Inducing and Combining Decision Structures for Expert Systems

A Thesis Submitted for the Degree of
Doctor of Philosophy
of
The Australian National University

Graham John Williams
January 1990

This thesis describes the original work of the author. Where this work is based on the work of others, this is clearly indicated in the text.

Graham J. Williams

Dedication

This thesis is dedicated to my wife Catharina, my son Sean, and our Anita. Over the years they have had to endure the excesses of the self indulgence that is the pursuit of learning. And yet, in times of difficulty they always stood by me. Their love and support has known no bounds. And now, another chapter of my life concludes.

"Herman Helmholtz compared himself with a mountain climber who, after a long and circuitous climb, eventually reaches the summit. There he is able to see clearly the route he should have taken."

(New Scientist, 15 December 1983, Number 1388, page 821.)

Acknowledgments

My sincere thanks go to my supervisor, Professor Robin Stanton of the Department of Computer Science, Australian National University, who over the time of this thesis has always supported and stood by me. His encouragement, insights, and friendship were of tremendous assistance.

My advisors deserve many thanks for their support and assistance in the development and deployment of my ideas. Thanks to Dr. Richard Davis, Dr. Malcolm Newey, and Hugh Mackenzie. Richard provided much assistance in the practical aspects of this research, with many insights into the application of expert systems technology to real world problems.

I have also benefited from discussions with numerous fellow researchers, including David Harper, David Morley, Jason Catlett, and Jon Patrick.

Many more friends and colleagues than I could hope to list individually have contributed much to the completion of this work. Thanks to them all.

And a special thanks to my family, without whom this would not have been possible.

Abstract

Decisions are central to our daily existence. Every activity requires us to make decisions, many subconsciously. Knowledge, defined as an acquaintance with facts, truths, or principles (Delbridge, 1982), is the key to correct decision making. Its representation and use by machine has been a major goal throughout the history of computing machinery. Research in the discipline of Artificial Intelligence (AI) explicitly investigates ways in which knowledge can be effectively represented and employed by computers in order to make intelligent and human-appreciable decisions. Knowledge-based expert systems (KBESs) are a family of successful, practical systems arising from AI research. These systems structure knowledge (as decision structures) in such a way that it can be efficiently employed to make decisions, and yet is easily understandable by humans.

Significant research problems relating to KBESs remain. One such problem area falls under the general categories of machine learning and knowledge acquisition. It is generally agreed that learning is one of the most important components of intelligence. The research reported here focuses upon the acquisition of decision structures for use by KBESs.

This thesis takes a well-established, practical tool for knowledge acquisition as its basis. Experiments are described, pinpointing some of the limitations of the tool. A new approach is then developed which builds upon this tool, introducing the idea of combining decision structures.

1 Introduction

This thesis is about learning. To learn is to acquire knowledge, or to gain skills, by study, instruction, or experience (Delbridge, 1982). Knowledge can be defined as an acquaintance with facts, truths, or principles, and thus to learn is to become ever more familiar with these facts, truths, or principles. In addition to this, the skill of applying knowledge appropriately must also improve. A system which can improve its performance at a given task over time is a system that can learn (Forsyth and Rada, 1986). Improvement is generally effected, at least in the context of computing machinery, by modifying (by, for example, adding to) some store of knowledge. Whilst there exists philosophical debate over the necessity for there to be performance improvement (Gaines and Boose, 1988b, Section 3.2), such a definition captures the intent of most computer-based learning systems and will suffice here. Performance, in the context of knowledge-based expert systems, is usually defined as the making of accurate decisions in an efficient manner (Langley, 1989).

In this thesis learning is considered in the context of knowledge-based expert systems. This chapter begins with a discussion of what "knowledge" means in the context of the work presented here. With this foundation, the chapter introduces knowledge-based expert systems, discussing their deficiencies, focusing upon those deficiencies addressed in this thesis. A general overview of knowledge acquisition and machine learning follows.

1.1 KNOWLEDGE

Knowledge, unquestionably, is a difficult concept to define, or even describe. Knowledge can be "a collection of specialized facts, procedures, and judgment rules" (Turban, 1990) or the "information available to the individual from internal or external sources about relationships and rules that describe organised human activities" (Hertz, 1990). For the purposes of this thesis four types of knowledge are identified: Terminological Knowledge, Inferential Knowledge, Situational Knowledge, and Meta-Knowledge.

Terminological knowledge is basic definitional knowledge. It includes the nouns, or as is popular today, the objects, of the language. It includes the verbs of the language, describing how actions are performed upon or by objects. It includes all the words of the language, and their meanings, and is the type of knowledge found in a dictionary. Such knowledge provides the blocks upon which to build the other types of knowledge.

Inferential knowledge describes relationships between objects. A relationship may be causal, or it might be one of similarity, or it might describe some set (as in super set or subset) relationship. Such knowledge is often of a general nature, expressed in terms of classes of objects rather than in terms of particular objects. Inferential knowledge also covers the rules of behaviour (often called heuristics), as well as the knowledge which encompasses a description of processes.

The knowledge which experts employ to solve problems is usually regarded as inferential in nature. Such knowledge is classified further by Klein and Methlie (1990, page 30) into theoretical knowledge (the known facts of the domain) and experimental knowledge (the ill-defined domain knowledge, usually referred to as heuristic knowledge or rules-of-thumb). Heuristic knowledge is that knowledge gained by an apprentice working closely with a Master.

Situational knowledge records information about particular instances of objects. This is the type of knowledge that is found in database systems. It

is dormant, rather than active knowledge. (Inferential knowledge, on the other hand, can be thought of as being active knowledge.)

Meta-knowledge is that which guides the deployment of other forms of knowledge. John McDermott describes this as the knowledge of "how to bring relevant knowledge to bear at the right time" (Mostow, 1985). Meta-knowledge is the knowledge which allows us to reason.

The endeavour of Artificial Intelligence research is the study of knowledge—in particular, its representation and use. Artificial Intelligence is often characterised as automated problem solving. Problem solving is the task of bringing the appropriate inferential knowledge, under the guidance of some meta-knowledge, to bear upon the appropriate situational knowledge.

1.2 KNOWLEDGE-BASED EXPERT SYSTEMS

A system is an interrelated collection of parts. In an expert system, these parts interrelate in such a way so as to bring knowledge to bear on a problem to provide a solution. There are many parts to expertise: learning; reasoning at different levels of abstraction; having different perspectives of the same problem; knowing when to break the rules; explaining the reasoning. No computer-based system has yet achieved such an integration of all parts. There are no truly expert computer-based systems. However, there is a class of practical, and indeed commercially successful, systems known as expert systems.

Such expert systems can be characterised as systems which contain a store of mainly inferential-type knowledge (the **knowledge base** or **decision structures**), using some inferencing mechanism (the **performance element**) in the context of some situational knowledge to make decisions. In other words, an expert system makes decisions based upon its store of knowledge operating upon a set of facts. Of central importance is the store of knowledge. A primary characteristic of expert systems is that this store of knowledge is separate from the mechanisms which are employed to use it.

An **expert system**, in the broadest sense, is any system which attempts to perform some task at the level of a human expert. While computers have traditionally excelled in performing numerically-oriented tasks, symbolically-oriented tasks, as performed by human experts, are now a primary area of attention. Such tasks involve the manipulation of symbols (rather than numbers) to obtain results. Symbols represent objects and concepts, and a result is a symbolic expression of the state of the objects. Expert systems manipulate symbols under the guidance of a symbolic knowledge-base which stores the truths and principles of some domain.

A **knowledge-based expert system** is a computer program in which a performance element operates upon a knowledge-base to make intelligent decisions within the confines of a given situation. A situation can be supplied

interactively by the users of such a system, whilst the system itself identifies the information it needs. The users may ask the system to justify requests made upon them, or even to demonstrate how the conclusions were reached.

A **performance element** is that part of a system involved in determining new facts or beliefs from previous knowledge. "Performance element" and "inference engine" are synonyms.

A knowledge-based expert system can be characterised by its knowledge-base structure, the mechanism it employs to make deductions, and its user feedback capabilities. The knowledge-base contains inferential knowledge encapsulating the "principles" of the domain. It is often represented in the form of rules and may be structured to model the domain. A simple, uniform representation of the knowledge is considered advantageous. Many systems provide a mechanism for representing uncertain or fuzzy information. The performance elements employ a variety of techniques for using the knowledge, with the so-called backward and forward chaining approaches being common.

There are numerous deficiencies with such systems, and the last decade has been witness to an explosion in research directed towards these problems. Deficiencies include the narrow domain of expertise of the systems, leading to the potential for incorrect behaviour in slightly different domains; the difficulty in representing certain types of knowledge; the slow and laborious task of constructing such systems; and their often inadequate explanation capabilities.

1.2.1 A Characterisation

Many different types of systems have been developed, and even post facto labelled, as expert systems. The use of the term "expert system" in this thesis is restricted to those systems characterised by the classic expert systems and their descendants. Systems such as DENDRAL (Lindsay et al., 1980), CASNET (Weiss et al., 1978), MYCIN (Shortliffe, 1976), and PROSPECTOR (Duda, Gaschnig, and Hart, 1979) are widely recognised as exemplar of the first generation of

expert systems. The common characteristics of these systems are summarised below.

Type of Knowledge: Expert systems attempt to capture heuristic knowledge. Such knowledge is usually represented by expert systems in a declarative, rather than procedural, manner. Newell and Simon (1972) have observed that when the expert is forced to put his/her knowledge into words, it is often expressed in terms of a situation implying a particular action to be taken. Thus, many expert systems encapsulate this heuristic knowledge as if-then rules.

