A Hybrid Heuristic-Exhaustive Search Approach for Rule Extraction

By

Daniel Rodić

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Science

in the Faculty of Natural & Agricultural Sciences

AT

University of Pretoria

Pretoria

8. December 2000

Abstract

The topic of this thesis is knowledge discovery and artificial intelligence based knowledge discovery algorithms. The knowledge discovery process and associated problems are discussed, followed by an overview of three classes of artificial intelligence based knowledge discovery algorithms. Typical representatives of each of these classes are presented and discussed in greater detail. Then a new knowledge discovery algorithm, called Hybrid Classifier System (HCS), is presented. The guiding concept behind the new algorithm was simplicity. The new knowledge discovery algorithm is loosely based on schemata theory. It is evaluated against one of the discussed algorithms from each class, namely: CN2, C4.5, BRAINNE and BGP. Results are discussed and compared. A comparison was done using a benchmark of classification problems. These results show that the new knowledge discovery algorithm performs satisfactory, yielding accurate, crisp rule sets. Probably the main strength of the HCS algorithm is its simplicity, so it can be the foundation for many possible future extensions. Some of the possible extensions of the new proposed algorithm are suggested in the final part of this thesis.

Opsomming

Die tema van hierdie tesis is kennisontginning en kennisontginningsalgoritmes gebaseer op kunsmatige intelligensie. Die kennisontginningsproses, en geassosieerde probleme word bespreek, gevolg deur 'n oorsig van drie klasse kunsmatige intelligensie kennisontginningsalgoritmes. Tipiese verteenwoordigers uit elke klas word aangebied en in detail bespreek. 'n Nuwe kennisontginningsalgoritme, genaamd Hybrid Classifier System (HCS), word dan aangebied. Dit word ook geevalueer teen een algoritme uit elk van die bespreekte klasse, naamlik: CN2, C4.5, BRAINNE en BGP. Resultate word bespreek en vergelyk. Vergelykings word gemaak deur gebruik te maak van 'n toetsbed van klassifikasie probleme. Hierdie resultate toon aan dat die nuwe kennisontginningsalgoritme bevredigend presteer, en akkurate, eenvoudige reëlversamelings lewer. Die hoofvoordeel van HCS is die eevound van die algoritme, wat dit as fondamet laat vir verdere uitbreidings. Moontlike uitbreidings van die voorgestede algoritme word voorgestel.

Acknowledgments

Let him who seeks not cease

from seeking until he finds

A friend

This thesis would probably not materialised, if it were not for the following people, to whom I extend my deepest gratitude:

- My supervisor, Dr A.P. Engelbrecht, for his time, advice and PATIENCE.
- My parents, for instilling a culture of learning since my childhood.
- My family, especially grandma for all her prayers.
- My beloved girlfriend, for all her support.
- My friends, especially colleagues and partners at work, for all their support.

I owe you much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since its invention, the computer has proven to be a useful tool. Computers were amongst the first tools that could help mankind with intellectual labour, as opposed to tools that were used, from the dawn of consciousness, to ease mechanical labour. Computers have been very successful in automation of many repetitive, computationally extensive, and mostly sequential tasks. The initial successful applications of computers gave a hope to optimism, expressed in the fifties, that soon computers would be able to mimic the way humans think. From this optimism the term "Artificial Intelligence" has been coined. Fifty years later, it is still a big question if true artificial intelligence is ever going to be a reality, or if it will stay a dream of computer scientists, as "elixir of wisdom" was to alchemists in the Middle Ages. Instead, computers were used to automate more and more complex tasks. Computers work perfectly when handling data in a well-defined procedure. However, many problems require solutions that are not well defined. Instead of following well-defined procedures, more often than not, there are only guidelines on how to achieve solutions to these problems. For most of such problems, previous experience and intuition of human experts are invaluable. Previous experience and intuition can be loosely labelled as knowledge about the problem domain.

Nowadays, a vast amount of data is captured and stored. Within that data lies an abundance of embedded knowledge. However, because of vast quantities of data, the extraction of knowledge can not be done efficiently by human experts. What is required is intelligent tools to extract the knowledge. Various algorithms have been designed to solve these types of problems. Only the artificial intelligence based knowledge discovery algorithms are considered in this thesis. Three main classes of the knowledge discovery algorithms based on artificial intelligence are: Artificial Neural Networks, Evolutionary Computing and Classical Machine Learning Algorithms. Many tools were developed based on algorithms from those classes. To name but a few: GA-Miner[25], C4.5 [54], CN2 [14] and DMT [4]. Each of the tools has its own advantages and disadvantages. None of the tools is superior at all time and in all applications. Some of the tools can not even be applied to certain classes of problems, for example problems with continuous values attributes, or correlated features.

In this thesis, a new knowledge discovery algorithm, called Hybrid Classifier System (HCS), is proposed. It is called hybrid because it employs both heuristic and exhaustive search methods. In each iteration, HCS explores the search space exhaustively and then uses heuristics to prune some of the candidate rules. The new algorithm is not based on any previously established algorithm, although it has its basis in schemata theory [37]. The new algorithm is then evaluated against a representative from each of the three above-mentioned classes of algorithms, namely CN2, C4.5, BGP and BRAINNE. The main characteristic of the algorithm is its simplicity, making it a good candidate for embedded, real-time applications. Despite the simplicity of the algorithm, HCS shows promising results, and its simplicity offers a lot of space for future improvement.

The rest of this thesis is outlined as follows: The process of knowledge discovery is defined in chapter 2. In the same chapter, some of the approaches to automated knowledge extraction are presented, with a detailed description of a typical representative for each of the different approaches. The new classifier system, HCS, is proposed in chapter 3. Chapter 4 presents the results of application of some of the algorithms presented in chapter 4, as well as HCS, to some of the benchmark problem domains. Finally, in chapter 5, a conclusion and suggestions for further research are discussed.

Chapter 2

Different Approaches to Knowledge Discovery

This chapter gives an introduction to knowledge discovery, the steps of a typical data mining process and the methods used during this process. The most frequently used methods of knowledge discovery, namely, connectionist, evolutionary and classical machine learning are then discussed.

2.1 Introduction to Knowledge Discovery

Since the advent of computers, vast amounts of data were compiled and stored. That data from those early days were transformed into information, and human experts then extrapolated knowledge from such information. In other words, computers were used to prepare information for the human experts. With almost an exponential rise in the computing and storage capacity of modern computers that we have been witnessing in the last two decades, the amount of information also increased at an equal rate.

In this abundance of information lies a vast amount of knowledge that needs to be discovered, or made available to non-experts. The extrapolation of knowledge from human experts is a well-known problem [41].

While knowledge extrapolation from human experts is not within the scope of this thesis, a few problems associated with knowledge extrapolation problems will, however, be mentioned next.

Human experts often express their knowledge about a problem domain in terms of exceptions, common sense and a "gut feeling". These concepts are extremely difficult, and sometimes even impossible, to implement in terms of computer instructions. Other avenues for automated knowledge extraction thus had to be explored. Several techniques have been proposed, including statistical analysis [2], data clustering [11,12], visualization [35,12], nearest neighbour approaches [9] and artificial intelligence based techniques. This thesis concentrates on knowledge extraction techniques from the field of artificial intelligence.

The last decade has witnessed an outburst in applications of knowledge discovery, as well as the development of numerous data mining techniques. In order to define knowledge discovery, the thesis first defines what is meant by the term *knowledge*. There are many definitions of knowledge, including:

- a collection of interesting and useful patterns in a database [1],
- a strategic resource for gaining a competitive advantage [44], and
- the body of truth, information, and principles about a system of interest [67].

However, computing devices, such as computers, do not operate directly with knowledge; humans do. Computers operate with binary coded data. But, the binary coded data alone does not mean much to humans.

Information can be represented through the combination of a symbolic representation of some concept with data. Such a symbolic representation transforms the one-dimensional scalar values that represent data into a two-dimensional vector representation of information consisting of value of data and symbolic representation. A combination of information into a rule representation, understandable to humans, represents knowledge discovered from the underlying data.

The format in which knowledge is represented should adhere to the following two conditions:

- The knowledge representation format must be complex enough to describe relationships in data (condition 1)
- The format should, however, be easily understood by humans (condition 2).

Symbolic knowledge representation satisfies the second condition. Some of the symbolic knowledge representations are easier to understand than others. For example, the production rule format is generally easier to understand by humans than relational algebra, but both the above mentioned representations are much easier to understand than non-symbolic knowledge representation schemes, such as numerical weights as used by ANNs. Various knowledge representation schemes are discussed in the next section, with reference to the two stated conditions.

2.1.1 Knowledge Representation Schemes

Various schemes have been developed to represent knowledge. This section reviews production rules, semantic nets, numerical weights (of Artificial Neural Networks (ANNs)), relational algebra and first order logic. The different representation schemes are criticised with reference to complexity and comprehensibility.

- Production Rules

The production rule format is one of the most frequently used knowledge representation schemes. The following expression is an example production rule:

if A_1 and A_2 then C

where A_1 and A_2 are attributes (information or values that describe or characterise the value of the concept C) and C is the concept. The production rule format satisfies both conditions; it can be used easily to represent the relationship between information (A_1 ,A_2) and the concept

(C) and it is easily understood by humans, because humans tend to think in terms of rules and exceptions to rules.

It is noteworthy to mention that most expert systems, especially expert systems that are required to provide an explanation facility, use the production rules knowledge representation format [41].

- Semantic Nets

Semantic nets are graphs where nodes are objects of the real world, and edges connecting nodes represent relationships between real-world objects [47,7]. . In a semantic network, each path, for example, *Node1 to Node2 via EdgeA*, can be read as: *Object Node1* is in relationship A with *Object Node2.* Figure 1 is an example semantic net, describing the object "MyDog":

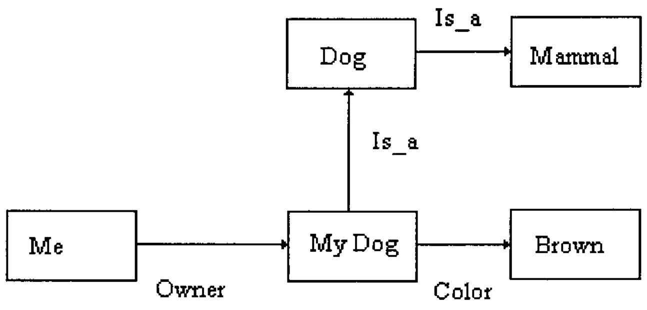


Figure 1: Semantic Nets Knowledge Representation

Semantic Nets do satisfy, to a great extent, the first condition (noted on page 5 before section 2.1.1) in that they provide a model complex enough to describe the relationship between data in an easy to follow graphical representation of knowledge. While semantic nets are easily followed and understood by humans (satisfying the second condition), they do become less comprehensible with increasing size and connectivity of the network. Some of the expert systems use the semantic nets or semantic nets based knowledge representation format, i.e. CENTAUR [67].

- Numerical weights

Numerical weights are the primary knowledge representation format for connectionist models, for example ANNs (see section 2.2). In ANNs, knowledge is represented by the numerical weights learned by the networks. The first condition is satisfied, in that the numerical weights give an accurate approximation of the relationship between information and the concept. The numerical weights are extremely difficult to comprehend. In order to make the knowledge embedded in the weights understandable, the numerical knowledge must be converted into a symbolic form, usually in the form of production rules. In this case output units represent the concept, or action, while the hidden and input units represent the conditions in the production rules. Algorithms that transform numerical weights to production rules are referred to as rule extraction algorithms.

- Relational Algebra

The Greek philosopher Plato described relational algebra in its basic form,

$$\forall o \in U : \mathit{condition1}(o) \rightarrow o \in C$$

The former expression is read as: if an object o from a set U, satisfies condition *condition1*, then object o is a member of class C. Relational algebra satisfies the first condition of symbolic knowledge, because sets can be viewed as relationships between the members of the set and vice-versa. While people with a mathematical background easily understand relational algebra, it is not the case for general knowledge discovery practitioners.

- First order logic

First order logic is an extension of relational algebra, represented in a format more acceptable to computers. The isfather relation below is an example of first order logic knowledge representation:

isfather(John, Jack)

The expression above is read as: John is related to Jack via function isfather. Knowledge representation in first order logic format is similar to that of semantic nets, and, thus satisfies both criteria of good knowledge representation. However, semantic nets are more easily read than long lists of logic relations, in case of complex relationships between data. First order logic is the main knowledge representation scheme for certain non-linear programming languages, such as PROLOG [10].

Each of the above mentioned knowledge representations have their own advantages and disadvantages. For the purpose of this thesis, the production rules representation of knowledge is assumed. In other words, only algorithms that, as the final result, produce production rules are considered in this thesis.

2.1.2 Knowledge Extraction

The process of finding information from data is referred to as knowledge extraction. There are various definitions of knowledge extraction, of which the following is the most comprehensive:

"...a nontrivial process of identifying valid, novel, potentially useful and ultimately understandable knowledge from data" [1].

If the production rules representation format is used, the definition of knowledge discovery can be simplified to:

"Process of converting data to a set of consistent production rules".

Knowledge discovery is a relatively new and very active research field in computer science. It is not surprising that only recently this area is experiencing major growth in industry. There are numerous causes for this growth, of which the two most important ones are listed below:

- The business sector has accumulated an enormous amount of data in their corporate databases, wherein lies a vast potential source of information that can be used to increase business profitability. This kind of knowledge discovery is sometimes referred to as data mining. At the first knowledge discovery conference in Montreal 1995, it was proposed that the term "data mining" referred only to the discovery stage of the knowledge discovery process. For the purpose of this thesis, the terms knowledge discovery and data mining will be interchangeable. In this thesis, the term knowledge discovery also refers to the data-mining phase of the complete knowledge discovery process, which is of pertinence to this thesis.

- Increase in cheap computing power. Knowledge discovery techniques are reasonably heavy on computer resources. Thus, the enormous increase of readily available computing power, that was characteristic of the last decade, has encouraged research in this area. Numerous new techniques appeared.

Rule extraction algorithms have been successfully applied to extract production rules from artificial neural networks (ANNs), for example the KT algorithm [27] and the n-of-m algorithm [65].

Evolutionary computing has also been used successfully to extract knowledge from data. Genetic algorithms have been used to develop one of the first evolutionary knowledge extraction tools, now frequently used as benchmark, namely SCS-1 [29]. Other more recent GA knowledge discovery tools include GABIL [19], GIL [42] and GAMINER [25]. Genetic programming has also been used to evolve decision trees [58].

Classical machine learning research has resulted in the development of several decision tree and rule induction algorithms. In decision tree algorithms (e.g. ID3 [53], AQ [49] and C4.5 [54]), production rules are directly read from the induced trees, whereas rule induction algorithms (e.g. CN2 [14]) directly induce production rules using a beam search. As mentioned above, this branch of artificial intelligence is relatively new, thus there are, understandably, many areas that are not thoroughly explored and researched. In the following section, the whole process of knowledge discovery is presented and explored in greater detail, while sections 2.2, 2.3 and 2.4 present a detailed overview of classes of knowledge discovery algorithms. The process of knowledge discovery is often long and laborious, usually consisting of the following six principal stages [1].

2.1.3 Knowledge Discovery Process

- **Data Selection**

  Data selection concerns the selection of the relevant parameters (or attributes) and exemplars, from a larger data base, to solve a specific problem. Several tools can be used to determine the relevance of attributes, collectively referred to as feature selection tools. (give examples). The question of the number of exemplars remains an open question, with current research attempting to find a solution [22]. The problem is how much data is sufficient to extract valid knowledge. Too much data unnecessarily increase the computational complexity of knowledge discovery.

- **Cleaning of the Data**

  More often than not, data is in an inconsistent format. Alternatively, we can say that data is polluted. Common pollution causes include duplication of data, lack of domain consistency and data capturing errors.

  For example, consider an attribute that represents the age of a person in the number of years. Assume for a particular example that the value for age is not available and entered as zero. The value zero can, however, represent the age of infants. This is an example of a lack of domain consistency. The purpose of the data cleaning stage is to eliminate such pollution as much as possible. Automated tools have been developed to clean data from inconsistencies, for example GritBot [60].

- **Data Enrichment**

  Enrichment is a process in which data is enriched by acquiring additional sources of data, for example census figures, geographical information data, or any other data that may have some relevance to the data being enriched. The actual merging of the data may present a problem due to pollution and different referencing (indexing) systems. Sometimes, a merge is not even possible without loss of data.

- **Data Coding**

  The data coding stage transforms the representation of the data into a form acceptable for the chosen data mining technique.

  Quite often, in order to speed-up the processing of the data, or perhaps because the chosen data mining technique does not support the original format of the data, a different coding needs to be applied to the data. One of the most common techniques is discretisation, which transforms continuous values to discrete values.

- **Data Mining**

  The data mining process is a process of extracting knowledge from the data. Numerous artificial intelligence data mining techniques exist, and have been applied successfully. These techniques are broadly categorized in the following classes: connectionist based algorithms, evolutionary computing based algorithms and classical machine learning based algorithms. These classes of algorithms are discussed in sections 2.2, 2.3 and 2.4 respectively. The data mining phase is the main topic of this thesis.

- **Reporting**

  Reporting can be viewed as a support function with the main purpose serving as the input point for further analysis, either by a human or a non-human expert.

Further on, this chapter presents a classification of rule extraction algorithms and overviews specific algorithms within each class. Rule extraction algorithms are divided into three classes according to the classifying method used by these algorithms:

- **Connectionist Based Algorithms**

The knowledge learned by an ANN is represented by the numerical weights (section 2.1.1) of the ANN. The numerical weights of an ANN are the coefficients of a non-linear function that represents the mapping between input and output space. Since the number of coefficients for real world problems is usually very high, it is almost impossible for humans to understand (and use for explanation) this numerically encoded knowledge. Additional algorithms are required to convert the numerically encoded knowledge into a comprehensible format, for example production rules.

There are basically two classes of rule extraction algorithms for ANNs, namely decompositional and pedagogical. Rule extraction algorithms are divided into these two classes according to the classification scheme proposed by Andrews et al [5]. Decompositional rule extraction algorithms view the ANN as being transparent, in the sense that knowledge about the topology of the ANN, i.e the number of weights and units, is available and necessary to extract rules. Pedagogical algorithms regard the ANN as a black box. Rules are extracted using only the mapping of input units to output units, without knowledge about the topology of the network. There are numerous rule extraction algorithms, for example: M-of-N [65], Rule-As-Learning [15], KT algorithm [27], and RULEX [6]. The M-of-N, KT and RULEX algorithms are examples of the decompositional approach, while Rule-As-Learning is a combination between the two approaches (decompositional and pedagogical).

