2-1-2011

Extensions to the ant-miner classification rule discovery algorithm

Khalid Magdy Salama

The American University in Cairo

School of Sciences and Engineering

EXTENSIONS TO THE ANT-MINER CLASSIFICATION RULE DISCOVERY ALGORITHM

A Thesis Submitted to

Department of Computer Science and Engineering

In partial fulfillment of the requirements for

the degree of Master of Science

By

Khalid Magdy Nagib Salama

B.Sc., Computer Science

The American University in Cairo

Under the supervision of

Prof. Ashraf Abdebar, Prof. Ahmed Rafea and Prof. Awad Khalil

November 2010

The American University in Cairo

School of Sciences and Engineering

EXTENSIONS TO THE ANT-MINER CLASSIFICATION RULE DISCOVERY ALGORITHM

A Thesis Submitted to

Department of Computer Science and Engineering

in partial fulfillment of the requirements for

the degree of Master of Science

has been approved by

Dr. . . . . .

Thesis Committee Chair / Adviser _____

Affiliation _____

Dr. . . . . .

Thesis Committee Chair / Adviser _____

Affiliation _____

Dr. . . . . .

Thesis Committee Reader / examiner _____

Affiliation _____

Dr. . . . . .

Thesis Committee Reader / examiner _____

Affiliation _____

Dr. . . . . .

Thesis Committee Reader / examiner _____

Affiliation _____

_____	_____	_____	_____
Department Chair/	Date	Dean	Date

ACKNOWLEDGMENTS

First, I thank god for enabling me to complete this thesis along with my work. It was hard to balance between working and studying. However, with god's help, I did it.

I am deeply indebted to my supervisor Prof. Ashraf Abdelbar who was very encouraging in the research and tried to pull the best out of me. His ideas and advice were great help for me. I hope we would be always together in the research.

I would like to express my gratitude to my parents, who were very supportive to me in to complete this work. Thanks to my mother for her love and care and praying for me. Thanks to me father for his support and encouragements and pushing me to the limits.

Special thanks to my college and friend, Ismail, who helped me to improve the English style and grammar for the thesis. Another special thanks to my Syrian friend, Ramiz, who helped me with his experience and advice.

I dedicate this thesis to my supervisor, family and my friends, Ismail, Ramiz, Yosri, Essam, Bassem and Risha.

ABSTRACT

Ant Colony Optimization (ACO) is a subfield of swarm intelligence which studies algorithms inspired by the observation of the behavior of biological ant colonies. It has been proposd by M. Dorigo and colleagues [8 – 9] as a meta-heuristic for solving combinatorial optimization problems. Ant-Miner is an application of ACO in data mining. It has been introduced by Parpinelli *et al.* [20] in 2002 as an ant-based algorithm for the discovery of classification rules. The classification rules are generated in the following form:

IF <Conditions> THEN <class>

The <conditions> part (antecedent) of the rule contains a logical combination of predictor attributes, in the form: term1 AND term2 AND... . Each term is in the form of <attribute = value>, where value belongs to the domain of attribute. Ant-Miner has proved to be a very promising technique for classification rules discovery. Ant-Miner generates a fewer number of rules, fewer terms per each rule and performs competitively in terms of efficiency compared to the C4.5 algorithm (see experimental results in [20]). Hence, it has been a focus area of research and a lot of modification has been done to it in order to increase its quality in terms of classification accuracy and output rules comprehensibility (reducing the size of the rule set).

The thesis proposes five extensions to Ant-Miner. 1) The thesis proposes the use of a logical negation operator in the antecedents of constructed rules, so the terms in the rule antecedents could be in the form of <attribute NOT= value>. This tends to generate rules with higher coverage and reduce the size of the generated rule set. 2) The thesis proposes the use stubborn ants, an ACO-variation in which an ant is allowed to take into

consideration its own personal past history. Stubborn ants tend to generate rules with higher classification accuracy in fewer trials per iteration. 3) The thesis proposes the use multiple types of pheromone; one for each permitted rule class, i.e. an ant would first select the rule class and then deposit the corresponding type of pheromone. The multi-pheromone system improves the quality of the output in terms of classification accuracy as well as it comprehensibility. 4) Along with the multi-pheromone system, the thesis proposes a new pheromone update strategy, called quality contrast intensifier. Such a strategy rewards rules with high confidence by depositing more pheromone and penalizes rules with low confidence by removing pheromone. 5) The thesis proposes that each ant to have its own value of α and β parameters, which in a sense means that each ant has its own individual personality.

In order to verify the efficiency of these modifications, several cross-validation experiments have been applied on each of eight datasets used in the experiment. Average output results have been recorded, and a test of statistical significance has been applied to indicate improvement significance. Empirical results show improvements in the algorithm's performance in terms of the simplicity of the generated rule set, the number of trials, and the predictive accuracy.

Keywords: Ant Colony Optimization (ACO), Data Mining, Classification, Multi-pheromone, Stubborn Ants, Ants with Personality.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Overview

Swarm intelligence is a branch of soft computing in which the biological collective behavior is applied [2]. Many animal groups, such as fish schools and bird flocks exhibit such a swarm behavior. This behavior can also be seen in insects like ants and bees that display structural order and integrated behavior (see figure 1.2). At a high-level, a swarm can be viewed as a group of homogenous agents cooperating in some purposeful behavior to achieve some goal. This collective intelligence seems to emerge from what are often large groups of relatively simple agents. The agents use simple local rules to govern their actions and via the interactions of the entire group, the swarm achieves its objectives. A type of self-organization emerges from the continuing actions of the group.

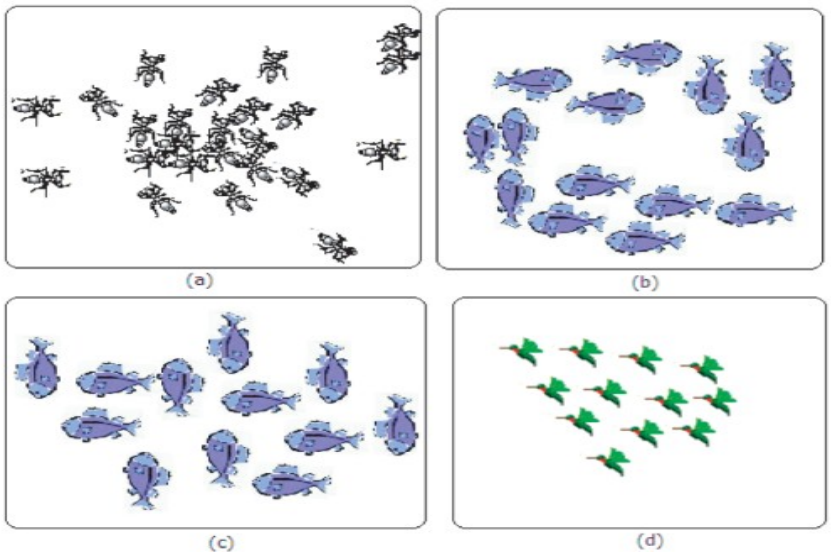

Figure 1.1 - Biological Swarm Behavior Examples. [2]

Since the early 90's, several collective behavior (like social insects, bird flocking) inspired algorithms have been proposed and applied studied optimization problems like NP-hard problems (Traveling Salesman Problem, Quadratic Assignment Problem, Graph problems), network routing, clustering, data mining, job scheduling and many other areas in order to solve problems that are combinatorial in nature.

Particle Swarm Optimization (PSO) and Ant Colonies Optimization (ACO) are the most popular algorithms in the swarm intelligence domain. PSO is a population-based search algorithm and is initialized with a population of random solutions, called particles [2]. Unlike in the other evolutionary computation techniques, each particle in PSO is also associated with a velocity. Particles move through the search space with velocities which are dynamically adjusted according to their historical behaviors. Therefore, the particles have the tendency to move towards better search areas over the course of search process. The following figure describes the basic structure for PSO algorithms.

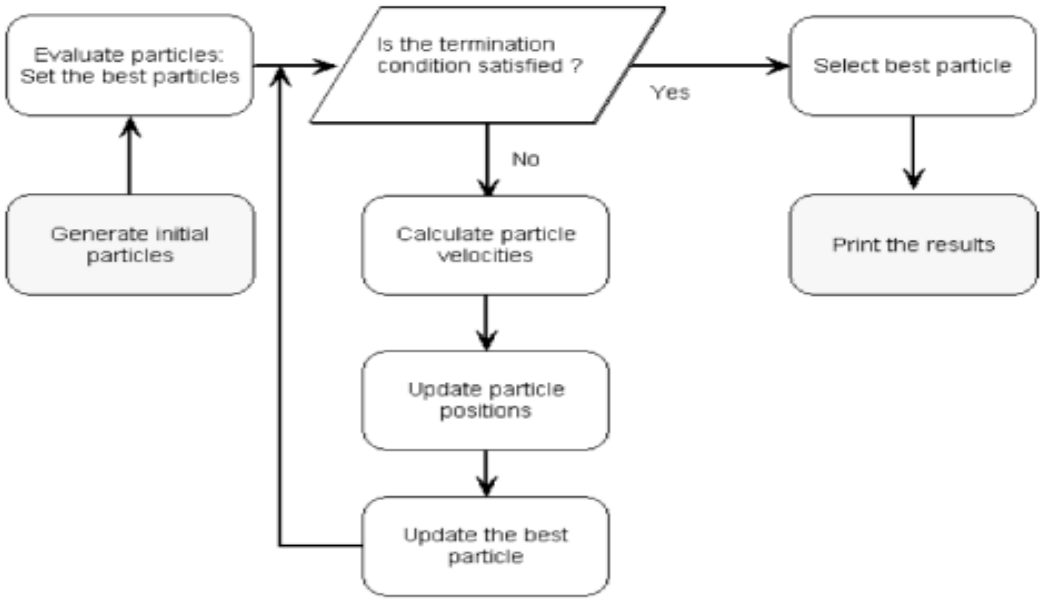

Figure 1.2 - Basic Structure of PSO. [2]

Ant Colonies Optimization (ACO) algorithms were introduced around 1990 [8], [9], [10], [12]. These algorithms were inspired by the behavior of ant colonies. Ants are social insects, living in colonies and exhibit an effective collective behavior. Although each ant is relatively a simple insect with limited individual abilities, a swarm of ants has the ability to find the shortest path from their nest to food. This idea was the source of the proposed algorithms.

When searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. Ants are guided by pheromone smell. Ants tend to choose the paths marked by the strongest pheromone concentration. When an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. The indirect communication between the ants via pheromone trails enables them to find shortest paths between their nest and food sources. As given by Dorigo *et al.* [13], the main steps of the ACO algorithm are given below:

1. Pheromone trail initialization.
2. Solution construction using pheromone.
3. State transition rule.
4. Pheromone trail update.

This process is iterated until a termination condition is reached. More details on the ACO algorithm are discussed in Chapter 3.

One of the most important application of swarm intelligence algorithms is data mining. Data mining is the application of specific algorithms for extracting patterns from

data. The additional steps in the Knowledge Discovery and Data mining process (KDD), such as data selection, data cleaning, and proper display and interpretation of the results are essential to ensure that useful knowledge is derived from the data.

The task of interest here is classification, which is the task of assigning a data point (a case in given a dataset) to a predefined class or group according to its predictive attributes. The classification problem and accompanying data mining techniques are relevant in a wide variety of domains such as financial engineering, medical diagnostic and marketing. The result of a classification technique is a model which makes it possible to classify future cases (in other words, predict the class of a new case) based on a set of specific attributes in an automated way, with a sufficient level of confidence.

In the literature, there is a lot of different techniques proposed for this classification task, some of the most commonly used being C4.5-based decision trees, logistic regression, linear and quadratic discriminate analysis, k-nearest neighbor, artificial neural networks and support vector machines. The performance of the classifier is typically determined by its predictive accuracy on an independent test set. Benchmarking studies have shown that the non-linear classifier generated by neural networks and support vector machines score best on this performance measure. However, comprehensibility can be a key requirement as well, demanding that the user can interpret the model to understand the motivations behind the model's prediction.

In some domains, such as credit scoring and medical diagnostics, the lack of comprehensibility is a major issue and causes a reluctance to use the classifier or even complete rejection of the model. In a credit scoring context, when credit has been denied the Equal Credit Opportunity Act of the U.S. requires that the financial institution

provides specific reasons why the customer's application was rejected, whereby vague reasons for denial are illegal. In the medical diagnostic domain as well, clarity and explainability are major constraints besides the classifier efficiency. The most suited classifiers for this type of problem are of course rules and trees. C4.5 is one of the techniques that construct such comprehensible, user-interpretable classification model with efficient predictive accuracy. On the other hand, other techniques, such as artificial neural network and support vector machine classifiers, are known for their predictive accuracy. However, they do not produce a comprehensive, explainable output.

Ant-Miner is an ACO algorithm, proposed by Parpinelli *et al*. [20], that discovers classification rules of the form:

IF <Term-1> AND <Term-2> AND . . . <Term-n> THEN <Class>

where each term is of the form <attribute = value>, and the consequent of a rule is the predicted class. Chapter 3 is dedicated to describe the Ant-Miner algorithm in detail, where its related work is discussed in Chapter 4.

1.2 Motivation

Ant-Miner performance was compared with the performance of the well-known C4.5 algorithm in six public domain data sets [26]. Overall the results show that, concerning predictive accuracy, Ant-Miner is competitive with C4.5. In addition, Ant-Miner has consistently found considerably simpler (smaller) rules than C4.5. Although applying ACO in the field of classification rule discovery was a new trend, Ant-Miner produced promising results compared to a well-known, sophisticated decision tree algorithm, which has been evolving from early decision tree algorithms for at least a couple of decades. This has motivated a lot of research to focus on such an algorithm.

Since the birth of this ACO-based classification algorithm, several ideas and modification have been applied to the original Ant-Miner version in order to enhance its performance, yet various enhancements and extensions can be investigated, tried and tested to develop Ant-Miner from the perspective of a classification algorithm. From another perspective, as an ACO-based technique, a lot of ACO-based ideas and updates that arise in the literature of swarm intelligence can be easily applied to the Ant-Miner algorithm.

1.3 Thesis Statement and Objective

According to the state of Ant-Miner as a new, promising classification rule discovery technique and its ACO-based algorithm nature, my objective is to:

"Implement effective extensions to the original version of Ant-Miner in order to improve its performance in terms of1) Produced model comprehensibility, via reducing the number of generated rules resulting in a smaller (simpler) model, 2) algorithm running time, via decreasing the number of iterations and the trials performed per iteration, and 3) produced model efficiency, via elevating the predictive accuracy of the generated rule set."

1.4 Thesis Contribution

The main contribution of this Master's thesis consists of five extensions on the original Ant-Miner algorithm:

1. **Logical Negation Operator:** this allows the usage of a logical negation operator in the antecedents of constructed rules, so that the constructed rules would have a higher coverage. This should decrease the number of the generated rules, thus improving output comprehensibility, as well as increasing its classification accuracy.

2. **Applying Stubborn Ants:** an ACO-variation in which an ant is allowed to take into consideration its own personal history. The technique was introduced in 2008 in [1]. The idea is to promote search diversity by having each ant be influenced by its own history of constructing solutions in addition to the pheromone trails left by other ants. This tends to reduce the number of trials needed to converge on a rule per iteration. Besides, stubborn ants produce better results in terms of classification accuracy.
3. **Multi-Pheromone Ant-Miner:** using multiple types of pheromone, one for each permitted rule class, i.e. an ant would first select the rule class and then deposit the corresponding type of pheromone. An ant is only influenced by the amount of the pheromone deposited for the class for which it is trying to construct a rule. In this case, pheromone is not shared amongst ants constructing rules for different classes. This allows choosing terms that are only relevant to the selected class. This improves the classification accuracy of the generated rules.
4. **Quality Contrast Intensifier:** A new pheromone updates procedure where a rule whose quality is higher than a specific threshold would be rewarded by allowing it to deposit higher quantities of pheromone. In the same manner, rules with lower levels of quality are penalized by removing pheromone from their terms in the construction graph. This is used to direct the ants to use the good tried paths and unexplored paths rather than the low-quality-tried paths. The result of such an extension is to reduce the trials per rule and find better classification rules in term of accuracy. Moreover, a new convergence test is applied in order to insure that the discovered rule satisfies a minimum quality threshold. Otherwise, new different rules should be sought.
5. **Ants with Personality:** we allow each ant to have its own value of α and β parameters, which represent the weight of the cognitive component and the social

component respectively in the state transition formula (see formula 2.2). This in a sense means that each ant has its own individual personality. This promotes search diversity and helps in finding new better solutions.

1.5 Thesis Overview

This thesis is structured as follows:

Chapter 2 consists of two parts. Part1 describes the Ant Colony Optimization (ACO) technique in detail. It starts by explaining the biological behavior of the swarms, and then it moves to the artificial collective behavior and ACO meta-heuristic algorithm. Some ACO variations are discussed in the end of the chapter. The second part of Chapter 2 talks about data mining and knowledge discovery. Knowledge discovery steps are explained, followed by discussion of various data mining tasks. Challenges of data mining are tackled and different applications of data mining are mentioned at the end of this chapter.

Chapter 3 introduces the original version of Ant-Miner algorithm. A detailed description of the algorithm steps, results and algorithm issues are tackled in this chapter as well.

Chapter 4 exhibits some of the most important related work to the original version of Ant-Miner.

Chapter 5 to Chapter 8 introduce the extensions that have been applied on the original version of the Ant-Miner algorithm in the following order: Chapter 5 explains the use of logical negation operator in rule construction, Chapter 6 describes the use stubborn ants, Chapter 7 explains multi-pheromone system, applying a quality contrast intensifier

in pheromone update as well as introducing the new convergence test, and Chapter 8 shows the use of ants with personality.

Chapter 9 describes the experimental approach that was used to test the performance of the new modifications on the algorithm. Experimental results and their discussion are shown in this chapter as well.

Chapter 10 summarizes this thesis and discusses options for future research.

Chapter 2

BACKGROUND

PART 1: ANT COLONY OPTIMIZATION

2.1 Introduction to ACO

Ant Colony Optimization (ACO) is subfield of swarm intelligence which studies algorithms inspired by the observation of the behavior of biological ant colonies. ACO was proposed by M. Dorigo *et al.* [8 – 9] as meta-heuristic method for solving optimization problems. As was described in Chapter 1, swarm intelligence algorithms are self-organizing systems that are made up of simple individuals cooperating with each other to achieve a goal, without any form of central control over the swarm members. Although ants are simple insects, ant colonies are able to solve complex problems such as finding shorts path from the nest to the food utilizing the collective behavior of the whole swarm communicating indirectly with each other via pheromone trails. This chapter illustrates the basic ideas of ACO and describes some variations in the literature for the algorithm. A comprehensive overview about ACO can be found in "Ant Colony Optimization", a book by M. Dorigo and T. Stützle [13].

2.2 Biological Ants Behavior

Social insect swarms like ant colonies are distributed systems that, in spite of the simplicity of their individuals, produce a collective behavior that enables a swarm of insects to accomplish complex tasks that, in some cases, far exceed the individual capabilities of a single insect [13]. The high coordinated, self-organizing structure that is exhibited by colonies of ants can be used to build an agent-based artificial system to solve

hard computational problems. Ants coordinate their activities via stigmergy, a form of indirect communication mediated by altering the environment.

As an example of stigmergy observed in colonies of ants, an ant drops a chemical substance called a pheromone while waking from source to food and vice versa. Other ants are able to smell this pheromone, and its presence influences the choices they make along their path. An ant is more likely to follow route containing high concentrations of pheromone over one that does not. The pheromone deposited on the ground forms a pheromone trail, which allows the ants to find good sources of food that have been previously identified by other ants. The similar types of behavior of ant colonies have inspired different kinds of ant algorithms, foraging, division of labor, brood sorting, and cooperative transport.

The "double bridge" is an effective experiment was done by Deneubourg *et al.* in the 90s [6] to explore the pheromone trail-laying and -following behavior of Argentine ant species. The experiment shows the collective behavior of ants that emerges through pheromone trial-based communication, which leads to converge on the shorter path from source to distention. The following section presents an overview of this experiment.

2.2.1 Double Bridge Experiment

The nest of the ants was connected to a food source by two bridges. In the first experiment, the two bridges were equal in length. The behavior of the ants in choosing which branch to take when searching for food and bringing it back to the nest was then observed over time. The ants start exploring the surroundings of the nest and randomly find one of the bridges and reach the food source. During their journey to the food source and back, the ants deposit pheromone on the bridge that they use. Initially, each ant

randomly chooses one of the bridges. After some time, there will be more pheromone deposited on one of the bridges than on the other. Because ants tend to prefer in probability to follow a stronger pheromone trail, the bridge that has more pheromone will attract more ants. This in turn makes the pheromone trail grow stronger, until the colony of ants converges toward the use of a same bridge.

In another experiment, the two bridges were not of the same length so that the longer branch was twice as long as the short one. At the beginning, ants leave the nest to explore the environment and arrive at a decision point where they have to choose one of the two branches. The two branches initially appear identical to the ants, they choose randomly. Therefore, it can be expected that, on average, half of the ants choose the short branch and the other half the longer branch. Because one branch is shorter than the other, the ants choosing the short branch are the first to reach the food and to start their return to the nest. Therefore, the pheromone intensity will increase faster on the short branch. Then, when other ants make a decision between the two bridges, the higher level of pheromone on the short branch will bias their decision in its favor, which will in time be used by all the ants because of the autocatalytic process described previously.

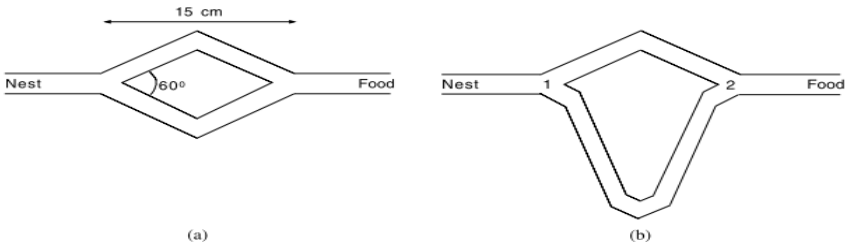

Figure 2.1 - Experimental Setup for the Double Bridge Experiment. [6]
(a) Branches have equal length. (b) Branches have different length.

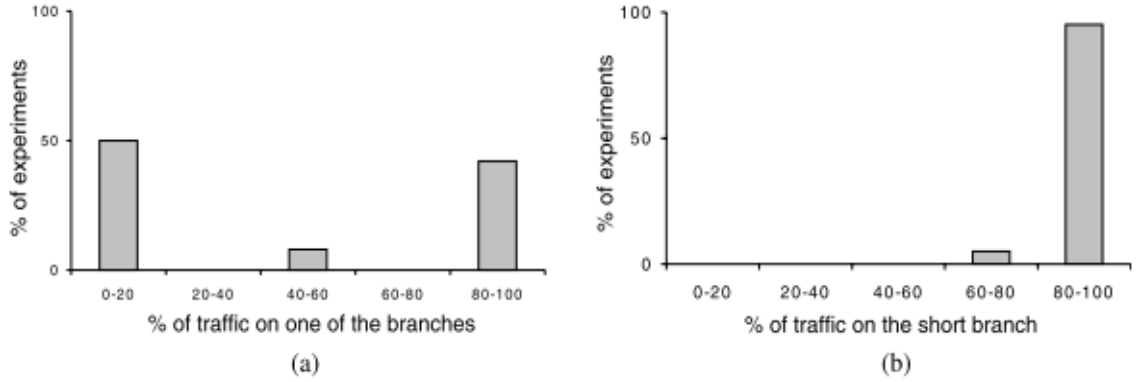

Figure 2.2 - Traffic Behavior for each Case in the Double Bridge Experiment. [6]
(a) Branches have equal length. (b) Branches have different length.

When compared to the experiment with the two branches of equal length, the influence of initial random fluctuations is much reduced, and stigmergy, autocatalysis, and differential path length are the main mechanisms at work. Interestingly, it can be observed that, even when the longer branch is twice as long as the short one, not all the ants use the short branch, but a small percentage may take the longer one. This may be interpreted as a type of ‘‘path exploration.’’ Figure 2.1 and 2.2 show the experimental setup and observed result for both experiments. Figures were taken from [13].

2.2.2 Related Algorithmic Model

A model was developed by Goss *et al*. [7] to explain the behavior observed in the double bridge experiment described in the previous section. As explained in [13], assuming that m_1 number of ants has taken the first branch and m_2 has taken the second one after t unites of time. The probability P_1 that $m+1$ ant selects the first branch is given by the following equation:

$$P_1(m+1) = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \tag{2.1}$$

where parameters K and h are needed to fit the model to the experimental data. The probability P_2 that the same ant chooses the second bridge is 1-$P_1(m+1)$. This model assumes that the amount of pheromone on a branch is proportional to the number of ants that used the branch in the past and no pheromone evaporation is considered by the model. So the at any given time t, the probability that that an ant chooses branch i depends on the number of ants that have m_i previously selected that branch. Assuming that branch i is the shorter one. At time t the number of ant that has taken branch i is probably larger as they take the path from the nest to the food and back in a shorter amount of time than the other branches. Therefore, the probability P_i of ant $(m+1)$ to select the shorter branch i would be larger than the probability of selecting other branches.

This basic model explains the foraging behavior of real ants in solving such an optimization problem, which is finding the shortest path, without any global sight or master control. Instead, stigmergic communication happens via the pheromone that ants deposit on the ground. This can be an inspiration to design artificial ants that solve optimization problems defined in a similar way. The following section describes ideas behind artificial ants.

2.2.3 Artificial Ants

The binary bridge experiments show that ant colonies exhibit a collective behavior that is able to solve optimization problems. With stigmergic communication, via pheromone depositing and the use of probabilistic rules based on local information they can find the shortest path between two points in their environment. An idea towards an artificial ant system is to represent the solution space for any optimization problem as a

set of nodes in a graph, representing the variable states of the solution. Artificial ants can visit these states to build a candidate solution for the problem. Artificial ants may simulate pheromone laying by modifying an appropriate pheromone variable associated with solution states they visit. They would have only local access to these pheromone variables according to the stigmergic communication model.

Ant Colony Optimization (ACO) is an artificial ants system that basically follows the previously described ideas of real ants' behavior. Both real and artificial ant colonies are composed of a swarm of simple individuals that use collective behavior to achieve a certain goal. In the case of real ants, the goal is to find the food using a good (short) path, while in the case of artificial ants, it is to find a good solution to a given optimization problem. A single ant (either a real or an artificial one) is able to find a solution to its problem, but only cooperation among many individuals through stigmergy enables them to find good solutions.

Artificial ants live in a virtual world, probably a graph of nodes representing the search space of the solution for a given problem. The use of pheromone, which is in the artificial system a numeric variable associated with each state in the search graph, depositing and influenced by it while searching in the solution states graph constructing a solution. A sequence of pheromone values associated with problem states is called artificial pheromone trail.

There are many similarities between real and artificial ants. However, there are some important differences between real and artificial ants. These differences are listed below as were described by M. Dorigo and T. Stützle in [13]:

- Artificial ants live in a discrete world—they move sequentially through a finite set of problem states.

- In real ants, there is the coupling between the autocatalytic mechanism and the implicit evaluation of solutions. As for the double bridge experiment, the fact that shorter paths are completed earlier than longer ones, and therefore they receive pheromone reinforcement quicker. So the shorter the path is, the sooner the pheromone is deposited, and the more the ants use the shorter path. On the other hand, artificial ants drop pheromone after the solution is constructed and its quality is evaluated. This may not have anything related to the quickness in which the pheromone accumulates on a path due to its length. Thus, the amount of the pheromone may vary according to the quality of the solution to simulate enforced catalytic mechanism toward the good paths.
- Artificial ants may use local heuristics, local search and other additional mechanisms.

The following section describes ant colony optimization meta-heuristic model in detail with illustration of the ACO algorithm.

2.3 Ant Colony Optimization Meta-Heuristic

"A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality." —Tabu Search, Fred Glover and Manuel Laguna, 1998.

In other words, meta-heuristic it is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems [13]. This can be seen as a general-purpose method designed to guide an underlying problem-specific heuristic toward promising regions of the search space containing high-quality solutions. A meta-heuristic is therefore a general algorithmic framework which can be applied to

different optimization problems with relatively few modifications to make them adapted to a specific problem.