Uniform Encoding of Knowledge: Whichever representation is chosen, it is often the case that the knowledge in the system is uniformly encoded as many separate units. This has allowed simple, yet powerful tools to be developed for managing all aspects of the knowledge.

Uncertainty in the Knowledge: Domain knowledge is often expressed in terms of uncertainties. This may be expressed by the domain expert as "If we are confident that x, y and z are true, then there is some evidence for belief in w". Various methods for handling such information have been developed (O'Neill (1986) provides an excellent review).

Structure of an Expert System: Moving away from the knowledge itself, expert systems are structured so that the performance element is generally a separate entity to the knowledge base. This, in theory, allows a domain-independent performance element to be developed, and then applied to any suitable knowledge base, adding to the flexibility of the system.

Explanation: A very important feature of expert systems is their explanatory ability. It is important to be able to justify why a particular answer is given. The lines of reasoning that lead to the answer, when made available, can give the user confidence in the answer.

1.2.2 Deficiencies

A large number of expert systems exist today, many of which are in daily commercial use (Michie, 1987). Whilst expert systems have been successful,

there is still room for improvement. The deficiencies of expert systems discussed below represents an amalgam of my own and other researchers' analyses (Williams, 1986).

Declarative Representation: Some time ago Clancey pointed to one of the weaknesses of rule-based systems when he said that rules "can only be read and understood by knowing the specific procedure that will be interpreting them" (Clancey, 1985). Much progress has been made toward the goal of separating the knowledge base from the performance element, but difficulties remain.

Domain of Expertise: A general and long standing criticism of expert systems is that they have too narrow a domain of expertise (Buchanan, 1982). What makes this worse though is that they are unable to recognise problems for which their own knowledge is inapplicable or insufficient (Hart, 1980).

The former is a problem of expectations. Expert systems typically have expertise in only one specific (often very narrow) domain. By developing small, specific systems, we can build a base from which a foundation can be laid, upon which the machinery required to build more general systems can be developed. Enlarging the domain of expertise often requires extending the knowledge base in use, introducing other problems. The CYC project (Lenat and Guha, 1989) is an example of a very ambitious development which attempts to encode large amounts of knowledge addressing such problems.

The problem of inappropriate application is a limitation that may be overcome with the focus on meta-knowledge. Already systems have been developed which are able to reason about their own knowledge bases, providing them with the ability to "know what they know". However, there is still much to be done. Researchers recognised early the importance of this (Buchanan, 1982).

Knowledge Representation: A most important aspect of expert systems which has had much attention is knowledge representation. The knowledge representation scheme employed by a system will influence such things as

the techniques employed for acquiring knowledge, the type of reasoning to be employed, and the type and complexity of explanations that the system can provide. A large number of problems can be classed as knowledge representation problems. These often derive from restrictions enforced by the language for expressing facts and relations and from the degree to which different types of knowledge are hidden.

Uniformity of representation allows relatively simple mechanisms to be employed to maintain large knowledge bases. If-then rules have the further advantage that they can be executed as procedural code and yet viewed as declarative expressions (Davis and King, 1984). However, any uniform scheme, it seems, inherently excludes some forms of knowledge from being naturally represented. Developers have often complained of their inability to represent such things as spatial relationships, time relationships, causality, and physical principles within a restricted representation scheme (Mackenzie, 1984; Weiss and Kulikowski, 1984). Within the rule-based paradigm, for example, simple algorithms have often been massaged into the if-then structure.

Aside from not being able to represent many things, it is often the case that too much is represented only implicitly. This has been a very common observation, and has significance to many other areas, including explanation and knowledge acquisition. Aikins (1983) provides examples of this hidden knowledge for production systems. Both Clancey (1983) and Mackenzie (1984) have made similar observations.

Several types of knowledge are often represented only implicitly. The context in which a rule is applicable can be implicit—often, some of the conditions of a rule test for applicability in the current context while the remaining conditions represent the expertise. The former often represent a type of meta-knowledge (or control knowledge). Control knowledge is also often embedded implicitly in the ordering of the conditions and the importance of the particular ordering is not explicit. The purpose of a rule is also often implicit. Some rules are "control rules", others are "summary rules", etc.

Knowledge Acquisition: Related to the knowledge representation problem is the knowledge acquisition problem. Knowledge acquisition is the process whereby a **knowledge engineer** extracts information from an expert in a particular domain (the **domain expert**). The knowledge engineer converts this information into a form suitable for use by the expert system. Test cases can then be used to exercise the acquired knowledge, and deficiencies (errors) can be referred back to the domain expert.

This form of knowledge acquisition requires considerable work on the part of those involved—the knowledge engineer and the domain expert. It was an early observation that "knowledge acquisition is ... the most limiting 'bottleneck' in the development of modern knowledge-intensive artificial intelligence systems" (Michalski, Carbonell, and Mitchell, 1983). There is a need to automate this task, and much research has focused on this area (Gaines and Boose, 1988a; Gaines and Boose, 1988b).

Weiss and Kulikowski (1984) refer to the closely related problem of adding new types of knowledge to the systems. This requires the ability to dynamically enhance the representation scheme, since full requirements may not be known beforehand, and different types of knowledge may be needed in solving a problem.

Explanation: An important reason for having an explanation facility is that the user must feel able to ask the system why any conclusion was reached, with adequate reasons being given. An explanation will not be required for each conclusion but provides important feedback during the knowledge acquisition phase.

The first generation systems have very basic, and often inadequate, explanation facilities. Buchanan (1982) noted the stylized explanations of a line of reasoning of many systems, whilst Hart (1980) noted that the explication of the reasoning processes are frequently silent on fundamental issues.

The problems with explanations often derive from problems with knowledge representation, many of which have been mentioned above, like the implicit embedding of control knowledge in rules. Mackenzie (1984) identified the need to be able to use contexts and typical situations to explain why certain rules are applicable, which has often been hidden in the rules.

Bramer (1982) also commented upon the problem with explanations when the knowledge base becomes very large. Although each item in the knowledge base may be comprehensible in itself, the overall operation of the system becomes incomprehensible. Expert systems must be able to group items of knowledge together at a suitable level of abstraction to be able to give comprehensible explanations.

Summary

Knowledge (and its representation, acquisition, and use), as the heart of intelligent behaviour, is of utmost importance to the development of intelligent systems. This thesis explores the process of automatically acquiring knowledge.

1.3 LEARNING

Historically, software systems have dealt with a fixed task, or at least with a fixed task environment (Lenat, Hayes-Roth, and Klahr, 1983). Many problems associated with expert systems, but certainly not confined to such systems, stem from this fact. Interest in systems which dynamically adapt themselves to change in their environment—systems that learn—has steadily grown over the past few years.

Computer-based learning algorithms address two of the most important aspects of knowledge-based expert systems—the acquisition of knowledge and the improvement in performance over time. Two corresponding and overlapping fields of research have emerged within the computer-based learning area.

Knowledge Acquisition is a generic name given to the task of building the various knowledge structures to be used in expert systems. Such an activity typically involves a domain expert. Classically, a knowledge engineer interviews the domain expert and translates the expert's knowledge into a form suitable for representation and use by computer. Knowledge acquisition systems assist this knowledge elicitation process, and the term covers the general development of tools and practices which can be employed by knowledge engineers.

Machine Learning covers research into techniques for automatically generating and improving knowledge bases. There are four broad categories of machine learning: inductive learning, analytic learning, genetic algorithms, and connectionist learning algorithms. This thesis deals with inductive learning where the system learns from a collection of examples presented to it.

Knowledge Acquisition and Machine Learning systems can be characterised by the type of knowledge they acquire. Many knowledge acquisition systems target the acquisition of terminological knowledge. Other systems focus on inferential knowledge, whilst some acquire control knowledge. The majority of machine learning systems attempt to learn inferential knowledge.

The usefulness of automating the task of acquiring knowledge was demonstrated by Michalski and Chilausky (1980) in an experimental comparison of manual and automated knowledge elicitation. They found that decisions made by decision structures generated by a learning system were more accurate than those made by the manually elicited decision structures. It was noted that the automatically derived decision structures "were viewed generally quite favorably by experts—with a few exceptions".

1.3.1 Knowledge Acquisition

Knowledge acquisition, knowledge extraction, and knowledge elicitation, are terms that have been used to denote the process of obtaining and formulating knowledge derived from experts. The activity of acquiring knowledge is often referred to as knowledge engineering (Klein and Methlie, 1990), or at least as a key component of the knowledge engineering process (Turban, 1990).

The classical paper on knowledge acquisition (Buchanan et al., 1983) identifies the five stages: identification, conceptualisation, formalisation, implementation, and testing. Identification involves understanding and characterising the problem domain. Conceptualisation explicitly identifies the concepts of the domain, pinpointing the objects of relevance, and the relations amongst them. A formalised structure is then put in place, organising the concepts so as to bridge the gap between the structure of the particular domain, and structures suitable for use in knowledge-based expert systems. A prototype system can then be implemented and refined by testing.

There is a literature dealing with manual methods of knowledge acquisition. Turban (1990, chapter 13) provides a review of such techniques. Many techniques borrow heavily from psychology and include structured interview, protocol analysis, observations of experts, questionnaires, and analysis of documented knowledge.

Tools have also been developed to assist the knowledge engineer in the rather labourious task of knowledge acquisition (Gaines and Boose, 1988a).

These range from sophisticated editors designed for the manipulation of knowledge bases (Abrett and Burstein, 1988; Musen et al., 1988) to dialogue managers which can themselves conduct interviews with the expert (Kitto and Boose, 1988).