- **Evolutionary Computing Based Algorithms**

Since the early works of Holland [36], with his proposed Classifier System 1 (CS-1) based on a genetic algorithm, many other evolutionary based, and especially genetic algorithm based, rule extraction algorithms have been developed. Evolutionary computing encapsulates genetic algorithms, genetic programming, evolutionary strategies and evolutionary programming. It is important to note that knowledge is represented uniformly throughout all the paradigms that make evolutionary based computing. For example in genetic algorithms, knowledge is encoded in chromosomes or specimen (section 2.3). In genetic programming, knowledge is encoded in a decision tree. Evolutionary based rule extraction algorithms include SCS-1 [29], GABIL [18, 19], GA-Miner [25], BGP [58].

- **Classical Machine Learning Based Algorithm**

Classical machine learning based rule extraction algorithms represent one of the earliest knowledge discovery paradigms, usually used as benchmark tests against any new knowledge discovery paradigms. Machine learning algorithms include the decision tree algorithms ID3 [53], AQ [49] and C4.5 [54] and the rule induction algorithm CN2 [14]. Other commercially available machine learning algorithms include CART [61], and WizWhy [73].

From each of the three abovementioned approaches, two algorithms will be explained in greater detail. The connectionist approach is discussed covering the KT algorithm [27] and BRAINNE (Building Representation for AI using Neural Networks) [21] in detail. Section 2.3 discusses the evolutionary programming approach, covering the SCS-1 algorithm [29] and BGP [58] in detail. The classical machine learning based approach is discussed in section 2.4. The C4.5 [54] decision tree algorithm and the CN2 [14] rule induction algorithm are discussed in detail.

2.2 Connectionist Based Rule Extraction Algorithms

This section reflects on general features of artificial neural networks. The term 'connectionist' comes from the property of ANNs as being constructed of nodes (units) with multiple weighted connections between them. As mentioned in section (2.1.1), the knowledge encapsulated in trained weights is not an ideal representation of knowledge. On the contrary, it is, more often than not, rather difficult to transform the numeric values of weights into symbolic knowledge. Furthermore, according to our working definition of knowledge, symbolic knowledge is the desired end product of a rule extraction (knowledge discovery) algorithm.

Section 2.2.1 provides an introduction to artificial neural networks. Section 2.2.2 and 2.2.3 present, in detail, the backpropagation neural network (BP ANN). BP ANN is by far the most used ANN, thus making it a good choice for comparison purposes. Section 2.2.4 classifies the different rule extraction algorithms for ANNs. While section 2.2.5 discusses the KT algorithm developed by Fu [27], section 2.2.6 discusses BRAINNE [21]. KT and BRAINNE both rely on the BP ANN and that was one of the selection criterion that resulted in the choice of KT and BRAINNE for this study.

2.2.1 Introduction to Artificial Neural Networks

In the last decade, artificial neural networks (ANNs) have gone through a renaissance. Initial research was done in the 1950's but was, more or less, neglected until the 1980's. The idea behind ANNs came from the research of biological neurons and the interaction between them. There is a loose analogy between biological neurons and nodes in ANNs. Although current ANNs are nowhere near the complexity of the brain, ANNs and the brain exhibit similar features, namely parallelism, capability to learn, high level of redundancy and interconnection, fault tolerance and graceful degradation. Since the revival of ANNs, ANNs have been successfully applied to various problem domains such as classification [48], pattern recognition [17], adaptive control [17], data mining [27,63] and natural language processing [15], to name but a few.

An ANN implements a function $y(\mathbf{z},\mathbf{w},A)$, where y is a function that maps inputs $\mathbf{z}$ to its output, according to parameters $\mathbf{w}$ (weights) and architecture A (i.e. the number of layers and the interconnection between neurons). Note that bold face, symbols in context of ANNs, denote vectors. For example, **z** denotes the vector $(z_1,z_2,\ldots,z_I)$.

One of the key features of ANNs is the ability to self-organise. The term self-organising is used for an ANN's ability to learn by adjusting the connection weights between its nodes (commonly called neurons). By virtue of being self-organising, it is meant that it is not necessary to go through the painful and time-consuming process of extracting knowledge from human experts. Knowledge is obtained through learning from a data set. Pattern recognition is a key role of ANNs. An additional feature is that, once trained, an ANN system is generally a faster classifier than, for example, an expert system. There is, however, one serious shortcoming of ANN systems: the lack of an explanation facility. There is a serious problem in trying to get some explanations from an ANN system, because of the associative nature of ANNs. The concepts and rules are buried in numerical connection weights between various neurons. The inability to get explanations from an ANN system seriously limits applications of ANNs in "safety critical" problem domains. This problem has inspired the creation of various rule extraction algorithms for ANNs.

But, before understanding the way rule extraction algorithms work, the working of ANNs need to be understood. There are many different ANN implementations, varying in topology, the number of layers, methods of learning, triggering functions and so forth. In the following section some of the most common topologies, methods of learning and triggering functions are presented. The backpropagation ANN (BP ANN) is then explained in detail.

2.2.2 Key Features of ANNs

This section introduces key features of ANNs and compares various implementations of these features. Figure 2 illustrates a typical artificial neuron.

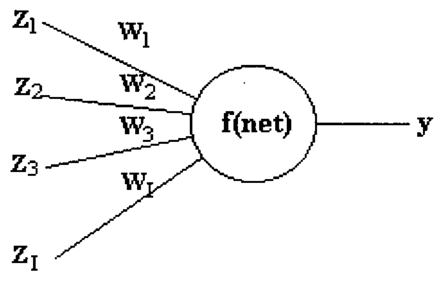


Figure 2: An Artificial Neuron

An artificial neuron behaves as follows:

An artificial neuron receives I input values z_1 .. z_I of which input z_I serves as the bias for that neuron. To each connected input is associated a weight value which strengthens or depletes the input signal. The artificial neuron uses a triggering function to compute an output signal.

At this point it is interesting to draw an analogy between a typical artificial neuron (perceptron) and a biological neuron. Rosenblatt [59], the early researcher of ANNs said the following:

> *Perceptrons are not intended to serve as detailed copies of any actual nervous system. They're simplified networks, designed to permit the study of lawful relationships between the organization of a nerve net, the organization of its environment, and the 'psychological' performances of which it is capable. Perceptrons might actually correspond to parts of more extended networks and biological systems; in this case, the results obtained will be directly applicable. More likely they represent extreme simplifications of the central nervous system, in which some properties are exaggerated and others suppressed. In this case, successive perturbation and refinements of the system may yield a closer approximation.*

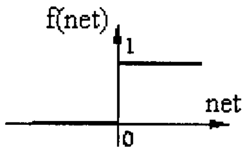
In the remainder of section 2.2.2, the key components and features such as triggering functions, learning methods and learning rules of an ANN are discussed in greater detail. Various types of ANNs are also presented in the remainder of this section.

- **Triggering Functions**

Triggering functions, also referred to as threshold or activation functions, calculate the output signal of a neuron as a function of the weighted input signal:

$$\text{net} = \sum_{i=1}^{I} z_i \, w_i \tag{2.1}$$

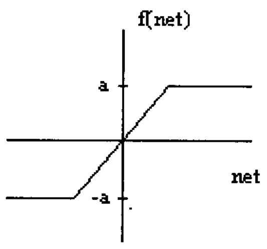
Assuming binary output values, the triggering function compares the net input to some threshold value. If the net input is higher than the threshold, the neuron is activated (in our analogy, the output is binary one). This simplified analogy is implemented using the following equation:



$$\mathbf{f(net)} = \begin{cases} 1 \text{ if net } \geqslant 0 \\ 0 \text{ if net } < 0 \end{cases}$$

Figure 3: Step Function

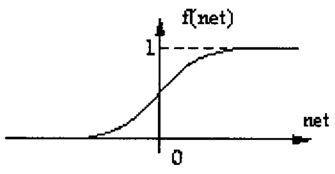
Another commonly used activation function is the linear function, f(net)=net. The linear and step functions can be calculated to form the ramp function:



$$f(\text{net}) = \begin{cases} a & \text{, if net} \geqslant a \\ \text{net} & \text{, if } |\text{net}| < a \\ -a & \text{, if net} \leqslant a \end{cases}$$

Figure 4: Ramp Function

The functions described on the previous page do have some advantages, as well as disadvantages. One advantage is that they are very simple to calculate. Hence their choice is recommended when processing time is scarce, as well as in situations when there are a huge number of neurons that need to be examined for activation. The main disadvantage is that the above-mentioned functions are linear and not differentiable. Most of the learning algorithms (described later in this section) require activation functions to be differentiable. Two widely used non-linear trigger functions are the sigmoid function (illustrated in figure 5) and the hyperbolic tangent function.



$$f(net) = \frac{1}{1 + e^{-net}} \tag{2.2}$$

Figure 5: Sigmoid Function

The disadvantage of non-linear functions is that they are computationally more expensive than former linear functions.

- **Learning Methods**

It has been noted fairly early in ANN research that the single neuron (also called the perceptron) has serious limitations, especially when trying to solve non-linear separable problems (such as the mapping of the XOR function) [50]. Combining multiple neurons into ANNs solved this limitation. The early work of Hebb [34] provided the theoretical background that was used for learning ANNs. Hebb's rule is discussed in more detail later in this section. Learning (or training) uses input data in order to adjust weights between layers. There are three main learning paradigms:

  - **Supervised Learning**, where the objective is to minimize the error between the actual output of the ANN and the given desired output. Probably the most common approach and very acceptable if we have historical data that we can use in preparation of the training set. In other words, there are training examples consisting of input and desired output vectors. Supervised learning is also called learning by example. The BPANN discussed in section 2.2.3 is an example of supervised learning algorithms.

  - **Unsupervised Learning**, also known as clustering, is based on the idea to leave it to the ANN to discover patterns or features within the given data. In unsupervised learning no desired output is provided to approximate.

- **Reinforcement Learning**, where weights are adjusted by punishing bad actions (as evaluated against right choices) and rewarding good actions.

Only the supervised learning paradigm is considered in this thesis.

- **Learning Rules**

Learning rules are used to adjust the weights of an ANN in order to minimize the error made by that ANN. That is, to minimise the distance between the target function that is approximated and the approximation by the ANN of that function. In other words, the function learned by the ANN and the actual function as encapsulated in the training data. From the finite training set, the ANN can only approximate the true function. The most frequently used metric function used to measure the above-mentioned distance or (error) is the square sum error (SSE), defined as:

$$SSE = \sum_{p=1}^{P} \sum_{k=1}^{K} (t_k^{(p)} - f_k^{(p)})^2 \qquad (2.3)$$

where $t_k^{(p)}$ represents the k-th target output for training pattern p, and $f_k^{(p)}$ represents the actual calculated k-th output of the network for the same pattern. The goal of the learning rule is to minimise this error. However, if the ANN is overtrained, the network cannot generalise well. In other words, the ANN is too specifically trained for recognition of the training set, and when presented with input that is not contained within the training set, the network cannot correctly classify this new input. For the purpose of this study, we will concentrate on classification problems, although ANNs are widely used for approximation of functions as well. This problem is also known as overfitting of the training set. The minimization of the error is achieved through weight adjustments by applying suitable optimisation algorithms. The most popular optimisation algorithm used for ANN training is gradient descent (explained in section 2.2.3). Other optimisation techniques commonly used include scaled conjugate gradient [51] and second-order methods [8]. Feedforward ANNs trained using gradient descent optimisation are referred to as backpropagation ANNs. Other well-known training rules, with their weight adjustments demonstrated on a case of a perceptron, are:

- **Generalised Delta Learning Rule**

  This learning rule assumes a differentiable (continuous in most of the cases) activation function. Let that activation function be the unipolar sigmoid function of equation (2.2) then

$\partial f / \partial net = f(1 - f)$ and $\partial \varepsilon / \partial w_i = -2(d - f)f(1-f)x_i$

where ε is the squared error, d is target (derived output), w_i is weight of the input x_i and x_i is the value of the i-th input.

- **Error Correction Learning Rule**

  This rule assumes a discontinuous binary valued activation function (e.g. a step function). Weights are then adjusted only when (d-f) = 1 or (d-f) =-1, using $w_i(t) = w_i(t-1) + 2\eta(d - f)\, x_i$.

  where the notation is as in the example above and η is the learning rate.

- **Widrow-Hoff Learning Rule**

  For this rule, as proposed by Widrow and Hoff [71], the activation function is a linear function, and, furthermore, f = net. Then, $\partial f / \partial net = 1$ and $\partial \varepsilon / \partial w_i = -2(d - f)x_i$

  weights are updated using $w_i(t) = w_i(t-1) + 2\eta(d - f)\, x_i$

  where ε is the squared error, d is target (derived output), w_i is weight of the input x_i and x_i is the value of the i-th input η is the learning rate..

2.2.2.1 Types of Supervised ANNs

There are various types of ANNs in existence, differing in the way neurons are interconnected, and the way that the net input signal is computed.

In feedforward ANNs input signals are fed from the input layer through the hidden layer to the output layer, with no feedback connections. Recurrent ANNs have feedback connections from hidden and/or output layers to serve as additional inputs [28]. This enables ANNs to learn temporal characteristics of data. Feedforward and recurrent ANNs are by no means the only possible types of ANN. Functional Link ANNs are of the feedforward type, where the input layer is expanded to include a number of "functional links", or higher-order combinations of inputs [40]. Another alternative to the feedforward network is product unit ANNs where the net input to a neuron is calculated as a weighted product of input signals, instead of a weighted sum [20].

ANNs also differ according to architecture. ANNs can have one or more layers. The simplest one layer ANN consists of only one neuron. This ANN is referred to as the perceptron. The perceptron was the first proposed ANN, and was subject to much analysis [50, 62]. Shortcomings of a perceptron have become apparent fairly early in research of ANNs [50] and the next step was a the construction of networks consisting of more than one node. A typical multilayered ANN consists of one input layer, one output layer and a number of hidden layers. There is no theoretical limit on the number of layers, but it has been proven that ANNs which have only one hidden layer can learn any continuous mapping if there are enough units in its hidden layer [16]. Choosing the right architecture poses a formidable problem when designing the ANN. If the number of neurons and connections are not sufficient, it may happen that the ANN is not capable of learning the desired mapping. However, if there are too many neurons and connections, the ANN may overfit the training data and waste time and memory resources. There is no general solution to the architecture selection problem, although various heuristics have been proposed. Some of the recent proposed solutions include:

- **Regularisation**

  In the regularisation approach, a penalty term is added to the objective (error) function to penalize the complexity of the ANN architecture [72]. The modified objective function is then given as:

  $$E = SSE + \lambda C$$

  where SSE is the standard sum squared error, λ regulates the influence of the penalty term and C is the penalty (regularisation) term. Examples of regularisation are weight decay and softweight sharing [52].

- **Modularity**

  In research done relatively early, in ANN research terms, it has been noted that an increased number of ANN inputs leads to an increased complexity and a deterioration in performance [45]. Various solutions were proposed, and many of which involved modularity. The two main techniques used for the modularity approach are decomposition and replication. Decomposition is a way of dealing with complex problems by means of breaking it into many less complex problems (sometimes called sub-problems). Replication is a technique used for knowledge reuse. Once the module is successfully applied and tested, it can be safely replicated. For each of these techniques, there is an equivalent in biological organisms. For example, replication can be seen by the fact that humans have two eyes, and both of which are functionally identical. Decomposition is often the way that many tasks are performed in nature, and some authors [62] see it as a sign of intelligent behaviour. One of the simplest proposed architectures involves using ANNs as building blocks instead of neurons. The output of these building block ANNs is then presented to the last layer in this architecture. The last layer is also an ANN. Figure 6 illustrates a modular ANN.

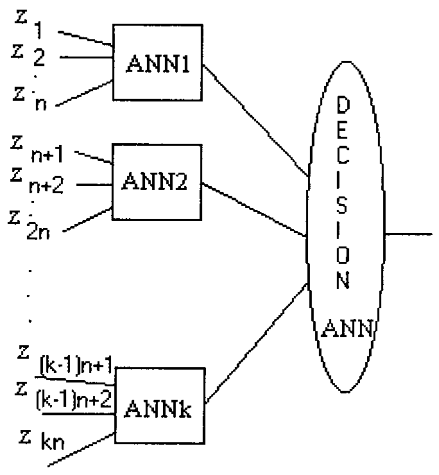


Figure 6: Modular ANN

The figure above illustrates the case when a network is presented with a total of kn input values. The problem is solved by presenting each of k sub-ANNs (ANN1 – ANNk) with n inputs. The outputs of these sub-ANNs are then presented (all k of them) to a decision ANN.

- **Pruning**

  The objective of pruning is to reduce the number of input and/or hidden unit weights (and, in most cases, consequently, the number of connections). Pruning starts with an oversized architecture, and prune irrelevant parameters. For this purpose a measure of parameter relevance is used. One measure of relevance is the significance of network parameters (which can be input units, weights or hidden units) to the output. There are various approaches to measuring significance such as:

  - Sensitivity analysis with regard to the objective function.

    In this approach, saliency is calculated for each parameter, resulting in a measure of sensitivity of the parameter change to the objective function, which is a measure of the ANN's approximation accuracy. Parameters with low saliency are then removed. Some of the better known techniques are "optimal brain surgeon" [31,32] and "optimal brain damage" [30,46].

- Sensitivity analysis with regard to ANN output function.

  Sensitivity analysis of the output function concerns a study of the sensitivity of output units to changes in input units, hidden units or weights. Two sensitivity relevance measures have been developed: (1) parameter significance, which is simply a norm computed from the parameter sensitivities over the training set [74,23], and (2) variance nullity, which is based on the premise that parameters with an approximately zero variance in sensitivity can be pruned.

2.2.3 Backpropagation ANN

The backpropagation ANN (BP ANN) is of the feedforward type where weights are adjusted using the gradient descent optimisation algorithm. Therefore, weights are adjusted by minimising the error between the actual network output and the expected output. The minimisation proceeds by moving towards the negative gradient of the error function. The BP ANN thus operates in two distinctive phases. In the first phase, referred to as the feedforward phase, the output of the network is computed for each of the presented patterns. This is done by presenting input patterns to the input layer and calculating the activation of each neuron in the succeeding layers, using the current weight values. The second phase, referred to as the error back propagation phase, adjust weight vectors as a function of the error made by the network. First the error in prediction, i.e. the difference between actual output and target output, is computed. This error is used to adjust the weights between the last hidden layer and the output layer. The error is then propagated back to previous layers and an accumulated error over all internal nodes is calculated. This accumulated error is then used to adjust the weights in preceding layers. The resulting learning rule is referred to as delta learning (section 2.2.2).

The backpropagation algorithm consists of four main parts: (1) initialisation, (2) repetitive cycles of feedforward pass and (3) backpropagation of an error, and (4) testing of stopping criterion. A brief algorithmic description of the backpropagation learning method is presented first and then all necessary calculations that are required.

BP ANN:

1. The connection weights are initialised to small random numbers.
2. Feedforward pass: input signals are fed from the input layer through the hidden layer to the output layer. The sigmoid function is used as triggering function.
3. Backpropagation of the error:

    a) The weights between the output and hidden layer are adjusted according to:

    $w_{kj}(t) = w_{kj}(t-1) + \Delta w_{kj}(t)$

where $w_{kj}(t)$ is the weight between output unit o_k and hidden unit y_j at time t (or the t-th iteration) and Δw_{kj} is the weight adjustment.