M. Dorigo *et al*. formalized an ACO meta-heuristic model using pheromone manipulation for solving Combinational Optimization Problems (COPs) [8]. This has since been used to tackle many combinatorial optimization problems. The model can be defined as follows; a model $P = (\mathbf{S}, \mathbf{\Omega}, f)$ of a COP consists of:

- A search space **S** defined over a finite set of discrete decision variables and a set **Ω** of constraints among the variables.
- An objective function $f: \mathbf{S} \rightarrow \mathbb{R}_{\mathbf{0}}^{+}$ to be optimized (minimized or maximized).

The search space **S** is a set of discrete variables X_i, $i = 1, \dots, n$, with discrete values $v_i^j \in \mathbf{D}_i = \{v_i^1, v_i^2, \dots, v_i^{|\mathbf{D}_i|}\}$. A variable instantiation is the assignment of value v_i^j to variable X_i, denoted by $X_i \leftarrow v_i^j$. An instantiated decision variable $X_i = v_i^j$ is called a solution component and denoted by c_{ij}, The set of all possible solution components is denoted by $\boldsymbol{C}$. Any solution $s \in \mathbf{S}$, that is a complete variables assignment in which each decision variable has a value assigned that satisfies all the constraints in the set **Ω**, is a feasible solution of the given COP. A solution $s^* \in \mathbf{S}$ is called a global optimum if and only if $f(s^*) \leq f(s)\ \forall\ s \in \mathbf{S}$ (for minimization). The set of all globally optimal solutions is denoted by $\mathbf{S}^* \in \mathbf{S}$. To solve a (COP), at least one $s^* \in \mathbf{S}$ needed to be found.

The aforementioned model for COP is the basis for pheromone manipulation used in ACO. A pheromone trail parameter T_{ij} is associated with each component c_{ij}. The set of all pheromone trail parameters is denoted by $\boldsymbol{T}$. The value of a pheromone trail parameter T_{ij} in a given time t associated with decision component c_{ij} is denoted by $\tau_{ij}(t)$. This pheromone value is then used and updated by the (ACO) algorithm during the

search. This allows modeling the probability distribution of different components of the solution.

In ACO, the described model is represented as a graph, called construction graph, which is traversed by artificial ants to build a solution for a given problem. The construction graph $G_c(\boldsymbol{V}, \boldsymbol{E})$ is a fully connected graph consisting of a set of vertices $\boldsymbol{V}$ and a set of edges $\boldsymbol{E}$. The set of components $\boldsymbol{C}$ may be associated either with the set of vertices $\boldsymbol{V}$ of the graph G_c, or with the set of its edges $\boldsymbol{E}$. An ant constructs a solution incrementally while moving from vertex to vertex along the edges of the graph. Additionally, the ant deposits a certain amount of pheromone on the components, that is, either on the vertices or on the edges that they visit. The amount of pheromone, $\Delta\tau$, deposited depends on the quality of the solution found. Subsequent ants are influenced by pheromone trails and use them as guides toward good decision components in the search graph. This increases the probability of choosing such decision components in the following ant trials. The ant colony optimization meta-heuristic technique is shown in the following algorithm.

Algorithm 2.1 - Ant Colony Optimization Meta-heuristic.

Set parameters, initialize pheromone trails.

WHILE termination conditions not met

DO

Construct a Solution

Apply Local Search {optional}

Update Pheromone

END WHILE

As shown in Algorithm 2.1, each ant in the swarm builds a solution by incrementally selecting solution components from the construction graph utilizing the pheromone on it. A local search might be applied to enhance the solution quality. Then the pheromone is updated on the ant trail during its navigation. The amount of pheromone to deposit may depend on an evaluation function used to determine the quality of the constructed solution. These steps are repeated until a predefined termination condition is met. The following is a more detailed explanation for the basic components of the algorithm.

2.3.1 Construct a Solution

Each $ant_l \in \{ant_1, ant_2, \dots, ant_m\}$ constructs a solution from elements of a finite set of available solution components $C = \{c_{ij}\}$ in the construction graph G_c, where i represents the index of the solution variable and j the index of the value belonging to the domain this variable. Each ant_l starts with an empty solution $s^l = \emptyset$. At each step in the solution construction, a valid solution component is added to the partial solution from a set of feasible neighbors to the current ant solution state $\boldsymbol{N}(s^l) \subseteq \boldsymbol{C}$. The process of constructing a solution can be viewed as a path in the construction graph G_c where the set of constraints among the variables $\boldsymbol{\Omega}$ defines the feasible neighbors $\boldsymbol{N}(s^l)$ at each step according to the current state of the partial solution s^l.

The decision component c_{ij} selection at each step is done probabilistically according to the following formula:

$$p(c_{ij}) = \frac{\tau_{ij}^{\alpha}(t).\eta_{ij}^{\beta}}{\sum \tau_{ij}^{\alpha}(t).\eta_{ij}^{\beta}}, \qquad \forall\, c_{ij} \in \boldsymbol{N}(s^l) \tag{2.2}$$

where:

- $\tau_{ij}(t)$ is the amount of the pheromone associated to component c_{ij} at time t.
- η_{ij} is a problem dependent heuristic value assigned to component c_{ij}.
- α and β are positive parameters, whose values determine the relative importance of pheromone versus heuristic information.

2.3.2 Apply Local Search

Local search is an optional solution that can be applied after the solution is constructed in order to enhance the solution by locally optimizing it. Local search a can be implemented as problem specific operation and is done before the pheromone update step. Then the locally optimized solutions are then used to decide which pheromones to update. Local search improves the quality of the solution constructed and enhances the overall output of the algorithm. However, it might be an expensive operation depending on the combinations scope that the operation searches in. Local search can be done after each ant constructs a solution or can be done iteration based on the best solution constructed by set of ants per iteration.

2.3.3 Update Pheromone

After a solution is constructed, the pheromone on the construction graph is updated to guide subsequent ants to good decisions to take while constructing their solutions. This pheromone update is done by two steps:

- **Pheromone Reinforcement**: this done by increasing the pheromone value associated with the decision components $c_{ij} \in s^l$ according to the quality of the constructed

solution, as they are good or promising components. The reinforcement is done by the following equation:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + f(s^l), \forall\, c_{ij}\, \epsilon\, s^l \qquad (2.3)$$

where $f: \mathbf{S} \rightarrow \mathbb{R}_{\mathbf{0}}^{+}$ is a fitness function that evaluates the quality of solution s^l.

- **Pheromone Evaporation:** this is done by decreasing the pheromone value associated with all $c_{ij}\, \epsilon\, \mathbf{C}$ in the construction graph so that the bad components (the ones that are not being chosen frequently) get their pheromone values decreased and give space to other components in unexplored regions in the construction graph to get selected. This is to avoid early convergence of the algorithm. Evaporation is done as follows [13]:

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \rho, \forall\, c_{ij} \epsilon\, \mathbf{C} \qquad (2.4)$$

where ρ is evaporation factor parameter $\rho\, \epsilon$ [0,1].

2.4 Traveling Sales Person Problem

This section describes the implementation of ACO and how it works to solve the famous NP-hard problem: traveling sales person. The TSP consists of a set of locations (cities) and a traveling salesman that has to visit all the locations once and only once. The distances between the locations are given and the task is to find a Hamiltonian tour of minimal length.

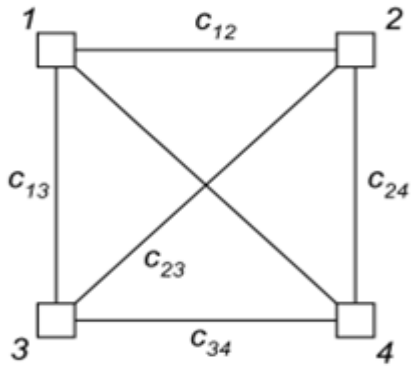

Figure 2.3 - Construction Graph for TSP with Four Cities.

The first thing to apply ACO on a problem is to have a construction graph that represents the solution components space for the problem. As shown in figure 2.3, the construction graph $G_c(\boldsymbol{V}, \boldsymbol{E})$ for TSP consists of vertices $\boldsymbol{V}$ representing the cities $c_i \in \{c_1, c_2, \dots, c_n\}$. $\boldsymbol{E}$ is the set of edges connecting the cities, which represents the solution components c_{ij}, and with which the pheromone is associated. The length value of each edge represents the distance between city c_i and c_j.

Each ant starts from a randomly selected location (vertex of the graph G_c). Then, at each construction step it moves along the edges of the graph, by which it selects a solution component. Each ant memorizes the solution components (edges) that it selected through its path, and in subsequent steps it chooses among the edges that do not lead to vertices that it has already visited (this constraint defines feasible movements to the ant according to its current partial solution $\boldsymbol{N}(s^l) \subseteq \boldsymbol{C}$) . At each construction step an ant chooses probabilistically the edge to follow using equation (2.2). An ant has constructed a solution once it has visited all the vertices of the graph.

Afterwards, the pheromone is updated according to the quality of the constructed path. A possible fitness function for TSP solution is:

$$f_{TSP}(s^l) = \frac{1}{\sum length(c_{ij})} \; \forall \; c_{ij} \in s^l \tag{2.5}$$

which is the inverse of the length of the tour constructed by the ant. Ant colony optimization has been shown to perform quite well on the TSP [25].

2.5 ACO Variations

2.5.1 Ants System

Ant System was introduced in the literature by M. Dorigo *et al*. in [9]. It is the first ACO algorithm to be proposed. Its main characteristic is that the pheromone values are updated by all the ants that have completed constructing the solution. In other words, after each ant constructs a solution, it updates its pheromone trial according to the quality of the solution it generated, unlike other techniques which update the pheromone after the best solution is selected among a set of ants that constructed solution in an iteration of the algorithm.

2.5.2 MAX-MIN Ant System

MAX -MIN Ant System is an improvement over the original Ant System idea. MMAS was proposed by T. Stützle and Hoos in [24], who introduced a number of changes of which the most important are the following: only the best ant can update the pheromone trails, and the minimum and maximum values of the pheromone are limited.

2.5.3 Ant Colony System

Another improvement over the original Ant System is Ant Colony System (ACS), introduced by L. M. Gambardella and M. Dorigo [11]. The most interesting contribution of ACS is the introduction of a local pheromone update in addition to the pheromone update performed at the end of the construction process (offline pheromone update). The main goal of the local update is to diversify the search performed by subsequent ants during the same iteration. In fact, decreasing the pheromone concentration on the edges as they are traversed during one iteration encourages subsequent ants to choose other edges

and hence to produce different solutions. This makes the possibility of several ants producing identical solutions per a given iteration less likely.

PART 2: DATA MINING AND KNOWLEDG DISCOVERY

2.6 Introduction to Data Mining

Since the widespread of transactional software that has automated various systems in different fields, a huge volume and variety of data has been continuously collected. Storing and retaining immense amounts of data in easily accessible form was availed effectively. As a matter of fact, this raw data potentially stores a huge amount of information and hidden patterns. Hence, the need of discovering these hidden patterns and convert them into useful knowledge arose.

The notion of finding useful patterns in data has been given a variety of names including data mining, knowledge extraction, information discovery, and data pattern processing. Data mining is the application of specific algorithms for extracting patterns from data. Research directions have emerged in the recent past for tackling the problem of making sense out of large, complex data sets. As conventional methods for sifting through huge amounts of data manually and making sense out of it is slow, expensive, subjective and prone to errors, the need to automate the process has been a research focus. Knowledge discovery from databases (KDD) evolved as a research with multi-disciplinary fields containing databases, machine learning, pattern recognition, statistics and artificial intelligence.

Data is stored in huge repositories with high dimensionality in different types and formats; numerical, textual, graphical, symbolic, linked. Typical examples of some such domains are the world-wide web, geo-scientific data, maps, multimedia, and time series

data as in financial markets. In addition, the type of the knowledge wished to be discovered varies in a wide range according to the domain of interest and it task of use needed from the knowledge. All these factors encourage developing advanced techniques for mining complex data.

2.7 Knowledge Discovery Steps

Basically, Knowledge discovery process has three essential parts: data preparation, data mining and knowledge presentation. Data mining is the core step where the techniques for extracting the useful hidden patterns are applied. In this sense, data preparation and knowledge presentation can be considered, respectively, to be pre-processing and post-processing steps of data mining.

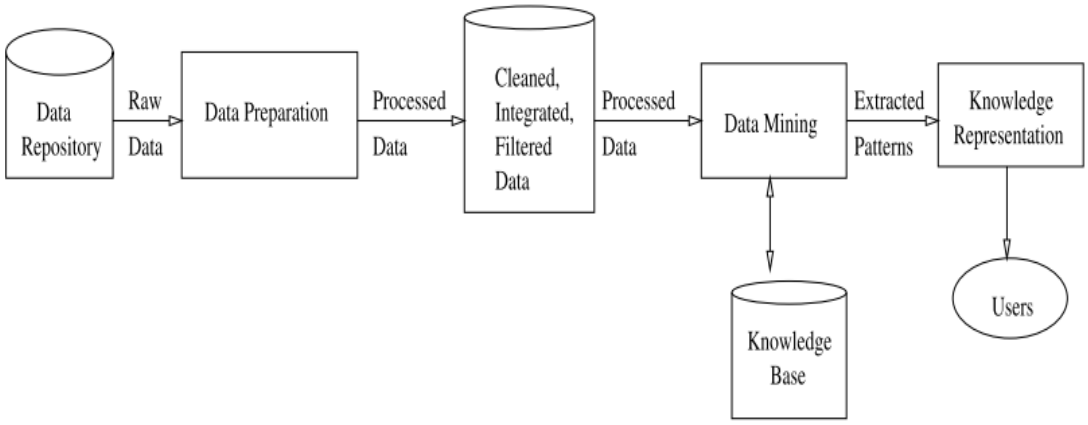

Figure 2.4 - Knowledge Discovery Process.

As shown in Figure 2.4, raw data in different types and formats is received from non-homogenous data sources. Various tasks of data preparation and data fusion are applied to the raw data to create a cleansed, filtered, integrated and malleable version that is appropriate for different task of information retrieval and knowledge extraction. As data mining algorithms are applied, generated models and discovered knowledge are

stored in a knowledge base for further usage. A neat presentation and visualization is required for the knowledge to facilitate user interaction.

2.8 Data Preparation

Data source repositories have data in different types and formats. Some errors may occur during the data recording and storing by the source system such as missing values, noise, inconsistency etc. In addition, among the huge amount of the available data, only some parts of it can be interesting or useful for a specific knowledge discovery task and other parts should be neglected. Data needs different structures and formats to be suitable for data knowledge discovery processing tasks. Therefore, before going to perform mining on the data, some kind of pre-processing [15] is required. Preprocessing of data is done in the following major ways:

- **Data cleaning**: This is performed to remove inconsistency, noise to fill up missing values and to filter needed portions.
- **Data integration.** This is needed to combine and unify data from multiple different sources like databases, data cubes, flat files etc. Correlation analysis, detecting data conflict, and resolving semantic heterogeneity are used for data fusion.
- **Data transformation.** The format of data in the repositories may not be suitable for processing. So, the format of the data should be transformed to a one suitable for a particular task. This is done for smoothing, aggregation, generalization, normalization, and attributes construction.
- **Discretization**. This step consists of transforming a continuous attribute into a categorical (or nominal) attribute, taking only a few discrete values - e.g., the real-valued attribute. Salary can be discretized to take on only three values, say "low",

"medium", and "high". This step is particularly required when the data mining algorithm cannot cope with continuous attributes. In addition, discretization often improves the comprehensibility of the discovered knowledge.

- **Data reduction**. This is needed to have a reduced version of data that can work effectively with a data mining algorithm. This data reduction is done in terms of dimensionality reduction, data cube aggregation, as well as data compression.
- **Data selection.** For the purpose of processing and analysis, relevant data are selected and retrieved in this step.

2.8.1 Data Mining

Data mining is the core part in the knowledge discovery process, which aims to discover and extract interesting, potentially useful hidden patterns from large amounts of data. Patterns discovered could be of different types such as associations, trees, profiles, sub-graphs, and anomalies. The interestingness and the usefulness of the knowledge to be discovered are relative to the problem and the concerned user. A piece of information may be of immense value to one user and absolutely useless to another. Often data mining and knowledge discovery are treated as synonymous, while there exists another school of thought which considers data mining to be an integral step in the process of knowledge discovery.

Different data mining techniques are used to carry out different knowledge discovery tasks. Classification, clustering, association analysis, regression and deviation detection are the most common data mining techniques that are used for different knowledge discovery task. These techniques are described in the following section.

Data mining techniques mostly consist of three components: a model, a preference criterion and a search algorithm [14]. The most common model functions in current data mining techniques include classification, clustering, regression, and link analysis and dependency modeling. A model is selected according to the intended discovery task and the nature of the useful knowledge to be extracted. Models vary in the flexibility of the model for representing the underlying data and the interpretability of the model in human terms. This includes decision trees and rules, linear and nonlinear models, example-based techniques such as NN-rule and case-based reasoning, probabilistic graphical dependency models (e.g., Bayesian network) and relational attribute models. The preference criterion is used to evaluate the efficiency of the model according the underlying dataset. Preference citation can determined which model to use for mining, as it best fits the current nature of data. It tries to avoid over-fitting of the underlying data or generating a model function with a large number of degrees of freedom. The search algorithm is then defined for the model that carries out the intended knowledge discovery task.

2.8.2 Knowledge Presentation

As the knowledge is extracted, the user should be able to interpret this knowledge and make use of the extracted patterns for decision making concerning his domain. The discovered knowledge will be interesting for the user if it is easily understood, valid, novel and useful. Presentation of the information extracted in the data mining step in a format easily understood by the user is an important issue in knowledge discovery. Data visualization and knowledge representation are important components. The following are some interesting ways of data presentation:

- Decision Trees.

- Graphs.
- Tables and cross-tabs
- Charts and Histograms.
- Natural language generated rules.

2.9 Overview of Data Mining Tasks

Data mining tasks vary according to what types of knowledge we are want to try and discover and how the discovered knowledge is intended to be used. In general, data mining tasks can be classified into two categories, descriptive and predictive [15]. The descriptive techniques provide a summary of the data and profile its general characteristics and properties. On the other hand, the predictive techniques learn from the current data in order to make forecasts or predictions about the behavior of new data. The following is description of most commonly used data mining tasks.

2.9.1 Classification

Classification is a type of supervised learning. In supervised learning, the data set contains objects with several attributes as input features for each object, and one attribute is considered the class (or the label) of this object. Classification is a process of building a model that can describe and classify the object class as a function of its input attributes. As shown in figure 2.5, the input for classifier model discovery is a training set that contains labeled cases (cases which their classes are known). A classification model is built upon relationship patterns discovered between the input attributes and the classes of the cases. Now the classification model is able to classify (find the class) of unlabeled input cases, whether they are a testing set of cases or whole new cases which their classes need to be predicted.

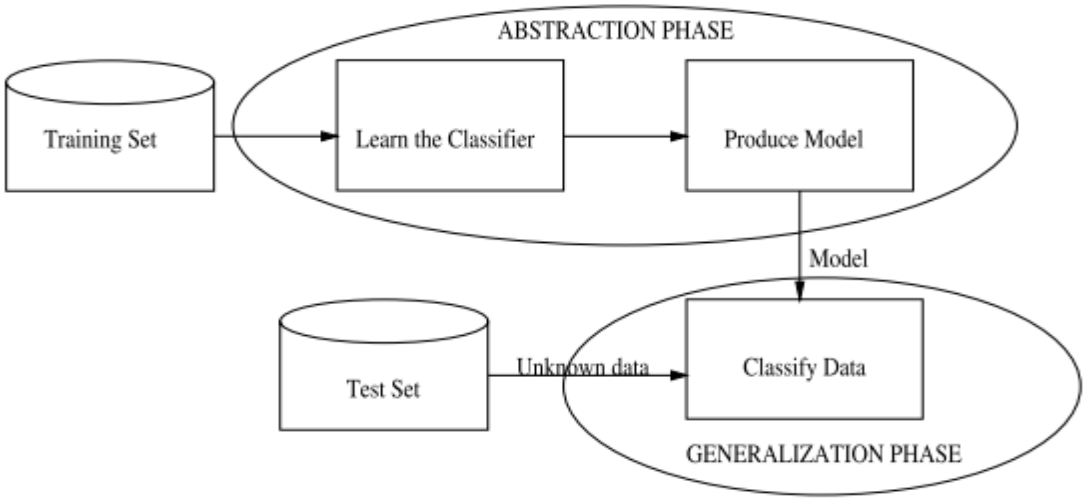

Figure 2.5 - Process of Building a Classification Model.

Note that some data mining techniques for classification generate a classifier that can only classify unlabeled cases without describing the relationships between the attributes of a case and its class. Examples of such techniques are the nearest neighbor classifier, Bayes maximum likelihood classifier and Neural Networks-based classifier. Other techniques can produce a classification model that not only can predict a class of an unlabeled case, but can also describe the relationships between the input features and the classes of the cases. This description can be in the form of rules or classification trees. Decision trees and rule induction are examples. The latter type of classification techniques has an advantage of model interpretation as it provides insight for the user regarding the data at hand and on the relationship patterns amongst it. Some of these techniques are now briefly described.

- **Nearest Neighbor Classifier:** It assigns the unlabeled cases the class of the nearest neighbor to it within the labeled training set. Given a training set with many labeled cases $\{x_1, x_2, \dots, x_n\}$, the distance D is calculated between the new unlabeled case

x_{new} and each case $x_i \in \{x_1, x_2, \dots, x_n\}$ in the training set using distance function. The new unlabeled case x_{new} is given the class of case that has the least value of distance D with it. If k-nearest neighbor is considered, the new case is assigned the class of the majority of the k nearest cases [15].

- **Naïve Bayes Classifier:** is a simple probabilistic classifier based on applying Bayes' theorem with attributes independence assumptions [15]. Let us consider having a data set with n attributes $\{x_1, x_2, \dots, x_n\}$ for each case. Assuming that attributes are conditionally independent of one another given class Y, we have:

$$P(x_1, x_2, .. x_n|Y) = \prod_{i=1}^{n} P(x_i|Y) \quad (2.6)$$

This is a dramatic reduction compared to the $2(2^{n-1})$ parameters needed to characterize $P(X|Y)$ if we make no conditional independence assumption. Naïve Bayes aims to train a classifier that will output the probability distribution over possible values of , for each new instance X that we ask it to classify. The expression for the probability that Y will take on its k-th possible value is the maximum value of the following equation calculated for each y_k:

$$y_k \leftarrow P(Y = y_k).\prod_i P(X_i|Y = y_k) \quad (2.7)$$

- **Decision Trees:** A decision tree is an acyclic graph. In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications. It is easy to convert any decision tree into classification rules. Once the training data set is available, a decision tree can be constructed from them from top to bottom using a recursive divide and conquer algorithm. This process is

also known as decision tree induction. A version of ID3 [15], a well-known decision-tree induction algorithm, is described below.

1. Create a node *N*.
2. If all training data points belong to the same class *(C)* then return *N* as leaf node labeled with class *C*.
3. If cardinality (features) is NULL then return *N* as a leaf node with the class label of the majority of the points in the training data set.
4. Select a feature *(F)* corresponding to the highest information gain, then label node *N* with feature *F*.
5. For each known value f_i of *F*, partition the data points as s_i.
6. Generate a branch from node N with the condition feature = f_i.
7. If s_i is empty, then attach a leaf labeled with the most common class in the data points left in the training set.
8. Else attach the node returned by Decision tree induction (s_i , (features-F)).

Assume we have a data set with n cases labeled with k classes. n_i is the number of cases belonging to class C_i. Suppose that each case has D features $\{f_1, f_2, \dots, f_d\}$. Each feature F can the cases into D subsets $\{s_1, s_2, \dots, s_d\}$. The information gain of a feature is measured by the following equation:

$$Gain(F) = I(n_1, n_2, \dots, n_k) - E(A) \tag{2.8}$$

where

$$E(A) = \sum_{j=1}^{s} \left(\frac{n_{1j}, n_{j2}, \dots, n_{kj}}{n}\right) . I(n_{1j}, n_{j2}, \dots, n_{kj}) \tag{2.9}$$

and

$$I(n_{1j}, n_{j2}, \dots, n_{kj}) = -\sum_{j=1}^{j=k} p_{ij}.\log(p_{ij}) \quad (2.10)$$

Here, p_{ij} is the probability that a data point in D_{ij} belongs to class C_i.

2.9.2 Clustering

Clustering is the process of partitioning the input data set into groups or segments, where each group is called a cluster. Each cluster contains a subset of the data points that are more similar to one another and less similar (dissimilar) to data points in other clusters. The similarity and dissimilarity are measured in terms of some distance function. Cluster analysis serves as a powerful descriptive model that can profile the data point according to its attributes and exhibits similarities and dissimilarities between the data clusters that are found.

Clustering is considered as an unsupervised learning, as the input cases to any clustering technique are not required to be labeled. The clustering algorithm should discover these labels as each cluster can be considered as a class for the data points that it contains after it is discovered.

K-Means algorithm [15] has been one of the more widely used ones; it consists of the following steps:

1. Choose K initial cluster center $z_1, z_2, .., z_k$ randomly from the domain space of the input data point $\{x_1, x_2, \dots, x_n\}$.
2. Assign each data point x_i to cluster $C_j, j \in \{1, 2, \dots, k\}$ if the distance between x_i and z_j is the least among the distance between x_i and all other cluster centers.
3. Computer new cluster centers as follows:

$$z_i^{new} = \frac{1}{n_i} \sum_{x_j \in c_i} x_j \ , i = 1,2, \dots, k \tag{2.11}$$

where n_i is the number of data points belonging to cluster c_i.

4. Terminate if no change in the centers occurs or upon meeting any other criteria.

Although K-means is one of the widely used clustering algorithms, it suffers from shortcomings. Outliers can affect the computation of centriods. K-medoid attempts to alleviate this problem by using the medoid, the most centrally located object, as the representative of the cluster. (PAM), (CLARA) and (CLARANS) are various implementations of K-medoid. Fuzzy K-Means cluster the data set with membership value associated with each data point for each cluster. Hierarchal clustering uses top down (divisive) or bottom up (aggregative) approach to find clusters with no initial cluster center and now initial clusters number. Density based clustering (DBSCAN) is another clustering technique that can discover arbitrarily-shape clusters, which is used for mining spatial data. [15].

2.9.3 Association Rules Mining

Discovery of association relationship among large set of data items is useful in decision-making. A typical example for association rules mining is market basket analysis, which studies customer buying habits by finding associations between the different items that customers place in their baskets. An association rule is thus a relationship of the form: "$IF < A > THEN < B >$", where A and B are sets of items and A. Such a rule generation technique consists of finding frequent item sets $A \cap B = \emptyset$ (set of items, such as A and B satisfying minimum support and minimum confidence)

from which rules like $A \rightarrow B$ are generated. The measures support is the percentage of transactions that contain both the item sets. Thus:

$$Supp(A \cup B) = P(A\ U\ B) \tag{2.12}$$

$$Conf(A \rightarrow B) = \frac{P(A\ U\ B)}{P(A)} \tag{2.13}$$

Although both classification and association rules have an IF-THEN structure, association rules can have more than one item in the consequent part, whereas classification rules always have one attribute (class label) in the consequent. In other words, for classification rules, predicting attributes and the goal attribute. Predicting attributes can occur only in the rule antecedent, whereas the goal attribute occurs only in the rule consequent.

2.9.4 Regression

Regression analysis includes any techniques for modeling and analyzing several variables (criterion), when the focus is on the relationship between a dependent variable and one or more independent variables (predictor). More specifically, regression analysis helps us understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Linear regression is a form of regression where the relationship between variables is modeled with a straight line (linear equation), learned using the training data points. A straight line, through the input vector X (known as predictor variable) and the output vector Y (known as response variable), can be modeled as $Y = \alpha + \beta.X$ where α and β are the regression coefficient and slope of the line, computed as:

$$\beta = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{\sum_{i=1}^{n}(x_i - \bar{X})^2} \quad (2.12)$$

$$\alpha = \bar{Y} - \beta\bar{X} \quad (2.13)$$

where $\bar{X}$ and $\bar{Y}$ are averages of vector X and vector Y respectively.

2.9.5 Deviation Detection

This is also known as the process of detection of outliers. Outliers are those patterns that are distinctly different from the normal, frequently occurring, patterns, based on some measurement. These patterns can be found in some data objects that do not comply with the general behavior of the data. They are inconsistent from the remaining set of data. These data objects are called outliers.

The wide range of applications of outlier detection includes fraud detection, customized marketing, detection of criminal activity in e-commerce, network intrusion detection, and weather prediction. The different approaches for outlier detection can be broadly categorized into three types [15]:

- Statistical approach: Here, the data distribution or the probability model of the data set is considered as the primary factor.
- Distance-based approach: An object O in a data set T is a $DB(p, D)$-outlier if at least fraction p of the objects in T lies greater than distance D from O.
- Deviation-based approach: Deviation from the main characteristics of the objects is basically considered here. Objects that "deviate" from the description are treated as outliers.