Another class of knowledge acquisition systems covers those which construct a prototype knowledge base from a database of example cases. Such systems include decision tree induction and rule induction systems which take, as their input, examples of decisions made by an expert. From these specific instances, general decision structures are induced which reflect the general pattern of decision making embodied in the examples.

Such inductive systems do not eliminate the need for an expert nor for a knowledge engineer. An expert is often required to provide example decisions, and to identify a maximal set of possible attributes that need to be considered in arriving at any decision. The induced decision structures are then only the first step in developing the knowledge-based expert system, typically requiring much refinement.

1.3.2 Machine Learning

The boundary between those systems which fall into the Machine Learning category and those which fall into the Knowledge Acquisition category is fuzzy. In general, machine learning takes the process of automating knowledge acquisition much further. A machine learning system is one which requires minimal, or at best no, direct human assistance.

A strong motivation for machine learning within the context of building knowledge-based systems is the existence of large databases containing information which is used statically (retrieved and updated). Machine learning systems may be viewed as tools which are capable of turning this mass of information into usable knowledge. Many machine learning (and knowledge acquisition) systems do this by summarising the information contained in the database in the form of rules or other knowledge structures.

Inductive learning, of which similarity-based learning is a prime example, begins with a so called training set of examples and builds a generalised description of those examples. Divide and conquer is a common approach. Examples in the training set are usually described in terms of a number of attributes. Given an initial training set, a partition based upon a particular attribute is sought. For each cell of a partition, another attribute is sought to further partition it. This recursive process continues until all examples in a partition are, in some sense, homogeneous. This process describes a tree-like structure, with the internal nodes representing particular attributes, and the links representing the various values that the attribute may attain. The leaf nodes of the tree correspond to decisions.

1.4 MIL: MULTIPLE INDUCTION LEARNING

This thesis deals with the problem of inducing decision structures from examples of decisions. The inductive learning approach, exemplified by the ID3 decision tree induction algorithm (Quinlan, 1986a), provides a basis for this work. Decision tree induction has proved popular as a knowledge acquisition tool. This can be attributed to the simplicity of the divide and conquer technique it employs, and to the numerous successful expert systems which employ knowledge generated using such decision tree induction algorithms (Michie, 1987). Below is summarised the genesis of the MIL algorithm, developing upon the decision tree induction approach.

A study of the application of a decision tree induction algorithm to an agricultural domain was undertaken. The induction algorithm's task was to develop a knowledge base which could be used to predict the viability of grazing cattle in the arid regions of Australia. A series of experiments then considered various aspects of the decision tree induction algorithm, including the choice of attributes and pruning.

A second domain of application was also considered. Experiments were carried out in building decision trees for determining the credit-worthiness of an applicant for a loan. This domain proved to be an interesting complement to the agricultural domain, providing further support for many of the observations.

One of the important observations made from these experiments was that the decision tree induction algorithm was often unable to distinguish between possible choices of partitions at each stage. Implementations of these algorithms have often assumed an implicit ordering on the attributes, and use this ordering when ties occur. Unexpected changes in the resulting decision trees resulted from simply reordering the attribute definitions.

The induction algorithm's inability to always choose between attributes was taken advantage of by allowing multiple decision trees to be induced. These

decision trees can then be merged to produce a single decision structure, in the form of a set of rules.

By building multiple decision trees another problem of many decision tree induction algorithms is redressed. Decision trees require a particular performance element which checks particular attributes and follows the branch associated with its value. The performance element must check the attribute which appears as the root of the tree for every example presented to it. Thus a value must always be known for the root attribute. (The same is true of sets of rules derived directly from a decision tree.) Multiple decision trees, with the possibility of different root nodes, may avoid this problem.

The MIL algorithm resolves and removes any conflicts which arise when two rule sets are combined. It is a tool to be used by the knowledge engineer to provide an initial (or suggested) implementation of the decision structures to be used in an expert system. The conflicts identified by MIL may provide important information to the knowledge engineer, in identifying limitations of the decision structures which have been induced. This system can thus be used as an aid in the knowledge acquisition process.

The problem of combining induced decision trees is a simpler version of the knowledge-base maintenance problem. In particular, one of the tasks of knowledge-base maintenance is the incorporation of new knowledge into an existing knowledge base. The maintenance task must ensure that the knowledge-base remains consistent and conflict-free. This thesis introduces an exploration into this more general case of combining arbitrary sets of rules.

1.5 OVERVIEW OF THE THESIS

This chapter has provided the general context of my research, clarifying, for the purposes of this thesis, the concepts of knowledge, expert systems, and machine learning.

Chapter 2 is an introduction to decision tree induction. A general decision tree induction algorithm is presented, together with particular implementations of this algorithm.

Chapter 3 demonstrates the application of a decision tree induction algorithm to actual learning tasks. Data from a geographic domain and from a financial domain are used in a series of experiments carried out to confirm a number of properties of the decision tree induction algorithm.

Chapter 4 introduces the MIL algorithm, developed as an approach to handling the multiple decision trees produced by a decision tree induction algorithm. The primary task of MIL is to identify conflicting rules, and to resolve these conflicts, whilst attempting to maintain the accuracy and coverage of the knowledge base. A series of experiments in Chapter 4 confirm the effectiveness of this approach.

Chapter 5 discusses alternatives that have either been implemented or considered during the development of the MIL algorithm. Suggestions for future directions are included there.

Chapter 6 summarises the primary results and conclusions of the research described in this thesis.

Appendix A lists the reference material for the thesis, forming an extensive bibliography of the decision tree induction literature.

Appendix B lists my publications each of which has played a role in the development of the work described in this thesis.

2 Decision Tree Induction

Inductive learning systems build decision structures from examples, summarising relationships between the attributes of the examples. Systems which generate such structures for use in decision making require that these examples have decisions (or classifications) associated with them. The family of decision-tree induction algorithms reviewed here accept examples described in terms of a finite, predetermined set of attributes and build decision structures based upon simple attribute-value tests.

Databases suitable for providing the training examples for inductive learning systems are readily available. Financial institutions, for example, maintain databases of customer records which document the history of successful and unsuccessful applications for credit. Universities maintain records of student progress, along with other background information, in addition to a final decision about each student (indicating whether they obtained a degree). And databases containing information about land use are regularly utilised by land use planners. It is the widespread existence of such data that makes learning from examples an attractive proposition: static data can be brought to life.

Such data can be analysed in a variety of ways, from manually scanning for patterns, through to the use of analytical tools such as discriminant analysis, and on to inductive generalisation. These techniques vary in the amount of human effort or intervention required. Studies have found though that decision making systems using knowledge-bases produced by inductive learning algorithms can outperform both discriminant models and human decision making (Messier Jr. and Hansen, 1988).

Decision tree induction systems share a common divide and conquer approach entailing the application of a recursive algorithm to ever smaller sets of training examples. The goal is to search for a decision tree which accurately

and efficiently reflects the decisions recorded for the training examples, and is general enough to be used to make accurate decisions for unseen objects. Some of the algorithms post process decision trees into collections of rules (a form more familiar to the expert systems developer), and carry out pruning.

This chapter reviews the decision tree induction family of systems. Of particular interest is the ID3 algorithm and its successors, which have been found to be useful tools for knowledge engineering.

2.1 THE TERMINOLOGY

Decision tree induction falls into the category of machine learning variously referred to as inductive learning, empirical learning, similarity-based learning, concept learning, and learning from examples. The terminology of decision tree induction is introduced below.

An **object** is a description of an entity and, in particular, of an example. The description consists of a list of the features of the example, represented as an attribute-value list. Such a list is effectively a conjunctive description of the example. An illustration of an object is:

Region 19481:	Soil is of type CC1, Distance to nearest seaport is 836 km, Average weekly winter moisture index is 21%.

This object can be interpreted as representing a geographical region identified as region 19481 and characterised by the specified attribute-value pairs.

An **attribute** describes some feature of an object. An attribute may take on any number of values from the domain of the attribute. In the systems described here attributes are single valued. For the object illustrated above, the attributes are the soil type, the distance to the nearest seaport, and the average weekly winter moisture index. A **categorical attribute** is one which has a finite, unordered, set as its domain. An **integer attribute** is one whose domain is the set of integers. The soil type in the above example is a categorical attribute, while the distance to the seaport and the moisture index are considered to be integer attributes. **Real attributes**, with their domain consisting of real numbers, will not be considered specifically here—results for integer-valued attributes in general also hold for real-valued attributes.

A **decision attribute** is a distinguished attribute associated with each object which identifies the decision class of that object. A decision class simply organises all objects with a common value for the decision attribute into a

single set. Such attributes are usually categorical attributes (in decision tree induction systems). For example, we can associate with the above object a decision attribute called the grazing viability. The values recorded for this attribute indicate the viability for grazing cattle in the particular region.

A **training set**, denoted as **Tr**, is a collection of examples (objects) with known values for the decision attribute. These decision values may be empirically derived or provided by a domain expert. A training set is used by the decision tree induction system to build generalised decision structures.