The weight adjustment is calculated as:

$$\Delta w_{kj} = -\eta \delta_{ok} y_j$$

where η is a trial independent learning rate ($0 < \eta < 1$) and δ_{ok} is the error gradient at k-th output unit o_k.

b) The weights between hidden and input layer are adjusted according to:

$$v_{ji}(t) = v_{ji}(t - 1) + \Delta v_{ji}(t)$$

where $v_{ji}(t)$ is the weight between hidden y_j and input unit z_i at time t and Δv_{ji} is the weight adjustment.

The weight adjustment is calculated as:

$$\Delta v_{ji}(t) = -\eta \delta_{yj} z_i$$

where and δ_{yj} is the accumulated backpropagated error.

4. Steps 2 and 3 are iterated until any one of the following stopping criteria is satisfied:
   - The error is acceptable (the mean squared (or sum squared) error)
   - The maximum number of epochs has been exceeded
   - The network starts to overfit the training data

- No decrease in error over consecutive epochs occurred

To justify the weight adjustments as presented in the algorithm outline above, assume that the sigmoid activation function is used:

$$f(net) = (1 + e^{-net})^{-1}$$

Then the activation function for k-th unit of output layer, o_k, is

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-net_{o_k}}} \tag{2.5}$$

and for the j-th unit of the hidden layer, y_j,

$$y_j = f_{y_j}(net_{y_j}) = \frac{1}{1 + e^{-net_{y_j}}} \tag{2.6}$$

For the purpose of notational convenience, pattern superscript is omitted in all formulae where applicable.

The error for output layer is computed as the sum of squares

$$E = \frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2 \tag{2.7}$$

where t_k is the desired output and o_k is the actual output of the ANN.

The output error backpropagated from the output layer is

$$\delta_{o_k} = \frac{\partial E}{\partial\, net_{o_k}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial\, net_{o_k}} \tag{2.8}$$

Where, from equation (2.7)

$$\frac{\partial E}{\partial o_k} = \frac{\partial}{\partial o_k}\left(\frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2\right) = -(t_k - o_k) \tag{2.9}$$

and from equation (2.5)

$$\frac{\partial o_k}{\partial \, net_{o_k}} = \frac{\partial f_{o_k}}{\partial \, net_{o_k}} = (1 - o_k) o_k \tag{2.10}$$

Then from equations (2.8), (2.9) and (2.10)

$$\delta_{o_k} = -(t_k - o_k)(1 - o_k) o_k \tag{2.11}$$

Now, for the hidden layer, the error that needs to be backpropagated is defined as

$$\delta_{y_j} = \frac{\partial E}{\partial \, net_{y_j}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \, net_{y_j}} \tag{2.12}$$

Where from equation 2.6 it follows that

$$\frac{\partial y_j}{\partial \, net_{y_j}} = (1 - y_j) y_j \tag{2.13}$$

and

$$\frac{\partial \, net_{o_k}}{\partial y_j} = \frac{\partial}{\partial y_j} \Big(\sum_{j=1}^{J} w_{kj} y_j \Big) = w_{kj} \tag{2.14}$$

From equations (2.7) and (2.14),

$$\begin{aligned} \frac{\partial E}{\partial y_j} &= \frac{\partial}{\partial y_j} \Big(\frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2 \Big) = \\ &= \sum_{k=1}^{K} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial \, net_o} \frac{\partial \, net_{o_k}}{\partial y_j} = \\ &= \sum_{k=1}^{K} \frac{\partial E}{\partial \, net_{o_k}} \frac{\partial \, net_{o_k}}{\partial y_j} = \\ &= \sum_{k=1}^{K} \delta_{o_k} w_{kj} \end{aligned} \tag{2.15}$$

Finally from equations (2.11) ,(2.12) and (2.15) follows that

$$\delta_{y_j} = (1 - y_j)\, y_j \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{2.16}$$

These calculations are done for each pattern through a predetermined number of epochs. An epoch includes presenting all the patterns in the training set, calculating activations, and modifying weights for each and every pattern in the training set.

2.2.4 Rule Extraction Algorithms

ANN rule extraction algorithms extract the knowledge buried in the numerical weights of the network and convert that knowledge into a more acceptable format – typically into a production rules format (refer to section 2.1.1). The rule extraction algorithms considered in this thesis are appropriate only for classification problems. It is important to note that all the rule extraction algorithms that use ANNs as starting point, assume that the ANN is already trained. Rule extraction algorithms try to solve the incomprehensibility problem of ANNs (i.e. that knowledge represented by an ANN is not symbolic and not easily understood by humans). Recent research in different rule extraction algorithms from ANNs has opened a whole new world of possibilities, architectures and applications [6,27]. For example, by extracting rules from an ANN, it is now possible to justify and explain the responses from that ANN. Rule extraction from trained ANNs comes at a cost, in terms of the resources (i.e. computer time and memory), and the additional effort of implementing the rule extraction algorithms. The question arises, then, if the rules extracted from an ANN, and presumably used in some other classifier mechanism (such as expert systems), offer any real advantage over the ANN itself? The answer is yes; the rule extracting algorithms do present definite advantages. Firstly, by using a rule extraction algorithm, an explanation capability is added to the ANN, although not as powerful as in ESs. For example, in a multilayered ANN concepts can be related to the input layer and output layer neurons as symbolic knowledge. Defining the relationship between these concepts in production rule format, transform the embedded knowledge into symbolic knowledge, and that is a highly desirable trait. Another advantage is that by extracting rules from a once trained ANN, better understanding of an ANN's behaviour is achieved. Probably the most important advantage is that, by using ANN rule extraction, a knowledge base for use with a symbolic system (typically an ES) is obtained, or the knowledge within an existing knowledge base is refined. It has become possible to build a hybrid system that will have "the best of both worlds", i.e. the explanation facility of ESs and the robustness of ANNs.

There are two main approaches to rule extraction from ANNs, depending on whether prior knowledge about the domain is available or not. If prior knowledge is available, it is possible to design an ANN in such a manner that the architecture reflects that knowledge. Such networks are referred to as knowledge based ANN (KBANN) [27]. The ANN is then biased towards prior knowledge. The ANN is then supplemented with additional hidden units (and consequently additional connections), creating a fully connected network. The goal of the additional connections is to extract additional characteristics (new rules, or a refinement of the existing rules) that were not reflected in the prior knowledge used for constructing the ANN. Once supplemented, the ANN is trained using available (preferably new) data, and, when training is complete, a rule extraction algorithm is applied to the trained ANN. Refined rules, or even new rules, can then be extracted from the trained ANN. It is possible to use these extracted rules to design a new ANN, and thereby repeating the process (closing the loop). Effectively, with this process the approximation of the desired function is improved, thus improving the performance of the ANN. It is also important to mention that this approach can be used for dynamic tuning of the ANN in situations where the target function changes with time.

Sadly, more often than not, prior knowledge is not available for most real-world applications. In this case, the process starts with an ANN designed using some of the heuristics and architectures as mentioned in section 2.2.2. The ANN is trained using the available data. Once trained, a rule extraction algorithm is applied to the trained ANN. Knowledge extracted (in the form of rules) can now be presented to some other classifier mechanism (e.g. an expert system). This approach is appropriate when the knowledge gained about the target function is static. In other words the target function should not be time dependent.

This section presents a classification of ANN rule extraction algorithms based on the level of abstraction of the ANN [5].

- **Taxonomy of Rule Extraction Algorithms from ANN**

  The first group of rule extraction algorithms is the decompositional rule extraction algorithms that view the ANN transparently, in the sense that the values of weights are available to the algorithm. The second group of algorithms view the ANN as a black box. This group of algorithms extract rules using only the behaviour of the ANN. Algorithms in this group are called pedagogical algorithms. The third group of rule extraction algorithms is the set of eclectic algorithms that are combinations of the first two approaches.

  - **Decompositional Rule Extraction Algorithms**

    The main characteristic of the decompositional rule extraction approach is the focus on extracting rules at the level of individual (hidden and output) units within the trained ANN. A basic requirement for the algorithms in this category is that the computed output from each hidden and output unit must be mapped into Boolean values. Some of the algorithms in this group are the

KT algorithm by Fu [27], RULEX [6] and the Subset algorithm by Towell and Craven [65].

- **Pedagogical Rule Extraction Algorithms**

While the decompositional approach assumes transparency of the ANN, the pedagogical approach views the ANN as a "black box". Techniques of the "Pedagogical" approach aim to extract rules that map inputs directly into outputs. Such techniques are usually used together with a symbolic learning algorithm. The VIA algorithm, developed by Thrun [64], is an example of the pedagogical algorithm.

- **Eclectic Rule Extraction Algorithms**

Eclectic rule extraction algorithms are combinations of "pedagogical" and "decompositional" approaches. "Rule extracting as learning", by Craven and Shavlik [6], is an example of the eclectic rule extraction class.

Two ANN rule extraction algorithms, both of them decompositional, are discussed in this thesis. The KT algorithm is discussed in section 2.2.5, while BRAINNE is discussed in section 2.2.6. BRAINNE is also used for experiments in chapter 4, where different classes of algorithms are compared.

2.2.5 KT Algorithm

This section describes the KT algorithm developed by LiMin Fu [27]. Section 2.2.5.1 lists the definitions necessary for better understanding of the algorithm. Section 2.2.5.2 classifies this algorithm using the classification scheme proposed by Andrews et al [5]. Sections 2.2.5.3 gives a short algorithmic description and example of the working of the KT algorithm. Section 2.2.5.4 discusses the heuristics used by the KT algorithm.

2.2.5.1 Definitions for the KT Algorithm

In order to understand the KT algorithm, the following definitions are needed:

- *Attributes* - Attributes can be either input or hidden units.

  Attributes are denoted as A_i , i = 1..I (in the case of input units) or A_j , j = 1..J (in case of hidden units.
- *Concepts* - Concepts can be either hidden or output units.

  Concepts are denoted as $C_{j,}$ j = 1..J (in the case of hidden units) or $C_{k,}$ k = 1..K (in the case of output units).
- *Pos-att* - A Pos-att for the concept C is an attribute designating a node which directly connects to the node C with a positive connection weight.

- *Neg-att* - A Neg-att for the concept C is an attribute designating a node which directly connects to the node C with a negative connection weight.
- *Non-conflicting attributes* - Two attributes are non-conflicting if they are not negatively correlated.
- *Rule* - A rule generated from an ANN has the form if

  A_1^+, A_2^+,...,NOT A_1^-, NOT A_2^-,..., then C_k (or NOT C_k)

  where A_i^+ is an attribute in the positive form, A_i^- is an attribute in the negative form, and C_k is the concept. The rule is called a confirming rule if the concept C is not negated, and disconfirming if the concept C is negated.

2.2.5.2 Algorithm Classification

The algorithm works without prior knowledge about the problem domain. The KT algorithm is a "decompositional" rule extraction algorithm, which assumes transparency of the ANN. This means that the algorithm has access to all the attributes of all the neurons and the links between them. The KT algorithm does not, in its original form, accept continuous input values. This algorithm has been developed for solving problems with binary values as inputs.

2.2.5.3 KT Algorithm Overview

A short algorithmic description of the KT Algorithm is given on the next page:

1. For each hidden and output unit, search for a set S_p of pos-atts whose summed weights exceed the threshold of that unit. In other words, which combination of pos-atts that will trigger the unit. The threshold is the bias value for the unit. The algorithm at this point tries to find a confirming rule, represented as the activated unit that represent that concept.

2. For each element p of the set S_p:

   a) Search for a set S_n of neg-atts n, such that the summed weight of element p and negated weight of the member of set S_n (element n) and summed weights of the remaining neg-atts not included in set S_n, exceed the threshold on the unit.

   b) With each element n of the set S_n, form a rule "if p and NOT n then the concept designated by the unit". In other words, if all remaining neg-atts (attributes with negative weights) are activated, the sum of pos-atts and negated weights of neg-atts will still be greater than threshold – activation level.

3. For each hidden and output unit, search for a set S_{nn} of neg-atts, whose summed weights do not exceed the threshold of the unit; in other words, which combination of neg-atts will prevent triggering of the unit. The algorithm at this point tries to find disconfirming rule.

4. For each element n of the S_{nn} set:

   a) Search for a set S_{np} of pos-atts p, such that the summed weight of p and weight of the member of set S_{nn} (element n) and summed weights of remaining pos-atts not included in set S_{np}, do not exceed the threshold of the unit

   b) With each element p of the set S_{np} set, form a rule "if n and NOT p, then NOT the concept designated by the unit". In other words, the combination of neg-atts that prevent triggering of the unit form the antecedent of the rule. The algorithm at this point tries to find a disconfirming rule.

To illustrate the KT algorithm, assume pos-atts A_1 (weight 3.7), A_2 (weight 3.5) A_3 (weight 0.5), A_4 (weight 1.5), concept C_1 (actually a hidden layer unit) and a bias unit of the concept (or threshold) with weight 3.6.

The algorithm starts with an empty set of potential rules (which can be seen as the root of the tree) and then creates a set of new rules by including only one attribute in each potential rule. Each combination of pos-atts represents a node in the search tree. A potential rule, which involves A_1, becomes a proper rule, because the weight of the attribute A_1 is greater than the threshold of C_1. That is the rule: if A_1 then C_1. Then, the nodes in the set are expanded, by combining all unused attributes (an unused attribute is an attribute which is not used as the premise of any rule confirming, or disconfirming a concept) with the nodes in the tree. The level of the search tree actually represents the number of pos-atts in the node at that level. Consider expansion $\{A_2, A_3\}$ (node at level two): this is still not accepted as a rule, because $3.5 + 0.5 < 3.7$ (the sum of the weights of the attributes in the expansion is still less than the threshold). However, the expansion $\{A_2, A_4\}$ is accepted as a rule, because $3.5 + 1.5 > 3.7$ (the sum of the weights of the attributes in the expansion is greater than the threshold). This is demonstrated on figure on the next page:

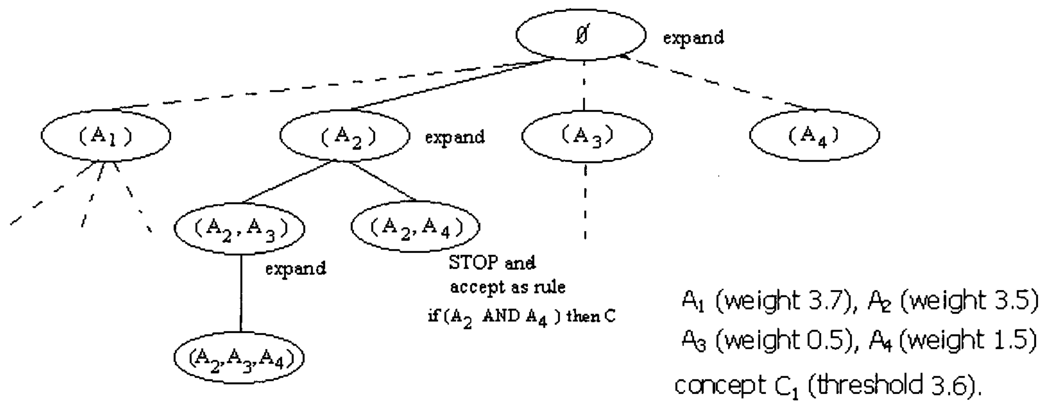

Ø
expand
(A1)
(A2)
expand
(A3)
(A4)
(A2,A3)
(A2,A4)
expand
STOP and accept as rule if (A2 AND A4) then C
(A2,A3,A4)
A1 (weight 3.7), A2 (weight 3.5)
A3 (weight 0.5), A4 (weight 1.5)
concept C1 (threshold 3.6).

Figure 7: Illustration of the KT algorithm

2.2.5.4 Heuristics

Without any heuristics, the search employed by the KT algorithm is exhaustive, in the sense that all possible combinations of attributes are checked. An exhaustive search is, however, prohibitively computationally expensive. In order to minimise the search space, the KT algorithm implements the following heuristics:

- H1: Given a combination G of g pos-atts, which form a node in the search tree, if the summed weights of these attributes and any other k - g strongest, non-conflicting pos-atts are not greater than the threshold on the concept node, then prune combination G (k is the maximum number of attributes forming a rule).

  To illustrate this heuristic, assume pos-atts A_1 (weight 1.1), A_2 (weight 2.0) and A_3 (weight 0.5), A_4 (weight 1.5), concept C_1 (hidden or output layer unit) and the bias unit of the concept with weight 3.7. Assume that k is 3. Combination {A_1, A_3} will be pruned, because the sum of the weights of the attributes in that combination is 1.6. The strongest pos-att (assume it is non-conflicting) is A_2 with the weight 2.0. The sum of the weights of A_1, A_3, A_2 is 3.6, which is still less than the threshold of 3.7. Therefore, combination {A_1, A_3} is pruned.

- H2: Given a combination of pos-atts, if the summed weights of these attributes and all non-conflicting neg-atts, are greater than the threshold on the concept node, then keep the combination, but stop generating its successors. This is actually not proper pruning: when the process of generating the successors of this combination is stopped, it means that the given combination represents a valid rule, and it is sensless to further expand that combination.

To illustrate this heuristic, assume pos-atts A_1 (weight 1.1) and A_2 (weight 3.0), neg-atts A_3 (weight -0.5) and A_4 (weight -1.5), concept C_1 and the bias unit of the concept with weight 2.0. For combination $\{A_1, A_2\}$, the algorithm will stop creating successors because the sum of the weights of the attributes A_1 and A_2 is 4.1, and the sum of the weights of the neg-atts is -2.0. In other words, if attributes A_1 and A_2 are triggered, even if the neg-atts are triggered, the sum is still greater than the threshold, so the concept is true ($1.1 + 3.0 - 0.5 - 1.5 > 2.0$). The combination then becomes a rule.

- H3: Given a combination of pos-atts and negated neg-atts, if the summed weights of the pos-atts and non-conflicting neg-atts not in the combination, are greater than the threshold on the concept node, then the combination is kept, but if it does not, generate its successors. Again, this is not actually pruning in the typical meaning of the word. When the process of generating the successors of this combination is stopped, it means that the given combination represents a valid rule.

To illustrate this heuristic, assume pos-att A_1 (weight 1.1) and neg-atts A_2 (weight -3.0), A_3 (weight -0.5) and A_4 (weight -1.5), concept C_1 and the bias unit of the concept with weight 2.0. For combination $\{A_1, \sim A_2\}$ (read $\sim A_2$ as not A_2, i.e. a negated neg-att) the algorithm stops to create successors, and the combination becomes a rule, because the sum of the weights of the attributes A_1 and A_2 is 4.1, and the sum of the weights of the neg-atts is -2.0. In other words, if attributes A_1 and A_2 are triggered, even if the neg-atts are triggered, the sum is still greater than the threshold. The concept is therefore true $(1.1 + (-(-3.0)) - 0.5 - 1.5 > 2.0)$.