2.10 Issues and Challenges in Data Mining

2.10.1 Data Issues

- **High dimensionality:** Datasets usually contain huge amounts of records, with considerably a large number of attributes. This affects the performance of the data mining algorithm not only in terms of running time, but also the efficiency and the accuracy of the produced model as several irrelevant features have to be considered during model training. Therefore, these considerations should be taken while developing a mining algorithm and can exploit the advantages of techniques such as dimensionality reduction, sampling, approximation methods as well as incorporation of domain specific prior knowledge.
- **Complex Types:** Databases may contain complex data objects such as: hypertext and multimedia, graphical data, transaction data, and spatial and temporal data. An efficient, specific algorithm should be developed to cope with these types of data, or special versions of existing techniques can be tailored to work on such types of datasets.
- **Missing, incomplete and noisy data:** The data preparation part plays an important role to solve such problems. As many errors may occur in recoding transactional data in the source systems, missing values and inconsistencies are born. This affects the quality of the generated model by the mining algorithms. Data cleaning techniques, more sophisticated statistical methods to identify hidden attributes and their dependencies, as well as techniques for identifying outliers are therefore required to address this issue.

2.10.2 Mining Techniques Issues

- **Problem Definition and Domain Characteristics:** A deep analysis on domain characteristics, available data nature and the problem to solve should be carried out before a specific mining model is recommended. This is needed as there is none that is equally applicable to a wide variety of data sets and can be called the universally best data mining technique.
- **Efficiency and accuracy:** Efficiency and accuracy of a data mining technique are key issues. Data mining algorithms should be efficient enough in terms of outcome, usability and confidence so much that the user should be able to rely on the results and take decisions upon them. A lot of effort is done to enhance the efficiency of already existing mining techniques as well as develop new ones that can work efficiently in some specific problem situations and fabricate more comprehensive results.

2.10.3 User Interaction Issues

- **Interpretation of the discovered patterns:** Some data mining techniques are preferred over others based on their ability to produce knowledge, represented in a natural language rules, graph or a tree, that is understandable, interpretable and traceable by the user. For example, neural networks classifiers and SVMs may produce better results than other algorithms such as rule induction. However, rule induction based classification can be preferable as they give the user insight into the discovered knowledge from his domain data.

2.11 Data Mining Applications

- **Spatial data mining.** A spatial database stores a large amount of space-related data, such as maps, preprocessed remote sensing or medical imaging data and VLSI chip layout data. They carry topological and/or distance information and are usually organized via a multidimensional structure utilizing data cubes. Spatial data mining refers to the extraction of knowledge like spatial relationships or other interesting patterns from large geo-spatial databases.
- **Web mining.** With the explosive growth of information sources available on the World Wide Web (WWW), it has become increasingly necessary for users to utilize automated tools in order to find, filter, and evaluate desired information and resources. Web mining can be broadly defined as the discovery and analysis of useful information from the WWW. In order to mine the web basically two ideas are used. Web content mining: here the idea is the automatic search and retrieval of the information. Web usage mining: the basic idea here is the automatic discovery and analysis of user access patterns from one or more web servers.
- **Text mining.** In recent days we can have databases, which contain large collections of documents from various sources such as news articles, research papers, books, digital libraries, e-mail messages, and various web pages which are called text databases or document databases. These text databases are rapidly growing due to the increasing amount of information available in electronic forms, such as electronic publications, e-mails, etc. Data stored in most text databases are semi-structured data, in that they are neither completely unstructured nor completely structured. For example, a document may contain a few structured fields, such as title,

authors, publication date, and category and so on, but also contain some largely unstructured text component such as abstract and contents. This type of text data presents challenges to traditional retrieval techniques. As a result, text-mining concepts are increasingly coming into light. Text mining goes one step beyond the traditional approach and discovers knowledge from semi-structured text data as well.

- **Image mining.** Actually image mining, i.e. mining the image databases, falls under the multimedia database mining, which also contains audio data, video data along with image data. Basically images are stored with some description against a particular image. Again images are nothing but some intensity values, which figure the image in terms of color, shape, texture etc. The mining task is based on using such information contained in the images. Based on this image mining techniques can be categorized in two places: description based retrieval, and content based retrieval.
- **Biological data mining.** Biological researches are dealing greatly with the development of new pharmaceuticals, various therapies, medicines and human genome by discovering large-scale sequencing patterns and gene functions. In the process of gene technology, DNA data analysis becomes significantly focused with data mining applications. Since the discovery of genetic causes for many diseases and disabilities and to discover new medicines as well as disease diagnosis, prevention, and treatment, DNA analysis is a must. The DNA sequences form the foundation of the genetic code of all living organisms. All DNA sequences are comprised of four basic nucleotides [i.e. Adenine (A), Cytosine (C), Guanine (G), and Thiamine (T)]. These four nucleotides are combined in different orders to form long sequences or chains in the structure of DNA. There are almost an unlimited number of ways that

the nucleotides can be ordered and sequenced which play important role in various diseases. It is a challenging task to identify such a particular sequence from among the unlimited sequences, which are actually responsible for various diseases. Now people are trying to use data mining techniques to search and analyze these sequence patterns. In addition to DNA sequencing, linkage analysis and association analysis (where the structure, function, next generation genes, co-occurring genes, etc.) are also studied. For all these, machine learning, association analysis, pattern matching, sequence alignments, Bayesian learning, etc. techniques are being used in bioinformatics recently.

- **Distributed Data Mining.** The evolution of KDD system from being centralized and stand alone along the dimension of data distribution signifies the emergence of Distributed Data Mining (DDM). Specifically, when data mining is undertaken in an environment, where users, data, hardware, and the mining software are geographically dispersed, will be called DDM. Typically such environments are also characterized by the heterogeneity of data, various user bases, and large data volumes.

2.12 Summary

Part 1 has presented Ant Colony Optimization technique which is a field in swarm intelligence inspired by the behavior of biological ants. (ACO) is a meta-heuristic algorithm that is used to solve combinational optimization problems (COP). Artificial ants live in a virtual world, called a construction graph, which represents the solution search space for the given problem. Elements in the construction graph are the solution components which each ant selects while traversing the graph to construct a solution. The pheromone deposited on the construction graph is the way of communication and sharing

information among the ants in the colony. The ant drops pheromone proportional to the quality to the solution that it constructed. The pheromone is considered the guide for subsequent ants to decision components in the construction graph with good or promising quality. The ant chose the next decision probabilistically according to the amount of pheromone associated with it and a heuristic value for that decision component. ACO has proven to be quite efficient and flexible. Ant colony optimization algorithms are currently state-of-the-art for solving many COPs.

Part 2 has given a wide overview on data mining and knowledge discovery concepts and issues. Data mining has become very important since the enormous growth of data in different domains with various types and formats. Knowledge discovery is known as the process of finding and extracting hidden useful pattern from raw data. There are three basic phases in knowledge discovery process. The first step is data preparation which involves data cleansing, integration, selection, reduction and transformation to be in a valid form processing. The second step, which is the core step in the process, is data mining. Several techniques of data mining exist and are applied to solve various types of knowledge discovery needs. Such techniques include classification, clustering, association rule discovery, regression, and outlier detection. Data mining has been utilized in several domains like web mining, text mining, image mining, spatial data mining, and biological data mining.

Chapter 3

ANT-MINER

3.1 Introduction

Ant-Miner was proposed by Parpinelli *et al*. [20] in 2002. Utilizing ACO techniques, Ant-Miner is a data mining algorithm that is designed to generate classification rules from a given dataset. As for a typical ACO algorithm, the ant is considered an agent that incrementally constructs and modifies a solution from the construction graph to the given problem. The problem is to build a classification model and the solution is a set of rules that can be used for classification. Therefore, each ant in the swarm tries to construct a rule that can be used in the classification model rule set. Basically, Ant-Miner is a rule-based induction algorithm that makes use of ACO's collective behavior. As mentioned before, the generated rules are expressed in the following form:

IF <Conditions> THEN <class>

The <conditions> part (antecedent) of the rule contains a logical combination of predictor attributes, in the form: term1 AND term2 AND... . Each term is a triple <attribute, operator, value>, where value is a value belonging to the domain of attribute, and the operator element in the triple is always "=". The original version of Ant-Miner deals only with categorical attributes. As such, continuous (real-valued) attributes are discretized as a preprocessing step. However, several modifications have been done to the algorithm to come up with new versions that can cope with continuous attributes efficiently. Some of these versions are mentioned in the next chapter. The <class> part

(consequent) of the rule contains the class predicted for cases in given dataset whose predictor attributes satisfy the <conditions> part of the rule.

Ant-Miner discovers an ordered list of classification rules. For each ant trial, an ant attempts to discover a rule by selecting terms probabilistically according to a heuristic function and pheromone amount for this term. After a rule is constructed, the ant updates the pheromone on its trial to lead next ants in their paths. The best rule is selected among the ants that have constructed rules and added to the discovered rule set. The algorithm is repeated until the discovered rules cover a sufficient portion of the given dataset. The first part of this chapter describes the algorithm of the original Ant-Miner in detail.

3.2 Ant-Miner Algorithm

The following pseudo code describes the outline of the original Ant-Miner algorithm. Algorithm 3.1 – Original Ant-Miner is taken from [20]. The following is detailed description on the algorithm.

Algorithm 3.1 - Original Ant-Miner.

TrainingSet = {all training cases};

DiscoveredRuleList = [ ]; /* initialize rule list with empty list */

WHILE (TrainingSet > Max_uncovered_cases)

$t = 1$; /* ant index, and also rule index */

$j = 1$; /* convergence test index */

Initialize all trails with the same amount of pheromone;

REPEAT

Ant_t starts with an empty rule and incrementally constructs rule R_t;

by adding one term at a time to the current rule;

Prune rule R_t; /* remove irrelevant terms from rule */

Update the pheromone of all trails by increasing pheromone in the trail followed by Ant_t (proportional to the quality of R_t) and decreasing pheromone in the other trails (simulating pheromone evaporation);

IF (R_t is equal to R_{t-1}) /* update convergence test */

THEN $j = j + 1$;

ELSE $j = 1$;

END IF

$t = t + 1$;

UNTIL (i ≥ No_of_ants) OR (j ≥ No_rules_converg)

Choose the best rule R_{best} among all rules R_t constructed by all the ants;

Add rule R_{best} to DiscoveredRuleList;

TrainingSet = TrainingSet - {set of cases correctly covered by R_{best}};

END WHILE

As an ACO-based algorithm, the decision components in the construction graph of Ant-Miner are the available attribute values, by which a rule's antecedent terms can be constructed. The algorithm consists of two nested loops: the outer loop where a single rule in each iteration is added to the discovered rule list and the inner loop where an ant in each iteration constructs a rule as follows. Each ant in the colony attempts to construct a rule's antecedents by selecting terms probabilistically according to a heuristic value (using a heuristic function that will be discussed later) and pheromone amount for this term. As an ant starts, it has an empty rule (a rule with no term in its antecedent and no

consequent class). As the ant moves through the construction graph (which will be described later), it tries to construct its empty rule premises by adding one term at a time to have a current partial rule corresponding to the current partial path followed by that ant in the construction graph.

The ant keeps adding terms one-at-a-time to its current partial rule until it faces a stopping condition that prevents it from adding more terms to its current rule it is constructing. This stopping condition can arise in two cases: the first case is when any term that could be added to the rule would make the rule cover a number of cases; less than a user-specified threshold, called **min_cases_per_rule** (minimum number of cases that should be covered by a rule). This condition exists in order to avoid constructing rules with low convergence, which may lead to extra running time for the algorithm and over fitting in the generated rules set. The second case that makes the ant stop is when all attributes have already been used by the ant, so that there is no more attributes to be added to the rule premises.

As the ant faces one of the two stopping condition, the ant has now completed building a rule antecedents (it has completed its path through the construction graph). The rule consequent is then chosen by determining the class value with maximum occurrence in the cases matching the rule antecedents. The constructed rule (premises with consequent class) is pruned in a post-processing step to remove irrelevant terms that might have been unduly included in the rule. Pruning the rule premises tends to enhance the quality of the rule in term of coverage and accuracy, since irrelevant terms may have been included in the rule due to stochastic variations in the term selection procedure and/or due to the use of a shortsighted, local heuristic functions - which consider only

one-attribute-at-a-time, ignoring attribute interactions. The pruning procedure will be described later in this chapter.

When an ant completes its rule, the amount of pheromone is updated on its trial according to the quality of the generated rule. Then another ant starts to construct its rule, using the new amounts of pheromone to guide its search. This process is repeated for at most a predefined number of ants. This number is a system parameter, called **no_of_ants**. However, this iterative process can stop earlier, when convergence occurs. Convergence occurs when stagnation is detected as the current ant has constructed a rule that is exactly the same as the rule constructed by the previous **no_rules_converg** – 1 ants. **no_rules_converg** (number of rules used to test convergence of the ants) is also a system parameter. This stopping criterion detects that the ants have already converged to the same constructed rule, which is equivalent to converging to the same path in real Ant Colony Systems. The best rule amongst the rules constructed by all ants is selected, added to the discovered rule set and considered for the classification rules model. The other rules are discarded. This completes a single iteration of the algorithm. After the best rule among the ants trial is selected, all the cases covered by this rule are removed from the training set.

This course of action is considered an iteration of the outer loop. When the next iteration of the Ant-Miner algorithm starts, it runs in a reduced training set. This process is repeated for as many iterations as necessary to find rules covering a sufficient portion of the cases in the training set. This sufficient portion is reached when the number of uncovered cases in the training set is less than a predefined threshold, called **max_uncovered_cases** (maximum number of uncovered cases in the training set), at which the algorithm stops execution. When a sufficient portion of the training set cases is

covered by the discovered rules, the search for further rules stops. At this point the system has generated a classification model consisting of an ordered rule list (in order of discovery), which will be used to classify new cases, unseen during training.

A default rule is added to the last position of the rule list. The default rule has an empty antecedent (i.e. no condition) and has a consequent predicting the majority class in the set of training cases that are not covered by any rule. This default rule is automatically applied if none of the previous rules in the list cover a new case to be classified.

Once the rule list is complete, it is ready to classify a new test case set. This is done by applying the discovered rules, in order. The first rule that matches the new case is applied and case is assigned the class predicted by that rule's consequent.

3.3 Construction Graph

As was described in Chapter 2, the ACO technique represents the solution space for a given problem as a graph, from where an ant can construct a solution. The solution is basically the path that the ant took in its trial from source to sink. The decisions (nodes) that the ant selected during its path are considered the components of the solution to the problem. In Ant-Miner, since the solution to be constructed for the classification problem is a rule that consists of set of terms, then the terms are considered the solution components for the current problem. Accordingly, the construction graph should contain all the available terms than can be used to construct a rule (solution). The Ant-Miner construction graph is typically a graph consisting of nodes, where each node represents an attribute value for each attribute values in the dataset. The set of nodes (N) in the construction graph nodes is:

$$N = \bigcup_{i=1}^{n} v_{ij} \ , \qquad j \in \{1,2,\dots,l\}$$

where i is i-th attribute, n is the number of nodes and v_i is the j-th value of i-th attribute. Thus, each node is selected to represent a term $< A_i = V_{ij} >$. The set of terms that the ant chosen in its path represents a rule:

$$\text{“}IF < A_i = V_{ij} > AND < A_k = V_{kl} > AND \ \dots\text{”}$$

Each node in the construction graph contains an amount of pheromone. At the beginning of each iteration, the pheromone is initialized for each term with the same value given by the function:

$$\tau_{ij}(t=0) = \frac{1}{\sum_{r=1}^{a} b_r} \tag{3.1}$$

where:

- a is the total number of attributes.
- b_r is the number of values in the domain of attribute i.

The construction graph does not include the class attribute values; it only includes terms contributing in constructing the rule premises. The rule consequent (class) is selected after the rule antecedents are constructed by determining the class value with maximum occurrence in the cases matching the rule antecedents.

Each node has a Boolean property indicating whether it is still available for use or not. A node can be ignored from the construction graph if all the cases in the training set containing the value of the attribute that the node represents are covered by the discovered rules. The Boolean property of the node helps the ant to consider the node during term selection or to ignore it.

A heuristic value is also associated with each node that represents the local quality of this term to be selected, which affects the node selection probability by the current ant. This value is updated after each rule is discovered and the training set is reduced. The used heuristic function is described in one of the following sections.

The amount of the pheromone is updated on each node after each ant trial to influence other ants' selection of the terms in the next trials. Rule construction and pheromone update are discussed each in separate following sections.

3.4 Rule Construction

A rule is constructed incrementally by adding a terms to the current partial rule that an ant holds. As mentioned before, $term_{ij}$ is in form of $A_i = V_{ij}$, where A_i is i-th attribute and V_{ij} is the j-th value of the domain of A_i. The probability that $term_{ij}$ is selected by the ant to be added to the current partial rule is given by the following equation:

$$P_{ij} = \frac{\eta_{ij} \cdot \tau_{ij}(t)}{\sum_{r=1}^{a} \sum_{s=1}^{br} (\eta_{rs} \cdot \tau_{rs}(t))} \tag{3.2}$$

where:

- η_{ij} is the value of a problem-dependent heuristic function for $term_{ij}$.
- $\tau_{ij}(t)$ is the current amount of the pheromone on $term_{ij}$ for the current ant through its current trial.
- a is the total number of attributes.
- b_r is the number of values in domain of the i-th attribute.

given that the Boolean property that indicates whether $term_{ij}$ can still be used is true.

As shown in equation (3.1), the probability of a term to be selected is calculated according to two components. The first is η_{ij} which is the value of a problem-dependent heuristic function, and the second is τ_{ij} which is the amount of the pheromone on $term_{ij}$. The first component is a problem-dependent heuristic function η_{ij}, which is a measure of the predictive power of $term_{ij}$. The higher the value of η_{ij} the better the $term_{ij}$ is in the context of the given problem (classification), and so the higher the probability of it being selected. The heuristic value for each term is calculated by the same function, which will be described in the following section.

The second component that affects the probability of selecting a term is the amount of pheromone τ_{ij} currently associated with $term_{ij}$, which is entirely dependent on the paths that other ants took during their previous trials in rule construction. As mentioned in Chapter 2, in the typical ACO technique, the amount of the pheromone on the construction graph acts as an indirect way of communication between the ants in the colony. It represents the experience of the pervious ants in constructing solution and gives advice to the next ants about the good paths to take in their trials to attempt in constructing better ones. In the beginning, all the terms have the same amount of the pheromone. However, as soon as an ant finishes its path, the amount of pheromone in each term visited by the ant is updated, as will be explained in detail shortly. The amount of the pheromone to be dropped on the trail depends on the quality of the rule constructed by taking this path; the better the quality of the rule constructed by the ant, the higher the amount of pheromone added to the terms selected during the trail. With time, and after several ants have attempted to construct rules, the "best" path (collection of terms to

be selected) will have a greater probability to be taken by upcoming ants as the amount of pheromone on this path increase.

The term to be selected and added to the partial rules is subjected to some restrictions: $term_{ij}$ cannot be selected if the current partial rule contains $term_{kj}$ (i.e. if the current partial rule contains another value V_k from the same domain of the attribute A_i. Another restriction is that a term cannot be added to the current partial rules if this makes the extended partial rule cover less than a predefined minimum number of cases, called the **min_cases_per_rule** threshold, as mentioned previously in section 3.2.

In rule construction process, the ant builds the rule premises only, without specifying the rule consequent to be assigned to the rule. The selection of the class that will be the rule consequent is decided afterwards. After rule premises are completed, the system chooses the rule consequent (predicted class) that maximizes the quality of the rule. This is done by assigning to the rule consequent the majority class among the cases covered by the rule.

3.5 Heuristic Function

Each node in the construction graph has a current heuristic value that represents the local quality of this node to be selected as part of a solution for the current problem context. This value is calculated for each node with the same problem-dependent heuristic function. As for Ant-Miner, the node in the construction graph represents a term that could be added to a rule. The context is a classification problem. The heuristic value to be calculated an estimate of the quality of a given term, with respect to its ability to improve the predictive accuracy of the rule. This heuristic function is based on information theory, introduced by T. Cover and J. A. Thomas in [5].

More precisely, the value of the heuristic function for a term involves a measure of the entropy (or amount of information gain) associated with that term [21]. For each $term_{ij}$ of the form of $< A_i = V_{ij} >$, where A_i is *i*-th attribute and V_{ij} is the *j*-th value of the domain of A_i, its entropy is given the following equation:

$$Entropy(T_{ij}) = -\sum_{w=1}^{k}\left(\frac{FreqT_{ij}^{w}}{|T_{ij}|}\right) \times log_2\left(\frac{FreqT_{ij}^{w}}{|T_{ij}|}\right) \quad (3.3)$$

where:

- k is the number of classes.
- $|T_{ij}|$ is the total number of cases in partition T_{ij} (partition containing the cases where attribute A_i has value V_{ij}).
- $FreqT_{ij}^{w}$ is the number of cases in partition T_{ij} that have class w.

If the value of $Entropy(T_{ij})$ is high, this means that value V_{ij} in attribute A_i is more uniformly distributed among the classes, and so the lower the predictive power of $term_{ij}$. The terms to be selected should have a high predictive power to be added to the current partial rule. Therefore, in Ant-Miner, the higher the value of $Entropy(T_{ij})$, the smaller the probability of an ant choosing $term_{ij}$ to be added to its partial rule.

The value of the heuristic function is normalized. The resultant normalized, information-theoretic heuristic function given by the following equation:

$$\eta_{ij} = \frac{log_2(k) - Entropy(T_{ij})}{\sum_{r=1}^{a}\sum_{s=1}^{b} log_2(k) - Entropy(T_{rs})} \quad (3.4)$$

where:

- a is the total number of attributes.

- b_i is the number of values in domain of the *i*-th attribute.
- $Entropy(T_{ij})$ is the entropy of $term_{ij}$ calculated by equation (3.3)
- k is the number of classes.

$Entropy(T_{ij})$ for $term_{ij}$ is always the same regardless the content of the current partial rule. $Entropy(T_{ij})$ is calculated for each $term_{ij}$ as a preprocessing step before each outer iteration in the Ant-Miner algorithm.

If attribute A_i does not occur in the training set, then $|T_{ij}| = 0$. In this case, $Entropy(T_{ij})$ is set to its maximum value; $log_2(k)$. This corresponds to assigning to $term_{ij}$ the lowest possible predictive power. If all the cases in the partition T_{ij} belong to the same class then $Entropy(T_{ij}) = 0$. This corresponds to assigning to $term_{ij}$ the highest possible predictive power. Note that the value of $Entropy(T_{ij})$ varies in the range:

$$0 < Entropy(T_{ij}) < log_2(k)$$

3.6 Rule Pruning

The main goal of rule pruning is to remove irrelevant terms that might have been unduly included in the rule. As mentioned above, Rule pruning potentially increases the predictive power of the rule, by increasing its coverage without sacrificing its confidence. This also helps to avoid it over-fitting to the training data. Simpler rules generated after rule pruning are more easily interpreted by the user as they are shorter and more general. That was another motivation for pruning the rules.

For each ant constructing a rule, as soon as the ant completes the construction of its rule, the rule pruning procedure is performed. The search strategy of rule

pruning procedure used in Ant-Miner is very similar to the rule pruning procedure suggested by Quinlan [22], although the rule quality criterion used in the two procedures are very different from each other.

The basic idea is to iteratively remove one-term-at-a-time from the rule while this process improves the quality of the rule. In the first iteration one starts with the full rule. Then one tentatively tries to remove each of the terms of the rule – each one in turn – and computes the quality of the resulting rule, using the quality function defined by equation (3.5) . This step may involve re-assigning another class to the rule, since a pruned rule can have a different majority class in its covered cases. The term whose removal most improves the quality of the rule is effectively removed from the rule, completing the first iteration. In the next iteration one removes again the term whose removal most improves the quality of the rule, and so on. This process is repeated until the rule has just one term or until there is no term whose removal will improve the quality of the rule. [20]

Another rule pruning procedure was introduced by A. Chan and A. Freitas in [4]. This new procedure has enhanced the quality of the generated rules. A brief description of the procedure is mentioned in the following chapter.

3.7 Pheromone Update

Each node in the construction graph, which represents a term to be selected by an ant for rule construction, has current amount of pheromone associated with it. This amount of pheromone changes as ants select nodes though their trials and drop pheromone on the selected nodes during their paths. All the terms are initialized with the same amount of pheromone. The initial amount of pheromone deposited at each path

position is inversely proportional to the number of values of all attributes, as given by the aforementioned equation (3.1).

As an ant finishes constructing the rule, the amount of pheromone in all nodes in the construction graph is updated. This pheromone updating has two operations:

a) Increasing the amount of pheromone associated with each term in the construction graph that was selected during the rule construction (terms occur in the constructed rule).
b) Decreasing the amount of pheromone associated with each term in the construction graph that was not selected in during the rule construction (terms that does not occur in the constructed rule). This acts as pheromone evaporation in the typical ACO algorithm.

As for increasing the pheromone on used terms – which is also known in ACO systems as pheromone reinforcement – each ant drops pheromone on the terms that were selected through its path during its trial after it completes rule construction. If $term_{ij}$ occurs in the constructed rule, this operation increases the probability of $term_{ij}$ being selected by ants in the future trials, as the current ant acknowledges the benefit of selecting such term. The amount of pheromone being dropped on each $term_{ij}$, selected by the ant, through its path is proportional to the quality of the rule constructed by the ant using these terms. The better the rule is, the higher the increase in the amount of pheromone for each $term_{ij}$ occurring in the rule.

The quality of the rule constructed by an ant, denoted by Q is computed by the formula: $Q(R_t) = sensitivity(R_t) \times specificity(R_t)$, as defined in the following equation:

$$Q(R_t), = \frac{TP}{TP+FN} \times \frac{TN}{TN+FP} \tag{3.5}$$

where:

- R_t is the rule constructed by current ant_t
- *TP* (true positives) is the number of cases covered by the rule that have the class predicted by the rule.
- *FP* (false positives) is the number of cases covered by the rule that have a class different from the class predicted by the rule.
- *FN* (false negatives) is the number of cases that are not covered by the rule but that have the class predicted by the rule.
- *TN* (true negatives) is the number of cases that are not covered by the rule and that do not have the class predicted by the rule.

The larger the value of Q, the higher the quality of the rule. Note that Q varies within the range: $0 < Q < 1$. Pheromone update is performed according to the following equation:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t).Q \tag{3.6}$$

This formula is applied for each $term_{ij}$ contained in the constructed rule. Therefore, the value of the pheromone associated with each term in the constructed rule is increased by an amount proportional to the rule quality calculated via formula (3.5).

Decreasing the pheromone in unused terms corresponds to the phenomenon of pheromone evaporation in real ant colony systems. In typical ACO systems, evaporation is obtained via an evaporation factor ρ to be multiplied to each τ_{ij} in the construction graph after pheromone reinforcement (as was described in the Chapter 2).

In this original version of Ant-Miner, the pheromone evaporation process is simulated by normalizing the value of each pheromone τ_{ij} for each $term_{ij}$ after pheromone reinforcement. More precisely, this normalization is performed by dividing the value of each τ_{ij} by the summation of all τ_{ij} on each node in the construction graph. When a rule is constructed, only the terms occurring in the rule constructed by an ant have their amount of pheromone increased by equation (3.6). Therefore, at normalization time the amount of pheromone of an unused term (the terms that did not occurred in the constructed rule) will be computed by dividing its current value (the pervious value that was not increased) by the total summation of pheromone for all terms (which was increased as a result of reinforcing the pheromone amount on the used terms). The final effect will be to reduce the normalized amount of pheromone for each unused term. Used terms will, have their normalized amount of pheromone increased due to application of equation (3.6).

3.8 Algorithm Parameters

This original version of Ant-Miner algorithm has the following parameters:

- **Number of Ants** (no_of_ants): this is also the maximum number of ant trials for constructing rule in each iteration of the system. In each iteration, the best rule constructed in that iteration is considered a discovered rule. Note that the larger the no_of_ants, the more candidate rules are evaluated per iteration, but the slower the system becomes.
- **Minimum number of cases per rule** (min_cases_per_rule): each rule must cover at least min_cases_per_rule, this guarantees certain degree of generality an coverage

in the discovered rules. This helps avoiding over-fitting to the training data and decreasing number of overall system iterations needed,

- **Maximum number of uncovered cases** (max_uncovered_cases): this threshold tells the system when to stop. The process of rule discovery is iteratively performed until the remaining cases in the training set that are not covered by any of the discovered rule is less than this threshold.
- **Number of rules used to test convergence of the ants** (no_rules_converg): If the current ant has constructed a rule that is exactly the same as the rule constructed by the previous no_rules_converg –1 ant, then the system concludes that the a stagnation has occurred, no ant can take another path to construct a different (possibly better) rule, and the whole colony has converged to a single rule (path).