A **decision tree** is a tree structure consisting of nodes connected by directional branches. A decision tree will be denoted by $\mathcal{T}$. The **root node** of a decision tree is the unique node with branches emanating from it but with no branches pointing to it, and is pictured as the top node of the tree. A **leaf node** of a decision tree is any node with branches pointing to it, but no branches emanating from it. A **trivial decision tree** is one in which the root node is a leaf node. Each non-leaf node of the decision tree is labelled with the name of an attribute. Each branch emanating from a node is labelled with a value for the attribute which labels the node. Each leaf node is labelled with a value for the decision attribute. A decision tree can thus be defined recursively as either a single node labelled with a value for the decision attribute (or Null in the case where no decision can be made), or a node labelled with a non-decision attribute from which a number of branches emanate, each branch leading to another decision tree. Figure 2.1 illustrates the simple structure of a decision tree. A **null decision tree** is a trivial decision tree with the label Null, indicating that no value for the decision attribute can be determined. When illustrating a decision tree, null decision trees are not usually shown. The depth of a decision tree is the number of branches between nodes in any path from the root node of

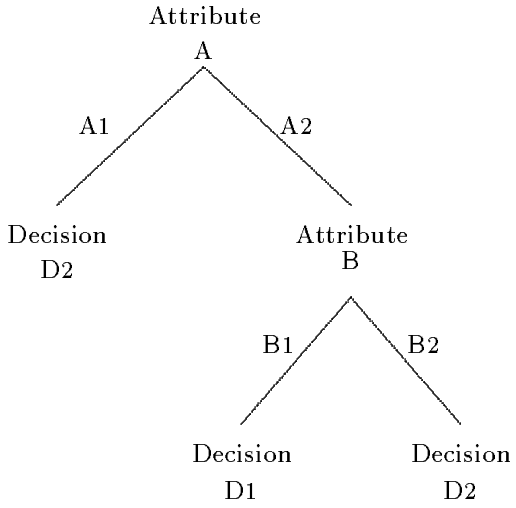


FIGURE 2.1: *A simple decision tree. The root node of this decision tree is labelled with attribute A. The branches emanating from this node correspond to each of the valid values for the attribute A. Leaf nodes correspond to values for the decision attribute, with the possible decision values being D1 and D2.*

the decision tree to a leaf node. The general concept of decision trees is covered quite comprehensively in Moret (1982).

A **performance element** uses such a decision tree to determine a value for the decision attribute for a given object. In its simplest form, this value is found by traversing the decision tree, beginning at the attribute-labelled root node, and following the branch corresponding to the actual value of that attribute as recorded for the object. This traversal continues until a leaf node is reached, whereupon the value labelling the leaf node is returned as the decision.

The **coverage** of a decision tree refers to the ability of a decision tree to give decisions for any object presented to it. Often, a decision tree will not have total coverage. This can be the case when attribute B in Figure 2.1 actually has three values rather than just the two shown, with no branch corresponding to the third value. An object with the value A1 for attribute A and B3 for attribute B is not covered by this decision tree.

A **Tr-consistent** decision tree is a decision tree for which the performance element, when applied to the examples contained in the training set, returns decisions which agree with those recorded in the training set. This definition of consistency says nothing about the ability of the performance element to make correct decisions for objects outside of the training set.

A decision tree is often thought of as representing a **concept**, and the decision tree induction system is referred to as a **concept learning system**—defined as a device for creating a concept corresponding to some partition of a sample of objects which have been classified by a pre-established rule for specifying a class (Hunt, Marin, and Stone, 1966).

2.2 THE DIVIDE AND CONQUER ALGORITHM

The task of the learning algorithm described here is basically one of taking a training set and generating a decision structure which can explain the decisions associated with the objects in that training set. The decision structure so constructed can then be used by a performance element to make decisions about previously unseen objects.

The general algorithm begins with a training set, **Tr**, consisting of a set of objects, each being an example of a domain expert's decision. A number of alternative partitions, S_i, of the training set are considered—each S_i represents an alternative branching pattern from the current node in the developing decision tree. The set of p partitions to be considered will be denoted as

$$\mathbf{S} = \{S_1, S_2, \ldots, S_p\}.$$

Each partition, S_i, consists of a number of **cells**, each cell containing objects from **Tr**. The n cells of a given partition are identified as $C_j,\ j = 1, 2, \ldots, n$.

A best partition, S^*, is chosen from **S** using some **selection criterion**. Such a criterion is often represented as a **cost function** which assigns a cost to each partition in **S**. Such cost functions are typically dependent upon the C_j's. The partition corresponding to the minimum value of this cost function is chosen.

The third step of the general algorithm involves constructing a **discriminating description** for each cell of S^*. Such a description of a cell categorises each object in that cell, and no other object in any other cell of S^*. These discriminating descriptions become the branch labels in the decision tree.

For the typical decision tree induction algorithm, a partition is based upon the values of a particular attribute. The objects in each cell have a common value for the chosen attribute. The discriminating descriptions are then simple tests on this attribute's values, often described as the **split points** of the

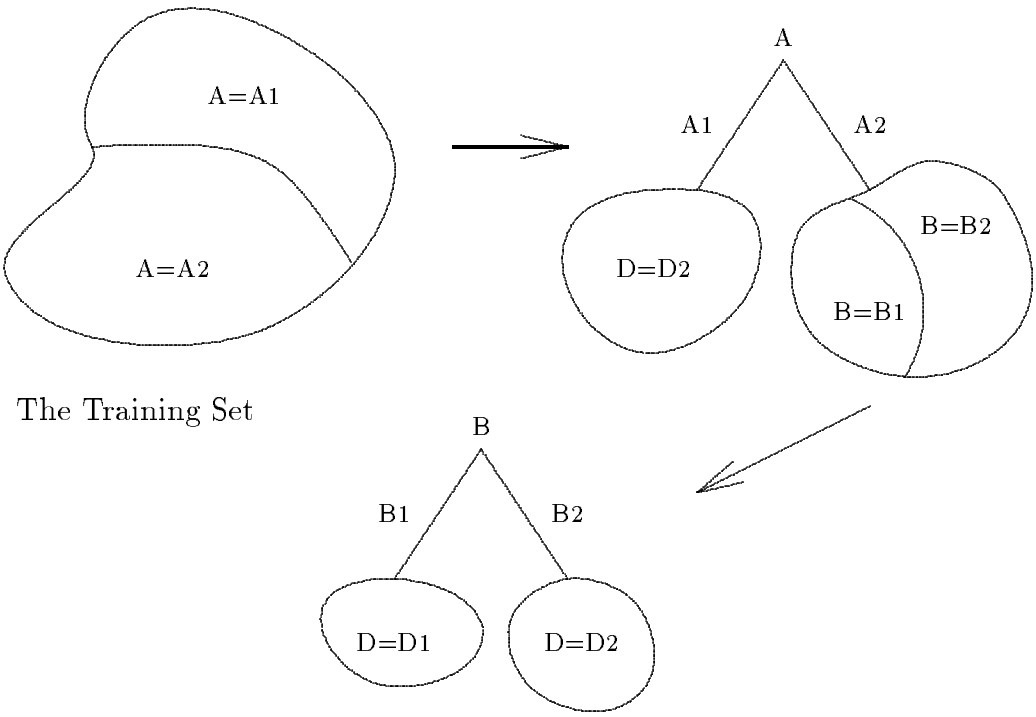


FIGURE 2.2: *An illustration of the divide and conquer technique of decision tree induction. The original training set is partitioned using attribute A. All objects in the cell of this partition corresponding to a value of A1 have a decision of D2. The other cell of the partition is further partitioned using the attribute B. Each of the cells in this second partition is homogeneous with respect to the values of the decision attribute. The resulting decision tree is that of Figure 2.1*

attribute. Thus a simple decision tree of the form presented in Figure 2.1 will result from this induction process, with the chosen attributes labelling the nodes, as illustrated in Figure 2.2.

The final step tests a **termination criterion** which is used to express the conditions under which this divide and conquer process should stop. If the criterion is not met, then the cells of S^* are each used as new training sets, and the algorithm is re-applied to each.

The termination criterion typically identifies a training set whose objects all record the same value for the decision attribute (i.e., homogeneous with respect

to the values of the decision attribute) as one for which no further processing is required. The common value for the decision attribute will then label the corresponding leaf node. Processing also terminates whenever no attributes remain upon which to partition the given training set. This may arise when only categorical attributes exist, and all attributes have been used. That is, each object has the same collection of attribute values, except for the decision attribute. This can result from noise in the training examples, or because of a lack of suitable attributes. In such a case the corresponding leaf node is labelled with a decision of Null.

As pointed out by various researchers, including Kononenko, Bratko, and Roškar (1984), for small training sets the selection criteria typically become unreliable, because of the small sample sizes involved when the training set is partitioned. Thus, a number of the decision tree induction systems terminate the divide and conquering when the objects in a training set fail to meet certain other conditions, leading to a form of tree pruning. (Quinlan (1982) and Arbab and Michie (1985), however, indicate that small training sets are capable of generating quite adequate decision trees.)

Tree pruning is the process of reducing the size of the decision tree, typically by removing leaf nodes of the tree and combining branches. Pruning is useful when the training set contains noisy data (data containing inaccuracies). It is also useful in noisy domains, where the accuracy of attribute values can not be assured.

In summary the general algorithm is:

1. Construct **S**, the set of candidate partitions of the training set.
2. Using a selection criterion, select the best S^* in **S**.
3. Find a discriminating description for each cell, C_i, in S^*.
4. For each C_i, test the termination criterion:
   4.1. If it is not met, then use C_i as a training set and repeat from step 1.
   4.2. If it is met, then a leaf node of the decision tree is formed, with an appropriate decision associated with it.