Heuristics H1 and H2 are applied to steps 1 and 3 of the algorithm, while H3 is applied to steps 2 and 4.

As mentioned in the introduction, KT can handle only binary valued attributes. One way of solving this problem would be to use a discretisation technique to divide the range of continuous values into finite number of intervals. Membership of a particular interval would then be encoded as a binary value. This approach would increase computational cost of the whole algorithm. Furthermore, there is an additional problem of determining interval boundaries which would have to be addressed.

2.2.6 BRAINNE Algorithm

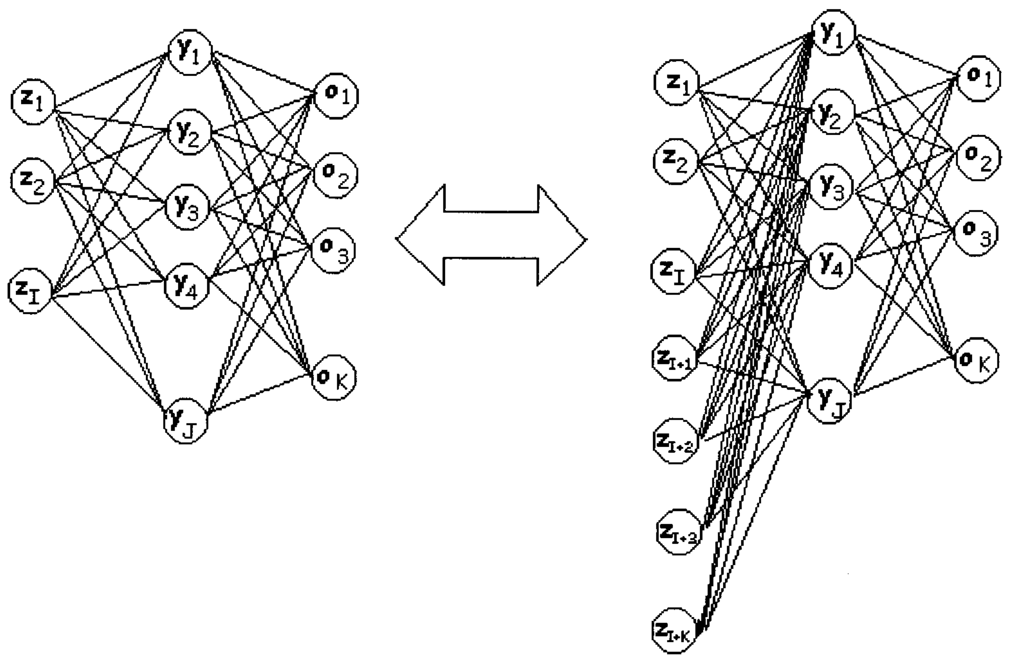
This section describes the Building Representation for AI using Neural Networks (BRAINNE) algorithm [21]. Section 2.2.6.1 gives an overview of the algorithm and classifies this algorithm using the classification scheme proposed by Andrews et al [5]. Sections 2.2.6.2, 2.2.6.3 and 2.2.6.4 respectively examine the way in which conjunctive rules are extracted, the way disjunctive rules are extracted and how continuous values are handled. It is important to keep in mind that sections 2.2.6.2 and 2.2.6.3 consider only binary inputs. The extension of the algorithm to discrete and continuous values is presented in section 2.2.6.4.

2.2.6.1 Overview of the BRAINNE Algorithm

The BRAINNE algorithm extracts rules using both single layered and multi layered ANNs at the same time. In BRAINNE, a standard BP ANN is modified by adding extra input nodes, as depicted by figure 8. The modified ANN architecture is organised such that the input layer consists of I+K units, with $z_{i+1},\ldots,z_{I+k}$ representing the k output units. The first part of the modified ANN consists of a standard BP ANN (multi-layered ANN with gradient descent learning rule), while the second part of modified ANN, consist of extended input units $z_{i+1},\ldots,z_{I+k}$, hidden layer units $y_1,\ldots,y_J$ and the links between. The second part can be viewed as a single layer network, trained using Hebb's learning rule. This representation was chosen to establish a direct association between input and output neurons.

Direct correlation is obtained by using Hebb's learning rule [34] that states: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency as one of the cells firing B is increased."

The Hebbian learning rule therefore specifies how much the weight of the connection between two neurons should be adjusted (increased or decreased) in proportion to the product of their activations. The result is a direct correlation between input and output neurons in the case of the single layered network. Presenting *inhibitory* examples, in the case of binary values – negated inputs, creates inhibitory links in a single layered ANN. See figure 8 for illustration:



1) "Usual" ANN architecture

2) BRAINNE Architecture

Figure 8: BRAINNE Modification of an ANN

After the ANN is trained sufficiently, conjunctive rules are first extracted (see section 2.2.6.2), after which disjunctive rules are formed (see section 2.2.6.3). The step to select conjunctive rules assumes availability of weight values between input and output neurons. In other words, BRAINNE is a decompositional rule extraction algorithm.

2.2.6.2 Extracting Conjunctive Rules

Conjunctive rules have the following form:

If A_1 [AND A_i]* then B

Where A_i are attributes (representing either input or hidden units), [AND A_i]* denotes conditional repetition, and B is a concept (either a hidden or an output unit).

The following steps are followed in order to extract conjunctive rules:

1. The ANN is extended as described in section 2.2.6.2 and shown in figure 8. The ANN is trained using backpropagation (section 2.2.3).
2. Calculate the measure SSE_{ik} between input neuron i and ouput neuron k. SSE_{ik} is defined as:

$$SSE_{ik} = \sum_{j=1}^{J} (w_{kj} - v_{ij})^2 \tag{2.17}$$

Where w_{kj} and v_{ij} follow usual notation, namely weights between output unit k and hidden unit j and input unit i and hidden unit j respectively.

3. Train a single layer ANN using Hebb's rule and inhibitory examples (negated original inputs) to get the inhibitory links between input and output neurons.
4. BRAINNE uses a measure to quantify the significance of input neurons to output neurons, as the product of the weights of inhibitory links between the input and output neurons and the measure SSE_{ik}. The creators of the BRAINNE algorithm refer to the criterion as "product". The product is defined as:

$$Product_{ik} = Weight_{ik} * SSE_{ik}$$

Where $Weight_{ik}$ is the weight of the inhibitory (negative) link between input i and output k determined by Hebb's rule.

This product actually combines two measures (inhibitory link measurement and SSE measurement) into a single metric. Once all the measurements on input-output couples are done, the products are sorted from maximum to minimum. A cut off point is determined to separate relevant products from irrelevant ones. All the input attribute products above the cut-off point are declared "not so relevant" to the output and input attributes whose products are below the cut off point are declared "relevant" to the output. The cut-off point is defined as the point where the product value between two consecutive input attribute products differ in magnitude of the two to three times.

Consider the following example:

$\text{Product}_{12} = 4.5$, $\text{Product}_{22} = 3.9$,

$\text{Product}_{32} = 0.4$, $\text{Product}_{42} = 0.25$,

Then the cut off point is between Product_{22} and Product_{32}, so input attributes 3 and 4 are considered relevant to output 2.

It is not clear why this order of magnitude was selected. Intuitively, the cut-off point is problem dependent.

To finish the process of extracting conjunctive rules (in form of production rules) all that need to be done is to connect the relevant attributes with connective AND the resulting rule for our example is:

If (z_3 AND $z_{4)}$) then o_2

2.2.6.3 Extracting Disjunctive Rules

Disjunctive rules have the following form:

If A_1 [OR A_i]* then B

where A_i are attributes (either input or hidden units), [OR A_i]* denotes conditional repetition, and B is a concept (either a hidden or an output unit).

For the purpose of extracting disjunctive rules, the training example set is divided into multiple subsets, each of them corresponding to one of the output classes.

Each of these subsets is then subdivided into more subsets, until each subset can be described by one conjunctive rule. Once that is achieved, all the conjunctive rules that describe each subset are identified. If the subsets are from the same initial set (in other words, corresponding to the same output class), then their conjunctive rules are combined using disjunctive OR.

The most frequently used attribute (or agroup of attributes) is used as an initial point to divide the training set into subsets. These attributes are selected according to their product values. The attribute with the smallest product value is selected and the training example set is divided into two subsets. Assume z_2 is the attribute with the smallest product value. The one subset consists of all examples that contain true values of z_2 , while the other contains all examples that contain false values of z_2, i.e. NOT z_2.

The process is summarised in a form of a recursive algorithm:

1. Let the example set A consist of K possible outputs (classes).
2. Let A_1, A_2, ... A_K denote K subsets of A, each corresponding to one output (class). Thus, $A = A_1 \cup A_2 \cup \ldots \cup A_K$ and $A_1 \subseteq A$, $A_1 \subseteq A, \ldots A_K \subseteq A$. Now for each A_k, k = 1 .. K

   2.a) Select an attribute with a minimum product value from the list of attributes and subdivide A_k into two subsets B_{k1} and B_{k2} by dividing according to the value of the selected attribute. The property that $B_{k1} \cap B_{k2} = \{\}$ must hold. For each $S=B_{kj}$,

- Form one conjunctive rule describing B_{kj} by collating the list of attributes selected so far.
- If the rule uniquely describes B_{kj} stop the further expansion of the rule, and process the next subset at step 2.a or next subset at step 2.
- If the rule does not uniquely describe B_{kj}, does the rule select a contradictory attribute for B_{kj}? If yes, further expansion of the rule is stopped and the next subset at step 2.1 or next subset at step 2 is processed.

If a contradictory attribute is not selected, further specialise the subset B_{kj} by letting $S = B_{kj}$ and repeating step 2.a.

2.2.6.2 Handling Discrete and Continuous Values

BRAINNE defines rules as a list of attributes and their values. In the case of discrete data it is a simple task to check the value against possible attribute values for a particular rule.

Two simple rules define if a presented pattern S is covered by a rule. For example:

a) If all of the attribute values of rule R are present in sample S, then rule R covers sample S (1)

b) If all of the attribute values of rule R are not present in sample S, then rule R does not cover sample S. (2)

To illustrate this, assume a rule that consists of two attributes, a binary attribute and a discrete attribute, say colour. The attribute values for which the rule holds are (1, Green). Now if samples (1, Solid, Red) and (1, Liquid, Green) are presented, a simple comparison is done according to rules (1) and (2) above. Only the second example (1, Liquid, Green) is covered by the rule.

The solution to the problem of dealing with continuous values is not that simple. One of the most common approaches to handling continuous values is to convert continuous values into discrete values. The BRAINNE algorithm follows this approach. When constructing a rule, which contains a continuous-valued attribute, BRAINNE divides the continuous values of that attribute into two sets. The first set of continuous values contains those that satisfy the rule and the other set contains the values that do not satisfy the rule. The set of attribute values that satisfies the rule is defined as an interval limited by a lower and an upper bound.

In other words, if the attribute value is in between those boundaries, then the rule holds. The process of constructing the rule that handles continuous values consists of three steps that are explained in detail:

1. Determining the error between an example and the rule
2. Choosing the threshold that defines coverage by the rule

3. Calculation of upper and lower bounds.

The first step is to determine the error between an example and the rule. To quantify the error between the rule R and an example S, BRAINNE uses the sum difference that is defined as:

$$SD = \sum_{i=1}^{I} (r_i - s_i)^2 \tag{2.18}$$

where I is total number of attributes that define a rule R and, r_i is i-th attribute value in a rule R and s_i is attribute value of i-th attribute in example S. If the attribute is present in the rule, then the attribute value in the rule is 1. Values of continuous attributes in example S are normalised to values between 0 and 1.

To illustrate, assume a rule with continuous attributes A and B and binary attribute C:

If (A AND NOT B AND C) then X

The NOT operator is valid for continuous attributes, since these attributes are represented by the range boundaries MinB and MaxB. A value 0 (false) for B therefore means attribute values are not in the interval (Minb, MaxB).

Assume the following attribute values (1, 0, 1). Condition (A AND NOT B AND C) can be read as:

> The value of the first attribute is in interval (MinA, MaxA) and the value of the second attribute is not in the interval (MinB, MaxB) and attribute C is 1.

where MinA and MaxA are the lower and upper boundaries of attribute value A for which the rule holds (the same applies for attribute B).

Consider sample S defined as (0.8,0.2,1), then the SD is $(1-0.8)^2 + (0 - 0.2)^2 + (1-1)^2 = 0.04 + 0.04 + 0 = 0.08$.

The second step is to determine the threshold. Once the error is quantified, it is compared to a threshold (in the case of the first sample, the threshold has a predetermined default value) to check if the rule covers the example. If the error is bigger than the predetermined threshold, the rule does not cover the example, otherwise the rule covers the sample. The threshold in BRAINNE is dynamically increased, if necessary, after the introduction of another continuous attribute to a set of attributes that forms a rule. The reason for this increase is that by introducing more continuous attributes, the error will grow. In the case of binary and discrete values this is not the case since the error between two binary and discrete attributes is always either 0 or 1. The increase in threshold value is guided by the heuristic:

> If the rules obtained by introducing the new attribute into the original rule set do not cover the same, or "sufficiently" same number of examples as the original rule, increase the threshold.

The third and final step is to determine the lower and upper bounds of continuous attributes for which the rule holds. The calculation of these boundaries is fairly simple, calculating the SD for all examples and determining which ones are covered by the rule (i.e. those rules that have their SD less than the threshold). Once calculations are done, the values of the continuous attribute are examined over all covered examples to find the minimum and maximum values. The minimum and maximum values obtained for each of the continuous attributes are then taken as the lower and upper boundaries of the intervals of attribute values for which the rule holds.

This completes the method of handling the continuous values as employed by the BRAINNE algorithm. This method assumes existence of an interval for which a continuous attribute satisfies the condition of a rule. If the values that satisfy the condition of a rule can be grouped in two or more intervals, then this method would fail. In this case, a different approach is necessary. For example, instead of dividing the set of continuous attribute values into two subsets, the continuous values can be discretised. In other words, the continuous values are divided into smaller intervals until all the values in an interval either satisfy the rule or not.

2.3 Evolutionary Computing Based Rule Extraction Algorithms

The exhaustive process of finding optimal solutions is very resource heavy for complex problems. All of the approaches discussed in this thesis employ some form of heuristics in order to minimise the search space. Evolutionary computing is the broad term for various paradigms that employ a similar type of heuristics based on natural evolution. The predominant idea behind evolutionary heuristics is the survival of the fittest, the same principle that is behind the theory of natural evolution. In an applied sense of evolutionary computing, this reads as the survival of the best of "near optimal" solutions.

Evolutionary computing includes the following popular paradigms:

- **Genetic Algorithms**

  The genetic algorithm (GA) was the first paradigm to fit into the evolutionary computing area. This paradigm is of special interest to this thesis, since most data mining and knowledge discovery algorithms from the evolutionary computing group are based on the GA paradigm. Algorithms that use a GA for knowledge extraction are also referred to as genetics based machine learning (GBML) in order to distinguish between "Classical" Machine Learning algorithms and GA based algorithms. The former are described in section 2.4. The GA concentrates on evolving genotypes that present encoded solutions. In the terms of rule extraction, an evolving genotype presents a potential rule. The theory behind GA and GBML was provided by Holland [37] in his pioneering work. His schemata processors paradigm effectively gave birth to the whole, previously unexplored, field of GBML. It took some time for the field to gain momentum, but despite this, the first genetic algorithm (GA) classifier system was presented and applied in 1978 [38]. This system was called Cognitive System Level One (CS-1). It was trained to learn a two-dimensional maze running problem. Since 1978 a number of different GA based classifier systems have been developed, with variable success on various problem domains. One of the GA based algorithms is presented in section 2.3.3.

- **Genetic Programming**

Genetic programming (GP) can be viewed as a specialisation of GAs. The main difference between GAs and GP lies in the interpretation of the genotype. In the standard GA paradigm a candidate solution is represented as a bitstring, whereas GP uses a tree representation to represent an executable program. In terms of rule extraction, GP represents a candidate solution as a decision tree. Executing a genotype, and evaluating its effectiveness, obtains a given genotype's (program's) fitness function. A GP approach to rule extraction is discussed in section 2.3.4.

- **Evolutionary Programming**

The evolutionary programming (EP) paradigm is substantially different from both GAs and GP. EP emphasise on behavioural evolution instead of genetic evolution. Evolutionary programming evolves behaviour from the space of behavioural models instead of a genotype. EP therefore implements only mutation and no crossover. EP has not been used (yet) for rule extraction, and therefore falls outside the scope of this thesis.

- **Evolutionary Strategies**

The evolutionary strategies paradigm is based on the evolution of evolution [55]. This approach combines both the behavioural (as in evolutionary programming) and the genotypic (as in GA and GP). The evolutionary strategies paradigm is also controlled by a set of strategies that is devised to improve the population. In a sense it is less random than other paradigms. There has been no implementation of this paradigm in the knowledge discovery problem domain, and is therefore not within the scope of this thesis.

Section 2.3.1 presents an overview of general characteristics of an evolutionary algorithm, while section 2.3.2 discusses evolutionary operators in general. The simple classifier system level one (SCS-1), is discussed in section 2.3.3, while a GP approach, BGP, is discussed in section 2.3.4

2.3.1 Evolutionary Algorithm General Characteristics

This section discusses the general characteristics of evolutionary algorithms. A pseudo code of a generalised evolutionary algorithm is given and the evolutionary operators discussed.

The typical evolutionary algorithm (EA) is an optimisation technique where a fitness function (discussed later in this section) is the function to be optimised. An EA operates on n-tuples (usually binary valued), which represent the encoding of n application parameters. The n-tuple is usually referred to as a parameter string, or chromosome. Each n-tuple represents a potential solution to the optimisation problem. The set of chromosomes or individuals is referred to as the population. For rule extraction applications a chromosome, or an individual, represents a potential rule, or rule set. One of the first obstacles for the successful implementation of an EA is that an EA works much better with binary values, but a vast majority of real-world problems include continuous-valued attributes. This obstacle can be overcome with either discretising initial continuous values, or by using some way of encoding (which is usually computationally expensive), like converting floating point into a binary representation.

An EA searches for an optimal individual from a population of potential solutions, rather than from a single solution that needs to be approved. The optimisation performed by an EA is done through iterations that are called generations. One way of thinking in natural evolution terms is the gradual improvement of the population. The best specimen (or their genome) have more chance to stay in the population, while the worst specimen are taken out of the population in analogy with the natural principle: survival of the fittest. The quantifier for quality of a specimen is some kind of payoff function, which will reward the more promising individuals and punish the less promising individuals in a population. The ideal payoff function needs to measure the distance from the optimal (or nearly optimal) solution to an individual. The payoff function is usually referred to as the fitness function.

In order to move from one population to another (a transition known as a generation), an EA uses a set of probabilistic genetics-based operators such as reproduction, mutation and crossover (which are discussed in section 2.3.2).