The experimental results that have been published in [20] – and will be discussed in the following section – were produced by running the algorithm with the following values of the aforementioned parameters:

- Number of Ants (no_of_ants) = 3000.
- Minimum number of cases per rule (min_cases_per_rule) = 10.
- Maximum number of uncovered cases in the training set (max_uncovered_cases) = 10;
- Number of rules used to test convergence of the ants (no_rules_converg) =10.

The next section will show the computational result that was produced by experimented the running the original Ant-Miner algorithm and was published in [20]. A brief discussion on the efficiency of the algorithm according to the published results is included in the following section. Some issues and considerations on the algorithm will

be highlighted as they represent the triggers for other enhancements and related work done on this original version, and the motivation of the modifications proposed in this thesis.

3.9 Ant-Miner Results Discussion

Ant-Miner has been evaluated across six public-domain data sets from the UCI (University of California at Irvine) data set repository (2000) [26]. The detailed description of the used datasets characteristics and the experimental results can be found in [20].

In three data sets, namely Wisconsin breast cancer, Hepatitis and Heart disease, Ant-Miner discovered a rule set that is both simpler and more accurate than the rule set discovered by C4.5. In one data set, Ljubljana breast cancer, Ant-Miner was more accurate than C4.5, but the rule sets discovered by Ant-Miner and C4.5 have about the same level of simplicity. (C4.5 discovered fewer rules, but Ant-Miner discovered rules with a smaller number of terms.) Finally, in two data sets, namely Tic-tac-toe and Dermatology, C4.5 achieved a better accuracy rate than Ant-Miner, but the rule set discovered by Ant-Miner was simpler than the one discovered by C4.5.

It is also important to notice that in all six data sets the total number of terms of the rules discovered by Ant-Miner was smaller than C4.5's one, which is a strong evidence of the simplicity of the rules discovered by Ant-Miner.[20]

As for the first implementation of the algorithm, Ant-Miner has proved to be a very promising technique for classification rules discovery. Ant-Miner generates a fewer number of rules, less number terms per each rules, and performs competitively in terms of

efficiency compared to C4.5 algorithm. Hence, it has been a focus area of research and a lot of modification has been done to it in order to increase its efficiency.

Considering some issues in the original version of the Ant-Miner algorithm:

a) This version copes only with the categorical attributes, and the continuous attributes should be discretized as a pre-processing step. Coping with real-valued attributes would be an important feature to avail.

b) The rule consequent (rule class) is selected after rule antecedents' construction. Selecting the consequent of the rule before rule construction should enhance the quality of the generated rules and improve its running time.

c) Pheromone is associated with graph nodes, which represent the available terms to construct rules, unlike the typical ACO techniques, where pheromone is associated with edges between nodes. Applying such an idea can introduce terms dependency and can generate better rules.

d) As for any ACO system, a balance between exploration and exploitation is needed. In the current implementation of the algorithm, exploitation is dominant as all the ants follow the pheromone of all previous ants. Giving some personality to each ant can improve the exploration part and enhance the algorithm performance.

e) Ant-Miner can by hybridized with other evolutionary computation techniques.

f) Different heuristic functions and quality evaluation functions can be tried, and different pruning procedure can be applied.

Most of the aforementioned issues have been tacked in further research concerning this area of interest. The following chapter describes the work that has been done on the Ant-Miner algorithm and focuses on the work related to the modifications on the algorithm introduced in the thesis.

3.10 Ant-Miner Implementation

The work of this thesis has used a re-written Ant-Miner program that was built using C# and Microsoft.NET technologies. Both of the original and the extended version of the Ant-Miner algorithm were developed using the same aforementioned technology. The following subsection describes the used data structures for the Ant-Miner implementation and some of the code optimizations that were used. A comprehensive profiling and analysis for the execution behavior of the code is exhibited as well. This is done in order to point to the time consuming operations and give some speculation about the need of the proposed extensions as well as comparing the execution performance of these extensions to the original one.

3.10.1 Data Structures and Operations

- **Construction Graph Node Representation:** A node in the construction graph is the decision component that an ant selects to construct its solution. In Ant-Miner, the decision component is the attribute value that represents a term in a rule. The following code shows the implementation of the node in the code.

```
public struct Node
{
      public int AttributeIndex;
      public int ValueIndex;
      public int [] ValueFrequency;
      public double PheromoneAmount;
      public double HeuristicValue;
      public double Probability;
      public bool UnusableValue;

 }
```

The node is represented as a structure which contains the data fields necessary to describe the node entity. `AttributeIndex` is the index of the attribute in the dataset. `ValueIndex` is the index of the value in the domain of this attribute.

`ValueFrequency` is an array which contains the frequency of occerance of this attribute value for each class (this is needed to faclitate calculating the heuristic value based on the information gain). `PheromoneAmount` represents the current amount of pheromone associated with this node. `HeuristicValue` represents the value for the heuristic function for this attribute value. Both `PheromoneAmount` and `HeuristicValue` are used to calculate the value of the `Probability` field for a given node. A boolean field is associated with each node, named `UnusableValue`, used to indicate whether this value is still in use or not. This field is set to true if it occures less than **min_cases_per_rule** in the remaining cases in the training set. If so, this attribute will not be considered for selection in rule construction procedure.

- **Construction Graph Representation:** The construction graph contains of all nodes (the decisions components) in which an ant traverses to construction a rule (solution). The construction graph is represented as a two-dimensional array of nodes, which is declared in the swarm class and initialized in `BuildConstructionGraph()` method, as shown in the following code.

```
private Node[][] _constructionGraph;

…

private void BuildConstructionGraph()
{

this._constructionGraph = new
Node[this._trainingSetDataTable.Columns.Count][];

for (int attributeIndex = 0; attributeIndex <
this._trainingSetDataTable.Columns.Count; attributeIndex++)
{
List<string> values = this.GetDistinctAttributeValues(attributeIndex);

this._constructionGraph[attributeIndex] = new Node[values.Count];
```

```
for (int valueIndex = 0; valueIndex <
this._constructionGraph[attributeIndex].Length; valueIndex += 1)
{

this._constructionGraph[attributeIndex][valueIndex].AttributeIndex =
attributeIndex;
this._constructionGraph[attributeIndex][valueIndex].ValueIndex =
valueIndex;
this._constructionGraph[attributeIndex][valueIndex].ValueFrequency = new
int[numberOfClasses];
}

}

      …

}
```

An additional array is used to support the construction graph, `attDistinctLeft`, is used to keep track of the remaning values in the attribute domain which still in used (`UnusableValue=flase`). This helps the ants to discard the attribute whose values became unsuable when constructing a rule. For exmaple, when `this._constructionGraph[i][j].UnusableValue` is set to true, `attDistinctLeft[i]` which represents the number of distinct values in attribute `i` is decreased. When `attDistinctLeft[i]` becomes 0, this attribute will not be used for rule construction in further iterations.

- **Ant Representation:** The ant entity is represented as a class that contains the data fields for an ant object to help constructing a rule. The following is the implementation code for the ant entity.

```
public class Ant
{

        private int _antNumber;
        private int[] _ruleAntecedents;
        private int _ruleclassIndex;
        private double _ruleQuality;
        private List<int> _instancesIndexList;
        private bool[] _memory;

…
```

```
}
```

As shown in the previous code, each ant has an array of integers, `ruleAntecedents`, which repesents the partial rule the the ant is currently constucting. The array elements are intially intialized with -1, and as the ant selects a node from the construction graph, the value index is added to the element of the array conspoding to its attribute index. Moreover, `_memory` array keeps track of whether an ant has selected a value for a given attribute or not. For example, if `_memory[i]=false`, this means that an ant can select a value from the domain of attribute `i` from the construction graph. Each ant also keeps track of the instance index of the cases that are coverd by the current rule, using `_instancesIndexList`. This helps applying the minimum cases coverd by a rule when adding a new term to the rule by searching in the occerances of this term only in `_instancesIndexList`. It also helps in determinding the rule class by calcualting the class value that has the highest occerance in the cases of `_instancesIndexList`.

- **Ant Colony Representation:** The AntColony class is the core of the Ant-Miner program. In contains all the algorithm parameters as well as the methods needed for running the algorithm. The following class diagram shows the design of the Ant-Miner class. Note that the class contains other data fields and helper methods that are not shown in the class diagram as they are only used for housekeeping operations.

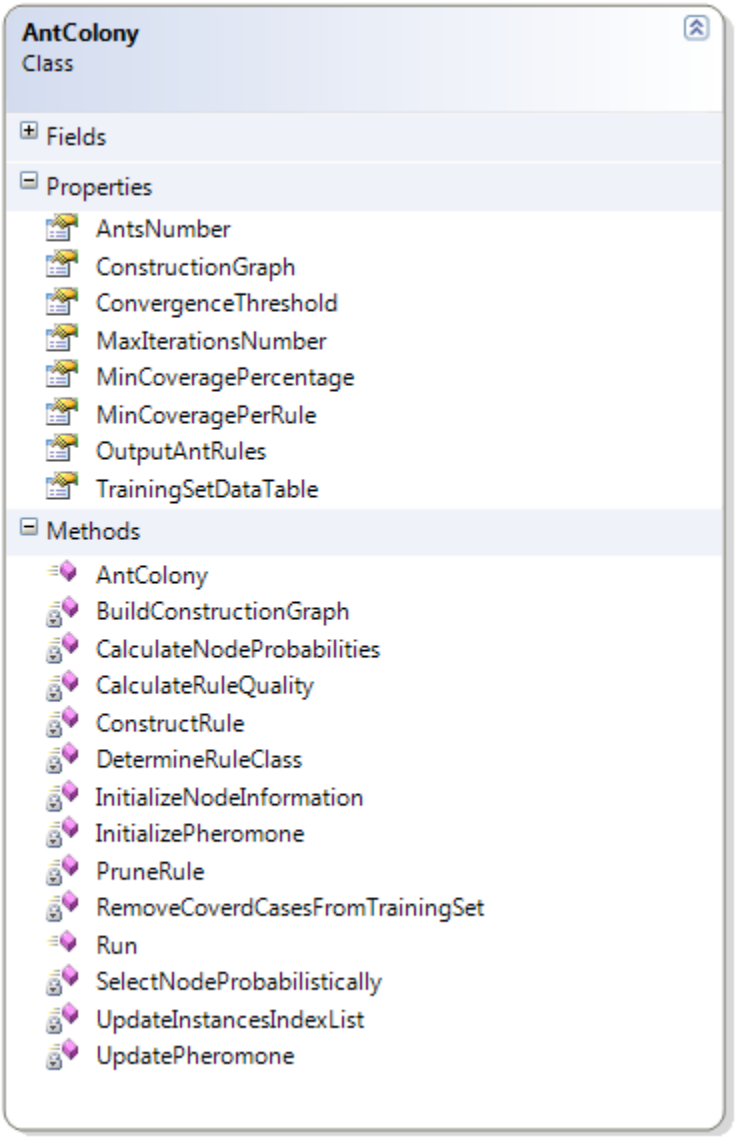

Figure 3.1 - AntColony Class Diagram.

As shown in figure 3.1, `AntColony` class contains the properties needed for running the Ant-Miner algorithm. It has `ConstructionGraph`, which represents the current instance of the construction graph for the dataset at hand. The `AntsNumber` property represents the number of permitted trials per iteration. The `MaxIterationsNumber` value is the maximum global iterations that the Ant-Miner can perform before it covers the minimum required cases from the training set by the generated rules. The

`ConvergenceThreshold` indicates when to determine that the ants have converged on a specific rule. This is considered when the current ant has constructed a rule that is the same as the previous ConvergenceThreshold-1 ant rules. . `MinCoveragePercentage` represents the minimum cases to be covered by the constructed rules before stopping to generate more rules. Finally, `OutptAntRules` is the generated rule list.

When a new object of the `AtColony` class is instantiated, the aforementioned parameters are passed to its constructor to be set, and the `BulildConstructionGraph()` is called, which is considered as a pre-processing operation for running the algorithm program.

The `Run()` method is the main operation for executing the Ant-Miner Algorithm. It starts as follows:

```
while (this._currentIterationNumber < MaxIterationsNumber &&
this._currentCoverage < this.MinCoveragePercentage)
{

      this.InitializePheromone();
      this.InitializeNodeInformation();

      …

}
```

At the beginning of each iteration, the methods `InitializeNodeInformation` and `InitializePheromone` are called. Both of them are called only once at the beginning of each iteration. Then, the inner loop for ant trials begins to discover a rule for the current iteration. As follows:

```
…

for (_currentIterationNumber = 0; _currentIterationNumber <
this.AntsNumber && !convergence; _currentIterationNumber++)
{
```

```
this._currentAnt = new Ant();
this.ConstructRule(this._currentAnt);
this.DetermineRuleClass(this._currentAnt);
this.CalculateRuleQuality(this._currentAnt);
this._currentAnt = this.PruneRule(this._currentAnt);
generatedAnts[_currentIterationNumber] = this._currentAnt;

if (_currentAnt.RuleQuality > generatedAnts[bestAntIndex].RuleQuality)
   bestAntIndex = _currentIterationNumber;

this.UpdatePheromone(generatedAnts[bestAntIndex]);

convergence=TestConvergence();
}
this.OutputAntRules.Add(generatedAnts[bestAntIndex]);
this.RemoveCoverdCasesFromTrainingSet(generatedAnts[bestAntIndex]);

…
```

The previous code shows the logical implementation of each iteration of Ant-Miner. On each iteration, several trials to discover a rule are performed until the maximum of trials is reached (`AntsNumber`) or `convergence` became true. In each trial, a new ant is created and referenced by `_currentAnt`.the `_currentAnt` constructs a rule by invoking `ConstructRule()` method. This method calls `SelectNodeProbablistically()`, which inturn calls `CalculateNodeProbabilities()` method and uses a rolette-wheel procedure to choose a node. Afer rule atecednets are chosen, `DetermineRuleClass()` is called, which uses the `_instancesIndexList` associated with `_currentAnt` to determined the class with the highest occerance in the covered cases by the current rule. Then the rule quality is claculated and set to `_currentAnt.RuleQuality`. The prunning procedure then takes palce by invoking `PruneRule()` method, which iteratively calls `DetermineRuelClase()` and `ClauclateRuleQuality()` methods after removing term by term. After that, the ant with the constructed, prunned rule is added to the `generatedAnts` and the index of the best ant is updated.

The best rule is selected after several ant trials. This discovered rule is added to OutputAntRules and the covered cases by this rule are removed from the training set.

3.10.2 Execution Profiling and Analysis

The performance of the algorithm has been discussed through the quality of its output in the section 3.9. The quality of the rule set generated using Ant-Miner was evaluated in terms of its classification accuracy and comprehensibility (number of rules and number of terms per rule). However, the execution of the algorithm should be profiled and analyzed in terms of running time. Such profiling helps in indicating which operation takes a longer time in execution, and how any modification to the algorithm could affect the running time. For example, a modification may be applied on the algorithm that increases the number of trials needed to converge on a rule per iteration, yet it could decrease the actual running time of a single iteration. Another modification could decrease the overall iterations needed to stop execution, on the other hand, it might be using a complex heuristic function or quality evaluation function that increases the overall running. The following table exhibits an execution profile of the algorithm on CarEvaluation dataset (see section 9.2 Chapter 9).

A metric of measure is presented to profile the execution of the Ant-Miner algorithm. Running time, number of method calls, average running time for a single call of the method and the percentage of the running time of the method to the whole execution are recoded. Such a profiling gives a deep insight about the performance of the execution of the Ant-Miner algorithm and highlights the points of the algorithm where enhancements can be directed to and modification can be applied.

Method	Time (m.sec)	Calls #	Avg. Time (sec)	% to Parent	% to Total
Run()	5704.2	1	5704.2	100%	100%
>ConstructRule()	43878.46	1112	39.459	13%	13%
>>CalculateNodeProbabilities()	51	4105	0.0124	2%	<1%
>>SelectNodeProbablistically()	0.9	4205	0.0002	< 1 %	<1%
>PruneRule()	12136.6	1112	10.914	42%	47%
>>CalculateRuleQuality()	16777.06	2242	7.483	19%	34%
>>DetermineRuleClass()	27162.86	2242	12.115	15%	21%
>UpdatePheromone()	2.81	1112	0.0025	< 1 %	<1%
>IntializeNodeInformation()	40744.29	8	5093.036	15%	14%
>IntializePheromone()	0.015	8	0.0018	< 1 %	<1%
>BuildConstructionGraph()	1.02	1	1.02	<1%	<1%

Table 3.1 - Ant-Miner Execution Profile.

As shown in Table 3.1, the `PruneRule()` method took the highest percentage of the total running time (47%). This is because it calls 3 time consuming methods each time it is called, namely `CalcuateRuleQuality()`, `DetermineRuleClass()` and `UpdateInstancesIndexList()`. The first method `CalcuateRuleQuality()` , which takes 24% of the total runing time, calculating the quality of the rule using $sensitivity \times specificity$, which needs to scan the whole training set each time it is called. `DetermineRuleClass()` uses the `InstanceIndexList` field associated to the ant that contains the covered cases indexes by the current rule to calculate the the class with the highest occuerance amoung these instances. This is done by scanning the `InstanceIndexList` and takes 21% of the running time. As rule term is removed during the prunning procedure, the

`InstanceIndexList` covered by the new rule changes, thus `UpdateInstancesIndexList()` is called, this takes 11% of the run time.

`IntializeNodeInformation()` comes after the previous methods in running time consumption (16%). This method involves scanning the training set to set attribute value frequncey for each class and claculate its heuristic value.

3.11 Summary

In summary, this chapter has described the original version of Ant-Miner that was published in 2002 [20]. Ant-Miner is an ACO based algorithm designed to discover classification rules. Iteratively, a swarm of ant tries to discover a rule to be used for building a rule-based classification model. Hence, each ant in the swarm wanders the construction graph looking for terms to select for rule construction. The ant is influenced in path selection via the amount of the pheromone on the terms and the heuristic value of each term. After an ant finish constructing a rule, the quality of the rule is evaluated, and the pheromone is updated on the trail that the ant took according to the rule quality. Rule pruning takes place to remove irrelevant terms from the rules. As all the ants finish their trials, the best generated rule is selected and added to the discovered rule list. The algorithm is then repeated on the reduced training set after the covered cases by the discovered rule are removed. After the minimum covered cases number is reached, the algorithm stops, and the discovered ordered list of rules is ready for classifying new unlabeled cases.

Chapter 4

ANT-MINER RELATED WORK

4.1 Introduction

In the previous chapter, a detailed description of the original version of Ant-Miner was introduced. Ant-Miner was proposed by Parpinelli *et al*. [20] in 2002 as an ACO-based algorithm for discovering classification rules from labeled cases. Empirical results have chosen competitive performance to C4.5 and CN2 concerning predictive accuracy on the test set and better generated rules in term of simplicity. However, Ant-Miner had some issues that were tackled in later versions. This chapter aims to present the literature review on Ant-Miner and the related work that has been done to improve it. The chapter lists the various versions of Ant-Miner that have been introduced in the literature in the order in which they were introduced in, along with a brief description of each. Section 4.2 presents Ant_Miner2 which introduced a new heuristic function for Ant-Miner. Section 4.3 presents Ant_Miner3 that suggested a new pheromone evaporation technique. A new pruning procedure is described in section 4.4. An Ant-Miner algorithm for multi-label classification is presented in section 4.5. Section 4.6 describes a new version which discovers unordered rule sets. AntMiner+ is discussed in section 4.7. cAntMiner, which is a version that copes with continuous attributes, is presented in section 4.8.

4.2 Ant_Miner2 [2002]

In Ant-Miner2, B. Liu *et al*. have introduced a density-based heuristic for rule discovery [16]. The idea is that the ACO algorithm does not need accurate information in this heuristic value since the idea of the pheromone should compensate the small potential errors in the heuristic values. In other words, a simpler heuristic value may do the job as well as the complex one. As a result, an easily computable density estimation equation, shown in the equation (4.1) was proposed to calculate a heuristic value η_{ij}:

$$\eta_{ij} = \frac{majority\_classT_{ij}}{|T_{ij}|} \tag{4.1}$$

where:

- T_{ij} is the size of partition that $term_{ij}$ occurred in.
- $majority\_classT_{ij}$ is the occurrence of the majority class in partition T_{ij}.

Although the density based function has less computational cost, Ant_miner2, with the simple heuristic function based on the density of the majority class, has introduced identical results to the original Ant-Miner, which were produced with entropy as a heuristic value measure [16].

4.3 Ant_Miner3 [2003]

B. Liu, H. A. Abbass, and B. McKay have introduced a new version (Ant_Miner3) [17] based on their previously proposed one (Ant_Miner2). They contributed with two new modifications on Ant_Miner2 concerning pheromone update state transition procedure. The modifications are described in the following subsections.

4.3.1 Pheromone Update Method

A new pheromone update method has been introduced in Ant_Miner3, show in equation (4.2). In this original version of Ant-Miner, the pheromone evaporation process is simulated by normalizing the value of each pheromone τ_{ij} for each $term_{ij}$ after pheromone reinforcement. More precisely, this normalization is performed by dividing the value of each τ_{ij} by the summation of all τ_{ij} on each node in the construction graph. When a rule is constructed, only the terms occurring in the rule constructed by an ant have their amount of pheromone increased by equation (3.6).In Ant_Miner3, the amount of pheromone associated with each term that occurs in the constructed rule is updated by equation (4.2), and the pheromone of unused terms is updated by normalization.

$$\tau_{ij}(t) = (1-\rho).\tau_{ij}(t-1) + \left(1 - \frac{1}{1+Q}\right).\tau_{ij}(t-1) \qquad (4.2)$$

where:

- ρ is the pheromone evaporation factor, which controls how fast the old path evaporates. This parameter controls the influence of the history on the current pheromone trail. A large value of ρ indicates a fast evaporation rate and vice versa. A value 0.1 was fixed and used for the experimentation of this modification.
- Q represents the quality of the contracted rule, which ranges in [0,1].

4.3.2 State Transition Procedure

Pheromone amounts in the construction graph represent the current knowledge of the colony which influences subsequent ants in choosing their paths. This benefits exploitation of prior knowledge. But it increases the probability of choosing terms belonging to previously discovered rules according to equation (4.2) In order to improve

exploration; Ant_Miner3 has introduced a new state transition procedure - shown in Algorithm 4.1 - that is taken from [17].

Algorithm 4.1 - Ant_Miner3 State Transition Rule.

If $q1 \leq \varphi$
  Loop
    If $q2 \leq \sum_{j \in J_i} P_{ij}$
    Then choose $term_{ij}$
  Endloop
Else
    Choose $term_{ij}$ with max P_{ij}

where:

- q1 and q2 are random numbers.
- φ is a parameter in [0,1].
- J_i is the number of i-th attribute values.
- P_{ij} is possibility calculated using equation (4.2).

Therefore, the results not only depend on the heuristic functions η_{ij} and pheromone τ_{ij}, but also on a random number, which increases the likelihood of choosing terms not used in previously constructed rules. If $q1 \leq \varphi$ then $term_{ij}$ is selected randomly as a favor for exploration. Else, $q1 > \varphi$ corresponds to an exploitation of the knowledge available about the problem, as $term_{ij}$ is selected based on heuristic functions $\eta_{ij}$and pheromone τ_{ij} from equation (4.2). φ, which represents the exploration/exploitation balancer, was set to 0.4.

Although Ant_Miner3 needed more ants to converge and find a solution, and the discovered rules by Ant_Miner3 are more than rules discovered by the original version of

Ant-Miner, the mean accuracy of the rule sets discovered by Ant_Miner3 is higher than that of Ant-Miner.

4.4 A New Rule Pruning Procedure [2005]

As was described in the previous chapter, Ant-Miner generates rules in the form of IF <antecedents> THEN <consequent>, where antecedents are the terms that were selected probabilistically during rule construction based on a heuristic value η_{ij} and pheromone value τ_{ij}. Irrelevant terms may have been included in the rule due to stochastic variations in the term selection procedure and/or due to the use of a shortsighted, local heuristic function, ignoring attribute interactions. Pruning can improve the quality of a rule by removing irrelevant terms from the rule antecedent. As a result, pruning can improve both the predictive accuracy and the comprehensibility of the rule. A. Chan and A. Freitas have introduced a new classification rule pruning procedure for Ant-Miner in [4]. The following section is a description of the original rule pruning procedure followed by a section that describes the new rule pruning procedure introduced in [4].

4.4.1 Original Ant-Miner Rule Pruning Procedure

In original version of Ant-Miner, the pruning procedure tries to improve the quality of the constructed rule (measured by the rule's predictive accuracy), by removing irrelevant terms from the rule antecedent. This is done by iteratively removing one term at a time while it improves on the rule's quality [20]. This iterative process stops when no term removal will further increase the quality of the current rule undergoing pruning. The pruned rule with the best quality is then selected for pheromone update. The procedure is described as follows in Algorithm 4.2.

Algorithm 4.2 - Rule Pruning Procedure of the Original Version of Ant-Miner.

Execute_pruning = true;

WHILE (Execute_pruning = true) AND (Number of terms in rule antecedent > 1)

FOR EACH (term t_i in the current rule to be pruned)

Temporarily remove t_i and assign to the rule consequent the most frequent class among the examples covered by the rule antecedent;

Evaluate rule quality;

Reinstate term t_i in rule antecedent;

END FOR

IF (rule quality was improved some iteration of the FOR loop)

THEN

Remove permanently the term whose removal improves current rule most;

ELSE

Execute_pruning = false;

END IF-THEN-ELSE

END WHILE

Rule pruning has proved to enhance the quality of the generated rules by Ant-Miner. However, the rule pruning operation is the most time consuming part of the algorithm (see Table 3.1 section 3.10.2 execution profile of Ant-Miner) as it is quite sensitive to the number of attributes of the input data set. This is due to the fact that the larger the number of attributes in the data being mined, in general the larger the number of terms in a constructed rule before pruning, and so the larger the number of iterations in

the loops of Algorithm 4.2. Moreover, in each iteration of the FOR EACH loop, a term is temporarily removed and the quality of the reduced candidate rule has to be computed by the rule quality evaluation formula (3.5). The quality evaluation formula is a quite computationally expensive operation as it scans the entire dataset to calculate values of TP, FP, TN and FN.

4.4.2 The New Hybrid Rule Pruner for Ant-Miner

The new rule pruning procedure proposed in [4] is a hybrid rule pruner, combining the original Ant-Miner's rule pruner with a rule pruner based on information gain. The basic idea is to combine the effectiveness of the original Ant-Miner pruner (in terms of maximizing predictive accuracy) with the speed of a rule pruner based on information gain. This latter is very fast, because it does not require any scan of the training set. If the number of terms in the rule antecedent of a generated rule exceeds the value of r, the rule first undergoes reduction of the number of terms to the value of parameter r. This reduction is obtained as follows. For each term within the rule antecedent, the rule pruner computes the probability of selecting that term. This probability measure is based on the pre-computed value of that term's information gain with respect to the class attribute. Then the rule pruner selects r number of terms with the probability of selecting each term proportional to the information gain of that term. Once r terms have been selected the resulting reduced rule is placed back into Ant-Miner's original rule pruner. A high-level description of the proposed hybrid rule pruner is described in the following algorithm. For more details about the algorithm, refer to [4].

Algorithm 4.3 - Hybrid Rule Pruning Procedure.

INPUT:

a) information gain of all terms individually, calculated using the entire current training set; /* previously done by another procedure of Ant Miner */

b) value of r /* user-defined parameter: number of terms in the current rule which will be given to Ant-Miner's original rule pruner */

Reduced_rule = {};

Num_terms_selected = 0;

IF (number of terms in current rule's antecedent > r)

THEN

WHILE (Num_terms_selected < r)

FOR EACH (term t_i in current rule's antecedent)

Calculate probability of selecting a term t_i as:

$$Prob(t_i) = \frac{InfoGain(t_i)}{\sum_{i=1}^{T} InfoGain(t_i)}$$

/ *T = number of terms in the rule antecedent */

END FOR

Create roulette wheel for selection and select one Term, called selected_term, by spinning the wheel;

Reduced_rule = Reduced_rule ∪ selected_term;

Remove selected_term from current rule's antecedent to avoid reselection;

Num_terms_selected = Num_terms_selected + 1;

END WHILE

Assign to the consequent of the Reduced_rule the most frequent class among all examples covered by the rule;

Run Ant-Miner's original rule pruner on Reduced_rule;

ELSE

Run Ant-Miner's original rule pruner on current rule;

END IF-THEN-ELSE

Experimental result has shown that, in general, the hybrid pruner significantly reduced the computational time of Ant-Miner, by comparison with the computational time taken with the original rule pruner, without sacrificing the accuracy of the generated rules. Moreover, shorter rule lengths were obtained in general by applying the new pruning procedure which enhanced the comprehensibility of the generated rules.