This decision tree induction algorithm can be viewed as an instance of the ubiquitous search paradigm. The search space consists of all simple decision trees of the form illustrated in Figure 2.1. Such decision trees have simple tests associated with the nodes (involving a single attribute) with branches corresponding to the possible outcomes. The search algorithm begins by determining which attribute should appear as the label of the root node of the decision tree. It then considers each child node recursively. Most decision tree induction algorithms are a best-first, depth-first search algorithm with no backtracking. The search space over which these systems operate is very large. All of the algorithms search through this space under the guidance of various heuristics.

Actual implementations of decision tree induction algorithms differ primarily along five dimensions: the type of attributes allowed; the type of descriptions which can label a node and its branches and the consequent branching (binary or n-ary); the selection criterion; the termination criterion; and whether tree pruning is carried out. The earlier algorithms catered for categorical attributes to the exclusion of integer attributes. This typically manifests itself in the context of the selection criterion employed. Decision tree induction systems typically use a single attribute, testing for its various values, as the discriminating description for the cells of a partition. More elaborate constructs are found in conceptual clustering systems (Michalski and Stepp, 1983). Implementations

also differ with respect to the number of cells allowed in a partition, which consequently impacts upon the structure of the resulting decision tree—a number of systems allow only binary partitions, leading to binary decision trees. The termination criterion also differs between the various implementations, with some using this criterion as a means of pruning. Other implementations begin pruning once the decision tree has been fully constructed. A further step in several implementations is to then convert the decision tree to a rule set, which may again be pruned.

2.3 THE DECISION TREE INDUCTION FAMILY

The Concept Learning System (CLS), developed by Hunt (Hunt, Marin, and Stone, 1966), is the earliest decision tree induction algorithm. ID3 (Quinlan, 1982) and CART (Breiman et al., 1984) marked the beginning of a renewed interest in decision tree induction, with ACLS (Paterson and Niblett, 1982) and ASSISTANT (Kononenko, Bratko, and Roškar, 1984) developing upon ID3. AOCDL, and in turn RG (Arbab and Michie, 1985), develop upon ideas introduced in ASSISTANT. More recent systems such as ID4 (Schlimmer and Fisher, 1986), ID5 (Utgoff, 1988), and ID5R (Utgoff, 1989) are natural progressions from the original ID3 allowing incremental learning. The following table lists those systems described in detail below. A summary box will follow the detailed description of each system to identify its main features, including the five dimensions introduced above. Figure 2.3 summarises the family tree of these systems.

CLS	*Concept Learning System*	(Hunt, Marin, and Stone, 1966)
	Quinlan's CLS	(Quinlan, 1979b)
ID3	*Iterative Dichotomiser 3*	(Quinlan, 1982)
ACLS	*Analogue Concept Learning System*	(Paterson and Niblett, 1982)
ASSISTANT		(Kononenko, Bratko, and Roškar, 1984)
	Structured Induction	(Shapiro, 1983)
AOCDL		(Bratko, 1983)
RG	*Rule Generator*	(Arbab, 1985)
C4		(Quinlan et al., 1986)
CART	*Classification And Regression Tree*	(Breiman et al., 1984)
ID4	*Incremental ID3*	(Schlimmer and Fisher, 1986)
ID5R	*Incremental ID3*	(Utgoff, 1989)

2.3.1 CLS

The forerunner to all of the systems described here is Hunt's Concept Learning System which has its roots in experimental psychology. "The original motivation for a CLS was to construct a simulation of human behaviour" (Hunt, Marin, and Stone, 1966). CLS was primarily a subroutine for inducing decision structures capable of being used for a pattern classification task. The pattern

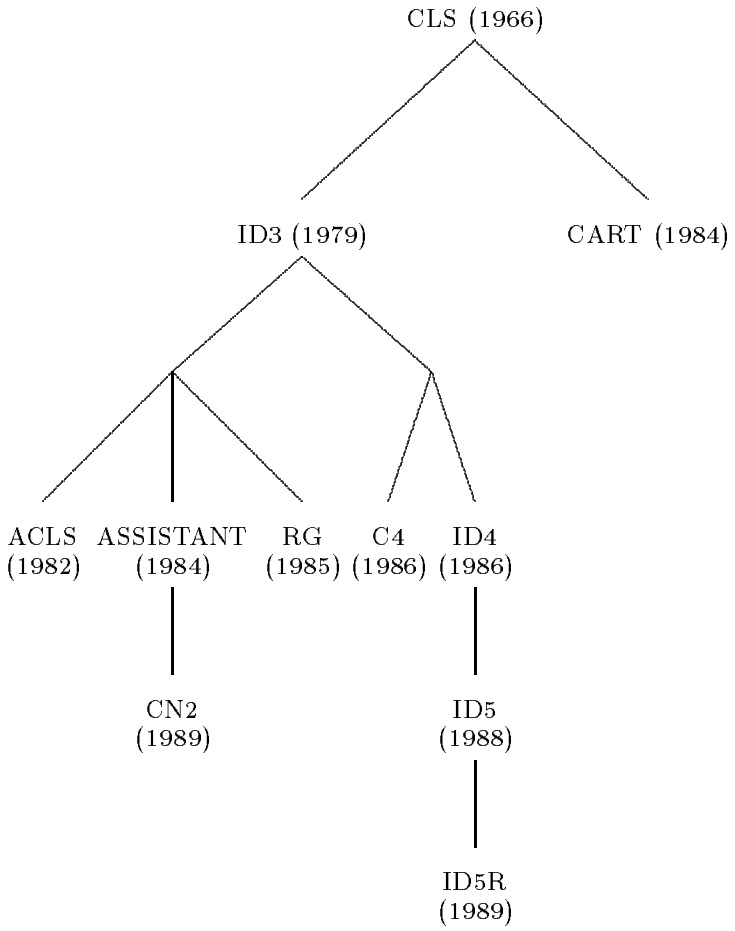


FIGURE 2.3: *A simplistic overview of the genealogy of the decision tree induction family. The dates are only indicative, and often reflect the date of publication of an early paper describing the system rather than the actual date of development. Refer to the text for clarification. After Quinlan (1986a).*

here is the vector of values associated with each object. Hunt, Marin, and Stone (1966) discuss a number of experiments with variations of the basic algorithm.

The CLS algorithm conforms to the general algorithm, with the exception that the first step is implicit, in common with many decision tree induction algorithms. CLS allows for categorical attributes only, and only binary partitions of the training set are considered—based upon particular attribute values. The examples contained in one of the cells of a partition have a common value for

the attribute upon which the partition is based. CLS begins by searching for the most discriminatory attribute-value combination.

The selection criterion employed by CLS is based upon costs associated with measuring attributes and with making incorrect decisions (Quinlan, 1979a). Under a scenario where one decision is associated with the training set, the sum of the resulting mis-classification costs associated with the incorrect decisions is determined. Repeating this for each possible value of the decision attribute and then taking the minimum, results in a measure of the cost of using a single decision value as our decision tree. Call this T_0. The training set is then partitioned. For each cell of the partition a T_0 is computed, and the sum of these is added to the cost of actually measuring the attribute. Computing this for each possible partition and taking the minimum gives a measure which we will call T_1. The minimum of T_0 and T_1 is then the cost of the resulting decision tree. If T_0 is the minimum, then there is no need to continue with tree induction from this training set. Otherwise, the corresponding partition is used to generate two new training sets.

This recursive algorithm for determining costs will ensure that the induced decision tree will have minimal cost. This calculation is prohibitively expensive, but can be approximated satisfactorily with much less calculation, resulting in low cost, or at best minimal cost, decision trees.

Each node of the decision tree induced by CLS, except for terminal nodes, will have only two branches, corresponding to the two answers to the question "Does attribute A have value A_i?". Thus, binary decision trees are generated, testing at each node for a specific value of a specific attribute. In general, such an approach may result in paths through the decision tree in which a single attribute may recur, testing for different values. The resulting decision trees are potentially deep (having many levels). Hunt restricts the algorithm to binary decision trees for the sake of simplicity. Multi-branching decision trees are considered by Hunt, but primarily in the context of "future work."

The box below summarises the CLS algorithm. CLS deals only with categorical attributes. It builds partitions based upon a single attribute and its values, resulting in decision tree nodes labelled with the single attributes. The branches emanating from a node correspond to the test for the value of the attribute. The selection criterion uses a combined measurement and mis-classification cost, and the divide and conquer process terminates whenever all objects in a training set belong to the same decision class (i.e., have the same value for the decision attribute), or whenever no attributes remain, or when further construction would result in increased cost.

CLS	
Attributes	Categorical.
Nodes	Single attributes (A).
Branching	Binary.
Branch Labels	$A = A_i$ and $A \neq A_i$.
Selection Criterion	Minimise the combined measurement and mis-classification cost of each attribute. The latter is computed using look ahead.
Termination Criterion	Homogeneous training set, or no attributes upon which to partition remain, or training set has the minimum cost.

2.3.2 Quinlan's CLS

The renewed interest in CLS-type systems was sparked by Quinlan (1979a) when he used a CLS-type system to induce decision trees in the domain of Chess. Quinlan (1979b) further considered the problem of handling large training sets in a memory-limited computer environment, again in the domain of Chess. The windowing technique was introduced here as a key component of Quinlan's approach. With this technique a subset of the training set, called the window, is used as the actual input to the induction algorithm. This window is augmented with other objects from the full training set whenever exceptions to the induced decision tree are found in the original training set.

The learning element described in these papers is a "simple and unsophisticated relative of Hunt's CLS" (Quinlan, 1979b). As with CLS, single attributes label the nodes. However, branches corresponding to the various values of the attribute are used, removing the restriction of binary trees. Only categorical

attributes were, once again, considered. The feature of the system of most interest is the selection criterion. The criterion is based upon the desire to produce a simple, and therefore general, decision tree, and uses estimates of the complexity of the sub-trees which result from choosing a particular attribute (Quinlan, 1979b).