The following pseudo code describes a generalised EA

```
1.    Initialise a population $C_g$ of N individuals,
2.  Reset generation counter ($g = 0$)
3.  While no stopping criterion is satisfied
        a. Evaluate the fitness $F_{EA}(C_{g,p})$ of each individual ($p = 1 ..P$) in the population $C_g$
        b. Perform crossover:
        c. Select two individuals $C_{g,i}$ and $C_{g,j}$ using one of the selection methods as described in section 2.3.2
        d. Produce offspring from individuals $C_{g,i}$ and $C_{g,j}$
4.  Perform mutation:
        a. Select an individual $C_{g,n}$
        b. Mutate $C_{g,n}$ according to mutation probability
5.  Select the new generation $C_{g+1}$ from existing population using fitness function
6.  Increase the generation counter $g = g + 1$
7.  Repeat steps 3 to 6 until a stopping criterion is satisfied
```

The stopping criterion can be defined in terms of:

- A Solution has been found that is sufficiently close to the optimum.
- No improvement on either the maximum or the average fitness function has been achieved after a certain period.
- The maximum number of predetermined generations has been exceeded.

- Some other resource-restricted threshold has been exceeded (e.g. time, implicitly stated in number of generations).

Each individual in the population is a potential solution. Since each individual is a potential solution, each individual is encoded with parameters that represent the parameters of a solution. The set of parameters that encode an individual is called a chromosome or a genome. These parameters are also known as the characteristics of an individual. For different evolutionary algorithms, a chromosome represents different things. For a GA, a chromosome is usually a set of binary encoded values (Boolean, integer, even discretised real numbers), for a GP, a chromosome represents a set of instructions for program execution. An evolutionary programming chromosome encodes real numbers.

The fitness function is used to quantify how close a potential solution is to the optimum. Usually, it is a mapping from n-dimensional space (the n parameters of the chromosome) into scalar space. The choice of the fitness function is crucial to the success of evolutionary algorithms and, understandably choosing the right fitness function is often not a trivial task. If the fitness function does not encompass all the criteria for the optimal solution, it is highly unlikely that even the best-implemented evolutionary algorithm will produce satisfactory results.

When choosing the initial population, it is possible to take into account any prior knowledge about the problem domain, by encoding existing knowledge and making it a part of the initial population. This choice can lead to faster convergence towards an optimal solution, but it can also bias towards a certain area of the search space, which in turn can lead to premature convergence to a local minimum. A more popular choice of the initial population is a complete random initialisation of the individuals.

2.3.2 Evolutionary Operators

In this section, the evolutionary operators (the operators that simulate natural evolution) are presented and discussed. Evolutionary operators can be broadly divided into two groups: one group is used for reproduction and the other is used for selection.

2.3.2.1 Reproduction Operators

The main idea behind evolutionary algorithms is that, using evolutionary techniques, the population moves towards an optimal solution. It can be said that the population "evolve" towards an optimal solution. Evolution progresses from one generation to another. The population evolves through reproduction from one generation to another. For the purposes of reproduction, the evolutionary algorithm applies two operators to the population:

- **Crossover**

Crossover is the process of generating offspring, or new solutions, by combining genetic material of two parent individuals. The easiest way of implementing this technique is the so called 1-point crossover, where chromosomes of the parents are divided into two parts, and then new chromosomes (individuals) consist of the one part belonging to one parent and the other part belonging to the other parent.

To illustrate this technique, consider two chromosomes

A: 10010011 and B: 01110111

and assume that the crossover point is at position 3 (for chromosome A, the value is 0 and for chromosome B, 1). The offspring chromosomes C and D are:

C: 10110111 and D: 01010011

- **Mutation**

Mutation is a technique used to inject new "possibilities" into an existing gene pool, thereby increasing the diversity of the population. Mutations are random changes of a genome. In a sense mutation increases the search space. This process also closely mimics natural processes. If there are too many mutations, it may happen that the best specimen from the population gets affected, which can degrade the "quality" of the whole population. There are different propositions on how to prevent this, ranging from a very low mutation rate to the prohibition of the mutation process to successful individuals (the individuals that show promise, i.e. usually the fittest individuals in the population).

2.3.2.2 Selection Operators

The objective of selection operators is to ensure that the population is evolving towards a better solution. Selection operators award a better specimen with bigger chances for reproduction, prevent extensive mutations on those better specimen, and, in certain cases, ensure that the fittest members are automatically qualified as members of the next generation (using the elitism operator). The selection operator usually uses the fitness function to select the best individuals. Fitness values are transformed in one of two ways before used in any selection process, namely:

- **Explicit Fitness Remapping**

The calculated fitness value of each individual is scaled into a new range (usually [0,1]). This new scaled value is then used as fitness measure for selection.

- **Implicit Fitness Remapping**

The fitness value of each individual is used with no transformation.

Several selection methods have been developed, of which the most frequently used are reviewed below:

- **Random Selection**

Random selection is by far the simplest method of selection. The main disadvantage of random selection is that it does not discriminate between "good" (higher quality) individuals and "bad" (lower quality) individuals.

- **Elitism**

Elitism is more of a strategy and an idea, than a clear selection criterion. The idea is that the selected best specimen are automatically included into the next generation, thereby ensuring that the best individuals survive to the next generation. The number of chosen specimen is often referred to as the generation gap. The choice of the specimen is usually based on its fitness.

- **Tournament Selection**

Tournament selection combines random selection and "the-winner-takes-all" approach. In short, a number of specimen are selected (usually using random selection) and then the best (one with the best fitness value) specimen is chosen.

- **Proportional Selection**

With proportional selection, the probability of the individual to be chosen is directly dependent on its fitness value. The fitter the individual is, the better are its chances of being selected. One of the most popular proportional selection techniques is "roulette wheel" selection. The individual's fitness is presented as a slice on a roulette wheel. The higher the fitness value, the bigger the slice, hence the bigger the chances of being selected. The probability of selection can be given as:

$$\mathrm{Prob}(C_n) = \frac{F_{EA}(C_n)}{\sum_{i=1}^{N} F_{EA}(C_n)} \tag{2.19}$$

where $F_{EA}(C_n)$ is the fitness of individual C_n, and N is the total number of individuals in the population.

2.3.3 Simple Classifier System 1 (SCS-1)

This section presents and discusses the Simple Classifier System 1 (SCS-1), developed by Goldberg [65]. SCS-1 is one of the earliest genetic algorithms and one of the most popular textbook examples of evolutionary computing algorithms used for rule extraction. SCS-1 is a classifier algorithm that incorporates a genetic algorithm to search for rules. SCS-1 operates under the premise that it receives stimuli from the environment (in other words, input data), tries to classify the stimuli using a population of potential classifying rules, and then rewards the successful rules and punishes the unsuccessful rules. SCS-1 consists of three main components, namely a message system (discussed in section 2.3.3.1), an apportionment of credit system (discussed in section 2.3.3.2) and a genetic algorithm (discussed in section 2.3.3.3). The main function performed by the message system is pattern matching between messages and potential rules. The apportionment of credit system provides the SCS-1 with a clean, if somewhat complicated, means of evaluating the potential rules. The apportionment of credit system bears similarity to a blackboard architecture framework [33], as used by certain expert systems, namely HEARSAY[24] and BB1[43]. The genetic algorithm in SCS-1 has as its sole purpose to inject new rules into an existing set of rules, that are then evaluated via the apportionment of credit system component. SCS-1 executes a predetermined number of cycles, each cycle representing a new generation. A cycle consists of sequential execution of all its components, starting with the rule message system and ending with the genetic algorithm.

2.3.3.1 Rule and Message System

The rule and message system is the pattern-matching part of the SCS-1 algorithm. A message is a term used by the author of SCS-1, and in a sense, can be either a sample presented to the SCS-1 system, or a consequent of a rule. The SCS-1 can handle only binary messages, where a message is defined as:

$$<\text{message}> ::= \{0,1\}^n$$

In other words, a message corresponds to a binary input pattern.

A classifier is defined as a production rule of the form:

<classifier> ::= <condition>:<message>

The classifier is a potential rule, and can be thought of as an If-Then statement. If the condition is satisfied (in our case, an incoming message matched against a condition pattern), then the classifier message is posted. The condition is a sample pattern recognition device, with the wild card character '#' added to the existing alphabet {0,1}.

Once the classifier condition is matched, the classifier is posted to the qualifying list of classifiers. Then the message is evaluated against the conditions of all the qualified classifiers, according to the strength of each of the classifiers. The stronger classifier is chosen over the weaker classifier.

To illustrate the working of the rule and message system, assume a message (sample pattern) from outside of the SCS-1 to be encoded as 1100, and assume two classifiers: 01##:0010 and #100:0011.

The message is not matched by the first rule, but it is matched by the second rule. The second rule then posts the message: 0011.

2.3.3.2 Apportionment of Credit System

In order to distinguish between potential good and bad (useful and useless) classifiers, some kind of reward and punishment system is desired. The reward and punishment system increases or decreases the fitness values. The most prevalent method of awarding successful classifiers is the "bucket brigade" algorithm. This algorithm consists of two main components: an "auction" component and a "clearinghouse" component. When the classifiers from the qualifying list are matched, they do not directly post their messages. Instead, they compete for the right to post a message by bidding only a predetermined (parameter driven value, usually 10%) percentage of their strength. Initially, all the individuals in the population of classifiers have the same strength. The highest bidder (the most successful rule so far, i.e. the one with the highest number of successfully matched messages) then posts its message. The message, as posted by the winning classifier, is then evaluated against a training example, and, if it matches, then the classifier was successful. The classifier is then awarded by extra strength points, which will make it more successful in securing future bids. The highest bidder must then clear (divide) its payment (payment is the sum of all bids posted by qualifying classifiers) through the process referred by Holand [65] as a clearinghouse, where the bid payment is distributed in some manner amongst the matching classifiers in the qualifying list. This distribution of the payoff helps to ensure that different rules can coexist in the population. The distribution mechanism does not award all the strength to only one rule, thus preventing the creation of one super rule, covering most of the examples.

2.3.2.3 Genetic Algorithm

The apportionment of the credit system ensures a clean procedure for rewarding (and punishing) classifiers. Since the initial population of potential rules is randomly created, there is no guarantee that these potential rules are sufficiently good rules. Therefore, we must find a way of enriching the variety of the rules population. This is the role of the genetic algorithm. It enriches the rules population by means of standard genetic algorithm techniques, such as mutation and crossover. The genetic algorithm employed in SCS-1 is the standard genetic algorithm with both 1 point crossover and mutation implemented (as explained in section 2.3.1.2). Roulette wheel selection, as described in section 2.3.1.3, is used.

For more information on SCS-1, the reader is referred to the work of Goldberg [29].

2.3.4 Building Blocks Approach to Genetic Programming

The building block approach to genetic programming (BGP), developed by Rouwhorst and Engelbrecht [58], combines decision trees and evolutionary computing. One of the unusual characteristics of this algorithm is that, unlike typical EAs, it uses a direct representation of the solution. Individuals, or candidate solutions, are represented in the form of decision trees. The search space is then the space of all possible decision trees, given a finite set of input parameters and conditions. The BGP algorithm starts with an initial population of simple individuals, consisting of only one building block. Individuals grow in complexity during the evolutionary process through the addition of new building blocks when the simplicity of the candidate solutions can not describe the complexity of the underlying data. This is contrary to standard GP approaches to evolve decision trees, which start with complex initial individuals representing entire decision trees. Section 2.3.4.1 discusses the decision tree representation used by BGP and section 2.3.4.2 discusses the implementation of the genetic operators.

2.3.4.1 Decision Tree Representation

Each individual in the BGP algorithm is represented as a decision tree. Each node in the decision tree represents one building block, which consists of three parts, namely an input attribute, a relational operator and a threshold. Input attributes are allowed to be of any type, e.g. discreet, nominal, binary or continuous. The relational operators == and <> are used for all attribute types, while the operators <, <=,>,>= are used for continuous valued attributes only. The threshold can be a numerical value, or another input attribute. The inclusion of attributes as thresholds add the ability to extract rules of the form

If $A_1 < A_2$ then class1

Each tree is parsed from the root to the leaf nodes. Each node is a condition, except for the leaf nodes. A leaf node corresponds to a class. The path from the root to a leaf represent a rule.

2.3.4.2 Genetic Operator Implementations

BGP implements various genetic operators, including, crossover, mutation and pruning. Each of these operators are applied with a certain probability as specified by the user.

The crossover operator randomly selects a random point in each of the parent trees. Swapping the subtrees of the two parents from the selected points then generates two offspring. The crossover operator is illustrated on a genetic programming example in figure 9, where the objective is to evolve a mathematical expression.

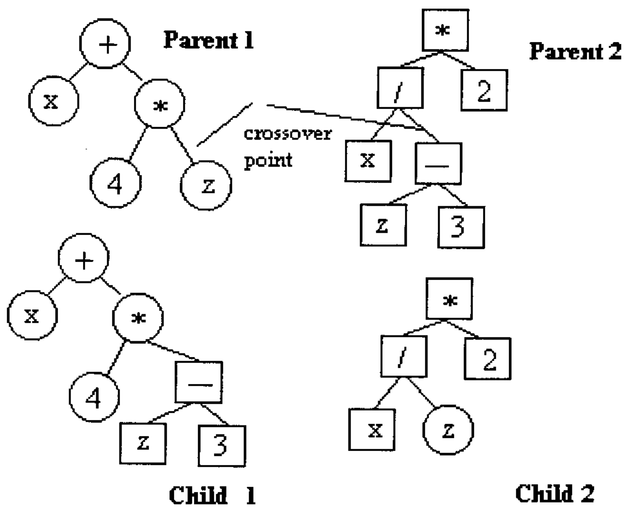


Figure 9: Genetic Programming Crossover

Mutations are implemented on three different levels. The first one is a mutation on the attribute value itself (data level) and the other two are mutation on the condition level (meta data level) and mutation of the threshold (meta data level).

- Mutation on attribute level is implemented as replacing of an attribute by randomly choosing another attribute, not changing the attribute value itself. This operation is under the restriction that the semantics of the rule still must be satisfied. For example, it is prohibited to replace a binary-valued attribute with a continuous-valued attribute.

- Mutation on the condition level is of such a nature that it takes the condition operator to a newly allowed one from the set {==,!=, >=, <=, <, > }. In other words, this mutation randomly selects a new relational attribute, while still adhering to operator type constraints.

- Mutation of threshold can be in one of two ways, depending on the threshold type:

  a) If the threshold is a value, then a new value is selected randomly.

  b) If the threshold is another attribute, that attribute is replaced with randomly selected attribute or value.

The pruning operator is another form of mutation. This operator randomly selects a non-leaf node, detects the corresponding subtree and replaces the non-leaf node with a leaf node indicating the class that are mostly covered by the corresponding rule.

2.3.4.3 BGP Algorithm

The BGP algorithm evolves decision trees from a training set, and determine the generalisation abilities from a test set of data patterns not seen during training.

A pseudo code algorithm for BGP is given below:

```
1. Read training data and randomly initialise a population of simple
   decision trees
2. While the best tree is still not "optimal enough" (error is higher than a
   predetermined error threshold) repeat:
      a) if more nodes can be added (if there are attributes available
      for subdivision) then add nodes to the trees in the current
      population;
      b) for the whole population do tournament selection of candidates
      in the population and apply evolutionary operators (as described
      in 2.3.4.2).
```

3. If a new best tree has been found, then declare it the new best tree, and repeat step 2.
4. Once an "optimal enough" tree is found, extract the rules from it.

After initialisation, the algorithm starts with a population of trees that consist of only one building block. The copy of the "best tree so far" (measured in terms of the number of examples correctly covered) is stored separately. Each cycle starts with a test to see if the current best tree can be accepted as satisfactorily, and to terminate the evolutionary process. The following test is used for this purpose:

$$\text{IF (MinRuleAcc} > e^{(c\,(\text{trainsize} / T(0)) - c(\text{trainsize}/T(t)))}\text{) THEN OPTIMAL TREE}$$

where MinRuleAcc is the accuracy of the worst rule in the tree, c is a parameter of the algorithm and train size is the number of training instances. T(t) is the temperature function that is implemented as :

$$T(t) = T(0) - t$$

where T(0) is the initial temperature, and t denotes the generation number.

For more information on BGP, the reader is referred to work of Rouwhorst et al [58].

2.4 Classical Machine Learning Rule Extraction Algorithms

The decision tree based algorithms by Hunt et al [39] were probably the first attempts to solve the rule extraction problem. Decision tree algorithms are similar to already established techniques used in solving problems, namely "divide and conquer", as well as the recursive approach to problem solving. It is therefore not surprising that this class of algorithms was historically first. For many years (it is important to remember that connectionist-based and genetic-based rule extraction algorithms reappeared only in the late eighties), rule extraction algorithms based on decision trees were the only AI based tools for rule extraction. For the above stated reason, this thesis uses the term "Classical Machine Learning" for the group of decision tree rule extraction and rule induction algorithms.

Section 2.4.2 presents some general characteristics of classical machine learning algorithms. Section 2.4.3 explores decision tree algorithms, and discusses the C4.5 algorithm developed by Quinlan [54]. Section 2.4.3 presents the CN2 rule induction algorithm.

Although there are many variations of the basic machine learning algorithms a set of general characteristics of decision tree algorithms can be defined. Both algorithms used in this study (C4.5 and CN2), share the general characteristics of the basic machine learning algorithm.

A basic machine learning algorithm has the following characteristics:

- The algorithm assumes that samples have target classes. Decision trees and rule induction are therefore algorithms from the supervised learning algorithm class.
- The algorithm produces as a result a "logical" classification model. In other words, the output (or the rule) is in the form of a logical expression.

The general decision tree algorithm works as a classifier and produces an output that is a rule. In other words, the algorithm sorts the input into discrete, delineated classes, rather than into a continuous value.

2.4.1 Decision Tree Approaches

2.4.1.1 Introduction

The decision tree is a widely used format for representing rules. A decision tree has distinctive node types: If a node does not have any descendants, then it is a leaf node which represents a class. If the node has descendants, then it represents a condition which is referred to as a branch node (see figure 10).

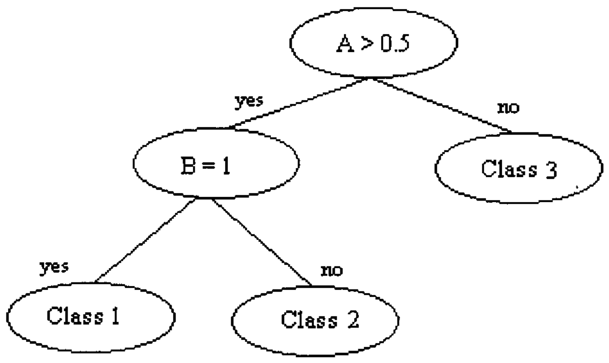


Figure 10: Decision Tree Example

Figure 10 illustrates a decision tree with three embedded rules:

- If (A < 0.5 AND B = 1) then Class 1
- If (A > 0.5 AND B <> 1) then Class 2
- If (A > 0.5) then Class 3

Decision tree based algorithms are based on ideas that go back to the late 1950s, to the work of Hunt and Hoveland [39]. Hunt published his experiments and proposed algorithms in his pioneering book "Experiments in Induction" [39], where several implementations of the "concept learning systems" (CLS) were described. Other researchers followed the initial research by Hunt, culminating in a few methods and algorithms similar to the Hunt method. The CART system [61], ID3 [53], and ASSISTANT [13] are examples of such decision tree algorithms. One of the most prolific researchers in this arena is Quinlan. Quinlan developed one of the most popular decision tree algorithms of the earlier years (before the late 1980s), namely ID3 [53]. ID3 was successful at two levels: firstly, it performed satisfactory, and secondly, it was simple enough to allow for modifications easily. AQ is another example of a "first generation" decision tree algorithm. ID3 and AQ, both early decision tree algorithms, formed the foundation for the next generation of decision tree algorithms such as C4.5 by Quinlan [39]. C4.5 evolved from the idea behind the ID3 algorithm. Since this study uses C4.5 for experimental results, C4.5 is discussed in detail in the next section.

