteiger, A. Gattiker, C. Hoogland, I. Ivanyi, R.D. Appel, and A. Bairoch. ExPASy: the proteomics server for in-depth protein knowledge and analysis. *Nucleic Acid Research*, 31:3784–3788, 2003.

[57] J.A. Gerlt and P.C. Babbitt. Can sequence determine function? *Genome Biology*, 1(5):1–10, 2000.

[58] A. González, R. Pérez, and J.L. Verdegay. Learning the structure of a fuzzy rule: a genetic approach. *Fuzzy System and Artificial Intelligence*, 3:57–70, 1994.

[59] H. Hermjakob, L. Montecchi-Palazzi, C. Lewington, S. Mudali, S. Kerrien, S. Orchard, M. Vingron, B. Roechert, P. Roepstorff, A. Valencia, H. Margalit, J. Armstrong, A. Bairoch, G. Cesareni, D. Sherman, and R. Apweiler. Intact: an open source molecular interaction database. *Nucleic Acid Research*, 32:D452–D455, 2004.

[60] P.G. Higgs and T. Attwood. *Bioinformatics and Molecular Evolution.* Blackwell Publishing, 2005.

[61] C.A.R. Hoare. Quicksort. *Computer Journal*, 5(1):10–16, 1962.

[62] N. Holden and A.A. Freitas. Improving the performance of hierarchical classification with swarm intelligence. In *Proceedings of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio 2008), LNCS 973*, pages 48–60, 2008.

[63] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–90, 1993.

[64] W.-C. Hong, Y.-F. Chen, P.-W. Chen, and Y.-H. Yeh. Continuous ant colony optimization algorithms in a support vector regression based financial forecasting model. In *Third International Conference on Natural Computation (ICNC 2007)*, pages 548–552, 2007.

[65] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. Langendijk-Genevaux, and M. Pagniand C. Sigrist. The prosite database. *Nucleic Acid Research*, 34:D227–D230, 2006.

[66] I. Iliopoulos, S. Tsoka, M.A Andrade, A.J. Enright, M. Carroll, P. Poullet, V. Promponas, T. Liakopoulos, G. Palaios, C. Pasquier, S. Hamodrakas, J. Tamames, A.T. Yagnik, A. Tramontano, D. Devos, C. Blaschke, A. Valencia, D. Brett, D. Martin, C. Leroy, I. Rigoutsos, C. Sander, and C.A. Ouzounis. valuation of annotation strategies using an entire genome sequence. *Bioinformatics*, 19(6):717–726, 2003.

[67] H. Jacobsson. Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review. *Neural Computation*, 17(6):1223–1263, 2005.

[68] N. Japkowicz and S. Stephen. The class imbalance problem: a systematic study. *Intelligent Data Analysis*, 6:429–450, 2002.

[69] L.J. Jensen, R. Gupta, N. Blom, D. Devos, J. Tamames, C. Kesmir, H. Nielsen, H.H. Stærfeldt, K. Rapacki, C. Workman, C.A. Andersen, S. Knudsen, A. Krogh, A. Valencia, and S. Brunak. Prediction of human protein function from post-translational modifications and localization features. *Journal of Molecular Biology*, 319:1257–1265, 2002.

[70] L.J. Jensen, R. Gupta, H.H. Stærfeldt, and S. Brunak. Prediction of human protein function according to gene ontology categories. *Bioinformatics*, 19(5):635–642, 2003.

[71] S. Kawashima and M. Kanehisa. Aaindex: amino acid index database. *Nucleic Acids Res*, 28:374, 2000.

[72] R. Kerber. Chimerge: Discretization of Numeric Attributes. In *Tenth National Conference on Artificial Inteligence (AAAI-92)*, pages 123–128, 1992.

[73] S. Kiritchenko, S. Matwin, and A.F.Famili. Functional annotation of genes using hierarchical text categorization. In *BioLINK SIG: Linking Literature, Information and Knowledge for Biology*, 2005.

[74] R. Kohavi and M. Sahami. Error-based and entropy-based discretization of continuous features. In *Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining*, pages 114–119. AAAI Press, 1996.

[75] A. Kumar, K. Cheung, P. Ross-Macdonald, P. Coelho, P. Miller, and M. Snyder. Triples: a database of gene function in s. cerevisiae. *Nucleic Acid Research*, 28:81–84, 2000.

[76] J. Kyte and R.F. Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology*, 157:105–132, 1982.

[77] A. Lægreid, T.R. Hvidsten, H. Midelfart, J. Komorowski, and A.K. Sandvik. Predicting gene ontology biological process from temporal gene expression patterns. *Genome Research*, 13(5):965–979, 2003.

[78] A.M. Lesk. *Introduction to Bioinformatics.* Oxford University Press, 3rd edition, 2008.