Consider a partition of the training set based upon a single attribute. The complexity of the tree that results can be related to the sum of the complexities of the trees that are constructed from each of the cells in the partition. Assuming a binary-valued decision attribute, we can further partition each of the cells of the first partition into two subcells corresponding to the two values of the decision attribute. If one of the subcells is empty, then a very simple, indeed trivial, decision tree will result, consisting of just a leaf node. Similarly, if one of the subcells is small, then it would be expected that the resulting decision tree will be relatively simple, but not trivial. Summing the square roots of the minimum of the number of items in the two subcells of each cell gives a lower estimate for the complexity of the resulting tree. The attribute which results in a partition which minimises this measure of complexity is chosen.

The termination criterion employed is the simple one of checking for empty cells, or for cells containing a single value for the decision attribute.

In summary, Quinlan's initial implementation of a concept learning system deals with categorical attributes, with branching based upon the values of a selected attribute. The selection criterion employed attempts to minimise the complexity of the resulting decision trees.

Quinlan's CLS	
Attributes	Categorical.
Nodes	Single attributes (A).
Branching	n-ary.
Branch Labels	A_i.
Selection Criterion	Minimise estimated complexity.
Termination Criterion	Homogeneous training set, or No attributes upon which to partition remain.

These early efforts by Quinlan to develop practical inductive learning systems spurred others into conducting further research. While the domain was limited to Chess, Quinlan's CLS served to prove the idea of using such a system with large collections of examples. These systems were capable of generating correct decision trees which sensibly summarised the input examples.

2.3.3 An Information-Theory Based Selection Criterion

Quinlan (1979b) makes mention of an information-theoretic model of complexity under experimentation. A selection criterion based upon this information-theoretic model is first presented in Quinlan (1982), and later in Quinlan (1983b) and Quinlan (1983a). The basic idea is to use an estimate of the amount of information gain that will result once a partition of the training set has been constructed (based upon a single attribute). Then, the attribute chosen is the one which best discriminates between the values of the decision attribute. This selection criterion can also be viewed as computing the amount of entropy associated with the attribute, defined in the context of the values for the decision attribute, and choosing the attribute associated with the least amount of entropy.

The information-theoretic model regards a decision tree as an information source. Given an example, the decision tree generates a message which is the value for the decision attribute. Information theory provides the formula $\sum_{j=1}^{m} -p_j \log(p_j)$ as a measure of the information content of a message, where p_j is the probability of making a particular decision, with m possible decisions in all. (This could be equivalently written as $\sum_{j=1}^{m} p_j \log(\frac{1}{p_j})$ which is of the more familiar form for an entropy function.) This measure of information content is maximal when all decisions are equally likely (the p_j are equal) giving rise to the most uncertainty. It is minimal (zero) when one decision only is certain ($p_j = 1$) and no other decision can ever be made (all other p_j's are 0). A useful analogy due to Mingers (1989b) is with horse racing: "the more runners and the more evenly they are matched, the greater the value of knowing the

winner". The probabilities are of course not known, but may be approximated by the relative frequencies of the occurrences of the decisions in the training set. This measure of information content will be denoted by $I(\mathbf{Tr})$. Thus, $I(\mathbf{Tr})$ is an estimate of the information content of a decision made by the decision tree induced from **Tr**, or equally, the information needed to make a decision, given only the decision frequencies in **Tr** (Mingers, 1989b).

Suppose that S is a partition of **Tr**, containing the n cells $C_1, C_2, \ldots, C_n$, each cell corresponding to a single value for the attribute A, which has n possible values. In line with the general decision tree induction algorithm, if we were to now consider each of these cells as new training sets from which to construct n new decision trees, then $I(C_i)$ is a measure of the information content of decisions made by the decision tree constructed from C_i. This is the information content of C_i given that we know that the attribute A has the value corresponding to this cell. Taking the average of these measures weighted by the number of examples in the cell leads to an estimate of the amount of information required, in order to make a decision for some object, given a value for the attribute A:

$$E(S) = \sum_{i=1}^{n} \frac{n_i}{N} I(C_i)$$

where n_i is the number of examples in C_i, and N is the number of examples in **Tr**.

Hence, by choosing a particular partition S of the training set we have a gain in information which is given by:

$$Gain(\mathbf{Tr},\ S) = I(\mathbf{Tr}) - E(S)$$

The best partition S^* then is the S for which $Gain(\mathbf{Tr}, S)$ is maximal, or equivalently, since $I(\mathbf{Tr})$ is constant over **S**, for which $E(S)$ is minimal.

Let n_{ij} be the number of examples in C_i with decision attribute value D_j (with m different decision values in all). Then:

$$I(C_i) = \sum_{j=1}^{m} -\frac{n_{ij}}{n_i} \log \frac{n_{ij}}{n_i},$$

and,

$$E(S) = \sum_{i=1}^{n} \frac{n_i}{N} (\sum_{j=1}^{m} -\frac{n_{ij}}{n_i} \log \frac{n_{ij}}{n_i}).$$

Minimising $E(S)$ is then equivalent to minimising

$$-\sum_{i=1}^{n} \frac{n_i}{n_i} \frac{1}{N} \sum_{j=1}^{m} n_{ij} \log \frac{n_{ij}}{n_i},$$

which is equivalent to minimising the final cost function:

$$Cost(P) = -\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} \log \frac{n_{ij}}{n_i}$$

recalling that $n_i = \sum_{k=1}^{m} n_{ik}$. This can be summarised as:

$$\text{Cost of partition} = -\sum_{cells} \sum_{decisions} \text{Examples in cell with this decision} \times \log \frac{\text{Examples in cell with this decision}}{\text{Number of examples in this cell}}$$

Consideration of this selection criterion leads to the observation that a cost of zero for any particular partition indicates that each cell of the partition contains examples having a single common value for the decision attribute. This is the most desired situation, as the decision tree constructed from such a partition is of depth 1, minimising the complexity. Further it is observed that there is a possibility of a number of candidate partitions being equally good with respect to the selection criterion. Intuition suggests that this selection criterion is unable to distinguish between attributes primarily in the case where a number of attributes have a cost of 0 since it is less likely for two or more

attributes to have the same non-zero cost computed for them, with respect to a particular training set.

These observations, whilst not developed any further in this chapter, are explored in Chapter 3 and taken advantage of by the MIL algorithm presented in Chapter 4.

The information-theoretic cost function describe above has been deployed as the selection criterion in a number of the decision tree building systems described below.

2.3.4 ID3

The ID3 algorithm is motivated by the desire to produce simple and efficient decision trees. ID3 presents an approach which reduces the potential complexity of the binary decision trees induced by CLS. As in CLS, the nodes of the ID3 decision trees are single attributes, but, by allowing n-ary branching, any attribute will now appear at most once on any single path through the decision tree. Given a training set, each attribute defines just a single partition. This partition consists of cells for which each member object has a common value for the said attribute. Each of these values thus corresponds to a branch from the appropriate node.

The information-theoretic selection criterion is used by ID3 to choose attributes at each stage. The attributes with the most information content are chosen early on, thus the decision trees tend to have minimal depth.

A number of conditions are checked in order to determine whether the dividing and conquering should terminate. As with CLS, a single decision cell (i.e., a homogeneous training set) indicates that a single decision can be made, and no further work is required of this training set. If no attributes remain upon which to partition the training set, then a decision of Null must be made. In addition, Quinlan (1983b) introduced a chi-square test for stochastic independence for deciding when to stop, to account for the possibility of there being noise in the training set. The basis of using the chi-square test statistic is that an attribute

will be selected for splitting upon only if the hypothesis that the attribute is independent of the decision class over the training set can be rejected with a high degree of confidence. The use of this extra termination criterion results in tree pruning leading to dramatic reductions in the size of the decision trees (Quinlan, 1983b).

In summary, the goal of ID3 as an inductive concept learning algorithm is to produce decision trees with high execution efficiency. The pruning method introduced in ID3 helps to reduce the complexity of the resulting trees.

ID3	
Attributes	Categorical.
Nodes	Single attributes (A).
Branching	n-ary.
Branch Labels	$A = A_i, i = 1 \ldots n$.
Selection Criterion	Information-theoretic.
Termination Criterion	Homogeneous training set, or No attributes upon which to partition remain, or Remaining attributes fail the chi-square test.

2.3.5 ACLS

ACLS (Michie, 1983; Paterson and Niblett, 1982; Shepherd, 1983) extends ID3 by allowing the input data to be described in terms of integer attributes, as well as categorical attributes. The information-theoretic selection criterion is again used to associate a cost with each attribute, whether categorical or integer. In terms of the general algorithm, the set of partitions, **S**, contains one partition corresponding to each categorical attribute (as in ID3), but with many partitions corresponding to each integer attribute. For an integer attribute A_I, the objects of the training set are considered as ordered with respect to that attribute. One partition of **Tr** then consists of two cells, C_1 and C_2, such that the maximum value for A_I in C_1 is less than the minimum value in C_2 (some number between the maximum and minimum is chosen as the split point). The selection criterion, as applied to these binary partitions, is thus quite simplified, although up to $n - 1$ (with n distinct values for the integer attribute found in **Tr**) possible partitions must now be considered for this single attribute.