2.4.1.2 C4.5 Algorithm

The C4.5 algorithm shares the general characteristics of the basic machine learning algorithm for rule extraction (see section 2.4.1). C4.5 creates a decision tree, which embeds rules describing the data. Leaf nodes indicate a class, and decision nodes indicate logical tests (conditions of a rule) to be applied on an attribute value.

The decision tree induction process consists of two phases, namely the construction of the decision tree and the pruning of the decision tree. After construction and pruning of the decision tree, an additional phase transforms the rules represented in the tree into a human friendly format of IF-THEN rules.

2.4.1.3 Constructing Decision Trees

C4.5 uses Hunt's method to construct decision trees [39], which is discussed next.

Let T denote the set of training cases and let C_n denote one of the n discrete classes. Considering training set T, there are three possibilities:

- If T contains one or more cases, all belonging to the same class C_1, then the decision tree for the training set T consists of one leaf that identifies C_1.
- If T is empty, then the decision tree for the training set is one node that does not denote any of the categories. In this case, we can say that the training set does not provide enough information to determine a class.
- If T contains cases that belong to more than one class, the algorithm uses the well-known "divide-and-conquer" strategy to proceed. At this point, the algorithm chooses a conditon applicable to a single attribute to divide the training set T into n subsets T_1,..,T_n according to n possible outcomes of the test, for $n \geq 2$. This process is repeated on each of the subsets.

Most real-world problems involve samples, which have many attributes. A criterion is therefore needed to select the most relevant attribute and condition upon that attribute to perform the split of the training set.

The ID3 algorithm uses the gain criterion to choose the attribute test that is used to split the training set. The gain criterion for a test X over training set T is defined as (based on concepts from information theory).

$$\text{gain}(X) = \text{info}(T) - \text{info}_X(T) \qquad (2.20)$$

where:

$$\text{info}(T) = -\sum_{i=1}^{K} \frac{\text{freq}(C_i, T)}{|T|} \times \log_2\left(\frac{\text{freq}(C_i, T)}{|T|}\right)$$

(2.21)

where freq (C_i, T) denotes the number of cases in training set T that belongs to class C_i, K is the total number of classes and | T | denotes the cardinality of set T (i.e. the total number of cases); info(T) measures the average amount of information, expressed in bits, needed to identify the class of a case in the training set T. Info$_X$(T) is calculated as:

$$\text{info}_x(T) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \text{info}\,(T_i)$$

(2.21)

Equation (2.19) expresses: The average amount of information needed to perform a classification if the training set is split on condition X, or more formally, the information gained by applying a test that divides training set T into n disjunctive sets, T_1 .. T_n, is expressed as the weighted sum over subsets T_1 ..T_n. | T | and | T_i | denote the cardinalities of sets T and T_i respectively where i = 1..n.

The C4.5 algorithm uses a modified version of the ID3 criterion, referred to as the gain ratio criterion. The gain criterion used by ID3 favours tests with many outcomes. This can potentially lead to an "overfitting" of the rules and to large decision trees with a high branching factor. The term "overfitting" refers to a situation where a rule is not general enough. In other words it is fine-tuned to fit only a specific example. To illustrate the overfitting problem, all individuals within a specific database can be classified (using any classification scheme) using his/her identity number. But building a classifier system that classifies n individuals according to some classification scheme will require the compilation of n rules (each of them having only one attribute – identity number), which is clearly not desirable.

The gain ratio criterion used by the C4.5 algorithm is expressed as the gain divided by the potential information generated by dividing T into n subsets:

$$\text{gainratio}(X) = \text{gain}(X) / \text{splitinfo}(X) \qquad (2.22)$$

where the potential information generated by dividing T into n subsets is given by:

$$\text{splitinfo}_x(T) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2\left(\frac{|T_i|}{|T|}\right)$$

(2.23)

2.4.1.4 Pruning Decision Trees

The divide-and-conquer process of constructing decision trees continues until there are no more possible subdivisions (no more attributes that can be used for further division), or until all the subsets contain cases of just one class, or until no test offers any improvement in the accuracy of a rule. This can result in a very complex decision tree that overfits the data. The damage is twofold: Firstly, a more complex tree requires more resources in order to be parsed, and, secondly, it can lead to a higher error rate when applied to a data set different from the training set. It is, therefore, desirable to prune the decision tree in order to prevent "overfitting". However, pruning comes at a cost. Pruning increases the computational complexity of the induction process, and decreases the accuracy on the training set, but can improve generalisation. At this point it is important to note that pruning is a heuristic process and the pruning criterion is often not easily identifiable.

There are two basic approaches to pruning. The first is based on a test that will stop expanding the decision tree at some point, for example, if the number of conditions in a rule exceeds a predetermined value. This method is often referred to as pre-pruning (or stopping) [54]. The second approach is where expansion of the tree is always allowed, and after the expansion, certain branches are removed from the tree and replaced by leaves or less complex branches. This approach is referred to as post-pruning. The C4.5 algorithm adopts the post-pruning strategy.

The heuristic used for pruning by C4.5 is based on a reduced-error pruning technique that assesses the error rate between the pruned and the non-pruned tree, based on the set of separate cases [54]. The heuristic used by C4.5 is illustrated with the following example:

Let N denote the number of training cases, from the training set T that are covered by a leaf C (a class) and let E denote the set consisting of cases from the training set T that are incorrectly classified as members of leaf C (class C). E is a subset of N. Let CF be a confidence level. Then, the upper level on the probability of error is found from the confidence limits for the binomial distribution. C4.5 then uses this upper level as the predicted error rate and compares this rate for the leaf (pruned and unpruned tree) and for the subtree.

If the predicted error rate of a subtree is higher than that for the leaf, the leaf replaces the subtree.

2.4.1.5 Rule Conversion

The output of the C4.5 algorithm is a pruned decision tree. The decision tree can be viewed as a flowchart to be followed in order to classify certain examples. This works very well, but it is not always easy to see the rules in the decision tree, especially in complex tree structures.

Rules are very simply extracted from the decision tree. Each path from the root to a leaf represents one rule. All intermediate nodes form the condition part of the rule, while the leaf node represents the rule consequent (or a class).

All the rules created using this method are then evaluated, and if possible, simplified. The simplification process uses a similar heuristic to heuristics employed by the pruning method used by the main part of the C4.5 algorithm. After the simplification process, all rules denoting the same class are then sifted in order to remove rules that do not contribute to the accuracy of the entire rule set, or to remove subsumed rules.

Finally, the sets of rules are ordered according to the class they represent. The default class is then selected as the class with the most cases in the training set. The final output is then a set of if-then rules that are usually as accurate as the original decision tree, but they are presented in a human friendly format.

2.4.2 Rule Induction Approach

2.4.2.1 Introduction

The most widely used rule induction algorithm is the CN2 algorithm. The CN2 algorithm was developed by Clark [14], by combining the best characteristics of ID3 and the AQ [49] algorithms. For example, ID3's ability to deal with noisy data was preserved in CN2. The CN2 algorithm employs the beam search approach of the AQ algorithm, where each beam direction is a direction to a single class. The AQ algorithm identifies all the rules applying to one class and then moves on to another class. The objective of the CN2 algorithm is to improve the AQ algorithm, in such a manner that CN2 is not so sensitive to specific examples and that the rule search space is increased by means of allowing rules to remain in consideration, even if they do not perform perfectly on training examples. CN2 employs, as most machine learning algorithms, heuristics to guide the search, namely entropy (to quantify the goodness of a rule) and significance (to express the reliability of a rule). Both of these are explained in the next section, as is the basic CN2 learning algorithm.

2.4.2.2 CN2 Induction Algorithm

In order to understand the CN2 algorithm, consider the following definitions:

- *Selector.* A test on an attribute is called a selector. The allowed test operations are defined as the set {=, ≤, >, ≠}.
- *Complex.* The conjunction of two or more operators is called a complex. This is the condition part of a rule.
- *Cover.* The disjunction of two or more selectors is called cover.
- *Rules.* Rules generated by CN2 are of the form:

  If <complex> then <class>

A pseudo code for the CN2 algorithm is presented below:

```
1. Initialise the rule list to empty. Initialise the complex set to non-
   specialised.
2. Find the best complex BESTCOMPLEX:
      a) Specialise all the complexes in the current complex set by
         intersecting the complex with all the selectors available. Store the
         results in the new complex list.
```

b) Remove all the complexes from the new complex list that are in the old list or are invalid (contradictory complexes, e.g. (if A = 1 AND A = 0))

c) For every complex from the new complex list, check if the complex is better than the current BESTCOMPLEX. If it is, declare that complex as the new BESTCOMPLEX and remove the worst complexes from the new complex set.

d) Declare the new complex set the current set.

3. Remove from the training set all examples covered by the BESTCOMPLEX

4. Let class C denote the most common class of all examples covered by the BESTCOMPLEX, then add the following rule to the rule list:

If BESTCOMPLEX then C

5. Repeat steps 2-4 until a stopping criterion is satisfied. The stopping criterion is satisfied if BESTCOMPLEX is invalid, i.e. contradictory or if all the examples from the training set are covered (training set is empty).

2.4.2.3 Heuristics used by CN2

CN2 employs two characteristics: the first heuristic is quantifies the quality of complexes (step 2.c in the pseudo code), using an *entropy* measure. Entropy of a rule (complex and class that it determines) is defined as:

$$\text{Entropy} = -\sum_{i=1}^{I} p_i \log_2(p_i) \tag{2.24}$$

where I is number of classes, p_i is the probability distribution of a class C_i in the set of all examples covered by the rule. This measure favours the rules that cover more examples (more general rules), which is a desirable characteristic.

The second heuristic employed by CN2 is used to quantify the significance of a rule. For example, a rule that covers say 90% examples is more significant than a rule that covers 10% of examples. Significance is calculated as the likelihood ratio statistic:

$$\text{Significance} = 2\sum_{i=1}^{N} f_i \log_2(\frac{f_i}{e_i}) \tag{2.25}$$

where $F = (f_1, f_2, \ldots, f_n)$ is the observed frequency distribution of examples among classes covering a rule, and $E = (e_1, e_2, \ldots, e_n)$ is the expected frequency distribution of the same number of instances under the assumption that the complex selects examples randomly.

This concludes the overview of the CN2 algorithm. More details on CN2 algorithms can be found in work of Clark et al [14].

Both algorithms, C4.5 and CN2, presented in section 2.4 are used for the experiments in chapter 4.

Chapter 3

Hybrid Classifier System

This chapter presents a new rule extraction approach, namely Hybrid Classifier System (HCS) [56]. HCS combines both heuristic and exhaustive search. The search space is the set of all possible rules that can be formed from a combination of all possible conditions on a finite set of attributes. The algorithm follows a building blocks approach to construct rules. HCS starts with a set of potential rules that contain a minimum number of attributes, i.e. one attribute. Rules are then expanded by adding one attribute at a time. This is similar to building block approach, where individuals improve in complexity when their simplicity cannot account for the complex relationships between attributes. The expansion of rules is exhaustive, but as soon as a new rule is produced, it is evaluated against a fitness function. If the potential new rule is not promising enough (e.g. if it does not exceed a certain parameterised and dynamic threshold), it is taken out of the population. By changing the parameters of the fitness function, the "exhaustiveness" of the algorithm can be adjusted. In other words, through adjustment of these parameters, upper and lower bounds are determined, thus controlling the accuracy and complexity. If the upper bound is set too low, accuracy is lowered. If the lower bound is too high, possible rules can be discarded too soon.

The HCS is loosely based on schemata theory [36], presented in section 3.1. Schemata theory leads to a building block approach, which should lead to the most general rules, and enable the algorithm to construct satisfactory results sooner.

3.1 Introduction to the Schemata Theory

Schemata theory [36] is based on the fact that the search that exploits similarities in the patterns is very efficient. It is this concept of a similarity template, or schema, which leads to the building block approach.

A schema is a similarity template describing a subset of strings with similarities at certain string positions. To illustrate, by example, consider the binary alphabet {0,1}. Adding the wild card character '#' to the binary alphabet results in the extended alphabet {0,1,#}. Assume a maximum string length of four. Using this extended alphabet, it is possible to create strings (schemata) over the ternary alphabet {0,1,#}, using the schema as a pattern-matching device. For instance, the schema 0*00 matches two strings, namely {0000, 0100}. The schema ##11 matches four strings, namely {0011, 0111, 1011, 1111}.

The wild card character '#' is just a device that enables the description of all possible similarities among strings of a particular length and alphabet. This notational device explicitly recognises all the possible similarities in a population of strings.

Each schemata has two attributes:

- Schema Order, o(H)

  Schema order is the number of fixed positions present in the similarity template. For example, o(011*1**) = 4 and o(0*****) = 1

- Defining length, d(H)

  Defining length is the distance between the first and last specific string position. For example, d(011*1**) = 4 because the last specific position is 5, and the first one is 1.

Schemata that are highly fit, according to the chosen fitness function, and with short defining length are referred to as the building blocks [26].
The objective of HCS is to build such schemata through a hybrid exhaustive and heuristic search strategy.

3.2 HCS Overview

In this section, an overview of the main parts of the HCS algorithm is given, as well as an example that is used to illustrate the working of HCS. The example in this section is used for the remainder of this chapter.

As noted in the introduction of this chapter, the HCS algorithm consists of two main parts:

- A rule pruning system, which evaluates the potential rules in the list of rules and prunes them accordingly (see section 3.2).
- A rule expansion system, which expands the surviving rules (the promising rules, not removed in the pruning process) (see section 3.3) by adding a new building block.

The HCS is illustrated using the following example:

Rules to be extracted are given by the Boolean function (A AND B) OR (A AND NOT C). The complete set of possible values for the three variables and the corresponding output of the Boolean function are taken as the training set for this example is listed in table 3.1.

A	B	C	(A AND B) OR (A AND NOT C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 3.1: Truth Table for Boolean function (A AND B) OR (A AND NOT C).

3.3 Initial Rule Population

The initial rule population consists of a complete set of rules with only one attribute initialised to either 0 or 1. If there are n attributes, the initial population therefore consist of 2n rules, with one attribute having a value, all other attributes have a value of #. For the example above, the initial population is:

{ 1## , 0## , #1# , #0# , ##1 , ##0 }

where '#' represents the wild card symbol in the sense of Holland's schemata theory [36]. It is also important to note that this example covers only binary values. The extension of HCS for discrete and continuous values is presented in section 3.4.

3.4 Rule Pruning System

The rule pruning system is the heart of the algorithm. Rules must be pruned, because the first part of the system is exhaustive. If rules are not pruned the algorithm reverts to a pure exhaustive search, which, due to the computational cost, is not desirable. Each rule is evaluated against the training samples, and the fitness value of that rule is calculated as the success rate in predicting the class of the matched samples. Three different scenarios can occur:

- If the rule matches the pattern of a particular class in more than the parameterised minimum percentage (supplied by the user), the rule is kept for future expansion.
- If the rule matches the pattern of a particular class in more than the parameterised maximum percentage (supplied by the user), the rule is declared as a valid rule and is taken from the rule population and no further children of the rule are created.
- If the rule matches the pattern of a particular class in less than the parameterised minimum percentage (supplied by the user), the rule is taken out from the rule population, i.e. the rule set is pruned by removing an irrelevant rule.

From this description it is easy to conclude the importance of the parameterised minimum and maximum acceptance percentage. If the minimum percentage is too low, the algorithm will be more resource costly, and if it is too high, potential rules may be discarded before they are expanded into valid ones. The algorithm starts with a small minimum acceptance percentage, which is increased with every pass. Similarly, for the maximum acceptance percentage: if the percentage is too low, less accurate rules are extracted, and, if too high, the rules are too specific. An additional disadvantage of the rules extracted in this manner is that they would be very resource costly to arrive at.

Considering the above example, the fitness values for the initial population are:

Individual	Fitness	Individual	Fitness
1##	75%	0##	0%
#1#	50%	#0#	25%
##1	0%	##0	50%

Table 3.2 Initial Population and its Fitness Values

If the initial minimum acceptance percentage is 50% and the maximum acceptance percentage is 90%, the following potential rules survive to the next iteration to be expanded in the next step:

{ 1## , #1# , ##0 }

3.5 Rule Expansion System

The rule expansion system represents the exhaustive part of the algorithm. Each candidate rule is expanded by an additional attribute in the following manner: A copy of each individual is made. The value of the next unused attribute is set to 0 for the one copy and to 1 for the second copy. This process further specializes each of the rules.

Continuing with the example, the rules are expanded to:

{ 11#, 10#, 1#1, 1#0, #10, #11 }

After the expansion phase, the new population is passed through to the rule set pruning phase. For the next pruning phase the fitness values are:

Individual	Fitness	Individual	Fitness
11#	100%	10#	50%
1#1	50%	1#0	100%
#10	50%	#11	50%

Table 3.3 Fitness Values of a New Generation

In this pass, the minimum percentage is greater than 50%, so rules 10#, 1#1, #10 and #11 are discarded, and rules 11# and 1#0 are declared valid rules, and are taken out of the population. The population is now empty and the process stops because there are no new possible rules to create.

A pseudo code for the HCS is presented below:

1. Initialise the population as described in section 3.3.
2. While there is a potential rule in the current population, remove the potential rule from the current population and expand it with an additional, previously unused attribute for all allowed values of that attribute.

   a) If the new potential rule is above the rule threshold, declare it a rule.
   b) If the new potential rule is below pruning threshold, prune it
   c) If the new potential rule is between the rule and pruning thresholds, make it member of the new population

3. If there are more potential rules in the population repeat step 2, if not make new population of the current population.
4. Repeat steps 2-3 until a stopping criterion is satisfied. The stopping criterion is satisfied if the current population is empty, or if all of the attributes are used.

This concludes the example of the working of the HCS algorithm. In next section, mechanism for handling the discrete and continuous values is presented and discussed.