[79] S. Letovsky and S. Kasif. Predicting protein function from protein/protein interaction data: a probabilistic approach. *Bioinformatics*, 19(1):i97–i204, 2003.

[80] D.J. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[81] H. Liu, F. Hussain, C.L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6:393–423, 2002.

[82] C. Manning and H. Schtze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[83] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens. Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11(5):651–665, 2007.

[84] H.W. Mewes, K. Heumann, A. Kaps, K. Mayer, F. Pfeiffer, S. Stocker, and D. Frishman. Mips: a database for genomes and protein sequences. *Nucleic Acid Research*, 27:44–48, 1999.

[85] T. Mitchell. *Machine Learning.* McGraw Hill, 1st edition, 1997.

[86] N.J. Mulder, R. Apweiler, T.K. Attwood, A. Bairoch, A. Bateman, D. Binns, P. Bork, V. Buillard, L. Cerutti, R. Copley, E. Courcelle, U. Das, L. Daugherty, M. Dibley, R. Finn, W. Fleischmann, J. Gough, D. Haft, N. Hulo, S. Hunter, D. Kahn, A. Kanapin, A. Kejariwal, A. Labarga, P.S. Langendijk-Genevaux, D. Lonsdale, R. Lopez, I. Letunic, M. Madera, J. Maslen, C. McAnulla, J. McDowall, J. Mistry, A. Mitchell, A.N. Nikolskaya, S. Orchard, C. Orengo, R. Petryszak, J.D. Selengut, C.J.A. Sigrist, P.D. Thomas, F.Valentin, D.Wilson, C.H. Wu, and C. Yeats. New developments in the interpro database. *Nucleic Acid Research*, 35:D224–D228, 2007.

[87] L. Oliveira, A.C.M. Paiva, and G. Vriend. A common motif in g-protein-coupled seven transmembrane helix receptors. *Journal of Computer-Aided Molecular Design*, 7(6):649–658, 1993.

[88] S. Oliver. A network approach to the systematic analysis of yeast gene function. *Trends in Genetics*, 12(7):241–242, 1996.

[89] F.E.B. Otero, A.A. Freitas, and C.G.Johnson. A Hierarchical Classification Ant Colony Algorithm for Predicting Gene Ontology Terms. In *Proceedings of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio 2009), LNCS 5483*, pages 68–79. Springer-Verlag, 2009.

[90] F.E.B. Otero, A.A. Freitas, and C.G. Johnson. *c*Ant-Miner: an ant colony classification algorithm to cope with continuous attributes. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A.F.T. Winfield, editors, *Proceedings of the 6th International Conference on Swarm Intelligence (ANTS 2008), Lecture Notes in Computer Science 5217*, pages 48–59. Springer-Verlag, 2008.

[91] F.E.B. Otero, A.A. Freitas, and C.G. Johnson. Handling continuous attributes in ant colony classification algorithms. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, pages 225–231. IEEE, 2009.

[92] G.L. Pappa, A.J. Baines, and A.A. Freitas. Predicting post-synaptic activity in proteins with data mining. *Bioinformatics*, 21(2):ii19–ii25, 2005.

[93] R.S. Parpinelli, H.S. Lopes, and A.A. Freitas. An ant colony algorithm for classification rule discovery. In H. Abbass, R. Sarker, and C. Newton,

editors, *Data Mining: a Heuristic Approach*, pages 191–208. Idea Group Publishing, 2002.

[94] R.S. Parpinelli, H.S. Lopes, and A.A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

[95] M. Pellegrini, E.M. Marcotte, M.J. Thompson, D. Eisenberg, and T.O. Yeates. Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. In *National Academy of Sciences of the United States of America*, volume 96, pages 4285–4288, 1999.

[96] G. Piatetsky-Shapiro and W. Frawley. *Knowledge Discovery in Databases.* AAAI Press, 1991.

[97] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[98] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[99] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[100] J.R. Quinlan. Improved Use of Continuous Attributes in C4.5. *Artificial Intelligence Research*, 7:77–90, 1996.

[101] V. Raghavan, P. Bollmann, and G.S. Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7(3):205–229, 1989.

[102] S. Raychaudhuri. *Computational Text Analysis for Functional Genomics and Bioinformatics.* Oxford University Press, 2006.

[103] R.L. Rivest. Learning Decision Lists. *Machine Learning*, 2(3):229–246, 1987.

[104] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-Based Learning of Hierarchical Multilabel Classification Models. In *Journal of Machine Learning Research*, pages 1601–1626. MIT Press, 2006.

[105] A. Ruepp, A. Zollner, D. Maier, K. Albermann, J. Hani, M. Mokrejs, I. Tetko, U. Guldener, G. Mannhaupt, M. Munsterkotter, and HW. Mewes. The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic Acid Research*, 32(18):5539–5545, 2004.