If an integer attribute A_I is chosen to label a node then that node will have two branches leading from it. One branch will be labelled with "$< V$" and the other with its complement "$\geq V$". The split point V is the mid-point of the two cells of the chosen partition P. That is,

$$V = \frac{max\ value\ of\ A_I\ in\ C_1 + min\ value\ of\ A_I\ in\ C_2}{2}$$

The ACLS system is written in Pascal, and a number of commercial derivatives of it have been developed, including Expert-Ease and Rule-Master (Michie et al., 1984).

ACLS	
Attributes	Categorical and integer.
Nodes	Single attributes (A).
Branching	One branch for each value of A for categorical A. Two branches for integer A.
Branch Labels	$A = A_i, i = 1 \ldots n$ for categorical A. $A < V, A \geq V$ for integer A.
Selection Criterion	Information-theoretic.
Termination Criterion	Homogeneous training set, or No attributes upon which to partition remain.

2.3.6 ASSISTANT

ASSISTANT (Kononenko, Bratko, and Roškar, 1984) is a variant of ID3. It was developed as an attempt to deal with a number of observed deficiencies of ID3. These deficiencies arose in the context of building diagnostic rules in a medical domain. ASSISTANT extends ID3 by considering the problems relating to the existence of multiple-valued attributes. Early descriptions of ID3 treated only binary-valued attributes. ASSISTANT introduces a new approach to tree pruning and is able to automatically select good training examples.

A key observation made by Kononenko, Bratko, and Roškar (1984) and empirically demonstrated again by Williams (1987) (and presented in Chapter 3 below) was that the information-theory based selection criterion was biased toward attributes with more values. Quinlan (1985) provides an analysis of this situation, leading to the same conclusion. The solution proposed in ASSISTANT

is to consider only binary decision trees. Although this is reminiscent of CLS, it is more general, allowing subsets of values of a categorical attribute to be associated with each branch of the decision tree, rather than just one. For integer attributes, the technique introduced in ACLS is followed.

The partitions considered are thus always binary. For a categorical attribute A_C, a subset of its values is identified for the purpose of partitioning the training set into two cells. One cell will contain all those objects of the training set which have a value for A_C contained in the identified subset of values. The other cell contains the remaining objects. Each distinct subset of the values of the attribute A_C describes a candidate partition. An attribute with n values leads to at most $2^{n-1} - 1$ candidate partitions after trivial and symmetric subsets of the values are removed. Such an approach removes the bias against binary partitions, since only binary partitions are considered, and it is claimed to lead to smaller decision trees with an improved classification performance (Kononenko, Bratko, and Roškar, 1984).

The termination criterion used by ASSISTANT also differs from ID3's termination criterion. The induction process terminates when a measure of the amount of information resulting from the construction of a sub-tree from the current training set is less than the expected amount of information required to make a decision for an object. Kononenko, Bratko, and Roškar point out that although this criterion is *ad hoc* it worked well in their experiments.

Two further features of ASSISTANT are that it can automatically select good training examples, and that it handles hierarchical classes of decisions and attribute values. ASSISTANT allows for the automatic selection of good training records based on Bayesian probabilities, which can be contrasted to the windowing technique introduced by Quinlan (1979b). Hierarchical classes allow decision trees to make decisions at different levels.

Whilst Kononenko, Bratko, and Roškar report that ASSISTANT results in better prediction and smaller decision trees, Quinlan (1985) expresses some reservation. ASSISTANT's method, it is reasoned, "could lead to decision trees

that are even more unintelligible to human experts than is ordinarily the case, with unrelated attribute values being grouped together and multiple tests on the same attribute". Kononenko, Bratko, and Roškar do point out, as does Shepherd (1983), that binary trees are often poorly structured.

ASSISTANT	
Attributes	Categorical and integer.
Nodes	Single attributes (A).
Branching	Binary.
Branch Labels	$A \in S$, $A \notin S$, or $A < V_i$, $A \geq V_i$.
Selection Criterion	Information-theoretic.
Termination Criterion	Homogeneous training set, or No attributes upon which to partition remain, or Information content < information required.

2.3.7 RG

The rule generator (RG) of Arbab (1985) is a descendant if ID3 which aims to produce linear decision trees. (A linear decision tree is one in which each node has at most one non-leaf child.) The motivation is the desire to produce decision trees which facilitate human understanding. This differs from ID3 which primarily aims to produce decision trees with high execution efficiency, rather than trees which are readily accessible by humans (Arbab and Michie, 1985). It is argued that a linear decision tree is more easily understood.

RG follows on from the linear decision tree induction system AOCDL developed by Bratko (1983). AOCDL implements a backtracking, heuristic search algorithm, aiming to produce an almost linear decision tree. A non-linearity measure is used to guide this search process. Little consideration is given to the execution efficiency of the resulting decision tree.

RG embodies the basic ideas introduced by AOCDL, with the additional goal of producing more efficient decision trees than does AOCDL. Although AOCDL produces linear trees, they are very inefficient compared to the trees produced by ID3. Efficiency is measured in terms of the average length of paths that are traversed when using the decision tree to classify an object.

The non-linearity measure RG uses, as applied to a decision tree T, is:

$$NL(T) = \frac{1}{n}\sum_{i=1}^{n}(NL(T_i) + (n-i) \times IN(T_i)).$$

Here, n is the number of branches emanating from the root of the decision tree T, T_i is the tree rooted at the ith branch, and $IN(T_i)$ is the number of internal (non-leaf) nodes of the tree T_i. The T_i are assumed sorted in increasing order of $IN(T_i)$. $NL(T) = 0$ if T is a leaf node.

In addition to this non-linearity measure, a measure for the average execution cost of the decision tree is given by Arbab and Michie (1985). Suppose the attributes $A_1 \ldots A_n$ appear in the decision tree T, and let $N_1 \ldots N_t$ be the labels of the non-leaf nodes of the decision tree (so $t \geq n$, and each N_j is just some A_i). Further, let $c(A_i)$ be some cost associated with attribute A_i (e.g., the cost of obtaining a value for it). Finally, denote by $T(N_j)$ the number of objects in the training set associated with node N_j of T. If N_1 is the root node, then $T(N_1)$ is just the size of the training set. If N_2 is a node appearing immediately below the root node, N_1, and N_1 is the attribute A_i and the branch from N_1 to N_2 is labelled with V_j, then $T(N_2)$ is the number of objects in the training set with $A_i = V_j$. The average cost of decision tree T is then:

$$\frac{\sum_{j=1}^{t} T(N_j) \times c(N_j)}{T(N_1)}$$

The strategy employed by RG is to use the AO* algorithm (Nilsson, 1980) with an over-optimistic estimate for the non-linearity of the decision tree. The And/Or tree used is made up of attribute nodes (the Or nodes) and attribute value nodes (the And nodes), the root node of the And/Or tree being an And node. (See Figure 2.4.) The attribute nodes represent alternatives for the choice of attribute at each node in the final decision tree, and the attribute value nodes represent the links of the final decision tree, there being one link (i.e., attribute

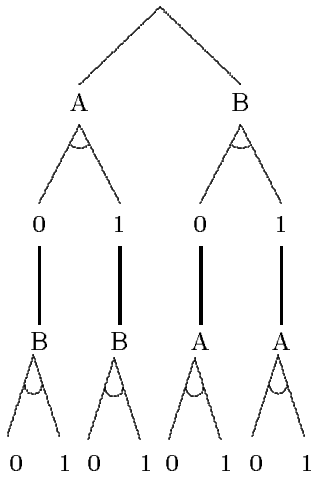


FIGURE 2.4: *The AND/OR tree structure searched by RG. This search tree embodies all possible decision trees based upon two binary-valued attributes, A and B.*

value node) for each value of the attribute. The And nodes can be thought of as subproblems.

RG	
Attributes	Categorical.
Algorithm	AO* using optimistic estimate for non-linearity.
Nodes	Single attributes (A).
Branching	n-ary.
Branch Labels	A = V.
Selection Criterion	Non linearity of partially constructed tree. Number of expected internal nodes. The attributes entropy measure.
Termination Criterion	All objects have same decision.

2.3.8 Quinlan's Gain Ratio Criteria

Reference was made in Section 2.3.6 to the inherent bias in the information-theoretic selection criterion toward multi-valued attributes. A solution to this problem was presented in ASSISTANT in the guise of restricting all branching to just binary tests. Another solution involves normalising the selection criterion. Kononenko, Bratko, and Roškar (1984) first considered this with their normalised informativity function, which has as a divisor the logarithm of the

number of values associated with an attribute. Thus, instead of maximising the gain in information, as represented by:

$$Gain(\mathbf{Tr},\ S) = I(\mathbf{Tr}) - E(S)$$

where S is a partition of the training set **Tr**, we now maximise:

$$NormalisedGain(\mathbf{Tr},\ S) = \frac{I(\mathbf{Tr}) - E(S)}{\log_2(Number\ of\ values\ of\ A)}$$

over all attributes A.

Kononenko, Bratko, and Roškar point out though that a new kind of bias is introduced. This arises in the case where the values of an attribute are not of equal importance. Indeed, it was this observation that lead them to restrict their decision trees to being binary.

Quinlan (1985) also identified a bias against multi-valued attributes under this normalised gain. He notes that "an attribute with eight values would have to achieve three times the information gain of a binary-valued attribute if it were to be the chosen attribute".

The Gain Ratio criterion was first introduced by Quinlan (1985) and further discussed in Quinlan (1986a), in response to this observation. It was again a normalised information-theoretic selection criterion. As with Quinlan's previous selection criterion this normalised function is based upon the idea of information content. A measure of the information content of just knowing the value of an attribute A can be formulated using the relative frequencies of the values of the attributes:

$$IV(A) = -\sum_{i=1}^{n} \frac{|C_i|}{N} \log \frac{|C_i|}{N}.$$

In this formula the partitions are as in ID3, and $|C_i|$ is the size of the cell C_i of the partition S based upon the attribute A. C_i corresponds to the attribute value A_i, with attribute A having n possible values. N is the size of the training set **Tr**.