4.5 Multi-Label Ant-Miner (MulAM) [2006]

Multi-label classification principles are similar to single- label classification ones; the aim is to find a classification model that is able to describe the class attribute(s) as a function of input attributes from labeled cases so that labels of new case can be predicted using this model. However, in multi-label classification there are two or more class attributes to be predicted. As a result, the consequent of a classification rule contains one or more attribute prediction, each prediction involving a different class attribute.

A. Chan and A. Freitas have introduced a version of Ant-Miner, called (MulAM) that copes with multi-label classification [3]. Although several workarounds have been used to apply traditional classification techniques to solve multi-label classification problems, none of them proved efficient in doing such a task. One approach is to split the original dataset into near identical datasets, where each contains all input

attributes and all cases, but each dataset produced in this way contains only one of the class attributes to be predicted. This results in requiring the classification algorithm to be trained on nearly the same dataset several times: as many as the number of the class attributes. This technique ignores possible correlations between class attributes, thus the resulting rules lakes the appropriate comprehensibility. Moreover, it's is computationally expensive. Another approach is to convert the existing class attributes into a single class attribute, where each value of this new class attribute represents a combination of the class attributes that were initially present in the data set. However, by doing such a workaround, the number of values of the new single-class attribute will increase exponentially with the number of original class attributes. Therefore, it becomes more difficult to predict a class value, as the number of cases associated with any given value of the new single class attribute decreases considerably, reducing the amount of information to effectively predict each class value.

The following algorithm was proposed in [3] as a version of Ant-Miner, an Ant Colony classification rule generation technique that copes with multi-label datasets:

Algorithm 4.4 - Multi-Label Ant-Miner (MuLAM).

TrainingSet = {set of all training examples}

DiscoveredRuleList = {}

WHILE (TrainingSet > MaxUncovExamples)

$t = 1$; /* ant index */

Calculate information gain of each term considering all class attributes based on current training examples;

For each class attribute C_i, initialize all cells of the pheromone matrix

REPEAT

Ant_t starts with an empty partial rule R_t;

Current ruleset RS_t= { };

WHILE ((there is at least 1 unused attribute) AND (there is at least 1 unpredicted class attribute))

Ant_t chooses, out of the unused terms, a term to be added to current partial rule R_t, with a probability proportional to the product of a heuristic function and the pheromone;

IF (after adding the chosen term to the partial rule R_t the rule will still cover more than MinExamplesPerRule)

THEN Add the chosen term to the current partial rule R_t ;

RuleCons = ∅;

FOR EACH (Class attribute C_i)

IF (partial Rule R_t predicts class attribute C_i with high confidence)

THEN

RuleCons = RuleCons È (predicted class for class attribute C_i);

Mark class attribute C_i as predicted;

END IF

END FOR EACH

IF (RuleCons ≠ ∅) THEN

Create complete rule CR_{ti} (with rule format IF$term_1$ … AND … $term_n$ THEN RuleCons);

$RS_t = RS_t \cup CR_{ti}$;

END IF

ELSE

Quit this **WHILE loop;**

END IF-THEN-ELSE

END WHILE

IF (there are still unpredicted class attributes) **THEN** Create one complete rule predicting each of those class attributes;

FOR EACH (class attribute C_i predicted by this rule)

Create a temporary $rule_i$ IF (ant_{ei}) THEN C_i;

Use original Ant-Miner pruning technique to prune this temporary rule. Instead of allowing the consequent to be modified during pruning, the current consequent is kept fixed, which will potentially produce a new ant_{ei} only;

END FOR

END IF

FOR EACH (rule in RS_t)

Update pheromone matrix for each predicted class attribute C_i in the rule, increasing pheromone of terms in rule antecedent and reducing pheromone (evaporation via normalization) of terms not used in the rule. Pheromone increasing is based on quality of partial rule predicting class attribute C_i only;

$t = t + 1$;

END FOR

UNTIL (t ≥ MaxNoAnts)

Choose best set of rules RS_{best} among those generated by all Ants in current population by using the rule quality measure;

Add RS_{best} to DiscoveredRuleList;

TrainingSet = TrainingSet – {set of examples where all the class attributes have been correctly predicted by RS_{best} };

END WHILE

In MuLAM, each ant does not produce a single rule like in the original Ant-Miner. Rather, each ant discovers a candidate rule set. The reason for this is due to addressing the multi-label classification task, where there are multiple class attributes to be predicted. Each ant discovers at least one rule and at most a number of rules equal to the number of class attributes, a different rule for each class to be predicted. An ant will discover a single rule only in the case where that rule is considered good enough to predict all class attributes. After an ant constructs its rule antecedents, the selection of prediction class value occurs. Before the algorithm makes a prediction for this current rule, it initializes the rule consequent with the empty set. This rule consequent holds all class attribute (with empty values) that are being predicted by the rule. The ant enters the FOR loop, where it processes each class attribute separately. So for every class attribute, the algorithm then decides under a certain pre-pruning criteria whether the current class attribute should be added to the rule consequent as a prediction. A detailed description of the algorithm is found in [3].

4.6 Ant-Miner for Discovering Unordered Rule Sets [2006]

J. Smaldon and A. Freitas have introduced one of the main important contributions to Ant-Miner, which is fixing in advance the class predicted by the rule [23]. The basic idea is to set the class as a consequent of the rule before selecting the terms that construct the rule antecedents. In the original Ant-Miner, ants chose terms for a rule with the goal of decreasing entropy in the class distribution of examples matching the rule in construction. The consequent of the rule is then assigned afterwards by determining the class value that would produce the highest quality rule. On the other hand, in Unordered Rule Set Ant-Miner, as the class is set before rule construction, the terms are chosen with respect to its relevance to the selected class. The approach has improved the quality of the generated rules in terms of accuracy. The following algorithm describes the proposed version in [23].

Algorithm 4.5 - Unordered Rule Set Ant-Miner.

Discovered Rule Set = {} /* initialize rule set with empty set */

FOR EACH Class

TrainingSet = {all training cases}

PositiveSet = {training cases of current class}

NegativeSet = TrainingSet – PositiveSet

WHILE (|PositiveSet| > max_uncovered_cases)

$t = 1;$

$j = 1;$

initialise all trails to the same amount of pheromone;

REPEAT

Ant_t starts with an empty rule and incrementally constructs a classification rule R_t by adding one term at a time to the current rule;

Prune rule R_t;

IF (LaplaceCorrectedConfidence(R_t) > RuleConfidenceThreshold)

THEN increase pheromone of terms in rule R_t

END IF

Update pheromones in all other terms by normalizing the pheromone values (simulating evaporation)

IF (R_t equals R_t -1)

THEN $j = j + 1$;

ELSE $j = 1$;

END IF

$t = t + 1$;

UNTIL (i ≥ No_of_ants) OR (j ≤ No_rules_converg)

Choose the best rule R_{best} among all rules R_t constructed by all ants;

Add rule R_{best} to DiscoveredRuleSet;

TrainingSet = TrainingSet – {set of positive cases covered by R_{best}};

PositiveSet = PositiveSet – {set of positive cases covered by R_{best}};

END WHILE

END FOR

As shown in algorithm 4.4, an extra For-Each loop is added as the outer loop of the algorithm, iterating over the values in the class attribute domain. Each value is set as a

rule consequent for the rules to be built by subsequent ants. Each iteration of the For-Each loop discovers an unordered set of rules, all of which predict the current class value. At the beginning of each iteration, the entire training set is reinstated, so that a maximal number of negative examples are available to the algorithm. Ants discover rules from the training data until the number of positive examples (belonging to the current class) remaining in the dataset that have not been covered by a discovered rule is less than or equal to the value determined by the **max_uncovered_cases** parameter.

As the class for the rules is known prior terms selection, a better heuristic function is used to focus on the terms that have more relevance to the current class. The Laplace-corrected confidence is used, as follows:

$$\eta_{ij,k} = \frac{|term_{ij},\mathrm{k}|+ 1}{|\mathrm{term_{ij}}|+ \mathrm{No\_of\_classes}} \tag{4.3}$$

where $|term_{ij}, \mathrm{k}|$ is the number of training cases having $term_{ij}$ and the current positive class k, $|term_{ij}|$ is the number of training cases having $\mathrm{term_{ij}}$ and **no_of_classes** is the number of values in the class attribute's domain.

As for rule quality evaluation and pheromone update, the same fitness function used in the original Ant-miner is used for this version (see equation 3.5). However, a threshold formula has been added to determine whether to accept this rule or not. The formula is defined as folows :

$$RuleConfidenceThreshold = MAX(0.5\ , \frac{|k|}{|TrainingSet|}) \tag{4.4}$$

where |k| is the number of training cases with the current (positive) class, and |training set| is the total number of cases in the current training set.

The new proposed version of the Ant-Miner, where the class of the rule is fixed before rule construction has shown to be a very good improvement in regards to the discovered rules in terms of rule accuracy and number of terms. However, the whole algorithm should be repeated for each class value, which increases the number of the overall iterations needed to discover rules covering the minimum needed cases. Moreover, the number of generated rules is larger than the number of rules generated by Ant-Miner, which affects the quality of the output in terms of simplicity.

4.7 AntMiner+ [2007]

AntMiner+, which was proposed by D. Martens *et al*. [18], is considered another important version of Ant-Miner with several modifications which enhanced the performance of the algorithm. The following section describes the main differences in the AntMiner+.

4.7.1 MAX-MIN Ant System

The first modification in AntMiner+ is utilizing the MAX-MIN Ant System [24]. As was described in Chapter 2 section 2.4.2, the MMAS has a maximum and a minimum value of the pheromone on the construction graph. Initially, the pheromone on the construction graph is initialized with τ_{max}. As the pheromone is updated on the construction graph (deposited and evaporated), the pheromone amount cannot exceed τ_{max} or go bellow τ_{min}. The idea behind this is to improve exploration and avoid early stagnation of the swarm. Moreover, only the ant that describes the best rule will update the pheromone of its path, which balances the exploitation aspect in front of the aforementioned pheromone clipping technique that keeps the exploration aspect.

4.7.2 Construction Graph

The construction graph for AntMiner+ is a directed acyclic graph (DAG) where the decision components are the edges which connect the nodes that represent the term constructing the rule antecedents. The following figure describes the (DAG) construction graph of AntMiner+:

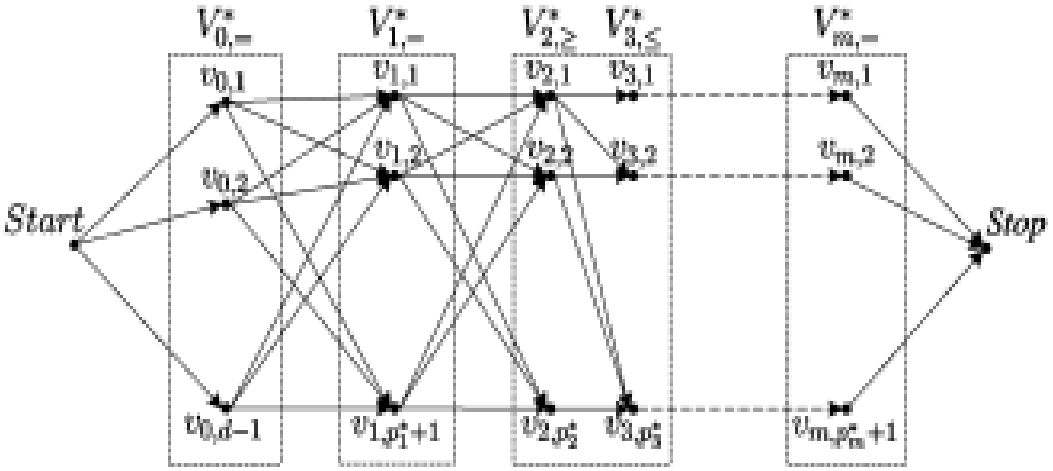

Figure 4.1 - Construction Graph for AntMiner+. [18]

As an ant starts to construct a rule, an ant should choose from each pool of terms which represent the values of an input attribute. After selecting a value from the current attribute it can then move to the next attribute. A value of nil is added to each attribute value's pool to give a probability of bypassing an attribute. Each value in attribute V_i is connected to all the values of the following attribute V_{i+1}. The pheromone is associated with edges between the nodes, in contrast to the original version of Ant-Miner where the pheromone is associated with terms themselves. This introduces attribute value dependency. However, the constructed rules can be sensitive to the attributes order in the construction graph.

4.7.3 A Class is Selected before Rule Construction

In AntMiner+ the ant selects the class value before constructing the rule. Thus, an extra vertex group is added that comes first in the construction graph. This is similar to considering the class variable as just one of the variables, treated as such when calculating the heuristic values and pheromone update. The class value is selected probabilistically according to the amount of pheromone on the edge leading to it. This amount of pheromone indicates that this class value has contributed in classifying a rule with high quality, which should be selected in subsequent trials. The following figure shows a path of an ant on the AntMiner+ construction graph:

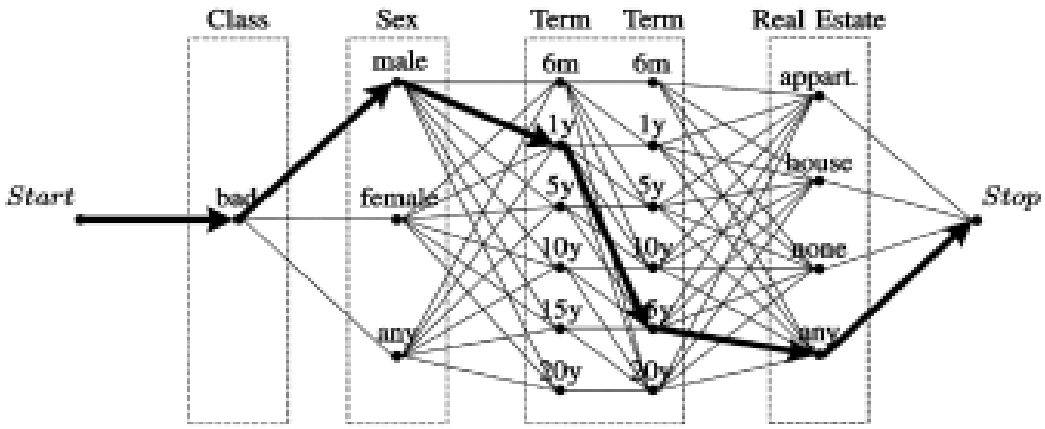

Figure 4.2 - A Path of an Ant in AntMiner+. [18]

Since the rule consequent is known before the rule construction, the heuristic function is defined as follows:

$$\eta_{v_{i,j}} = \frac{|T_{ij}\ \&\ Class = C_t|}{|T_{ij}|} \tag{4.5}$$

where

- T_{ij} is the portion of the dataset covered by $term_{ij}$.

- C_t is the current selected class.

The quality is evaluated as rule confidence + rule coverage.

Multiple ants in the same iteration can construct rules with different class as a consequent of the rule. However, the pheromone is shared by all ants constructing rules with different consequents. Any ant is influenced by the pheromone dropped by any other ant constructing similar or different labeled rule. The term that leads to construct a good rule with class C_x as a consequent does not necessary lead to construct a good rule with C_y as a consequent. This could affect the quality of generated rules.

4.7.4 Handling Continuous Attributes

In the original Ant-Miner version, the continuous-valued attributes should be discretized as a pre-processing step. In AntMiner+, each continuous attribute is represented by two pools of values V_i^{Upper} and V_i^{lower}. The values selected by the ant during the rule construction from V_i^{Upper} and V_i^{lower} define the range of the continuous value suitable for the current constructed rule. However, both V_i^{Upper} and V_i^{lower} have a discrete set of values. Figure 4.2 shows the idea on the construction graph.

4.7.5 Weight Parameters

In the typical (ACO) state transition formula, the heuristic value component (η) and the pheromone value component (τ) are each raised to the power of α and β respectivly. The powers are used to gives different emphasis on each component. In the Original Ant-Miner, α equals β equaling to 1.0. AntMiner+ allows other values to be chosen and actually lets the ants themselves choose suitable values. This is done by introducing two new vertex groups in the construction graph: one for each weight

parameter. The values for the weight parameters were limited to integers between the value of 1 and 3.

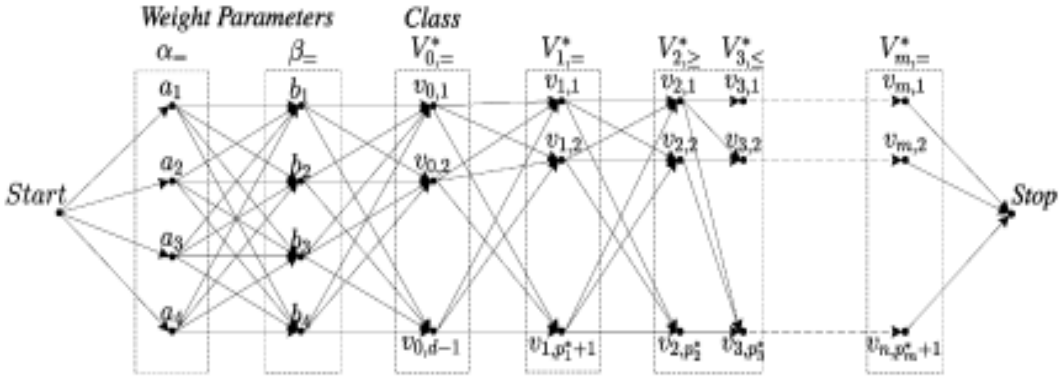

Figure 4.3 - The Complete Construction Graph for AntMiner+. [18]

4.8 *c*Ant-Miner [2008 – 2009]

Otero *et al*. [19] have proposed an Ant-Miner extension — named *c*Ant-Miner (Ant-Miner coping with continuous attributes) — which can dynamically create thresholds on continuous attributes' domain values during the rule construction process. Since cAntMiner has the ability of coping with continuous attributes "on-the-fly", continuous attributes do not need to be discretized in a preprocessing step. Firstly, *c*Ant-Miner includes vertices to represent continuous attributes in the construction graph. Secondly, in order to compute the heuristic information for continuous attributes, *c*Ant-Miner incorporates a dynamic entropy-based discretization procedure: a threshold value v needs to be selected in order to dynamically partition the set of examples into two intervals: $y_i < v$ and $y_i \geq v$. The best threshold value v is the value v that minimizes the entropy of the partition, computed as:

$$entropy(y_i, v) = \frac{|S_{y_i<v}|}{|S|} . entropy(S_{y_i<v}) + \frac{|S_{y_i\geq v}|}{|S|} . entropy(S_{y_i\geq v}) \quad (4.6)$$

where:

- $|S_{y_i<v}|$ is the total number of examples in the 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 of training examples where the attribute y_i has a value greater than or equals v.
- $|S|$ is the total number of training examples.

Thirdly, when a continuous attribute vertex (y_i) is selected by an ant to be added to its current partial rule, a relational operator and a value is computed using a similar procedure as for the heuristic information. Fourthly, the pheromone updating procedure has been extended to cope with continuous attribute vertices. In the case of continuous attributes, pheromone values are associated with continuous attribute vertices not considering the operator and threshold value, that is, there is a single entry in the pheromone matrix for each continuous attribute, in contrast to multiple entries for nominal attributes — nominal attributes have an entry for every (x_i, v_{ij}) pair.

4.9 Summary

This chapter has presented a literature review on Ant-Miner and the related work that has been done on the original version. Several modifications have been introduced to improve the quality of the algorithm by trying different heuristic functions, setting the class before constructing the rule antecedents, applying a new rule pruning procedure, proposing different pheromone update strategies, handling multi-label classification, and trying to cope with continuous attributes. However, a lot of other ideas can be applied to

enhance the exploration\exploitation behavior of the algorithm. Moreover, different pheromone update (deposit and evaporation) can be tried.

Chapter 5

USING LOGICAL NEGATION OPERATOR

5.1 Introduction

The first extension to the original Ant-Miner algorithm presented in this thesis is using the logical negation operator in constructing rule antecedents. In which case, terms contained in the constructed rules can be in the form of <attribute Not= value>. Such terms match more cases than the original form, leading to constructing rules with a possible higher coverage. The advantage of this extension is that it can reduce the number of the generated rules, which in turn improves the comprehensibility of the output. In order to apply this extension, allowing using logical negation, a simple modification is done on the construction graph. Results in Chapter 9 shows that using logical negation operators not only decreases the size of the generated rule set, but also increases its classification accuracy. This chapter describes in detail this extension and the modifications on the algorithm to support it. Data structure updates and program execution performance are discussed to show the implication of enabling such a modification on the algorithm.

5.2 Using Logical Negation

In the original and various versions of Ant-Miner, the construction graph consists of nodes representing attribute values of the dataset. These nodes are the decisions components (terms) that are selected to construct a solution (rule antecedents) by ants traversing the construction graph. The set of nodes (N) in the construction graph is:

$$N = \bigcup_{i=1}^{n} v_{ij} \ , \qquad j \in \{1,2, \dots, l\}$$

where:

- i is i-th attribute of the features that describes the case in the dataset.
- n is the number of attributes in the construction graph.
- v_{ij} is the j-th value of i-th attribute.

Thus the constructed rule antecedent will be in the form of:

$$\boldsymbol{IF} \ < A_i \ = \ V_{ij} > \ \boldsymbol{AND} \ < A_k \ = \ V_{kl} > \ \boldsymbol{AND} \ \dots$$

To allow using the logical negation operators in the antecedents of constructed rules, the values and their negation per attribute will be added to the construction graph. The set of nodes (N) in the construction graph will be:

$$N = \bigcup_{i=1}^{n} v_{ij} \ \cup \ \bigcup_{i=1}^{n} \overline{v_{ij}}, \qquad j \in \{1,2, \dots, l\}$$

Thus, the available decision components in the construction graph allow constructing rule antecedents in the form of:

$$\boldsymbol{IF} \ < A_i \ = \ V_{ij} > \ \boldsymbol{AND} \ < A_k \ \boldsymbol{NOT} = \ V_{kl} > \ \boldsymbol{AND} \ \dots$$

Negation values are added for the attribute that has more than two values in its domain. This supports constructing terms in the form of <attribute=value>. An example of a generated rule using logical negation operator is: "IF <price = low> AND <condition NOT = bad> THEN <Class=Buy>". Terms that have logical negation match more cases on the regular terms. This leads to construct rule with high coverage. More precisely, assume we have the following subset of 8 cases taken from a dataset that has two attributes and the class:

Condition	Safety	Class
Excellent	Bad	Buy
Very Good	Very Good	Buy
Good	Good	Buy
Good	Very Good	Buy
Bad	Very Good	Wait
Bad	Very Good	Wait
Bad	Good	Don't Buy
Bad	Bad	Don't Buy

If the logical negation operator is used for constructing classification rules for the previous dataset, 3 ordered rules will be needed to correctly classify the whole dataset. These rules are as follows:

1) **IF** <Condition **NOT** = Bad> **THEN** <Class =Buy>
2) **ELSE IF**< Condition = Bad > **AND** <Safety=Very Good> **THEN** <Class=Wait>
3) **ELSE** <Class= Don't buy>

Because the rules generated with the logical negation operator have a higher coverage, the output rule set size becomes smaller than the rule set generated without using logical negation. The following is the generated rules without using logical negation:

1) **IF**< Condition = Bad > **AND** <Safety=Very Good> **THEN** <Class=Wait>
2) **ELSE IF**< Condition = Bad > **AND** <Safety=Good> **THEN** <Class=Don't Buy>
3) **ELSE IF**< Condition = Bad > **AND** <Safety=Bad> **THEN** <Class=Don't Buy>
4) **ELSE** <Class= Buy>

At least 4 rules are needed to correctly classify the pervious labeled cases, or three rules that classify some cases incorrectly.

Using negative attributes doubles the size of the construction graph. However, it enables constructing rules that have greater coverage of the training cases. Hence, it produces a lower number of rules, which improve the comprehensibility of the output. Moreover, a reduced number of iterations are needed to reach the threshold of the number of cases to be covered, which reduces the overall runtime of the algorithm (see section 5.4.2 execution profile). Results in Chapter 9 also show that it has a better performance in terms of accuracy in addition to the reduced number of iterations and the simpler (smaller) rule set.

5.3 Algorithm Modifications

No modification is needed in the Ant-Miner algorithm to support using logical negation operator in constructing classification rules. Pheromone update procedure is done regularly with the negation values and the heuristic value is calculated for the negation attribute values the same as it is calculated for regular attribute values, using formula (3.4) in Chapter 3, which involves information gain, or any other heuristic functions that were used in various Ant-Miner versions (i.e. density based, or Laplace-corrected confidence) (see Chapter 4).

It is worthy to mention that the choice of the rule evaluation function can affect the efficiency of the generated output rules in terms of classification when using logical negation. Because using logical negation operators generates rules with high coverage, the confidence of the rule (which affects its classification accuracy) may be damaged. Using quality evaluation function that put more emphasis on the rule confidence should

overcome this drawback and balance between the coverage of the rule and its classification accuracy.

5.4 Logical Negation Operator Implementation

This section discusses the modifications that have been done on the code of the original Ant-Miner program to avail the use of logical negation operator in constructing rule antecedents. The execution performance with the use of this extension is discussed as well.

5.4.1 Data Structure and Operation

A few code modifications have been added to the original Ant-Miner program in order to implement using logical negation operator. First, a new data field has been added to the data structure that represents the node in the construction graph. `IsNegation` is a Boolean data field that indicates whether this node represents an attribute value or its negation value. The code for the new node data structure is described as follows:

```
public struct Node
{
      public int AttributeIndex;
      public int ValueIndex;
      public bool IsNegation;
      public int [] ValueFrequency;
      public double PheromoneAmount;
      public double HeuristicValue;
      public double Probability;
      public bool UnusableValue;

 }
```

The second modification is in building the construction graph. Each distinct value in the domain of an attribute is added twice in the construction graph, the first time the added, `IsNegation` is set to false, while the second time it is set to true. By this, each attribute value has two exisitances in the construction graph, with and without negation. This only applies on the attributes that have more than two values in their domains. Such

modification affects the logic of the `BuildConstructionGraph()` method. The new implementation of the method is as follows:

```
private void BuildConstructionGraph(bool useLogicalNegation)
{

this._constructionGraph = new
Node[this._trainingSetDataTable.Columns.Count][];

for (int attributeIndex = 0; attributeIndex <
this._trainingSetDataTable.Columns.Count; attributeIndex++)
{
List<string> values = this.GetDistinctAttributeValues(attributeIndex);

this._constructionGraph[attributeIndex] = new Node[values.Count];

for (int valueIndex = 0; valueIndex <
this._constructionGraph[attributeIndex].Length; valueIndex += 1)
{

this._constructionGraph[attributeIndex][valueIndex].AttributeIndex =
attributeIndex;
this._constructionGraph[attributeIndex][valueIndex].ValueIndex =
valueIndex;
this._constructionGraph[attributeIndex][valueIndex].IsNegation= false;
this._constructionGraph[attributeIndex][valueIndex].ValueFrequency = new
int[numberOfClasses];

if(useLogicalNegation && values.Count>2)
{
this._constructionGraph[attributeIndex][valueIndex].AttributeIndex =
attributeIndex;
this._constructionGraph[attributeIndex][valueIndex].ValueIndex =
valueIndex;
this._constructionGraph[attributeIndex][valueIndex].IsNegation= true;
this._constructionGraph[attributeIndex][valueIndex].ValueFrequency = new
int[numberOfClasses];

}
}

}

      …

}
```

When a node with `IsNegation` field set to true added to the rule antecedent, the treatment of the rule case matching differs, as a case would be a match if the value of a given attributes in the case does not equal the value of the selected node (term) in the

rule with the logical negation. This implies a need of a modification in two methods, namely `CalculateRuleQuality()` and `IntializeNodeInformation()`. The former involves rule case matching, and the latter involves calculate attribute value frequencies and heuristic values.

5.4.2 Execution Profiling and Analysis

The following table exhibits the execution profile of Ant-Miner program after implementing the use of logical negation operator. Results of the output quality in terms of predictive accuracy and comprehensibility are shown in Chapter 9. The following profile shows how such a modification has affected the program running time.