3. 6 Extension of HCS for Discrete and Continuous Values

In order to handle discrete and continuous values, the HCS algorithm requires minor modification. The pattern matching process and heuristics that guide the pruning of a potential rule are exactly the same. The first part of the algorithm (initialisation of the population) is modified in such a manner that it caters for all possible values for a discrete attribute. For example, consider the discrete attribute *TrafficLight*. The attribute *TrafficLight* can have three values (Red, Orange, Green) that can be encoded as (1, 2, 3). Then the initialisation of the *TrafficLight* attribute (assume *TrafficLight* is the second encoded attribute, and there are only three possible attributes) would yield the following initial population:

{ #1#, #2#, #3#}

The second part of the HCS algorithm, the rule pruning system stays unchanged. The third and final part of the algorithm, the rule expansion system, requires only minor modifications that are similar to the modifications done to the first part of the algorithm: The rule expansion system expands each attribute to all possible values.

When presented with continuous valued attributes, HCS employs a simple discretisation technique. The range of continuous values for an attribute is divided into a parameterised number of intervals. The default value for the number of intervals is six. The default was chosen intuitively and proper method that applies to a certain set of discrete values should be investigated. In the case of a rule that applies to the range of continuous values of an attribute, the HCS algorithm converts the rule into a rule that applies to certain set of a discrete values of that attribute. Once that is achieved it is a simple task to check the value against valid attribute values for a particular rule. To illustrate this technique, consider the following example:

The minimum attribute value is 0.5 and the maximum 3.5. Assume that the attribute for a training sample has a value of 2.7, and the attribute range for a rule is from 2.5 to 3.0.

The first step is to divide the range of the continuous values for the attribute into six intervals. The size of the interval SI is defined as:

SI = (MaxAttrValue – MinAttrValue) / NoIntervals

where MaxAttrValue is the maximum value for the considered attribute, MinAttrValue is the minimum value for the considered attribute and NoIntervals is the parameter that defines the number of intervals.

In our example:

SI = (3.5 – 0.5) / 6 = 3 / 6 = 0.5

The discrete attribute value is calculated as:

DV = (CV – MinAttrValue) DIV SI

where CV is original continuous value of the considered attribute and the DIV operator is a division between the integers.

Following our example, the discrete value for the considered attribute is:

DV = (2.7 – 0.5) DIV 0.5 = 2.2 DIV 0.5 = 4

The rule from this example is valid for the following discrete values:

DV1 = (2.5 – 0.5) DIV 0.5 = 2.0 DIV 0.5 = 4 and

DV2 = (3.0 – 0.5) DIV 0.5 = 2.5 DIV 0.5 = 4

The rule in our example is valid only for one discrete value, namely 4. In our example, the rule applies to the training sample because its attribute value is the same as the valid rule attribute value (in other words, the rule covers the sample).

This method has shortcomings that are mentioned, together with some of the ideas on how to alleviate them, in section 5.2. The method used, however, performs satisfactorily, especially when the number of intervals is high, because it leads to a finer differentiation between the attribute values which are valid for the rule and attribute values which are not valid for the rule. But more intervals increase the complexity, since more expansions need to be performed. There is therefore a trade-off between accuracy of the rules in terms of number of examples correctly covered, and complexity of the search process.

If the HCS is presented with a problem with continuous outputs, some pre-processing would be necessary. Specifically, the output should be discretised into discrete levels (classes) by the expert of the field.

In this chapter, the HCS algorithm was presented and briefly discussed. In the next chapter, the HCS algorithm is compared with representatives from each of the previously discussed classes of algorithms. Comparisons are done using well-known benchmark problems.

Chapter 4

Experimental Results

Various criteria to evaluate rule extraction algorithms are presented in this chapter, followed by the actual experimental results obtained from the algorithms chosen for comparison. The algorithms chosen in this study are: BRAINNE (section 2.2.6), representing the connectionist approach-based algorithms, BGP (section 2.3.4), representing the evolutionary programming approach-based algorithms, C4.5 (section 2.4.1) an example of decision tree approach-based algorithms, CN2 (section 2.4.2) as a representative of the rule induction algorithms, and, lastly, the HCS algorithm proposed in this thesis (chapter 3). All of the above-mentioned algorithms were tested on three benchmark data sets, namely: Monks, Iris and Diabetes [66].

4.1 Rule Extraction Algorithms Evaluation Criteria

Rule extraction algorithms can be evaluated according to the efficiency of the algorithm, and according to the quality and the consistency of the extracted rules:

- **Efficiency of the algorithm**

  The majority of rule extraction algorithms suffer from being computationally expensive. For example, in the case of decompositional ANN rule extraction algorithms, the need to explore most of the links between the neurons; or for decision tree based algorithms, the vast number of branches in the tree adds to complexity of the search process. Heuristics are usually employed to limit the size of a search space. In most cases, when we talk about efficiency of a rule extraction algorithm, we are talking about efficiency of its heuristics.

- **Consistency of the extracted rules:**

  A set of rules is consistent if there are no contradicting rules within it. In terms of contradiction, it is meant when there are two or more applicable rules, if applied, will classify an example as a member of different classes.

- **Quality of the extracted rules**

  One of the first rule quality evaluation criteria was proposed by Towell and Shavlik [65]. The proposed measurement for rule quality comprises of:

  - Accuracy: A set of rules created by a rule extraction algorithm is accurate if unknown (or previously not presented) examples are classified correctly.

- Fidelity: Fidelity is present if the extracted set of rules imitates the behaviour of the original classifier system from which those rules are extracted. This evaluation criterion, however, is not applicable for all rule extraction algorithms. It is applicable for most algorithms that use knowledge embedded in ANNs, genetic algorithms and decision trees.
- Comprehensibility: The comprehensibility of a set of extracted rules can be quantified by (1) the number of rules in the rule set and (2) the number of antecedents per rule.

The efficiency of rule extraction algorithms is not considered a measurement criterion in this thesis, although observations of the execution time will be mentioned in the conclusion (section 4.3.3). All algorithms, in their original form or with minor adjustments, produce rules in the form of production rules (section 2.1.1). Therefore, the readability of the rules will not be used as a measurement criterion. In the experimental results, no contradictory rules were detected (except in the case of the Monks dataset using BRAINNE, but those were discarded), so the emphasis is on the quality of the extracted rules. In the reminder of this chapter, algorithms are compared according to accuracy of the extracted rules (section 4.3.1), the number of extracted rules (section 4.3.2) and the number of the antecedents per rule.

4.2 Data Sets Used for Comparison of Algorithms

All data sets used for the purpose of this thesis were downloaded from the data repository of University of California, Irvine, Department of Information and Computer Sciences [66]. The following data sets were chosen because they are considered as benchmarks: the Monks data set has been "agreed" upon as a benchmark test, while the Iris data set is probably the most used data set in the area of machine learning, and finally, the diabetes data set is a well known and used data set.

All datasets were split into 30 training and validation sets. The details on numbers of samples used in each set are discussed in the appropriate sections below. At this point it is important to mention that the implementation of the BRAINNE algorithm in the Data Mining Tool (DMT) application [4], unfortunately, does not allow for training sets with more than 300 samples. This limitation has created serious problems with some datasets.

4.2.1 Monks dataset

The monks dataset consists of three artificial datasets that share attributes and values. The only difference between the datasets is the rule used for classification of those datasets. The output can be either of two classes: "Monk" and "Not Monk". An instance of each monk dataset consists of six attribute values (A_1,..A_6). Attributes A_3 and A_6 are binary valued attributes with values {1,2}, while attributes A_1 , A_2 and A_4 are discretised attributes with a range of three discrete values, {1,2,3} . Attribute A_5 is also a discretised attribute but with a range of four discrete values, {1,2,3,4}.

The rule for the first of the MONK'S problems (Monk1) is

if $A_1 = A_2$ or $A_5 = 1$ then Monk

The rule for the second of the MONK'S problems (Monk2) is

if EXACTLY TWO OF

$A_1 = 1$, $A_2 = 1$, $A_3 = 1$, $A_4 = 1$, $A_5 = 1$, $A_6 = 1$ then Monk

The rule for the third and last of the MONK'S problems (Monk3) is

if ($A_5 = 3$ and $A_4 = 1$) or ($A_5 \neq 4$ and $A_2 \neq 3$) then Monk

It is important to note that the MONK'S problem was the basis of the first international comparison of learning algorithms [64]. The whole data set consists of 432 instances. The data set was divided into two subsets: a randomly chosen set of 132 instances was used as a validation set and the remaining 300 were used as a training set. At this point it should be noted that BRAINNE failed to extract any rules from the monks databases due to abnormal application termination.

4.2.2 Iris Plants dataset

The iris dataset is probably one of the most used datasets in the machine learning related research. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Each instance consists of four continuous attributes, namely: sepal length, sepal width, petal length and petal width. It is important to notice that one class is linearly separable from the other two. The other two are not linearly separable from each other. There are 150 instances in the whole set. They were divided into two groups: 50 were randomly chosen for the validation set, while the remaining 100 were used as a training set.

4.2.3 Pima Diabetes dataset

This dataset consists of 768 instances. Several constraints were placed on the selection of these instances from a larger medical database [52]. In particular, all patients here are females of at least 21 years old and have a Pima Indian heritage. Each instance consists of eight continuous attributes, namely: number of times pregnant, plasma glucose concentration in an oral glucose tolerance test, diastolic blood pressure (mm Hg), triceps skin fold thickness (mm), 2-Hour serum insulin (mu U/ml), body mass index (weight in kg/(height in m)^2), diabetes pedigree function and age (years). The output can be one of two classes: "Diagnosed with diabetes" or "Not diagnosed with diabetes". The data set was divided into two groups – 268 were randomly chosen for the validation set, while the remaining 500 were used as a training set. All algorithms, except for BRAINNE, used the above-mentioned division. In the case of BRAINNE, this was not possible because of the limitation noted under section 4.2. In case of BRAINNE, the training set was 240 randomly selected samples and the validation set was 60 samples.

4.3 Results

Results presented here were obtained from data collected by executing 30 runs of each algorithm. Each of the 30 runs was using different splits for training and cross-validation. Default parameter values were used for all algorithms evaluated, except BGP [57]. Confidence intervals were calculated in order to predict the performance of algorithms over infinite number of runs. The 95% confidence intervals were calculated as:

$$\bar{a} \pm 1.96\sqrt{\frac{\bar{a}\,(1-\bar{a})}{n}} \qquad (4.1)$$

where $\bar{a}$ is the average value of the performance criterion under consideration for n runs.

4.3.1 Accuracy of Extracted Rules

Tables 4.1 and 4.2 show the mean accuracy on the training and validation sets over 30 runs for each algorithm, respectively. The CN2 algorithm consistently has the highest score for each of the tasks, which can be explained by the large number of rules, compared to other algorithms, that reach good accuracy with much less rules (refer to table 4.3). The BRAINNE algorithm, as implemented in DMT [4], could not run on any of the original Monk problems. The DMT application crashed repeatedly. Even a modified Monk dataset, when using a reduced number of possible attribute values for attribute A_1, i.e. with two attribute values instead of three, did not perform well with the implementation of BRAINNE algorithm of DMT. Using the modified Monk dataset, the DMT application executed successfully in approximately 70% of run instances, but the results were disappointing. The extracted rules were contradictory in nature. The Monks results for the BRAINNE algorithm are, hence, omitted. The BRAINNE algorithm also performed unsatisfactorily on the Pima diabetes dataset, but these results were, nevertheless, included (the application at least executed every time). An explanation for this can be the reduced number of training samples, due to the restriction on the maximum number of samples allowed by BRAINNE. It is important to note that various parameter settings were tried, but the results were still not satisfactory.

Problem Domain	***BGP***	***CN2***	***C4.5***	***BRAINNE***	***HCS***
Monk1	0.994 ± 0.026	1.000 ± 0.000	0.999 ± 0.008	**N/A**	1.000 ± 0.000
Monk2	0.715 ± 0.161	0.992 ± 0.030	0.769 ± 0.150	**N/A**	0.813 ± 0.139
Monk3	0.934	1.000	0.951	**N/A**	0.758
Iris	0.967 ± 0.064	0.987 ± 0.040	0.982 ± 0.047	0.944 ± 0.082	0.974 ± 0.057
Pima	0.766 ± 0.152	0.887 ± 0.113	0.855 ± 0.126	0.482 ± 0.179	0.877 ± 0.117

Table 4.1: Accuracy on Training Sets

HCS performed reasonably well, except in the case of the Pima diabetes dataset. The explanation for this is simple. All the values in the Pima diabetes set are continuous and they were discretised using only 6 intervals to encode the full range of attribute values. It is interesting that the same default value for the parameter that defines the number of intervals was found to produce excellent results for the Iris dataset, which also consists of just continuous valued attributes. Again, the explanation is simple. The range of the values of the attributes in the Iris dataset is much lower than the range of the values of attributes in the Pima dataset, and the class boundaries of the Iris problem are well defined.

Problem Domain	***BGP***	***CN2***	***C4.5***	***BRAINNE***	***HCS***
Monk1	0.993 ± 0.029	1.000 ± 0.000	1.000 ± 0.000	**N/A**	1.000 ± 0.000
Monk2	0.684 ± 0.166	0.626 ± 0.173	0.635 ± 0.172	**N/A**	0.789 ± 0.145
Monk3	0.972	0.907	0.963	**N/A**	0.631
Iris	0.941 ± 0.085	0.943 ± 0.083	0.945 ± 0.082	0.923 ± 0.049	0.925 ± 0.096
Pima	0.725 ± 0.160	0.739 ± 0.157	0.734 ± 0.158	0.391 ± 0.175	0.682 ± 0.167

Table 4.2: Accuracy on Validation Sets

All the algorithms executed on different hardware (although all were Intel PC based) and software platforms, so accurate comparison of the execution time is not possible. However, an observation is given. The HCS algorithm had a typical execution time on all problem datasets of less than a second. The execution time of CN2 and C4.5 algorithms were equally fast. The other two algorithms, depending on the dataset, required an execution time of up to 200 times longer. The longer execution time for these two algorithms is attributed to the fact that BRAINNE firstly trains an ANN and this is done over multiple iterations, while BGP also evolves an optimal specimen over multiple generations, where each population consists of a number of decision trees.

4.3.2 Comprehensibility of Extracted Rules

In table 4.3, the average number of extracted rules is listed. This comparison is not very favourable for the HCS algorithm, especially in the case of continuous values, because HCS, in its current form, does not support relational operators. All other compared algorithms support relational operators. However, results for HCS are included, although it is a comparison of two very different categories of rules. The extension of HCS to include relational operators would be fairly simple, and it is discussed in the section dealing with suggested future research, but it goes against the basic idea of the HCS, which is a simple, fast, pattern matching algorithm. Implementation of some kind of notation to recognise the relational operators would drastically reduce the number of rules. Another interesting fact is that the number of the rules extracted by the CN2 algorithm for the Monk2 dataset is very high, showing a sign of overfitting. This is confirmed in tables 4.1 and 4.2, where CN2 performs exceptionally well on the training set, while on the validation set it performs significantly worse.

Problem Domain	***BGP***	***CN2***	***C4.5***	***BRAINNE***	***HCS***
Monk1	4.70	18.00	21.50	**N/A**	4.00*
Monk2	6.00	122.80	13.90	**N/A**	61.13*
Monk3	3.00	22.00	12.00	**N/A**	102.70*
Iris	3.73	5.33	4.10	5.00	88.16*
Pima	3.70	35.80	12.73	12.72	280.30*

Table 4.3: Average Number of Extracted Rules

(HCS results marked * are results using only the equality operator)

When comparing the number of conditions per rule, the algorithms compared perform more or less in the same manner (refer to n table 4.4.).

Problem Domain	***BGP***	***CN2***	***C4.5***	***BRAINNE***	***HCS***
Monk1	2.22	2.37	2.73	**N/A**	1.75
Monk2	2.96	4.53	3.01	**N/A**	4.23
Monk3	1.67	2.17	2.77	**N/A**	3.89
Iris	2.02	1.64	1.60	3.71	2.13
Pima	1.97	2.92	3.90	5.39	2.44

Table 4.4: Average Number of Conditions per Rule

Again, the BRAINNE algorithm demonstrates the worst performance. The HCS algorithm, as stated previously, does not support relational operators, but the results obtained are applicable. The reason is that the number of conditions per rule is independent of any operator used in a condition.

The BRAINNE algorithm failed on the Monks datasets, but to illustrate the connectionist based rule extraction approach, the results obtained by using another connectionist based rule extraction algorithm, namely ANNSER [69, 70], are summarised next. The results are taken from the research done by Viktor et al [68]. At this point it is important to note that an exact comparison is not possible due to different split between training and validation sets, as well as the different number of runs used for each datasets. The purpose is not to compare, but to illustrate the applicability of the connectionist based rule extraction approach to Monks problems.

Problem Domain	***Accuracy on Validation Set***	***Number of Extracted Rules***
Monk1	0.970	4
Monk2	0.633	50.75
Monk3	0.938	22.67

Table 4.5: ANSSER Results on Monks Datasets

Unfortunately, results for the number of conditions per rule or accuracy on training set were not reported.

4.3.3 Conclusions

In this chapter, a comparison was made between different algorithms, using well-known datasets. All algorithms, with the exception of BRAINNE, performed satisfactorily. BRAINNE, as implemented in DMT [4], did not execute on the Monks datasets, and underperformed on the Pima diabetes dataset. At this stage it is not clear if the performance of the BRAINNE algorithm was inadequate because of the implementation specifics or because the algorithm is flawed. The HCS algorithm faired well in most comparisons, except on the number of extracted rules. HCS could be further improved by simple modifications, which are presented, together with the conclusion, in the next chapter.

Chapter 5

Discussion

In this chapter, a brief overview of what the main objective of this thesis was, is presented, followed with an evaluation of what was achieved. The remainder of this chapter consists of a discussion of suggested future research.

5.1 Conclusion

The main objective of this thesis was to propose a new knowledge discovery algorithm, HCS. A secondary objective was to present and compare different, artificial intelligence based approaches to the knowledge extraction problem. A representative from each of the three main classes of knowledge extraction algorithms was chosen for comparison with the HCS, namely: BRAINNE, BGP, C4.5 and CN2. The comparison was done using well-known benchmark datasets, namely: Monks datasets, Iris dataset and the Pima diabetes dataset. The algorithms were judged according to the accuracy of the extracted rules, both in terms of training error and generalisation, the number of rules per dataset and the number of antecedents per rule.

The BRAINNE algorithm, as implemented in the DMT application, performed extremely badly and some of the results obtained by BRAINNE had to be discarded because the extracted rule sets were contradictory. HCS, CN2, C4.5 and BGP performed well, extracting clean, crisp rule sets. The main idea behind HCS was simplicity, and that lead, as expected, to a very fast rule extraction algorithm. CN2 and C4.5 executed equally fast, while BGP and BRAINNE needed much longer execution time. All algorithms, except BGP, which was fine-tuned, used their default parameter values. The HCS performed satisfactorily, and because of its simplicity, it can serve as an excellent foundation for further development. Some of the ideas for future research are presented in the next section.