[106] R.E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[107] A. Secker, M.N. Davies, A.A. Freitas, J. Timmis, M. Mendao, and D. Flower. An experimental comparison of classification algorithms for the hierarchical prediction of protein function. In *Third UK Knowledge Discovery and Data Mining Symposium (UKKDD-2007)*, pages 13–18, 2007.

[108] J. Smaldon and A.A. Freitas. A new version of the ant-miner algorithm discovering unordered rule sets. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 43–50, 2006.

[109] S.F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, 1980.

[110] S.F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the 8th International Conference on Artificial Intelligence*, pages 422–425. Morgan Kaufmann, 1983.

[111] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[112] K. Socha. Aco for continuous and mixed-variable optimization. In M. Dorigo, L. Gambardella, F. Mondada, T. Stützle, M. Birratari, and C. Blum, editors, *Proceedings of the 4th International Conference on Swarm Intelligence (ANTS 2004), Lecture Notes in Computer Science 3172*, pages 25–36, 2004.

[113] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operations Research*, 185(3):1155–1173, 2008.

[114] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45:427–437, 2009.

[115] T. Stützle and H.H. Hoos. Improvements on the Ant System: Introducing $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. In *Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms*, 1997.

[116] T. Stützle and H.H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[117] A. Sun and E.-P. Lim. Hierarchical Text Classification and Evaluation. In *Proceedings of the 1th IEEE International Conference on Data Mining*, pages 521–528. IEEE Press, 2001.

[118] A. Sun, E.-P. Lim, and W.-K. Ng. Hierarchical text classification methods and their specification. *Cooperative Internet Computing*, pages 236–256, 2003.

[119] A. Sun, E.-P. Lim, and W.-K. Ng. Performance Measurement Framework for Hierarchical Text. *Journal of the American Society for Information Science and Technology*, 54:1014–1028, 2003.

[120] A. Sun, E.-P. Lim, W.-K. Ng, and J. Srivastava. Blocking reduction strategies in hierarchical text classification. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1305–1308, 2004.

[121] S. Swaminathan. Rule Induction Using Ant Colony Optimization for Mixed Variable Attributes. Master's thesis, Texas Tech University, 2006.

[122] B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov Networks. In *Neural Information Processing Systems Conference (NIPS03)*, 2003.

[123] W. Tian and J. Skolnick. How well is enzyme function conserved as a function of pairwise sequence identity? *Journal of Molecular Biology*, 333(4):863–882, 2003.

[124] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104, 2004.

[125] G. Tsoumakas and I. Katakis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.

[126] S. Tsutsui. Ant colony optimisation for continuous domains with aggregation pheromones metaphor. In *Proceedings of the 5th International Conference on Recent Advances in Soft Computing (RASC-04)*, pages 207–212, 2004.

[127] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[128] G. Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. *Machine Learning: ECML-93*, pages 280–296, 1993.

[129] K. Wang, S. Zhou, and S.C. Liew. Building Hierarchical Classifiers Using Class Proximity. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 363–374. Morgan Kaufmann, 1999.

[130] E. Webb. *Enzyme Nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology.* Academic Press, 1992.

[131] W. Weinert and H. Lopes. Neural networks for protein classification. *Applied Bioinformatics*, 3(1):41–48, 2004.

[132] J.C. Whisstock and A.M. Lesk. Prediction of protein function from protein sequence and structure. *Q Rev Biophys*, 36(3):307–340, 2003.

[133] H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2nd edition, 2005.

[134] I. Xenarios, D.W. Rice, L. Salwinski, M.K. Baron, E.M. Marcotte, and D. Eisenberg. Dip: The database of interacting proteins. *Nucleic Acid Research*, 28:289–291, 2000.

[135] H. Xie, A. Wasserman, Z. Levine, A. Novik, V. Grebinskiy, A. Shoshan, and L. Mintz. Large-scale protein annotation through gene ontology. *Genome Research*, 12:785–794, 2002.

[136] M.-L. Zhang and Z.-H. Zhou. A k-nearest Neighbor Based Algorithm for Multi-label Classification. In *Proceedings of the IEEE International Conference on Granular Computing*, volume 2, pages 718–721, 2005.

[137] X.-M. Zhao, L. Chen, and K. Aihara. Protein function prediction with high-throughput data. *Amino Acids*, 35:517–530, 2008.

Appendix A

Software Availability

All the algorithms presented in the thesis have been made publicly available under the GNU Lesser General Public License (LGPL) as an open source software package, named myra, hosted in the SourceForge.net repository. Myra is a cross-platform ant colony optimisation (ACO) framework written in Java, providing a specialised data mining module to support the application of ACO to classification problems. The source code, binary code and documentation can be downloaded from `http://sourceforge.net/projects/myra/`.