Method	Time (m.sec)	Calls #	Avg. Time (m.sec)	% to Parent	% to Total
`Run()`	4188.4	1	4188.4	100%	100%
`>ConstructRule()`	586.376	395	1.484	14%	14%
`>>CalculateNodeProbabilities()`	18.84	515	0.813	2%	<1%
`>>SelectNodeProbablistically()`	0.84	515	0.813	< 1 %	<1%
`>PruneRule()`	1968.548	595	3.308	47%	47%
`>>CalculateRuleQuality()`	1130.868	514	2.2	20%	27%
`>>DetermineRuleClass()`	628.26	514	1.222	11%	15%
`>UpdatePheromone()`	48.04	595	0.703	< 1 %	<1%
`>IntializeNodeInformation()`	1047.1	5	209.42	25%	25%
`>IntializePheromone()`	0.24	5	83.768	< 1 %	<1%
`>BuildConstructionGraph()`	1.8	1	418.84	<1%	<1%

Table 5.1 - Ant-Miner with Logical Negation Execution Profile.

As shown in Table 5.1, the overall running time of the Ant-Miner program with the use of logical negation operator has decreased by 30 % in comparison to the running time of the original An-Miner shown in Table 3.1. Although the running time of all of the

methods that takes the largest amount of the execution decreased from 30% to 60%, the overall all running time reduction is because the application has executed a fewer number of iterations, and needed less number of trials per each iteration. Therefore, the number of calls to the time consuming methods decreased, thusly decreasing the overall running time.

It is reasonable that the running time for `ConstructRule()` and `IntializeNodeInformation()` increases with the use of logical negation operator. This is due to the duplication of the number of nodes in the construction graph. For the former method, the number of to select from increases, so the `CalculateProbabilities()` and `SelectNodeProbablistically()` take more time. For the latter method, more time is needed to set node information regarding occurrence frequencies and heuristic values, as more nodes are available in the case of using logical negation. Nonetheless, according to the running time compared to the original one, the increase of running time of each method did not affect the overall running time.

5.5 Summary

This chapter has introduced the first extension to the original Ant-Miner algorithm, which is the use of logical negation operator. Allowing the use of logical negation operator in constructing rule antecedents produces rules with higher coverage and decreases the number of rules needed to cover the minimum coverage needed to stop execution. This enhances the output in terms of comprehensibility and decreases the overall running time. Moreover, results show that it has a positive effect on the classification accuracy of the generated rules.

Chapter 6

INCORPORATING STUBBORN ANTS

6.1 Introduction

Stubborn ants were introduced in 2008 in [1]. The idea is to promote search diversity by having each ant be influenced by its own history of constructing solutions in addition to the pheromone trails left by other ants. Basically, each ant does several trials in the execution of the algorithm. Each ant memorizes the best solution that it has constructed during its own trials. If a term belongs to the antecedents of rule, then the term will have an amplified probability of being selected by the ant, with the degree of amplification depending on the quality of the solution. Such a technique helps in finding different solutions as each ant will have a partially different search path in the construction graph, which leads to improving the quality of the output rules in terms of classification accuracy.

6.2 Stubborn Ants

In the original version of Ant-Miner algorithm, the state transition procedure depends on the heuristic value for a node representing a given term and its pheromone level currently associated with this node (see equation 3.2). Thus, the probability of selecting $term_{ij}$ does not differ from an ant to another. In other words, an ant does not have any identity or special behavior in selecting terms and constructing a rule. The idea behind stubborn ants is to promote search diversity by having each ant be influenced by its own history of constructing solutions.

Stubborn ants were first introduced by A. M. Abedlbar in [1]. Originally it was used to solve optimization problems such TSP. In stubborn ants, not only can an ant learn from the experience of other ants via pheromones - which gives a clue about the quality of the selected decisions (term) in the previous trials, each ant can also learn from its own history of constructing solutions. Consequently, each ant will have a partially different search path, which introduces diversity in the colony. Basically, each ant does several trials in the execution of the algorithm. Each ant_t memorizes the best solution R_t^+ that it has constructed during its own trials. The probability $term_{ij}$ to be selected by ant_t is amplified by the quality of the best solution R_t^+ that the ant memorizes from its history if the $term_{ij}$ belongs to the antecedents of rule R_t^+.

To incorporate stubborn ants in the Ant-Miner algorithm, the following pseudo-code shows the required modification on Ant-Miner algorithm:

Algorithm 6.1 - Ant-Miner with Stubborn Ants.

TrainingSet = {all training cases};

DiscoveredRuleList = [ ]; /* initialize rule list with empty list */

AntList=Ants[Ants Number];

WHILE (TrainingSet < Min_covered_cases)

$t = 1$; /* ant index*/

$j = 1$; /* convergence test index */

$i = 1$; /* trial index */

Initialize all trails with the same amount of pheromone;

REPEAT

FOR EACH Ant_t in AntList

Ant_t starts with an empty rule and incrementally constructs a classification rule R_t by adding one term at a time to the current rule; /* influenced by pheromone amount, heuristic function value and best history rule*/

Prune rule R_t; /* remove irrelevant terms from rule */

Update the pheromone of all trails by increasing pheromone in the trail followed by Ant_t (proportional to the quality of R_t) and decreasing pheromone in the other trails (simulating pheromone evaporation);

IF (R_t Quality > Ant_t History Best Rule Quality)

THEN Ant_t History Best Rule = R_t /* update best history rule */

IF (R_t is equal to R_{t-1}) /* update convergence test */

THEN $j = j + 1;$

ELSE $j = 1;$

END IF

$t = t + 1;$

END FOR EACH

$i = i + 1;$

UNTIL (t ≥ No_of_Ants) OR (j ≥ No_rules_converg)

$t = 1;$

UNTIL (i ≥ No_of_trials)

Choose the best rule R_{best} among all rules R_t constructed by all the ants;

Add rule R_{best} to DiscoveredRuleList;

TrainingSet = TrainingSet - {set of cases correctly covered by R_{best} };

END WHILE

As shown in Algorithm 6.1, Ant-Miner with stubborn ants, a set of ants does several trials to construct classification rules. Therefore, an extra outer loop is added before the FOR EACH ant loop. This outer loop helps each ant to do several trials during the execution of the algorithm. Note that the size of the colony affects the behavior of the stubborn ants; as the number of the ants decreases, the stubbornness effect is more applied, given that the total number of trials per iteration is fixed. For example, given that the maximum trials allowed per iteration is 3000, if the colony has 3000 ants, then each ant will do only one trial, which is the case of the original algorithm. On the other hand, if the size of the colony is 30, in such case each ant can do up to 100 trials. And if the size of the colony is 10, then the number of iterations that can be performed by a single ant is 300. Therefore, the number of ants and the number of trials per ant should be adapted for each data set according to the required average trials per iteration in order to amplify the effect of stubborn ants to the appropriate amount.

Each ant memorizes the rule that has best quality from the rules that it constructed in the previous trials. The quality of the best rule influences the ant's decision in choosing of a term in the current rule construction. The probability that a term will be added to the current rule is given by the following formula:

$$P_{ij}(t) = \frac{V_{ij}}{\sum_{r=1}^{a}\sum_{s=1}^{br}(V_{rs})} \tag{6.1}$$

where:

- $V_{ij} = \left(\eta_{ij} \cdot \tau_{ij}(t)\right) + \left(\eta_{ij} \cdot \tau_{ij}(t)\right).Q(R_t^+)$ if $term_{ij}$ belongs to current ant's history best rule, R_t^+, otherwise $V_{ij} = \eta_{ij} \cdot \tau_{ij}(t)$.
- η_{ij} is the value of a problem-dependent heuristic function.

- $\tau_{ij}(t)$ is the amount of pheromone associated with $term_{ij}$ at iteration t.
- a is the total number of attributes.
- b_i is the number of values in domain of the i-th attribute.
- $Q(R_t^+)$ is the quality of the current ant best history rule.

Stubborn ants add individuality to each ant, which promotes exploration and diversity in the algorithm. This tends to discover better solutions. Results show an increase of rule accuracy when using stubborn ants as well as a decrease in number of trials per iteration.

6.3 Stubborn Ant Implementation

This section discusses the modifications needed on the implementation of the original Ant-Miner program in order to enable the use of stubborn ants. Implications on the execution running time are discussed as well.

6.3.1 Data Structures and Operations

The first modification on the code that was made to enable the use of stubborn ants is on the data structure representation of the ant. The following code shows the new ant data structure representation to cope with stubborn ants:

```
public class Ant
{

      private int _antNumber;
      private  int[] _currentRuleAntecedents;
      private int _currentRuleclassIndex;
      private double _currentRuleQuality;
      private  int[] _historyBestRuleAntecedents;
      private double _historyBestRuleQuality;
      private List<int> _instancesIndexList;
      private bool[] _memory;
…
}
```

In order to enable the ant to memorize its best rule constructed thus far, two new data fields are added to the ant data structure: `hisoryBestRuleAntecedents`, an array of integers representing the indices of attributes values structuring the best rule discovered by the ant so far, and `historyBestRuleQuality`, which is the quality of the this rule.

The second modification is in the logic of executing the algorithm. First, an array of ants is intialized with the number of ants in the colony, so that each ant can live to perform more than a trial, memorizing its best constructed rule. In case of stubborn ants we have three nested loops. The (while) loop that represents global iterations in which each iteration a rule is discovered. Inside it, a new loop is added, which is (for) loop that repesents the number of trials that each ant would perform. Finally, in each iteration of the previous loop, a (for each ant) loop iterates on the ants in the colony so that each perfrom a rule discovery trial. The following code shows the implementation of the logic:

```
Ant[] ants = new Ant[AntsNumber];
while (this._currentIterationNumber < MaxIterationsNumber &&
this._currentCoverage < this.MinCoveragePercentage)
{

      this.InitializePheromone();
      this.InitializeNodeInformation();
      this.InitiazlizeAnts();

      …

for (_currentIterationNumber = 0; _currentIterationNumber <
this.AntsNumber && !convergence; _currentIterationNumber++)
{

      foreach (Ant ant in ants)
      {
            this._currentAnt = ant;
            this.ConstructRule(this._currentAnt);

            …
```

After each ant constructs a rule, if the new constructed rule has a better quality than the current memorized rule, then this new rule is set to `hisoryBestRuleAntecedents` and becomes the ant best constructed rule, as follows:

```
…

generatedAnts[trialIndex] = this._currentAnt;

if (this._currentAnt.CurrentRuleQuality >
this._currentAnt.HistoryBestRuleQuality)
{
this._currentAnt._historyBestRuleQuality  =
this._currentAnt._currentRuleQuality;
this._currentAnt._historyBestRuleAntecedents =
this._currentAnt._currentRuleAntecedents.Clone() as int[];
}

…
```

The third important modification that is done on the code to use stubborn ants is in the `CalculateNodeProbabilities()` method in order to amplify the probability of selecting a node if its term exists in the current ant's `_historyBestRule`. The code is modified as follows:

```
…

for (attributeIndex = 0; attributeIndex < this._constructionGraph.Length
- 1; attributeIndex++)
{
for (valueIndex = 0; valueIndex <
this._constructionGraph[attributeIndex].Length; valueIndex++)
{
if (!ant.Memory[attributeIndex])
{
double value =
this._constructionGraph[attributeIndex][valueIndex].HeuristicValue
*
this._constructionGraph[attributeIndex][valueIndex].Pheromone;

if (ant.HistoryBestRule != null && ant.HistoryBestRule[attributeIndex]
!= -1)
      {value += value * ant._historyBestRuleQuality;}
```

```
value = value / sum;

this._constructionGraph[attributeIndex][ValueIndex].Probability = value;
}
else

this._constructionGraph[attributeIndex][ValueIndex].Probability = 0.0;
}
}
```

6.3.2 Execution Profiling and Analysis

The following table exhibits the execution profile of Ant-Miner program after implementing the use of stubborn ants (as described in the previous subsection). The execution profile shows how such a modification has affected the running time of the program.

Method	Time (m.sec)	Calls #	Avg. Time (m.sec)	% to Parent	% to Total
Run()	5017.3	1	5017.3	100%	100%
>ConstructRule()	652.249	907	0.719	13%	13%
>>CalculateNodeProbabilitie()	87.03	4105	0.021	2%	<1%
>>SelectNodeProbablistically()	0.9	4205	0.0002	< 1 %	<1%
>PruneRule()	2358.131	907	2.599	42%	47%
>>CalculateRuleQuality()	1705.882	2242	0.76	19%	34%
>>DetermineRuleClass()	1053.633	2242	0.469	15%	21%
>UpdatePheromone()	501.73	907	0.553	< 1 %	<1%
>IntializeNodeInformation()	702.422	7	100.346	15%	14%
>IntializePheromone()	0.015	7	0.002	< 1 %	<1%
>BuildConstructionGraph()	1.02	1	1.02	<1%	<1%

Table 6.1 - Stubborn Ants Excution Profile.

As shown in the previous execution profiling, the `ConstructRule()` running time increased than its original version with 3% because it calls the method `CalculateNodeProbabilities()` whose running time has increased.

Nonetheless, the overall running time of the algorithm, represented in the `Run()` method, is almost the same as the execution of the original Ant-Miner. This is because the number of overall iterations has decreased from 8 to 7 and the number of total trials has decreased from 1112 to 809. This compensated for the increase of running time of the aforementioned methods, which are called in each trial.

6.4 Summary

This chapter has presented the use of stubborn ants in the context of Ant-Miner classification rule discovery algorithm. The motivation is to introduce search diversity by giving identity to each ant in the colony. Each ant learns from its own history besides the experience of other ants in constructing classification rules. Each ant memorizes its own history best rule that it has constructed during its previous trials. When constructing a new rule, the probability of selecting a term for a rule is amplified by the quality of the memorized rule if this term exists in it. Imperial results have shown improvements in the classification accuracy of the generated rules. Moreover, the running time has not been damaged by applying such a modification on the program original of the Ant-Miner algorithm.

Chapter 7

UTILIZING MULTI-PHEROMONE ANT SYSTEM

7.1 Introduction

Multi-pheromone is a new ACO system where multiple types of pheromone are used. In Ant-Miner, one type of pheromone for each permitted rule class can be deposited. In essence, an ant would first select the rule class and then deposit the corresponding type of pheromone. Unlike the original version of Ant-Miner where the class is selected after rule antecedents construction, in multi-pheromone system the ant chooses the terms that are specifically related to the classification of the previously selected class. Moreover, the ant constructing a rule labeled by C_x is only influenced by the pheromone corresponding to this class which was deposited by ants previously constructed rules labeled by C_x. Such a modification led to other changes in the algorithm in order to maximize the quality of the discovered rules in terms of comprehensibility and classification accuracy. A different heuristic function, which focuses on the confidence of the term to be selected given the pre-selected class, is used. An even more appropriate rule quality evaluation function is customized for evaluating rules constructed using such a system. A new proposed pheromone update strategy, named Quality Contrast Intensifier, is used. This aims to reward a rule whose quality is higher than a certain threshold by depositing more pheromone and penalizing a low-quality rule by removing pheromone from its terms in the construction graph. Finally, a new rule convergence test

is used to ensure that the produced rule satisfies a minimum quality threshold. Otherwise, this convergence should be ignored, re-initialize, and start looking for better rules.

7.2 Multi-Pheromone Ant System

In the original Ant-Miner, the consequent of a rule is chosen after its antecedents are selected by determining the class value with maximum occurrence in the cases matching the rule premises. Thus, a term is chosen for rule antecedents in order to decreasing entropy in the class distribution of cases that match the rule in construction However, selecting the rule class before constructing the rule antecedents allows choosing antecedent terms that are specifically related to the classification of the pre-selected rule class. The idea of selecting the rule consequent prior to rule construction was introduced in different flavors. These ideas are introduced in Chapter 4 - Ant-Miner Related Work. The following brief description on them:

- J. Smaldon and A. Frietas in [23] introduced an algorithm that tries to construct rules for each class independently: an extra For-Each (class value) loop is added as an outer loop for the original algorithm. The consequent of the rule is known by the ant during rule construction and does not change. An ant tries to choose terms that will produce the rule predicting the class value in the current iteration of the For-Each loop with an optimum level of accuracy. This approach generates better rules in comparison with the original Ant-Miner where a term is chosen for a rule only in order to decrease entropy in the class distribution of cases matching the rule under construction. However, the entire execution (with the complete training set) is repeated separately for each class value until the number of positive examples (belonging to the current class) remaining in the dataset that have not been covered by the discovered rules is

less than or equal to max uncovered cases. This increases the algorithm running time. Moreover, the number of the generated rules by this version is increased, which damages the simplicity of the output. For a more detailed description of the algorithm, refer to [23].

- D. Martens introduced the same idea in Ant-Miner+ [18]. An extra vertex group is added at the start in the construction graph containing class values to allow the selection of class first. This is similar to considering the class as another variable. Rules with different classes can be constructed in the same iteration. Different heuristic values are applied according to the selected class in order to choose the term that is relevant to the prediction of the selected class. However, the pheromone is shared by all ants constructing rules with different consequents. In other words, any ant is influenced by the pheromone dropped by any other ant constructing similar or different labeled rules. This can negatively affect the quality of the constructed rules, as the terms that lead to constructing a good rule with class C_x as a consequent do not necessarily lead to constructing a good rule with C_y as a consequent for a classification rule.

Unlike the version of Ant-Miner in [23], our proposed multi-pheromone Ant-Miner system executes the course of operations only once during the entire training process. Ants in the multi-pheromone system can construct rules with different consequent classes in the same iteration simultaneously. Nonetheless, the ant is only influenced by the ants that have constructed rules with the same consequent, using a multiple types of pheromone system.

First, an ant probabilistically selects the rule consequent prior to antecedents based on pheromone information as described below. Then, it tries to choose terms that are relevant to predicting this class. The rule is then evaluated and the pheromone is updated. But, unlike the version of Ant-Miner in [18], the ant drops different kinds of pheromone as many as the permitted classes. The next ant is only influenced by the amount of the pheromone deposited for the class for which it is trying to construct a rule. In this case, pheromone is not shared amongst ants constructing rules for different classes. This allows choosing terms that are only relevant to the selected class. The algorithm is shown in Algorithm 7.1 – Multi-pheromone Ant-Miner.

Algorithm 7.1 - Multi-pheromone Ant-Miner.

TrainingSet = {all training cases};

DiscoveredRuleList = [ ]; /* initialize rule list with empty list */

WHILE (TrainingSet < min_covered_cases)

t = 1; /* ant index, and also rule index */

Is_convergence=false /* a flag for convergence test*/

Initialize pheromone of class value nodes.

Initialize pheromone type of the term nodes dedicated for the class of previously constructed rule, and leave the other pheromone types as they are. /* if it is the first iteration, all pheromone array elements are initialized in each node.*/

REPEAT

Ant_t Probabilistically **selects a rule consequent class** according to the pheromone information associated to the class values.

Ant_t starts with an empty rule and incrementally constructs a classification rule R_t by **adding one term** at a time to the current rule.

Prune rule R_t; /* remove irrelevant terms from rule */

Update the pheromone type of corresponding to R_t class value in the construction graph using **Quality Contrast Intensifier**;

Update the pheromone of R_t class in the class value nodes /* this will affect the selection of the class for subsequent ants*/

Apply Convergence Test;

UNTIL (i ≥ no_of_ants) OR (Is_convergence)

Choose the best rule R_{best} among all rules R_t constructed by all the ants, add rule R_{best} to DiscoveredRuleList;

TrainingSet = TrainingSet - {set of cases correctly covered by R_{best}};

END WHILE

As shown in Algorithm 7.1 – Multi-pheromone Ant-Miner, the idea of multi-pheromone Ant-Miner is that each class has a different pheromone to be deposited on the terms in the construction graph. In essence, we are replacing the traditional two-dimensional pheromone structure (attribute, value) by a new three-dimensional pheromone structure (attribute, value, class). The same applies as to the heuristic value structure; class-based structure.

During rule construction, the rule class is already set and an ant is only influenced by the amount of pheromone in the pheromone array element dedicated to its rule class. Similarly in pheromone update, an ant deposits pheromone in the array element dedicated

to the current rule class in each node belonging to the trial. Class values are also represented in nodes in the construction graph, and pheromone can be deposited on them. This pheromone affects the probability of selecting the rule class for subsequent ants. The pheromone is initialized in the node of class values as follows:

$$\tau_c = \frac{freq(c)}{|TrainingSet|} \tag{7.1}$$

where:

- As where $freq(c)$ is the number of instances labeled with class C.
- $|TrainingSet|$ is the size of the training set.

In pheromone update, the of pheromone level increases in the node of the constructed rule class according to the quality of the rule, as follows:

$$\tau_{c_r}(t) = \tau_{c_r}(t-1) + \tau_{c_r}(t-1).Q_r \tag{7.2}$$

where:

- C_r is the class of the constructed rule.
- Q_r is quality of the constructed rule.

The problem dependent heuristic function chosen is the Laplace-corrected confidence for each term as in [27], given by:

$$\eta_{ij,k} = \frac{|term_{ij}, \mathrm{k}| + 1}{|\mathrm{term_{ij}}| + \mathrm{No\_of\_classes}} \tag{7.3}$$

where:

- $\eta_{ij,k}$ is the heuristic value for $term_{ij}$ given that class k is selected.

- $|term_{ij}, k|$ is the number of training cases having $term_{ij}$ and the current selected class k.
- $|term_{ij}|$ is the number of training cases having $term_{ij}$.
- No_of_classes is the number of values in the class attribute's domain.

The probability of selecting $term_{ij}$ given that class k is chosen is calculated is follows:

$$P_{ij,k} = \frac{\eta_{ij,k} \cdot \tau_{ij,k}(t)}{\sum_{r=1}^{a} \sum_{s=1}^{b_r} \left(\eta_{rs,k} \cdot \tau_{rs,k}(t)\right)} \tag{7.4}$$

where:

- $\eta_{ij,k}$ is the value of a problem-dependent heuristic function for value *j*-th in attribute *i*-th for class *k*
- $\tau_{ij,k}(t)$ is the amount of pheromone associated with $term_{ij}$ for class *k* at iteration *t*.
- a is the total number of attributes.
- b_r is the number of values in domain of the *r*-th attribute.

The rule generated via multi-pheromone system is evaluated, to update the pheromone levels (as described in the following subsection), by a function that balances between the support and the confidence of the rule, as follows:

$$Q(R_t) = Confidence(R_t) + Support(R_t) \tag{7.5}$$

where:

- $Confidence(R_t) = \frac{TP}{|Matches|}$, represents the ratio of the number of cases that match rule R_t's premises and are labeled by its class to the total number of cases that match R_t's premises.

- $Support(R_t) = \frac{TP}{|TrainingSet|}$, represents the ratio of the number of cases that match R_t's premises and are labeled by its class to the total number of cases in the training set.

After the best iteration rule is selected, the cases covered by this rule are removed from the training set and the pheromone is initialized but only in the pheromone array element dedicated to the class of this rule. Leaving the pheromone in the array element of other classes tends not to waste the wisdom that has been collected by the ants in the previous trails for the rest of the classes, leading to faster convergence in the next iterations.

Note that for applying multi-pheromone Ant-Miner system with stubborn ants, each ant in the swarm should memorize the best rules it has generated, one for each class value. Hence, when an ant tries to select a term from the construction graph, knowing that the class is already set, the probability of selecting this term is amplified by the best rule that the ant memorizes for this current class if this term occurs in this rule.

Multi-pheromone Ant-Miner system generates better rule sets in terms of predictive accuracy with a smaller number of rules, which improves the Ant-Miner performance as a classification algorithm in terms of efficiency and comprehensibility. The reasons that make multi-pheromone technique outperform the original one are summarized in the following points:

1. The rule consequent (class) is chosen prior to rule antecedents (terms): this allows the ant to select terms that are relevant to the classification of the selected class, not to decrease entropy in the class distribution of cases matching the rule under construction. A better heuristic function is used in multi-pheromone (equation 7.3),

which is related to the confidence of a term given the selected class. Such a heuristic function leads to better terms that have classification relevance to the selected class.

2. A better rule evaluation function is used: the evaluation function used for multi-pheromone Ant-Miner works better in determining the classification accuracy of the generated rule, as it evolves the rule support and its confidence. Such an evaluation function is suitable in multi-pheromone system as the rule is constructed to improve its confidence given the selected class. Unlike the original version, where the rule is constructed to reduce entropy of the class distribution in the cases of the dataset. Moreover, as the evaluation function balances between the coverage of the rule and its classification accuracy, the size of the output rule set is reduced.
3. The pheromone in the construction graph is a three-dimensional structure (attribute, value, class). This is behind calling this system multi-pheromone. After rule construction, an ant deposits on the selected terms a specific type of pheromone corresponding to the rule class. Subsequently, the following ants that select the same class are only influenced by this type of pheromone in term selection. In other words, an ant constructing a rule labeled by C_x is not influenced by pheromone deposited by previous ants constructed rules labeled by C_y or C_z . An ant constructing a rule labeled by C_x is only influenced by pheromone deposited by previous ants constructing rules labeled by C_x. Such a technique prevents selecting irrelevant terms to the classification of the currently chosen rule class. This is unlike AntMiner+[18] where the pheromone is shared between all the ants constructing similar or different labeled rules.

4. The rule class is selected probabilistically, based on the heuristic information of the class: this allows constructing rules with different classes in the same algorithm iteration. Hence, best rules among all the available classes are constructed first, leading to a better classification accuracy rule set output with fewer rules to be generated, in comparison to the new Ant-Miner version proposed in [23], where the rule classes are selected iteratively. In latter versions, the whole algorithm is repeated for each class with entire training set. This produced an unordered rule set with more rules and terms per rule.

The following is a sample output of the rules generated by both the original Ant-Miner and the multi-pheromone Ant-miner applied on Car Evaluation data set (see section 9.2 Chapter 9).

Original Ant-Miner			**Multi-pheromone Ant-miner**		
Rule	Sup.	Conf.	Rule	Sup.	Conf.
IF <Persons=2> **Then** <Class=unacceptable>	0.33	1	**IF** <Safety=Low> **Then** <Class=unacceptable>	0.33	1
IF<Safety=medium> **Then** <Class=acceptable>	0.15	0.46	**IF** <Persons=2> **Then** <Class=unacceptable>	0.33	1
IF <Buying=Very high> **Then** <Class=unacceptable>	0.19	0.75	**IF** <Safety=high> **Then** <Class=acc>	0.27	0.73
IF <Buying=High> **Then** <Class=unacceptable>	0.22	0.66	**IF** <luggage=Small> **Then** <Class=unacceptable>	0.47	0.74
IF <Safety=High> **Then** <Class=acceptable>	0.24	0.47	**IF** <Safety=medium> **Then** <Class=unacceptable >	0.55	0.5
IF <Doors=2> **Then** <Class=unacceptable>	0.25	1			
IF <Buying=medium> **Then** <Class=unacceptable>	0.48	1			
IF <Persons=more> **Then** <Class=acceptable>	0.51	1			
	Cov. 97%	Acc. 77%		Cov. 100%	Acc. 84%

As shown in the previous table, multi-pheromone Ant-miner has produced a rule set with a higher classification accuracy (84% compared to 77%) and with fewer rules (5 rules compared to 8 rules). It has been noticed that about 85% of the runs of multi-pheromone algorithm produce the first 2 rules, and 75% of the runs produce the first three rules in order. These top rules have a confidence of 100% and the highest possible support. Note that the rules constructed at first have a higher confidence than the rules constructed later on. This is unlike the original version, where better rules could be constructed first. This proves that multi-pheromone targets the best relevant terms to the classification accuracy of a given class. By constructing best rules first, the number of generated rules is reduced. Moreover, as the class with the value "unacceptable" has the largest number of the cases, the multi-pheromone system tends to construct rules labeled by this class in order to generate rules with higher coverage and reduce the number of the generated rule set.

7.3 Quality Contrast Intensifier

In the pheromone update procedure for typical ACO algorithms, the amount of pheromone deposited is based on the quality of the trial. The idea is to intensify the contrast between bad solutions, good solutions and better ones as well as the unvisited solution. Quality contrast intensifier takes place as a new strategy for the pheromone update procedure. An ant that constructed a solution with good quality is rewarded by amplifying the amount of the pheromone to be dropped in its trail. By contrast, the ant that constructed a bad rule is penalized by removing pheromone from its trial according to the weakness of the constructed solution.