5.2 Suggestions for Future Research

HCS is a very simple algorithm, and as such offers numerous possibilities for expansion. Some of them are outlined below:

- **Parallelism**

  The HCS algorithm could be, with modification, easily executed in parallel. The modification would be simple enough. For example, when the algorithm perform rule expansion, it would be a simple task to spawn a new process that can execute in parallel for each of the candidate rules. As an effect, execution time would be reduced significantly. This saving, in turn, can be used for finer discretisation of continuous valued attributes, resulting in greater accuracy of the rules extracted.

- **Hardware Embedding**

  Because of its simple pattern matching nature, the HCS algorithm could be relatively easy implemented, as a whole or only its pattern matching part, on a hardware level using simple logical circuits. The implementation of the HCS algorithm on a hardware level would increase speed tremendously, making it a prime candidate for real time autonomous systems that require fast rule extraction and classification subsystems. It can be employed even as a control system, with the ability to learn, for control of real time autonomous systems.

- **Different Approach to Handling of Continuous Values**

  The HCS algorithm uses a very simple discretisation technique when handling continuous values. Although the technique performed satisfactorily on most of the datasets used in this thesis, it could be improved to include criteria such as the standard deviation and the probability distribution when considering the interval boundaries. This improvement should increase overall performance of the whole algorithm whenever continuous valued attributes are present in a dataset.

▪ **Different Rule Quality Measurement**

The HCS algorithm uses a simple quality measurement when making a pruning decision. The method used does not take into account the number of samples covered by a rule. This can be easily rectified using a different rule evaluation criterion, for example, entropy as used in CN2 algorithm. The computational overhead would be marginal, and pruning would be more effective. For example, assume two rules at an early stage of expansion:

Rule A: ###1## and Rule B: ###2##

with the same observed accuracy of 20%. However rule A covers 1 sample, and a rule B covers 10 samples. Assume a pruning threshold of 20%. It would make sense to try to keep rule B, because, potentially, it is more significant than rule A. That could be achieved by modifying the pruning criteria to include entropy.

▪ **Implementation of Relational Operators**

As currently implemented, the HCS algorithm uses only the equality operator in the condition part of a rule. This is a heritage of the simple, pattern matching origins of the algorithm. The pattern matching approach works well when handling binary and discrete valued attributes, but in the case of continuous valued attributes, it creates a few problems. The main problem is that for a range of values, for which the rule holds, HCS creates a number of rules, depending on the size of the intervals used for discretisation of a continuous value. The modification of the HCS algorithm should not be too difficult, and it is envisaged that it would take a form of post-processing. Once the rules are extracted, rules could be processed and then the intervals, if applicable, could be combined to form a range of rules. To illustrate the method envisaged, consider the following example:

A sample consists of four attributes. The range of the values for the second attribute is from 1 to 6 and the interval size is 1. Consider the set of extracted rules for class A: { #2##, #3##, #4##, #5## }. The rules are read as:

If $A_2 = 2$ then ClassA,

If $A_2 = 3$ then ClassA,

If $A_2 = 4$ then ClassA,

If $A_2 = 5$ then ClassA,

(Condition $A_2 = 2$ reads as: if the value of attribute A_2 of a sample is between 2 and 3 (see section 3.6) then the sample belongs to class A.)

All the rules from our example can be combined using operational operators in one rule:

If $A_2 > 2$ then ClassA

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

This modification should dramatically reduce the number of extracted rules extracted by the HCS algorithm, and make the comparison with other algorithms more favourable when considering the number of extracted rules (see table 4.3).

To sum, although in its basic form the HCS algorithm performs satisfactorily, it can be easily expanded and definitely warrants further research. Also further empirical analysis is needed to confirm the results and conclusions of this thesis.

APPENDIX A Artificial Neural Networks Notation

Figure A.1 presents the ANN related notation used in this thesis. The notation used was copied from the course material for the Honours degree at University of Pretoria [21], with the permission of the author.

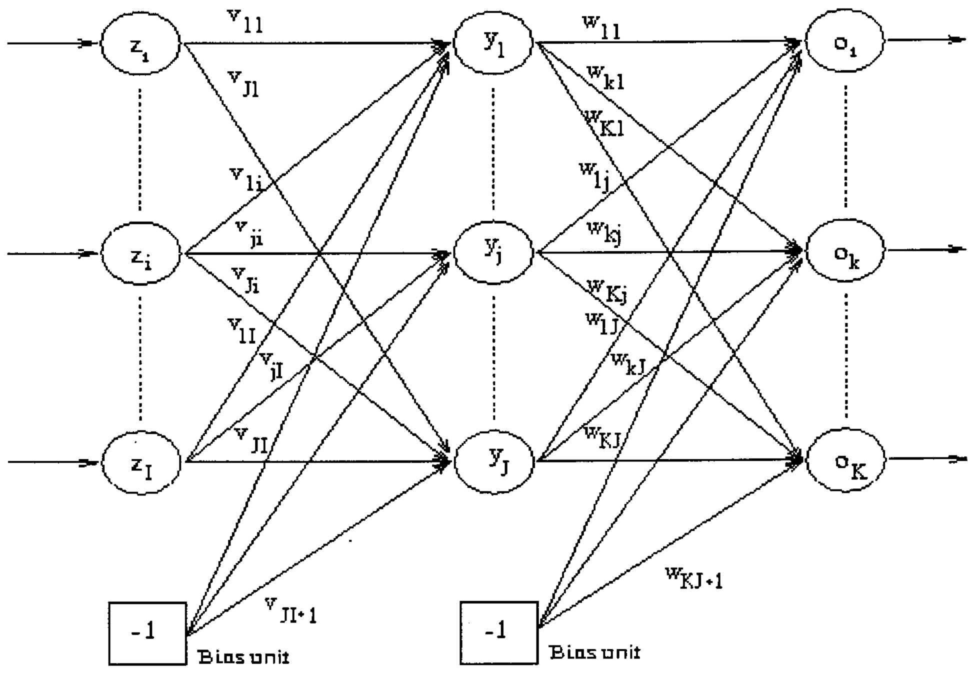


Figure A.1: Typical ANN

Explanation of the figure A.1:

z_i	the ith input unit
y_j	the jth hidden unit
o_k	the kth output unit
v_{ji}	the weight between hidden unit y_j and input z_i
w_{kj}	the weight between output unit o_k and hidden unit y_j

APPENDIX B Bibliography

1. Adriaans P., Zantinge P.; *Data Mining* ; Addison-Wesley ; 1996. (ISBN 0-201-40380-3)

2. Afifi A., Azen S.; *Statistical Analysis: A Computer Oriented Approach* ; Acad Press, New York; 1979.

3. Aikins J.S.; *Prototypical knowledge for Expert Systems* ; Artificial Intelligence; Vol 20; pp163-210.

4. AKT Systems PTY.LTD, Data Mining Tool, *User Manual,* Version 4.0

5. Andrews R., Diederich J., Ticke A.B; *A Survey And Critique of Techniques For Extracting Rules From Trained Artificial Neural Networks* ; Neurocomputing Research Centre ; Brisbane, Australia; 1995.

6. Andrews R., Geva S.; *Rule extraction from a constrained error back propagation MLP* ; Proceedings 5th Australian Conference on Neural Networks; pp9-12; 1994.

7. Barr A., Feigenbaum E.A. (eds): *The Handbook of Artificial Intelligence* ; Vol 1; William Kaufmann, Inc; Los Altos ; 1981.

8. Battiti R.; *First- and Second-Order Method for Learning: Between Steepest Descent and Newton's Method,* Neural Computation, Vol 4, pp 141-166; 1992.

9. Bishop, C.M.; *Neural networks for pattern recognition*; Oxford University Press; 1995.

10. Bratko I.; *Prolog Programming for Artificial Intelligence* ; Addison-Wesley, Wokingham, England; 1986.

11. Buhmann J.; *Data Clustering and Learning* ; in Arbib M. (ed): Handbook of Brain Theory and Neural Networks ; Bradfort Books/MIT Press; 1995.

12. Buhmann J.M.; *Stochastic Algorithms for Exploratory Data Analysis: Data Clustering and Data Visualization* ; in Jordan M. (ed): Learning in Graphical Models; Kluwer Academics; 1997.

13. Cestnik B., Kononenko I., Bratko I.; *ASSISTANT-86 A knowledge elicitation tool for sophisticated users* ; Progress in machine learning ; Bled Yugoslavia; 1987.

14. Clark P., Niblett R.; *The CN2 Induction Algorithm* ; Machine Learning, Vol 3; pp 261 – 284; 1989.

15. Craven M W., Shavlik J W.; *Using sampling and queries to extract rules from trained neural networks* ; Machine Learning: Proceedings of the Eleventh International Conference; 1994.

16. Cybenko G. ; *Approximation by Superposition of a Sigmoidal Function* ; Mathematical Control Signals Systems 2, pp 303, 1989.

17. Dayhoff J.E.; *Neural Network Architectures: An Introduction* ; Van Nostrand Reinhold; 1990.

18. De Jong K. A., Spears W. M.; *Learning Concept Classification Rules using Genetic Algorithms* ; Proceedings of the 12th International Joint Conference on Artificial Intelligence; pp651-656; 1991.

19. De Jong K. A., Spears W. M., Gordon D.F.; *Using genetic algorithms for concept learning.* ; Machine Learning Journal, 13(2-3): November-December; pp 161-188; 1993.

20. Durbin R., Rumelhar D.E.; *Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks.*

21. Engelbrecht A. P.; *Computational Intelligence, An Introduction* ; Graduate Course, Department of Computer Science, University of Pretoria, South Africa, 2000

22. Engelbrecht A. P.; *Data Generation Using Sensitivity Analysis* ; International Symposium on Computational Intelligence, Kosice, Slovakia; 2000.

23. Engelbrecht A. P., Cloete I,; *A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks* ; IEEE International Conference in Neural Networks, Washington, Vol 2; pp 1274-1277; 1996.

24. Erman L.D., London P.E., Fickas S.F.; *The design and example use of HEARSAY-III* ; Proceedings of the National Conference on Artificial Intelligence, pp 409-415; 1983.

25. Flockhart I. W.; *GA-MINER Parallel Data Mining with Hierarchical Genetic Algorithms* ; Report; University of Edinburgh; 1995.

26. Forrest S., Mitchell M.; *Relative building-block fitness and the building-block hypothesis,* ; in D. Whitley (ed.): Foundations of Genetic Algorithms 2; Morgan Kaufmann, San Mateo, CA;1996.

27. Fu L. M.; *Neural Networks in Computer Intelligence*; McGraw Hill; 1994.

28. Giles C. L., Kuhn G. M., Williams R. J.; *Dynamic Recurrent Neural Networks: Theory and Applications* ; IEEE Transactions on Neural Networks, Vol 5, Num 2; 1994.

29. Goldberg D. E., *Genetic Algorithms in Search, Optimisation and Machine Learning'* ; Addison-Wesley; 1989.

30. Gorodkin J., Hansen L.K., Krogh A., Svarer A.; *A Quantitative Study of Pruning by Optimal Brain Damage* ; International Journal of Neural Systems, Vol 4, pp 159-169; 1993.

31. Hassibi B., Stork D.G.; *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon* ; in Lee Giles C., Hanson S.J., Cowan J.D.(eds): Advances in Neural Information Processing Systems, Vol 5, pp 164-171. 1993.

32. Hassibi B., Stork D.G, Wol G.; *Optimal Brain Surgeon: Extensions and*

Performance Comparisons ; in Cowan J.D., Tesauro G., Alspector J.(eds): Advances in Neural Information Processing Systems, Vol 6, pp 263-270; 1994.

33. Hayes-Roth B.; *Blackboard architecture for control.* Artificial Intelligence, chapter 26, pp 251-321.

34. Hebb D.O.; *The Organization of Behavior*; John Wiley & Sons; New York; 1949.

35. Held M., Puzicha J., Buhmann J.M.; *Visualizing Group Structure* ; Advances in Neural Information Processing Systems 11 (NIPS'98), pp. 452-458, MIT Press, 1999.

36. Holland J.H.; *Adaptation in natural and artificial systems* ; University of Michigan Press; 1975.

37. Holland J.H.; *Outline for a logical theory of adaptive systems* ; Journal of the Association for Computing Machinery, Vol3, pp 297-314; 1962.

38. Holland J.H., Reitman J.S.; *Cognitive Systems Based on Adaptive Algorithms* ; in Waterman D.A., Hayey-Roth F. (eds.): Pattern Directed Inference Systems, pp. 313 –329; Academic Press, New York; 1978.

39. Hunt E.B., Marin J., Stone P.J.; *Experiments in Induction* ; Academic Press; New York; 1966.

40. Hussain A., Soraghan J.J., Durbani T.S.; *A new neural network for non-linear time series modelling* ; NeuroVest Journal, January, pp 16-26; 1997.

41. Jackson P.; *Introduction to Expert Systems* ; Introduction to Expert Systems; Addison–Wesley ; 1990. (ISBN 0-201-17578-9)

42. Janikow C. Z.; *A knowledge-intensive genetic algorithm for supervised learning.* Machine Learning, 13(2-3): November-December ; pp 189-228, 1993.

43. Johnson M.V., Hayes-Roth B.; *Integrating Diverse Reasoning Methods in the BB1 Blackboard Control Architecture*, Technical Report No KSL 86-76. Knowledge Systems Laboratory, Stanford University.

44. Khosla R., Dillon T.; *Engineering Intelligent Hybrid Multi-Agent* ; Kluwer Academic Publishers; 1997. (ISBN 0-7923-9982-X)

45. Kohonen T., Barna G., Chrisley R.; *Statistical pattern recognition with neural networks: benchmarking studies.* ; Proceedings of IEEE International Conference on Neural Network; pp 61-67; San Diego; 1988.

46. Le Cun Y., Denker J.S., Solla S.A.; *Optimal Brain Damage* ; in Touretzky D. (ed): Advances in Neural Information Processing Systems, Vol 2, 1990.

47. Lehmann F.(ed): *Semantic Networks* ; Part of International Series in Modern Applied Mathematics and Computer Science, Vol 24; 1992. (ISBN: 0-08-042012-5)

48. Masters T.; *Practical Neural Network Recipes in C++,* Academic Press, 1990.

49. Michalsk R.S.; *Discovering classification rules using variable valued logic system* ; Third International Joint Conference on Artificial Intelligence, Vol 1, pp 162-172; 1973.

50. Minsky M., Papert S.; *Perceptrons, An Introduction to Computational Geometry* ; MIT Press; Cambridge, MA; 1969.

51. Moller M.F.; *A scaled conjugate gradient algorithm for fast supervised learning* ; Neural Networks, Vol 6, pp 525—533; 1993.

52. National Institute of Diabetes and Digestive and Kidney Diseases, Pima diabetes set extracted by Sigillito V. Research Center, RMI Group Leader, Applied Physics Laboratory, The Johns Hopkins University

53. Quinlan J.R.; *Induction of deciscion trees* ; Machine learning, Vol1, pp 81 –106 ; Kluwer Academic Publishers; 1986.

54. Quinlan J.R.; *C4.5: Programs for Machine Learning* ; Morgan Kaufmann; 1992.

55. Rechenberg I.; *Evolution Strategy* in Zurada, J.M. Marks II R., Robinson C. (eds): Computational Intelligence – Imitating Life ; pp 147 – 159; IEEE Press; 1994.

56. Rodich D., Engelbrecht A.P.; *A Hybrid Exhaustive and Heuristic Rule Extraction Approach*; In Proceedings on the International Conference on Artificial Intelligence ICAI99, Vol 1, pp 25-29; 1999.

57. Rouwhorst S. E.; *Searching the Forest: A Building Block Approach to Genetic Programming for Classification Problems in Data Mining* ; Masters Thesis, Vrije Universiteit, Amsterdam, 2000.

58. Rouwhorst S.E., Engelbrecht A.P.; *Searching the Forest: Using Decision Trees as Building Blocks for Evolutionary Search in Classification Databases*; Proceedings on the 2000 Congress on Evolutionary Computation, CEC2000, Vol. 1, pp 633-638; 2000.

59. Rosenblatt F.; *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* ; Spartan books; Washington D.C; 1962.

60. RuleQuest (Web Site) www.rulequest.com accessed on 03/12/1999

61. Salford Systems (Web Site) www.salford-systems.com accessed on 21/11/1999

62. Schmidt A., Bandariasted Z.; *Modularity a Concept for new Neural Network Architectures* ; International Conference Computer Systems and Applications – CSA '98; Irbid, Jordan ; 1998.

63. Sestito S., Dillon TS.; *Automated Knowledge Acquisition* ; Prentice Hall; 1994.

64. Thrun S.B., Bala J., Bloedorn E., Bratko I., Cestnik J., De Jong K., Dzeroski S., Fahlman S.E., Fisher D., Hamann R., Kaufman K., Keller S., Kononenko I., Kreuziger J., Michalski R.S., Mitchell T., Pachowicz P., Reich Y., Vafaie H., Van de Welde W., Wenzel W., Wnek J., Zhang J.; *The MONK's Problems - A Performance Comparison of Different Learning algorithms*; Technical Report CS-CMU-91-197, Carnegie Mellon University.

65. Towell G., Shavlik J.; *The extraction of refined rules from knowledge based neural networks* ; Machine Learning ; Vol 131 ; pp71-101; 1993.

66. UCI Machine Learning Repository: http://www.ics.uci.edu/~mlearn/MLRepository.html accessed on 14/08/1998.

67. University of Maryland (Web site) http://ctsm.umd.edu/v&v/s3.htm, accessed on 01/02/2000.

68. Viktor H.L., Cloete I.; *Improved Generalisation using Cooperative Learning and Rule Extraction* ; IEEE International Joint Conference on Neural Networks; Washington DC ; 1999.

69. Viktor H.L., Engelbrecht A.P., Cloete I.; *Incorporating Rule Extraction from ANNs into a Cooperative Learning Environment* ; Neural Networks & Their Applications, pp 336-391; Marseilles, France; 1995.

70. Viktor H.L., Engelbrecht A.P., Cloete I.; *Reduction of Symbolic Rules from Artificial Neural Networks using Sensitivity Analysis* ; Proceedings of IEEE International Conference on Neural Networks ICNN'95, pp 1788-1793 ; Perth, Australia; 1995.

71. Widrow B., Winter., Baxter.; *Learning phenomena in layered neural networks*. First 1st Int. Conference Neural Nets, San Diego, volume 2, pp 411. I have this.This gives a nice description of training linear units and the ideas of linear separability; 1987.

72. Williams P.M.; *Bayesian Regularization and Pruning Using a Laplace Prior, Neural Computation* ; Vol 7, pp 117-143; 1995.

73. WizWhy for Windows User Guide. Wizsoft; 1996.

74. Zurada J.M., Malinowski A., Cloete A.; *Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network* ; IEEE International Symposium on Circuits and Systems; London, May 30 - June 3; 1994.

75. *Lessons in Neural Network Training: Overfitting May be Harder than Expected,* ; Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI-97; pp 540-545; AAAI Press, Menlo Park; 1997.