In order to apply such an idea in Ant-Miner, we consider the quality of the generated rule, which involves both support and confidence of the rule (see equation 7.5). If the confidence of the constructed rule exceeds an upper threshold φ_1, the pheromone to be deposited for this rule is amplified. On the other hand, if the confidence of the rule gets below a lower threshold φ_2, pheromone should be removed from the trial of this rule. This is shown as follows:

$$Ph_{ij}(\mathrm{t}) = \begin{cases} 2.Q(R_t), & if\ confidence(R_t) > \varphi_1 \\ Q(R_t), & if\ \varphi_2 < confidence(R_t) < \varphi_1 \\ 2 - Q(R_t), & if\ confidence(R_t) < \varphi_2 \end{cases} \quad (7.6)$$

where:

- $Ph_{ij}(\mathrm{t})$ is the amount of pheromone to be deposited in iteration t.
- $Q(R_t)$ is the quality of the rule R_t generated by the aforementioned rule quality evaluation function (7.5).
- φ_1 and $\varphi_2$are the upper and lower thresholds for the rule confidence at which the quality is contrasted. Typical used values are 0.85 and 0.35 respectively, given that both support and confidence values ranges from 0 to 1.

Such a strategy comes with several advantages. First, higher quality rules get significantly more pheromone than other normal and low quality solution, which leads to faster convergence. Second, it ensures the balance in the quality of output between the number of the generated rules (which is affected by the rule support) and the classification accuracy of these rules (which is affect by the confidence of the rule). For example, some attribute values have a very high occurrence among the training set cases. This increases the support value in the quality evaluation, which increases the quality of the rule in general, regardless of the rule confidence. Thus, this quality contrast intensifier

works in the favor of the rule confidence in order not to generate a significantly fewer rules with low classification quality. Finally, penalizing bad rules by removing pheromone from its trial gives opportunity to unvisited nodes to be selected in further iterations, as their pheromone amount probably gets higher than the already tried bad nodes. This enhances the exploration aspect of the algorithm.

7.4 New Convergence Test

In the Ant-Miner algorithm, the best rule in each iteration is selected to be added to the discovered rule list. This is done after a certain number trials per iteration, or when a convergence occurs. A convergence occurs when there no more better rules are being generated after a certain number iterations (**no_rules_converg**). Sometimes, an early convergence occurs and causes stagnation in the ant colony, while the best discovered rule yet has an insufficient quality, or at least, better rules could have been discovered if it was not for the early stagnation.

The new proposed convergence test tries to overcome such a problem. The idea is to set a minimum threshold for the quality of the solution to be selected. If the algorithm converged on a solution that satisfies this threshold, then it is considered. Otherwise, re-initialization with some sort of randomization occurs so that better solution could be discovered in the subsequent trials. As for Ant-Miner, if the best discovered rule yet has a confidence that is lower a certain threshold (0.5) then pheromone levels in the nodes in the construction graph should be re-initialized randomly, in order to introduce noise in the colony so that better rules could be discovered. The **no_rules_converg** counter for the convergence test is rest as well. If no rule with the sufficient confidence threshold is discovered after a certain number of iterations, a convergence is now considered for the

algorithm. Note that the convergence test threshold is set only on the confidence of the rule not on the overall quality of the rule. The idea behind that is to prevent selecting rule with high support and low confidence. A rule to be considered for selection should satisfy the confidence threshold (if possible), and then the rule with the best overall quality is selected. This tends not to sacrifice the discovered rule set classification accuracy in favor of its size.

7.5 Multi-pheromone Implementation

The program code implementation for the multi-pheromone Ant-Miner system is described in this section. Several modifications have been done on the data structure used in the algorithm as well as the operations in order to apply the multi-pheromone behavior. The quality contrast intensifier procedure for pheromone updating is described, along with the implementation of the new convergence test logic. Running time implication of these modifications is exhibited through an effective execution profiler.

7.5.1 Data structure and Operations

- **Construction Graph Node Representation**: The most significant modification in the Ant-Miner algorithm used data structure is in the construction graph. The attribute value node, which represents the decision component in the construction graph, has the pheromone represented in an array, where the length of this array is the number of permitted classes. Each element in the array contains the pheromone amount for each class value. And similarly, each node has an array of heuristic information, one array information element for each class. The following code shows the implementation of the node data structure in the multi-pheromone system.

```
public struct Node
{
```

```
        public int AttributeIndex;
        public int ValueIndex;
        public int []ValueFrequency;
        public double []PheromoneAmounts;
        public double []HeuristicValues;
        public double []Probabilities;
        public bool UnusableValue;

    }
```

As shown in the previous code snippet, `PheromoneAmounts` represents the pehromone array for each node, `HeuristicValue` is the heuristic value array and `Probabilities` is the probability array where the probablity of selecting this node given a spcific class is calcualted and stored.

- **Class Value Node Representation:** A new data structure has been added to the construction graph that represents the available classes in the domain of the current dataset. This data strucutre contains the current pheromone amount existing per each class value, which intialy set to the frequency of the occerance of the class value in the dataset. This data sturcture is used for calculating the probablity of selecting a class value by an ant at the begineeing of an itration. This data structure is a member in the `AntColony` class, and considered as a part of the whole construction graph.

```
private int[] _classFreq;
…
this._classFreq = new int[numberOfClasses];
```

- **Construction Graph Intialization:** The procedure for intializing construciton graph nodes in the multi-pheromone system changed, as several attributes in the node data structure became 2-dimensional members (a pheromone value for each different class, and a heuristic value for each different class). Thus, the construciton graph nodes are intiliaization as follows:

```
…
```

```
for (int attributeIndex = 0; attributeIndex <
this._constructionGraph.Length; attributeIndex++)
{
for (int valueIndex = 0; valueIndex <
this._constructionGraph[attributeIndex].Length; valueIndex++)
{

this._constructionGraph[attributeIndex][valueIndex].Probabilities = new
double[numberOfClasses];

this._constructionGraph[attributeIndex][valueIndex].PheromoneAmounts =
new double[numberOfClasses];

this._constructionGraph[attributeIndex][valueIndex].HeuristicValues =
new double[numberOfClasses];

this._constructionGraph[attributeIndex][valueIndex].Frequency =
new int[numberOfClasses];
}

}

…
```

- **Execution behaviour implementation:** The `Run()` method is the main operation that executes the Ant-Miner. As for the multi-pheromone Ant-Miner system, the folowing code shows the modifications on the `Run()` operation to cope with the multi-pheromone logic.

```
while (this._currentIterationNumber < MaxIterationsNumber &&
this._currentCoverage < this.MinCoveragePercentage)
{

this.InitializePheromone();
this.InitializeNodeInformation(); //using laplace-corrected confidence

…

for (_currentIterationNumber = 0; _currentIterationNumber <
this.AntsNumber && !convergence; _currentIterationNumber++)
{
this._currentAnt = new Ant();
this.SelectRuleClass(this._currentAnt); //before rule construction
this.CalculateProbabilities(this._currentAnt);
this.ConstructRule(this._currentAnt);
this.CalculateRuleQuality(this._currentAnt);//using support+confidence
this._currentAnt = this.PruneRule(this._currentAnt);
generatedAnts[_currentIterationNumber] = this._currentAnt;

if (_currentAnt.RuleQuality > generatedAnts[bestAntIndex].RuleQuality)
bestAntIndex = _currentIterationNumber;
```

```
this.UpdatePheromoneUsingQualityContrastIntensifier
(this._currentAnt);

convergence=ApplyNewConvergenceTest(this._currentAnt);

…

}

this.OutputAntRules.Add(generatedAnts[bestAntIndex]);
this.RemoveCoverdCasesFromTrainingSet(generatedAnts[bestAntIndex]);

…

}
```

As shown in the previous code, pheromone levels and heuristic information are intialized for each node in the construction graph at the begineeing of each iteraion. As for multi-pheromone, the pheromone is intialized in `PheromoneAmounts` array elements that are corssponding to the available classes. The heuristic information is claculated using laplace-corrected function (see equation 7.3) for each attribute value-class and set in the `HeuristicValues` array. For each trial in an iteration, the ant first select the rule consquent class before constructing the rule antecedents terms. This selection is done probablistically according to the pheromone information in `_classFreq` data structure. After the class is selected, the ant constructs the rule antecedents, following only the pheromone amount and the heuristic information associated with the current selected class, as shown in the implementation of `CalculateProbabilities()` method as follows:

```
private void CalculateProbabilities(Ant ant)
{
double sum = 0.0;
int attributeIndex = 0, valueIndex = 0;
for (attributeIndex = 0; attributeIndex <
this._constructionGraph.Length-1; attributeIndex++)
//if the attribute has not been used...
if (!ant.Memory[attributeIndex])
for (valueIndex = 0; valueIndex <
this._constructionGraph[attributeIndex].Length; valueIndex++)
```

```
sum +=
this._constructionGraph[attributeIndex][valueIndex].
HeuristicValues[ant.CurrentRuleClassIndex] *
this._constructionGraph[attributeIndex][valueIndex].
PheromoneAmounts[ant.CurrentRuleClassIndex];

for (attributeIndex = 0; attributeIndex <
this._constructionGraph.Length-1; attributeIndex++)
{
      for (valueIndex = 0; valueIndex <
      this._constructionGraph[attributeIndex].Length; valueIndex++)
      {
      if (!ant.Memory[attributeIndex])
      {

      this._constructionGraph[attributeIndex][valueIndex].
      Probabilities[ant.CurrentRuleClassIndex]
      =
      this._constructionGraph[attributeIndex][valueIndex]
      .HeuristicValue[ant.CurrentRuleClassIndex] *
      this._constructionGraph[attributeIndex][valueIndex]
      .PheromoneAmounts[ant.CurrentRuleClassIndex / sum;

      }
      else
      this._constructionGraph[attributeIndex][valueIndex].
      Probabilities[ant.CurrentRuleClassIndex] = 0.0;
      }
}
}
```

After the rule is constructed, its quality is evaluated using the fitness funciton discussed in the previous section (see equation 7.5). This evaluation function calculates the support in the confidence of the generated rule, as the sum of them represents the overall quality of the rule. The quality of the rule is then used in the pheromone update procedure, carried out by `UpdatePheromoneUsingQualityContrast-Intensifier()` method. The following is its code implementation.

```
private void UpdatePheromoneUsingQaualityContrastIntensifier(Ant ant)
{
//update pheromone for class nodes

this.classFreq[ant.CurrentRuleClassIndex] +=
this. classFreq[ant.CurrentRuleClassIndex]*ant.CurrentRuleQuality;

//normalize pheromone in class nodes

…
```

```
//update pheromone for used terms
for (int attributeIndex = 0; attributeIndex < ant.CurrentRule.Length;
attributeIndex ++)
{
if (ant.CurrentRule[attributeIndex] != -1)
{
int valueIndex = ant.CurrentRule[attributeIndex];
double currentPheromoneValue =
this._constructionGraph[attributeIndex][valueIndex].
PheromoneAmounts[ant.CurrentRuleClassIndex];

if (ant.CurrentRuleConfidence >= this._phi1)
{
      currentPheromoneValue += 2 * ant.CurrentRuleQuality;
}
else if (ant.CurrentRuleConfidence <= this._phi2)
{
      currentPheromoneValue -= ant.CurrentRuleQuality;
}
else
{
      currentPheromoneValue += ant.CurrentRuleQuality;

}
This._constructionGraph[attributeIndex][valueIndex].
Pheromone[ant.CurrentRuleClassIndex] = currentPheromoneValue;
}
}

//normalize pheromone

…

}
```

Where `_phi1` and `_phi2` are the variables that represent the user deifned thresholds for the quality contrast intensifier. These variables are data members in the `AntColony` class.

The following code shows the implementation of the new convergence test procedure, in which the algorithm ensures that the discovered rule satisfies a minimum quality threshold before it is selected. Otherwise, the colony is considered to have an early stagnation and re-initialization with some randomization.

```
…

private int _ruleConvergenceCount;
```

```
private int _convergenceDeltaCount;
private int _maxReintializationCount;
private int _currentReinitlizationCount;
private double _confidenceThreshold;

…

public bool ApplyNewConvergenceTest(Ant ant)
{

bool convergence = false;

if (ant.CurrentRuleQuality ==
generatedAnts[currentTrialIndex-1].CurrentRuleQuality)
      _convergenceDeltaCount++;
else
      _convergenceDeltaCount=0;

if (_convergenceDeltaCount == _ruleConvergenceCount)
{
if (ant.CurrentRuleConfidence > _confidenceThreshold ||
_currentReinitlizationCount == _maxReintializationCount)
      convergence = true;

else
{
      this.ReintializePheromoneLevelRandomly(ant);
      this._convergenceDeltaCount=0;
      this._currentReinitlizationCount++;

}

}

return convergence;
}
```

7.5.2 Execution Profiling and Analysis

This section shows the effect of the multi-pheromone system on the program execution time of Ant-Miner. Table 7.1 exhibits the running time of the algorithm after applying such an extension and how the behavior of this new technique along with the used functions for quality evaluation and heuristic information calculation have affected the execution of the algorithm. A detailed analysis for the execution profile is discussed in the table below.

Method	Time (m.sec)	Calls #	Avg. Time (m.sec)	% to Parent	% to Total
`Run()`	3645.7	1	3645.7	100%	100%
`>ConstructRule()`	546.855	617	0.8863	15%	15%
`>>CalculateNodeProbabilities()`	47	2405	0.0195	2%	<1%
`>>SelectNodeProbablistically()`	0.9	2805	0.00032	< 1 %	<1%
`>PruneRule()`	1640.565	617	2.658	41%	45%
`>>CalculateRuleQuality()`	1020.796	2242	0.455	18%	28%
`>UpdatePheromoneWith-QualityContrastIntensifier()`	3.17	617	0.00513	< 1 %	<1%
`>IntializeNodeInformation()`	692.683	5	138.536	20%	19%
`>IntializePheromone()`	0.107	5	0.0214	< 1 %	<1%
`>BuildConstructionGraph()`	1.94	1	1.94	<1%	<1%

Table 7.1 - Multi-pheromone Ant-Miner Execution Profile.

As shown in the previous table, multi-pheromone Ant-Miner outperforms the original version of Ant-Miner in terms of execution time. The average running time is less than 70% of the running time of the original algorithm. Multi-pheromone system takes less running time because of two main reasons. The first reason is that the number of overall iterations that the algorithm takes is less than the original one. This is due to the high coverage of the generated rules besides its confidence. Thus, less number of rules is needed to cover a sufficient portion of the training set. Consequently, a fewer number of iterations the algorithm executes. The second reason of the reduced execution time using multi-pheromone system is the fact that the single ant trial takes less time. In each ant trial, `ConstructRule()`, `PruneRule()`, and pheromone update methods are called. The profilers show that both rule construction and pheromone update methods takes almost the same time in both algorithms. However, the most time consuming method, `PruneRule()` (47% of the execution time of the algorithm), takes less time in

the multi-pheromone algorithm than its corresponding method in the original Ant-Miner algorithm. This is caused by the fact that the `CalculateRuleQuality()` method , that is called each time by `PruneRule()`, takes less time in the multi-pheromone algorithm. As in the original Ant-Miner algorithm, the program has to scan the whole dataset cases in order to calculate the quality of the rule using equation 3.5 ($Q = sensitivity \times specificity$). On the other hand, in the multi-pheromone system, the program only needs to scan the `_instancesIndexList` associated with the ant (which contains the indcies of the cases covred by the rule premises) in order to evaluate the rule quality using equation 7.5 ($Q = Support \times Confidence$). Moreover, the `DetermindRuleClass()` method (which is also a time consuming method) is no longer called in `PruneRule()` method. Such an improvement has given the multi-pheromone algorithm an edge in the execution time, although the number of trials in the multi-pheromone system is more than the number of trials in the original algorithm due to the new convergence test.

7.6 Summary

This chapter has introduced a genuine Ant-Miner variation that leads to improve the quality of the output in terms classification accuracy and rule set size. Multi-pheromone is a new version of Ant-Miner in which the ant selects the rule consequent class prior to rule antecedent construction. The typical two-dimensional structure for pheromone and heuristic information is replaced by a new three dimensional one (attribute, value, class). In rule terms selection, the ant is only influenced by the pheromone amount and the heuristic information values associated to the selected class for a given term. This leads to the choice of terms that are more relevant to the

classification of the selected class, instead of choosing terms that minimize entropy among the class values. Laplace-corrected is used as a heuristic function. New quality evaluation function, which balances between the support of the rule and its confidence, is used in rule evaluation. The quality value is used to update the pheromone on the construction graph, using the quality contrast intensifier procedure. This procedure rewards rules that exceed a certain level of confidence with more pheromone, and penalizes low confidence rules by removing pheromone. This helps in producing rules with higher confidence. A new convergence test was introduced to ensure the classification quality of the generated rule before it is added to the discovered rule list. Chapter 9 experimental results show that multi-pheromone Ant-Miner system outperforms the original Ant-Miner in the terms of output efficiency and comprehensibility, without compromising execution running time.

Chapter 8

GIVING ANTS PERSONALITY

8.1 Introduction

Ants with personality were proposed in the future work section of [1]. The idea is to give each ant a personality so that each ant would have its own special behavior in selecting terms during the rule construction procedure. The aim is to introduce diversity in the search among the colony and empower the exploration aspect in the swarm behavior. One idea to apply this was the use of stubborn ants, introduced in Chapter 6. Another idea is to have each ant in the colony using its own weights for the social component and the cognitive component in the state transition rule (see equation 3.2).

Applying such a modification should lead to discover new, and probably better, solution during the execution of the algorithm. Empirical results show enhancements in the quality of the output in terms of classification accuracy. However, such diversity has increased the number of trials needed per iteration to converge.

8.2 Stagnation and Early Convergence

One of the most important challenges that face the ACO systems is the problem of stagnation and early convergence. Stagnation occurs when several ant trials are done without change or increase in the quality of the solutions that are found. This results in converging on a solution that might not be good enough, or at least much better solution

could be found if there was more diversity in the search amongst the ants in the colony. One of the reasons of early convergence is that the exploration aspect is not empathized during the solution construction procedure. Ants would tend to exploit the best solution that has been discovered so far, following the intense pheromone trial, without trying to explore different solution.

One idea to face such a challenge is to adapted the pheromone update procedure and introduce sophisticated evaporation strategies to avoid stagnation. This was introduced in many previous related works (see Chapter 4). Another idea is to give each ant a different behavior (or personality) in selecting decision components during solution construction.

8.3 Ants with Personality

For typical ACO systems, the probabilistic transition function is calculated as follows:

$$P_i = \frac{\eta_i^{\alpha} \cdot \tau_i^{\beta}(t)}{\sum_i^a \left(\eta_i^{\alpha} \cdot \tau_i^{\beta}(t)\right)} \tag{8.1}$$

As shown in the previous equation, the probability of an ant to select a node i depends on two components. The firs is heuristic value component (η) , which represents the cognitive aspect to the ant, and the pheromone value component (τ) ,which represents the its social aspect. The former represents the tendency of the ant to choose the node according to its quality. The later represents the tendency of the ant to choose the node according to the experience of the previous ants in selecting such a term. Both components are raised to the power of α and β respectivly. The exponents α and β are

used adjust the relative emphases of the pheromone and heuristic information terms. In the original Ant-Miner, α equals β equals to 1.0.

The idea of giving personality to each ant refers to assign different α equals β values for each ant in the colony. This was proposed in the future work section of [1]. The values of α and β are drawn from a random number generator using Gaussian distribution function with a mean of 2 and standard deviation σ value that ranges from 0 to 1. Hence, some ants may tend to choose terms according to their predictive quality - using heuristic value of the term - regardless of the selection of other previous ants. Other ants may tend to follow the experience of the previous ants - via pheromone trials - during rule construction. The aim of such modification is avoid stagnation and early convergence by empowering the exploration aspect in the colony so that better rules could be found. Note that a higher standard deviation σ value used, Note that a higher standard deviation value would introduce a higher range of diversity between ant behaviors in term selection. However, this could increase the number of trials needed in each iteration to converge on a rule.

8.4 Ants with Personality Implementation

This section discusses the modifications needed on the implementation on the original Ant-Miner program in order to enable the use of different values of α and β for each ant. Implications on the execution running time are discussed as well.

8.4.1 Data Structure and Operations

The only modification that should be done in the data structures of the Ant-Miner program is to add two double-value data fields representing values of α and β in the data structure representing the ant entity. The new ant structure is as follows:

```
public class Ant
{

      private int _antNumber;
      private double _alpha;
      private double _beta;
      private  int[] _ruleAntecedents;
      private int _ruleclassIndex;
      private double _currentRuleQuality;
      private List<int> _instancesIndexList;
      private bool[] _memory;
…
}
```

Both of the `_alpha` and `_beta` data fileds are set for the ant as it is created and intialized in its constructore using a Gaussian distribution function with mean of two and variable standard deviation, as show in the following code:

```
//inside ant constructor
…

this._alpha =
Utilities.RandomUtility.GetNextDoubleFromGaussianFunction(mean, stdv);
this._beta = 3 - this._alpha;

…
```

The last modification to be done is in calculating attribute value probability in selecting a node during the rule construction. The change is to use the `_alpha` and `_beta` values of each ant when calculating node probability. The folowing exhibit the change of the method `CalculateNodeProbabilities()`:

```
…

for (attributeIndex = 0; attributeIndex <
this._constructionGraph.Length-1; attributeIndex++)
{
for (valueIndex = 0; valueIndex <
this._constructionGraph[attributeIndex].Length; valueIndex++)
{
if (!ant.Memory[attributeIndex])
{

double result =
Math.Pow(
this._constructionGraph[attributeIndex][valueIndex].HeuristicValue,
ant._alpha)
```

```
*
Math.Pow( this._constructionGraph[attributeIndex][valueIndex].Pheromone,
ant._beta) / sum;

this._constructionGraph[attributeIndex][valueIndex].Probability =
result;

}
else
this._constructionGraph[attributeIndex][valueIndex].Probability = 0.0;
}
}

…
```

8.4.2 Execution Profiling and Analysis

The following table exhibits the execution profile of Ant-Miner program after implementing the use of values of α and β for each ant (as described in the previous subsection). Results of the output rules quality of the algorithm in terms of classification accuracy and comprehensibility are described in Chapter 9. The following execution profile shows how such a modification affects the running time of the program.

Method	Time (m.sec)	Calls #	Avg. Time (m.sec)	% to Parent	% to Total
Run()	9126.2	1	9126.2	100%	100%
>ConstructRule()	1186.406	1779	0.6668	13%	13%
>>CalculateNodeProbabilities()	62	6568	0.009	2%	<1%
>>SelectNodeProbablistically()	0.9	6702	0.0001	< 1 %	<1%
>PruneRule()	4289.314	1779	2.411	42%	47%
>>CalculateRuleQuality()	3102.908	3587	0.8650	19%	34%
>>DetermineRuleClass()	1916.502	3587	0.534	15%	21%
>UpdatePheromone()	3	1779	0.0016	< 1 %	<1%
>IntializeNodeInformation()	1277.668	8	159.708	15%	14%
>IntializePheromone()	0.07	8	0.0087	< 1 %	<1%
>BuildConstructionGraph()	0.6	1	0.6	<1%	<1%

Table 8.1 - Ants with Personality Execution Profile.

As shown in table 8.1, the average running time of each leaf method did not increased comparing to the original Ant-Miner methods running time. However, the number of trials in the execution of the algorithm increases as more diversity was introduced and led to late convergence. Consequently, the number of calls to `ConstructRule()` and `PruneRule()` methods increased which led to increase the overall running time to 60%. The increase of the number of trials was planned for, as more diversity was intentionally added to the colony. The advantage is that the output rules produced better results terms of accuracy.

8.5 Summary

The idea of giving ant personality was introduced in this chapter. The challenge was to overcome the problem of stagnation and early convergence. The proposed extension to avoid such a problem is to giving each ant its own values of the α and β parameters, different from those of the rest of the colony. In our experimental results, we use values of α and β drawn from a random number generator using a Gaussian distribution with a mean of 2 and a standard deviation (σ) that ranges from 0 to 1.

Chapter 9

EXPERIMENTS AND RESULTS

9.1 Introduction

This chapter describes the experiments that were conducted to examine the performance of the Ant-Miner algorithm using the proposed extensions that have been presented in the previous chapters. Modifications were tested individually and together on several public-dmain datasets with different properties. The next section describes the datasets used for the experiments. Section 9.3 presents the experimental approach carried out for testing the new extensions. Used values for the algorithm parameters are shown in section 9.4. Section 9.5 exhibits the results of the experiments with analysis on each dataset. Section 9.6 summarizes the results and concludes the experiments.

9.2 Datasets

The performance of the extended Ant-Miner with the new modifications was evaluated using eight public-domain data sets from the UCI (University of California at Irvine) dataset repository[26] – available from:

http://www.ics.uci.edu/~mlearn/MLRepository.html.

The main characteristics of the datasets are shown in Table 9.1. The extended version of Ant-Miner does not deal directly with continuous attributes, as they should be discretized in pre-processing steps. The chosen datasets include only categorical attributes in order to avoid the interference of the quality of the discretization method on the experiment.

Dataset	Number of cases	Number of attributes	Number of classes
Car Evaluation	1728	6	4
Nursery	12960	8	5
Tic-Tac-To	958	9	2
Mushrooms	8124	22	2
Dermatology	366	33	6
Soybean	307	35	19
Contraceptive Method Choice	1473	9	3
BDS	1248	8	2

Table 9.1 - Description of Dataset Used in the Experiments.

As shown in Table 9.1, eight datasets are used for experimenting the new extensions. The number of cases for each dataset ranges from 958 to 12960 cases. Each dataset has a set of features containing from 6 to 35 attributes. Two datasets have 3 values for class attribute, namely Tic-Tac-To, Mushrooms and DBS datasets. The other five datasets, Car Evaluation, Nursery, Dermatology, Contraceptive Method Choice and Soybean datasets have more than 2 values for the class attributes. As was mentioned, all the attributes in the dataset contains only categorical values to avoid discretization before running the experiments.

9.3 Experimental Approach

Ten-fold cross validation was used to split the dataset into a training set and testing set with ratio of 90% and 10% respectively. Each pair of training and testing data was used for experimenting with each combination of modifications (original, using negative attributes, using stubborn ants and multi-pheromone) 10 times and the average was taken. The average of the ten folds was taken to conclude on cross validation run. The cross-validation experiment process was carried out 10 times for each dataset with different random partitioning of training\testing cases.

The number of rules generated (which represents the comprehensibility of the output), the average number of trials per iteration (number of ant trials needed to converge) and the accuracy of the generated rules were recorded to evaluate the quality of the experiment.

For ants with personality, the algorithm has been executed on the eight datasets with different values for the standard deviation parameter (σ). Each value of standard deviation is tried 10 times for each training/testing pair taken from each dataset.

9.4 Algorithm Parameters

The following fixed parameter values where used for all experimental runs:

- **Number of Ants (number_of_ants) =5**. Note that each ant would do multiple trials. This is done to support Stubborn Ants.
- **Number of trials per Ant (number_of_trials per_ants) = 100**. This represents the number of trials that the set of ants in the swarm would do per iteration. Thus, each ant in the swarm would do this exact number of trials.
- **Number of trials to converge (no_rules_converg) =10.** This parameter is needed to test whether the whole swarm has converged to a specific rule or not. If the same rule is discovered by 10 consequent ants, this is considered convergence, so the iteration is exited and the rule is extracted.
- **Maximum Uncovered Cases (max_uncovered_cases) =10%**. This is the maximum percentage of cases are allowed to be left uncovered by the generated rules. If the number of uncovered cases is larger than this given parameter, the algorithm should continue discovering rules to cover more cases.

- **Number of Global Iterations =50.** This indicates the maximum number if global iterations needed to discover rules that can cover the minimum coverage of cases in the training set.
- **Quality Contrast Intensifier thresholds (φ_1, φ_2):** These are the upper and the lower thresholds for the rule confidence at which the quality is amplified and deducted respectively. Used values for the experiments are 0.75 and 0.35 respectively, given that both support and confidence values ranges from 0 to 1.

Note that total trials per iteration equals to **number_of_ants** multiplied by **number_of_trials_per_ant**, which equals to 500. Also note that it is the maximum number of trials per iteration as any iteration can stop execution if the **no_rules_converg** was met.

9.5 Experimental Results

The following is the results produced by testing the performance of the Ant-Miner with the new extensions on the chosen datasets. Results for each dataset are presented in a separate subsection. Each dataset subsection has tree items: 1) a table for experimental results summary of applying each extension individually and with other ones, for both multi-pheromone system and the original Ant-Miner system, 2) analysis of the results, and 3) detailed results used for the test of statistical significance (ANOVA) of each extension compared to the original one, along with the generated F-value and the significance type. Note that the critical F-value for 10 runs to indicate normal significance is 4.41. This means that there is a probability of 95% that the difference in results is significant and did not occur randomly. In other words, the hypothesis of difference in algorithm performance can be accepted by confidence of 95%. The F-value for 10 runs

that indicates strong significance is 8.29. This means that there is a probability of 99% that the difference in results is significant and did not occur randomly. Similarly, this tells that the hypothesis of difference in the algorithm performance can be accepted with confidence of 99%.

Ant with personality has a separate table of results which contains the results for each dataset using different values of standard deviation parameter given to the Gaussian distribution function used to generate the values for α and β for each ant.

9.5.1 Car Evaluation Dataset Results

9.5.1.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	8.46 ±0.09	87	76.75 ±0.38	5.98 ±0.18	154	80.03 ±0.31
Logical Negation	5.96 ±0.37	105	77.62 ±0.54	**3.06 ±0.05**	169	76.61 ±0.20
Stubborn Ants	8.47 ±0.11	68	78.01 ±0.63	6.02 ±0.19	91	80.74 ±0.35
Negation & Stubborn	6.35 ±0.15	84	77.88 ±0.49	4.2 ±0.21	114	**81.62 ±0.12**

Table 9.2 - Car Evaluation Dataset Experimental Results Summary.

9.5.1.2 *Results Analysis*

As shown in Table 9.2, using logical negation reduced the average number of rules generated by the algorithm, as the generated rules have a higher coverage of the training cases. Stubborn ants improved the average accuracy of the generated rules and reduced the average number of trials per iteration. Multi-pheromone system improved the average accuracy with most of the scenarios compared to the original version. Using multi-pheromone system along with stubborn ants and logical negation operator produced the best average accuracy with a reduced number of rules and a smaller number of trials per iteration.

9.5.1.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	8.44	76.90	5.74	77.27	8.26	78.84	5.86	79.93
2	8.30	76.66	5.68	77.75	8.50	77.13	6.10	80.42
3	8.46	76.29	6.08	77.04	8.56	77.10	5.96	80.30
4	8.44	77.15	5.24	77.65	8.52	78.21	6.24	80.00
5	8.58	76.73	6.26	78.53	8.34	78.02	5.61	79.96
6	8.60	77.48	6.36	77.76	8.58	78.26	6.22	79.78
7	8.52	76.81	5.73	77.63	8.60	77.64	5.94	79.39
8	8.48	76.82	6.32	78.03	8.48	78.95	5.86	80.31
9	8.34	76.37	6.38	78.00	8.34	78.28	6.02	79.88
10	8.46	76.23	5.85	75.68	8.54	77.71	5.98	80.32
F- Value			414.11	7.3	0.015	29.27	1386.4	430.37
Significance Type			**Strong**	**Normal**	**-**	**Strong**	**Strong**	**Strong**

Table 9.3 - Car Evaluation Dataset Detailed Results for ANOVA Test.

9.5.2 Tic-Ta-To Dataset Results

9.5.2.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	6.63 ±0.32	89	70.1 ±0.20	5.8 ±0.13	148	69.9 ±0.22
Logical Negation	5.3 ±0.20	120	71.6 ±0.25	**3.1 ±0.24**	109	70.8 ±0.19
Stubborn Ants	6.9 ±0.25	59	71.9 ±0.75	5.9 ±0.27	83	70.3 ±0.55
Negation & Stubborn	4.9 ±0.11	97	**72.4 ±0.34**	3.2 ±0.21	99	71.8 ±0.29

Table 9.4 - Tic-Tac-To Dataset Experimental Results Summary.

9.5.2.2 *Results Analysis*

In the Tic-Tac-Toe dataset (Table 9.4), the class attribute has two values. Multi-pheromone did not improve the accuracy of the generated rules. However, it produced a smaller rule set. Using logical negation reduced the average number of generated rules. Stubborn ants enhanced the average accuracy of the rules. Using logical negation with stubborn ants in the original version produced the best average accuracy while using multi-pheromone with logical negation produced the least number of rules.

9.5.2.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	7.03	70.1	5.61	71.76	7.2	72.4	6.04	70.42
2	7.00	70.51	5.33	71.68	7.24	73.12	6.01	69.82
3	6.44	70.32	5.42	71.65	7.11	72.13	5.83	70
4	6.84	70.2	5.37	71.81	6.73	72.1	5.84	69.74
5	6.75	70.00	5.14	71.88	6.74	72.03	5.72	69.79
6	5.98	70.06	5.85	71.57	7.08	71.87	5.65	69.71
7	6.32	69.93	5.23	70.98	6.44	71.88	5.81	70
8	6.80	70.22	5.41	71.61	6.92	72.1	5.66	70.21
9	6.55	69.85	5.38	71.65	6.70	70.67	5.73	79.87
10	6.67	69.87	5.21	71.45	6.93	70.6	5.72	69.85
F- Value			122.78	206.59	4.2	5.2	56.77	2.88
Significance Type			**Strong**	**Strong**	**-**	**Normal**	**Strong**	**-**

Table 9.5 - Tic-Tac-To Dataset Detailed Results for ANOVA Test.

9.5.3 Mushrooms Dataset Results

9.5.3.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	6.20 ±0.29	51	91.04 ±0.54	4.28 ±0.18	143	91.79 ±0.9
Logical Negation	4.60 ±0.30	69	90.8 ±0.60	3.68 ±0.14	287	91.02 ±1.07
Stubborn Ants	6.34 ±0.18	48	92.14 ±0.8	4.28 ±0.25	97	**92.88 ±0.62**
Negation & Stubborn	4.97 ±0.34	49	90.25 ±0.9	**3.50 ±0.08**	139	91.70 ±0.41

Table 9.6 - Mushrooms Dataset Experimental Results Summary.

9.5.3.2 *Results Analysis*

The Mushrooms dataset (Table 9.6) has a two-valued class attribute, as in Tic-Tac-Toe. However, multi-pheromone system produced better results in terms of average accuracy. Stubborn ants performed well in enhancing the average accuracy of the generated rules. Using logical negation produced the least number of rules with a low number of trials, but the average accuracy of the rules declined. Multi-pheromone with stubborn ants produced the best average accuracy with an appropriate number of generated rules.

9.5.3.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	6.32	90.51	4.76	91.13	6.48	92.07	4.08	92.10
2	6.64	91.38	4.40	89.73	6.40	92.17	4.20	91.63
3	5.96	90.67	4.20	90.31	6.34	91.26	4.16	91.65
4	6.28	91.03	4.56	91.79	6.56	91.35	4.60	91.98
5	6.48	90.92	4.16	91.28	6.52	92.48	4.44	92.33
6	6.52	91.08	4.92	90.27	6.40	92.08	4.16	92.60
7	5.92	90.98	4.60	91.29	5.96	92.17	4.56	92.98
8	6.20	90.78	5.00	90.55	6.26	92.44	4.08	91.56
9	5.92	90.26	4.96	90.65	6.38	91.37	4.28	89.33
10	5.80	91.19	4.52	90.96	6.14	93.03	4.24	91.76
F- Value			144.33	0.911	1.64	16.68	304.9	4.48
Significance Type			**Strong**	**-**	**-**	**Strong**	**Strong**	**Normal**

Table 9.7 - Mushrooms Dataset Detailed Results for ANOVA Test.

9.5.4 Nursery Dataset Results

9.5.4.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	8.11 ±0.11	115	79.98 ±0.73	6.48 ±0.29	215	81.44 ±1.09
Logical Negation	5.16 ±0.18	110	76.04 ±0.7	**3.5 ±0.11**	253	76.94 ±0.47
Stubborn Ants	8.14 ±0.06	95	80.93 ±0.66	6.72 ±0.30	147	**82.08 ±0.82**
Negation & Stubborn	5.13 ±0.15	90	76.00 ±0.57	4.15 ±0.13	237	77.60 ±0.45

Table 9.8 - Nursery Dataset Experimental Results Summary.

9.5.4.2 *Results Analysis*

Experiments on the Nursery dataset (Table 9.8) have shown similar results to the Mushrooms dataset. Using logical negation reduced the number of generated rules, but came with a negative effect on the accuracy. Stubborn ants improved the average accuracy of the generated rules, especially when used with multi-pheromone system, as this combination produced the best average accuracy.

9.5.4.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	8.14	80.29	5.25	75.98	8.14	81.57	6.70	81.02
2	7.88	78.14	4.82	75.91	8.17	80.32	6.64	83.20
3	8.00	80.13	5.11	77.14	8.17	80.57	6.85	81.49
4	8.02	80.26	5.28	76.06	8.20	80.76	6.14	80.50
5	8.14	80.37	5.22	75.48	8.05	79.98	6.44	81.84
6	8.14	80.08	5.14	77.00	8.11	80.82	6.58	81.64
7	8.14	79.31	5.11	75.80	8.17	81.91	5.85	79.33
8	8.25	80.53	5.28	75.55	8.14	81.54	6.61	81.69
9	8.22	80.51	4.91	74.63	8.25	81.57	6.58	82.73
10	8.17	80.19	5.42	76.54	8.05	80.29	6.41	80.96
F- Value			1938.43	78.2	0.72	8.25	270.47	12.30
Significance Type			**Strong**	**Strong**	**-**	**Normal**	**Strong**	**Strong**

Table 9.9 - Nursery Dataset Detailed Results for ANOVA Test.

9.5.5 Dermatology Dataset Results

9.5.5.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	8.72 ±0.26	94	74.72 ±0.49	10.39 ±0.22	170	82.77 ±0.64
Logical Negation	**7.31 ±0.14**	59	80.84 ±0.43	7.68 ±0.15	120	**86.47 ±0.39**
Stubborn Ants	8.80 ±0.28	79	74.80 ±0.49	10.33 ±0.43	159	83.01 ±0.43
Negation & Stubborn	7.32 ±0.16	53	81.56 ±0.41	7.54 ±0.18	107	86.06 ±0.41

Table 9.10 - Dermatology Dataset Experimental Results Summary.

9.5.5.2 *Results Analysis*

Experiments on Dermatology dataset have shown superiority in results when using logical negation operator. As shown in Table 9.10, using logical negation operator produced less number of rules, compared to the original version. Moreover, the accuracy of the generated rule was enhanced significantly with the help of the logical negation operator in rule construction. Stubborn ants did not improve the quality of the output in terms of classification accuracy. However, they reduced the number of trials. Multi-pheromone system produced better results in terms of classification accuracy. Using multi-pheromone with logical negation operator produced the best classification accuracy, while using logical negation with the original version of Ant-Miner produced the lowest number of rules with good classification accuracy.

9.5.5.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	8.64	75.18	7.24	81.15	8.82	74.27	10.46	82.32
2	9.2	74.29	7.36	80.27	8.82	75.13	10.22	83.62
3	8.68	75.43	7.06	80.72	9.22	74.91	10.06	82.59
4	8.76	74.62	7.3	81.56	8.62	74.9	10.5	83.35
5	8.38	75.24	7.38	81.16	8.88	74.31	10.28	81.40
6	8.56	74.43	7.54	80.24	8.74	75.94	10.42	83.35
7	8.4	74.21	7.56	80.54	8.92	74.54	10.48	82.54
8	8.82	74.67	7.24	80.72	9.18	74.89	10.78	82.91
9	8.7	74.02	7.24	80.78	8.24	74.59	10.1	82.48
10	9.12	75.17	7.32	81.24	8.56	74.43	10.56	83.13
F- Value			207.6	890.15	0.35	0.11	226.5	970.7
Significance Type			**Strong**	**Strong**	**-**	**-**	**Strong**	**Strong**

Table 9.11 - Dermatology Dataset Detailed Results for ANOVA Test.

9.5.6 Soybean Dataset Results

9.5.6.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	11.15 ±0.17	181	48.00 ±0.30	11.30 ±0.19	327	56.13 ±0.46
Logical Negation	**9.23 ±0.19**	251	45.62 ±0.40	10.79 ±0.16	434	54.28 ±0.29
Stubborn Ants	11.08 ±0.16	141	48.08 ±0.5	11.50 ±0.3	317	**56.84 ±0.47**
Negation & Stubborn	9.26 ±0.22	167	46.87 ±0.68	10.9 ±0.2	425	54.27 ±0.23

Table 9.12 - Soybean Dataset Experimental Results Summary

9.5.6.2 *Results Analysis*

As shown in the Table 9.12, utilizing multi-pheromone system improved the accuracy of the generated rules from the Ant-Miner algorithm when experimented on Soybean dataset. Using logical negation operator made its expected effect on the output, which is reducing the number of rules, generated, thus improving the comprehensibility of the output. Stubborn ants did not improve the quality of the output in terms of classification accuracy very much. However, they needed fewer trials to produce the better output rules than the original algorithm in terms of simplicity and accuracy. Using logical negation with the original Ant-Miner version produced the least number of rules. While using multi-pheromone with stubborn ants produced the highest classification accuracy.

9.5.6.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	11.54	47.79	8.86	45.35	11.13	47.51	11.30	55.82
2	11.28	47.89	9.21	45.23	10.84	47.89	11.45	55.87
3	10.97	48.31	9.26	45.77	11.17	48.44	11.25	56.35
4	11.15	48.52	9.30	45.17	10.93	48.33	11.56	56.34
5	11.00	47.78	9.45	46.42	11.26	48.87	11.07	56.20
6	11.04	47.71	9.10	45.17	10.97	47.77	11.54	55.80
7	11.00	47.65	9.17	45.93	11.36	47.49	11.06	56.68
8	11.19	47.96	9.28	45.93	10.93	47.54	11.45	55.18
9	11.17	48.03	9.54	45.51	11.17	48.47	11.04	56.40
10	11.21	48.38	9.08	45.67	11.04	48.54	11.32	56.68
F- Value			565.34	220.9	0.966	0.19	3.33	2146.3
Significance Type			**Strong**	**Strong**	**-**	**-**	**-**	**Strong**

Table 9.13 - Soybean Dataset Detailed Results for ANOVA Test.

9.5.7 Contraceptive Method Choice Dataset Results

9.5.7.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	9.01 ±0.18	94	43.45 ±0.37	4.94 ±0.19	330	45.97 ±0.32
Logical Negation	6.93 ±0.29	60	43.53 ±0.38	**3.01 ±0.12**	490	45.60 ±0.50
Stubborn Ants	9.18 ±0.27	87	44.08 ±0.35	4.96 ±0.17	303	45.79 ±0.43
Negation & Stubborn	6.96 ±0.29	57	45.07 ±0.38	4.25 ±0.13	417	**46.24 ±0.38**

Table 9.14 - Contraceptive Method Choice Dataset Experimental Results Summary.

9.5.7.2 *Results Analysis*

Table 9.14 shows the results of testing the Ant-Miner extensions on the Contraceptive Method Choice dataset. Logical negation operator produced fewer rules compared to the original version. Using logical negation with multi-pheromone produced the least number of rules. Stubborn ants have improved the quality of the output in terms of classification accuracy. Similarly, using multi-pheromone system improved the classification accuracy of the generated rules. The best accuracy was produced form this dataset by using logical negation with stubborn ants in the multi-pheromone system.

9.5.7.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	8.88	43.66	7.12	43.47	9.14	43.89	4.82	45.40
2	9.18	43.68	6.92	43.62	8.74	43.85	5.20	46.10
3	9.24	43.71	6.98	43.17	8.98	44.14	5.02	45.85
4	8.80	42.87	7.06	43.24	9.74	43.80	5.20	45.74
5	9.16	42.93	6.82	43.18	8.98	43.93	4.06	46.48
6	8.70	43.74	6.76	43.82	9.24	44.02	4.98	46.17
7	9.20	43.67	7.00	43.28	9.36	44.37	4.86	46.35
8	9.02	43.70	6.68	43.75	9.14	44.55	4.72	45.98
9	8.92	42.94	7.10	44.40	9.38	43.97	5.04	45.64
10	9.02	43.64	6.90	43.37	9.18	44.33	5.04	46.01
F- Value			2275.9	2.23	289.	19.43	778.1	285.52
Significance Type			**Strong**	**-**	**-**	**Strong**	**Strong**	**Strong**

Table 9.15 - Contraceptive Method Choice Dataset Detailed Results for ANOVA Test

9.5.8 BDS Dataset Results

9.5.8.1 *Results Summary*

	Original			Multi-Pheromone		
	Rules Number	**Trials/ Iteration**	**Accuracy (%)**	**Rules Number**	**Trials/ Iteration**	**Accuracy (%)**
None	11.34 ±0.07	147	69.53 ±0.70	10.98 ±0.10	66	77.13 ±0.52
Logical Negation	4.58 ±0.13	166	64.89 ±0.50	3.38 ±0.08	105	71.45 ±0.26
Stubborn Ants	11.26 ±0.28	107	71.07 ±0.36	11.03 ±0.05	62	**77.85 ±0.36**
Negation & Stubborn	4.62 ±0.15	152	65.77 ±0.86	**3.37 ±0.04**	99	71.47 ±0.97

Table 9.16 - BDS Dataset Experimental Results Summary.

9.5.8.2 *Results Analysis*

The last dataset, BDS, exhibited similar behavior to the previous dataset when experimenting the Ant-Miner extensions on it. Results in Table 9.16 show that using logical negation reduced the number of rules generated by the algorithm. However, the accuracy of the generated rules was reduced as well. Stubborn ants increased the accuracy level of the generated rules without producing lager number of rules compared to the original version. Multi-pheromone on the other hand improved the quality of the generated rule in terms of classification accuracy noticeably, as well as reducing size of the generated rule set, especially when used with logical negation operator. The highest classification accuracy was produced when using multi-pheromone with stubborn ants. While the lowest number of rules was generated when using logical negation operator with multi-pheromone and stubborn ants.

9.5.8.3 *Test of Statistical significance*

	Original		Negation		Stubborn		Multi-pheromone	
Run	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**	**Rules**	**Acc.**
1	11.32	69.17	4.80	64.31	11.36	70.88	10.94	77.68
2	11.38	69.96	4.46	64.09	11.35	71.01	10.86	77.33
3	11.28	69.42	4.56	64.57	11.24	70.76	11.04	77.33
4	11.28	70.22	4.64	65.53	11.95	71.80	11.04	77.17
5	11.4	70.92	4.46	65.33	11.00	71.23	10.98	77.09
6	11.24	69.07	4.80	64.60	11.02	70.70	10.92	76.80
7	11.26	68.76	4.44	65.47	11.14	70.76	11.14	76.15
8	11.36	69.58	4.56	64.96	11.00	71.34	11.04	77.84
9	11.38	69.68	4.54	64.76	11.38	71.38	11.1	77.46
10	11.5	68.57	4.62	65.269	11.25	70.81	10.78	76.50
F- Value			227.9	287.7	0.58	37.45	68.2	751.42
Significance Type			**Strong**	**Strong**	**-**	**Strong**	**Strong**	**Strong**

Table 9.17 - BDS Dataset Detailed Results for ANOVA Test

9.5.9 Ants with Personality Experimental Results

	Rules Number		Trial\Iteration		Accuracy	
Dataset	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 0.5$	$\sigma = 1$
Car Evaluation	9.00 ±0.18	8.85 ±0.010	156	302	76.2 ±0.31	77.03 ±0.42
Tic-Tac-To	6.22 ±0.30	6.50 ±0.21	198	634	71.64 ±0.27	71.89 ±0.31
Mushrooms	6.40 ±0.22	6.20 ±0.17	167	538	92.72 ±0.55	91.44 ±0.38
Nursery	8.11 ±0.18	8.21 ±0.25	230	876	80.32 ±0.73	80.82 ±0.64
Dermatology	8.69 ±0.16	8.64 ±0.11	197	307	75.50 ±0.45	75.57 ±0.52
Soybean	11.12 ±0.18	11.21 ±0.22	361	815	48.60 ±0.28	48.42 ±0.31
CMC	8.99 ±0.10	9.01 ±0.12	142	296	43.26 ±0.42	43.66 ±0.35
BDS	11.30 ±0.14	11.24 ±0.12	301	640	70.02 ±0.77	69.74 ±0.82

Table 9.18 - Ants with Personality Experimental Results

The pervious table shows the results of using Ant with personality – where each ant has its own values for α and β drown from a Gaussian distribution random generator with mean of 1 and standard deviations of 0.5 and 1. Average rules number did not change significantly when using different standard deviation for all datasets. However, average accuracy and trials number has different values for each value of standard deviation. 0.5 produced better average accuracy comparing to the original version of Ant-Miner but with higher trials per iteration. Standard deviation of one produced even better

average accuracy than the results produced by using standard deviation of 0.5. However, number of trials needed to converge per iteration has increased significantly when using standard deviation of 1.

9.6 Summary

This chapter presented the experiments that were used to test the performance of the proposed modifications on the original version of Ant-Miner. The proposed modifications are: Using Negative Attributes, Stubborn Ants, Quality Contrast Intensifier, Multi-pheromone Ant System and Ants with Personality. Each modification was test alone and with other modifications on four public datasets. The results shown the effect of each these modification on the performance of the algorithm in terms of number of rules generated, number of trials needed for each iteration to converge and the accuracy of the generated rules. The next chapter concludes the outcomes of the research and mentions some future work on the field of Ant Colony Optimization in general and Ant-Miner in specific.

Chapter 10

CONCLUSION AND FUTURE WORK

10.1 Conclusion

Ant-Miner is an Ant Colony algorithm for discovering classification rules. It has been introduced in 2002 and proved to produce competitive results to the well-known classification algorithm. Since then, a lot of modification has been applied to the algorithm in order to develop its efficiency. This thesis proposes five new extensions to Ant-Miner. First, we proposed using logical negation in rule antecedents' construction. The aim was to discover rules with higher coverage in order to reduce the overall number of the generated rules, which in turn improves the comprehensibility of the output. Second, we proposed the use stubborn ants with Ant-Miner. Stubborn ants are variation of ACO in which an ant can use its own past experience in rule constructing besides the shared experience in the colony. The aim is to add individuality and promote search diversity. Third, we proposed a new Ant-Miner system; multi-pheromone. In multi-pheromone, the ant selects the class prior to constructing the rule antecedents. Each ant can drop different type of pheromone as many as the permitted class, and it can only follow the pheromone dedicated the class of the current rule being constructed. The aim is to select terms that are more relevant to the classification of the selected class, so that better rules in term of predictive accuracy are discovered. Fourth, we propose a new strategy for pheromone update, aims to intensify the contrast between the quality of the decision components i.e. rewarding good rules by amplifying the pheromone to be dropped and penalizing bad rules by removing pheromone amounts. Finally, we proposed

the use of different values of α and β in term selection formula for each ant. The aim is to give personality to each ant and promote search diversity.

10.2 Results Summary

In summary, experimental results indicate that using logical negation tends to produce a lower number of rules. However, since the number of nodes in the construction graph increases, the number of trials per iteration increases. Using logical negation does not sacrifice the accuracy of the generated rules. On the other hand, using stubborn ants reduces the number of trials needed per iteration to generate a rule and enhances the accuracy of the rules. Multi- pheromone increases rule quality in terms of accuracy. Furthermore, it produces a smaller rule set because of the evaluation function that balances between a rule's classification accuracy and its coverage. As for ants with personality, using $\sigma = 1.0$ produces better results in terms of accuracy than using $\sigma = 0.5$. Nonetheless, the algorithm needs less trials using $\sigma = 0.5$. Note that a standard deviation of 0.5 produces better results in terms of generated rules accuracy compared to the original version of Ant-Miner.

10.3 Future work

Experimental results on four popular datasets indicate that these extensions are promising and worthy of further exploration (see Chapter 9). In the future, we would like to explore using a weight coefficient for stubbornness when using stubborn ants, which may start at a small value and increase gradually over time. When using ants with personality, we would like to explore gradually decreasing over time the value of the standard deviation of the Gaussian distribution function used to generate the individual α's and β's.

REFERENCES

[1] Abdelbar, A. M.: "Stubborn ants". *Proceedings IEEE Swarm Intelligence Symposium*, pp. 1–5 (2008).

[2] Abraham, A., Grosan, C., Ramos V.: "Swarm Intelligence in Data Mining". *Studies in Computational Intelligence*, vol. 34, (2006).

[3] Chan, A., Freitas, A.: "A new ant colony algorithm for multi-label classification with applications in bioinformatics". *Proc. Genetic and Evolutionary Computation Conf.* pp. 27-34 (2006).

[4] Chan, A., Freitas, A.: "A new classification-rule pruning procedure for an ant colony algorithm". *Artificial Evolution, Lecture Notes in Computer Science*, vol. 3871, pp. 25–36 (2005).

[5] Cover, T. M., Thomas, J. A.: *Elements of Information Theory*. New York: John Wiley & Sons (1991).

[6] Deneubourg, J. L., Aron, S., Goss, S., Pasteels, J.M., "The self-organizing exploratory pattern of the Argentine ant". *J. Insect Behav.* , pp. 3-159, (1990).

[7] Deneubourg, J. L., Aron, S., Goss, S., Pasteels, J.M., "Self-organized shortcuts in the Argentine ant". *Naturwissenschaften*, pp. 76- 579, (1989).

[8] Dorigo, M.: "Optimization Learning and Natural Algorithms, Ph.D. thesis (in Italian)". *Dipartimento di Elettronica, Politecnico di Milano*. (1992).

[9] Dorigo, M., Colorni, A., Maniezzo, V.: "The Ant System: Optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics-Part B.*, vol. 26, pp. 29–41 (1996).

[10] Dorigo, M., Di Caro, G.: The ant colony optimization meta-heuristic, *in New Ideas in Optimization*. McGraw-Hill., p.11, (1999).

[11] Dorigo, M., Gambardella, L. M. Ant Colony System: A cooperative learning approach to the travelling salesperson problem. *IEEE Transactional on Evolutionary Computation*, vol. 1, pp. 53 – 66.

[12] Dorigo, M., Maniezzo, V., Colorni, A., "Positive Feedback as a Search Strategy, Technical report". *Dipartimento di Elettronica*, pp. 91-016 (1991).

[13] Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT Press. (2004).

[14] Fayyad, U., G. Piatetsky-Shapiro and P. Smyth,: "The KDD process for extracting useful knowledge from volumes of data". *Communications of the ACM* , vol. 39, 27–34 (1996).

[15] Jaiwei, H., Kamber, M.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, (2006).

[16] Liu, B., Abbass, H. A. , McKay, B.: "Density-based heuristic for rule discovery with ant-miner". *In Proc. 6th Australasia-Japan Joint Workshop on Intell. Evol. Syst.,* pp. 180–184 (2002).

[17] Liu, B., Abbass, H. A., McKay, B.: "Classification rule discovery with ant colony optimization". *In Proc. IEEE/WIC Int. Conf. Intell. Agent Technol*., pp. 83–88 (2003).

[18] Martens, D., Backer, M. D., Haesen, R., Vanthienen, J., Snoeck, M., Baesens, B.: "Classification with ant colony optimization". *IEEE Transactions on Evolutionary Computation*. vol. 11, pp. 651–665 (2007).

[19] Otero, F., Freitas, A., Johnson, C. G.: "cAnt-Miner: an ant colony classification algorithm to cope with continuous attributes". *Ant Colony Optimization and Swarm Intelligence. Lecture Notes in Computer Science*, vol. 5217, pp. 48–59 (2008).

[20] Parpinelli, R.S., Lopes, H.S., Freitas, A.: "Data mining with an ant colony optimization algorithm". *IEEE Transactions on Evolutionary Computation*, vol. 6, pp.321–332 (2002).

[21] Quinlan, J. R.: C4.5: *Programs for Machine Learning*. Morgan Kaufmann. (1993).

[22] Quinlan, J. R.: "Generating production rules from decision trees". *Proc. Int. Joint Conf. on Artif. Intel.*, pp. 304-307 (1987).

[23] Smaldon, J., Freitas, A.: "A new version of the Ant-Miner algorithm discovering unordered rule sets". *Proceedings Genetic and Evolutionary Computation Conference (GECCO)*, pp. 43–50 (2006).

[24] Stützle, T., Hoos, H. H., "MAX-MIN Ant System". *Future Generation Comput. Syst.*, pp. 889,(2000).

[25] Stützle, T., Dorigo, M.: "ACO algorithms for the traveling salesman problem". *Evolutionary Algorithms in Engineering and Computer Science, Wiley*, pp. 163. (1999).

[26] UCI Repository of Machine Learning Databases. Retrieved July 2009 from, URL:http://www.ics.uci.edu/ mlearn/MLRepository.html

[27] Wang, Z., Feng, B.: "Classification Rule Mining with an Improved Ant Colony Algorithm". *Advances in Artificial Intelligence, Lecture Notes in Computer Science*, vol. 3339, pp. 357–367 (2004).