Quinlan uses the original information-theoretic based *Gain* function to filter out the below average attributes before using the normalised function. That is, only those attributes with a computed *Gain* at least equal to, or greater than, the average gain in information, will be compared using the normalised function. The chosen attribute is the one which maximises the gain ratio:

$$\frac{I(\mathbf{Tr}) - E(S)}{IV(A)}$$

The heuristic of selecting from among the average or greater attributes (with respect to *Gain*) helps to avoid favouring poor attributes which have small values of $IV(A)$.

2.3.9 C4

The C4 induction system (Quinlan et al., 1986) evolved out of ID3. It incorporates many of the enhancements to the original ID3 introduced in the systems described above: C4 allows integer attributes and employs the gain-ratio selection criterion. It further differs from ID3 in the pruning that it carries out *after* constructing a decision tree. Such an approach to pruning was introduced by Breiman et al. (1984) in their CART algorithm. C4 retains the windowing approach of ID3 where subsets of the whole training set are employed in constructing the decision tree.

C4 attempts to prune a decision tree by replacing sub-trees with a leaf. This process is performed under the guidance of the information contained in the training set. When such a modified decision tree is applied to the original training set classification errors will result. The goal though is to decrease the complexity of a decision tree (measured in terms of path length, where a path begins at the root of the tree and ends at a leaf), whilst maintaining its accuracy at classifying objects (usually from the training set). Thus a sub-tree is replaced by a leaf if it has the least ratio of the increased error rate to the complexity of the sub-tree over all sub-trees, and this ratio is significantly worse than the average over all sub-trees of the decision tree.

Later developments of C4 have introduced further pruning techniques, in parallel with the conversion of decision trees to sets of rules by way of the enumeration of all paths through the decision tree (Quinlan, 1986b).

As with ID3 and ACLS, commercial systems based upon the ideas introduced in C4 have become available, including TODAY-ES (Morley et al., 1988).

C4	
Attributes	Categorical and integer.
Nodes	Single attributes (A).
Branching	One branch for each value of A for categorical A. Two branches for integer A.
Branch Labels	$A = A_i, i = 1 \ldots n$ for categorical A. $A < V, A \geq V$ for integer A.
Selection Criterion	Gain ratio: $\frac{Information\ Gain}{Information\ Content}$.
Termination Criterion	Homogeneous training set, or No attributes upon which to partition remain.
Pruning	Replace sub-tree by leaf if ratio of increased error rate to complexity of sub-tree is worse than average.

2.3.10 CART

At about the same time that Quinlan was developing ID3, Breiman et al. (1984) independently developed the CART decision tree induction algorithm. The approach adopted by CART is basically the same as that of ID3. Input examples consist of attribute-value lists, each example having a decision associated with it. Both categorical and continuous attributes are handled. The divide and conquer process is followed, but allowing only binary partitions. In this regard CART and ASSISTANT are similar. Primary differences between CART and the ID3 descendants are to be found in the selection criterion, and the emphasis upon pruning.

The CART algorithm uses a selection criterion known as Gini (although others are considered by Breiman et al. (1984), including the so called twoing selection criterion). The Gini selection criterion involves the use of the Gini index of diversity which is applied to a training set and can be summarised as:

$$\sum_{i=1}^{m} \sum_{j=1}^{m} p_i p_j, i \neq j$$

Here m is the number of possible values of the decision attribute, and p_i is the proportion of examples in the training set with decision value i. This Gini index of diversity is then calculated for the original training set, and for each of the training sets that results from the binary partition of the original training set. The partition which maximally decreases the diversity is chosen. Building upon earlier work in statistics, CART employs a technique which dramatically reduces the amount of computation required in order to choose the best partition from amongst the $2^{n-1} - 1$ possible partitions for each attribute (Crawford, 1989).

The pruning employed by CART uses a heuristic based upon cost and complexity. The cost here is calculated as the error (or mis-classification) rate of a tree, and the complexity is measured in terms of the number of leaves of the tree. The error rate is determined by application of the decision tree to examples not drawn from the training set. The CART algorithm searches for the smallest decision tree having an acceptable mis-classification rate. Crawford (1989) provides a detailed description of the process, as well as alternative techniques within the CART framework.

CART	
Attribute	Categorical and integers.
Nodes	Single attributes (A).
Branching	Binary.
Branch Labels	$A \in S$, $A \notin S$, or $A < V_i$, $A \geq V_i$.
Selection Criterion	Gini index of diversity.
Termination Criterion	Homogeneous training set, or No attributes upon which to partition remain.
Pruning	Cost/complexity heuristic.

2.4 DISCUSSION

The collection of systems presented here constitute the founding core of the decision tree induction family of systems. These systems represent the starting point of the work to be presented in the following chapters. Below is a review of other research related to decision tree induction. This is followed by a discussion of some relevant issues concerning the decision tree induction process and algorithm.

2.4.1 Other Related Research

Much interest in decision tree induction has emerged in the research and development community as a result of the development of a number of successful expert systems using decision tree induction algorithms (see, for example, Michie (1987)). Variations and enhancements to the basic decision tree induction approach, changing in some basic way the usage of the decision tree induction algorithm, have been suggested and implemented. The early idea of structured induction, and the more recent advances in incremental decision tree induction are two such developments.

The method of structured induction was introduced by Shapiro (1983) in his doctoral thesis and later published in book form (Shapiro, 1987). This approach to decision tree induction tackles the task in a style reminiscent of structured programming. Structured induction is more a methodology than an induction algorithm *per se*. This methodology introduces the idea of constructing a number of decision trees, each corresponding to different levels of detail. Structured induction is presented by Shapiro in the context of an ID3-based decision tree induction system.

Shapiro's approach begins with a collection of "super-attributes", which directly determine the class of an object. These super-attributes are the top-level attributes of the decision tree, and are distinguished from those attributes used to describe the objects of the training set. For each of these super-attributes, a decision tree is constructed which will determine a value for this attribute.

These decision trees may possibly make use of new super-attributes. This process continues until, at the lowest level, the decision trees are expressed solely in terms of the actual attributes used to describe the objects in the training set.

Such a process has the primary advantage of producing simple decision trees. The training set used to construct any one decision tree is typically small and thus the resulting decision trees are also typically small. Consequently, the resulting knowledge, represented as a collection of decision trees rather than a single, usually opaque, large decision tree, is more amenable to human understanding.

A relevant point to note regarding structured induction is that it is a methodology which is largely independent of the actual induction algorithm employed.

A deficiency of decision tree induction algorithms identified by a number of researchers is their inability to effectively handle new training instances that become available after the decision tree has been built. ID3's approach, for example, would be to reconstruct the decision tree using the original training set suitably augmented. ID3 is essentially a non-incremental algorithm. Consideration of this problem has lead to enhancements to the basic ID3 algorithm, resulting in the incremental decision tree induction systems of ID4, ID5, and ID5R. Only categorical attributes are considered in the following descriptions.

The ID4 algorithm (Schlimmer and Fisher, 1986) represents an early attempt at building an incremental version of ID3. ID4 takes advantage of the observation that the information-theoretic selection criterion used in ID3 operates upon a count of the number of objects belonging to each decision class for each cell of the various alternative partitions of the training set. For a given node in a decision tree, ID4 maintains a record of this information. Consider, for example, an attribute B which can potentially label a particular node, where B has the values $B_i, i = 1, 2, \ldots, n$. A count is kept of the number of examples at this node with a value of B_i for B and a decision of D_j for the decision attribute for each possible D_j. It is these counts for each of the possible attributes that

are used in the ID3 cost function. They thus embody all the information needed to select an attribute to label a particular node, without recourse to the original training examples.

From this store of information ID4 is able to restructure the decision tree appropriately as each new example is presented to it. This is effected by re-computing the cost function once the counts have been updated with the new example. If this re-calculation for a particular node does not change the choice of attribute, then the example is propagated to the appropriate child node, and the process repeated. If on the other hand the re-calculation indicates a different choice of attribute to label the current node, then all sub-trees below this current node are removed, and re-generated as new instances arrive. Usually, ID4 builds the same decision trees as ID3 since the information-theoretic-based criterion makes good choices.

Utgoff (1989) identifies a condition which can lead to the ID4 algorithm thrashing. This arises when the cost function is unable to distinguish between attributes or when it can only marginally distinguish between them. New examples may cause the choice of attribute to oscillate between two or more, thus requiring new sub-trees to continually be built.

The ID5 algorithm (Utgoff, 1988) and its successor ID5R (Utgoff, 1989) further develop the idea of caching the information contained in the training examples, as introduced in ID4. However, they go one important step further by retaining all the necessary information from sub-trees as they are collapsed rather than discarding it completely. This allows the ID5 algorithms to rebuild the sub-trees as if the original training examples were still available. To do this ID5 records at the leaves of a decision tree those portions of an example's description not implicitly recorded in the decision tree. When an attribute at a node is earmarked for removal, to be replaced by a better attribute, a process which restructures the sub-trees below this node is invoked.

This restructuring process will, through recursion, ensure that each sub-tree below the current node for which a change of attribute (from B to C say)

is required, has the attribute C at the root. These sub-trees are then severed at the original node (originally labelled with B), forming a number of disjoint trees, corresponding to each value of the attribute B. For each of these trees, the attribute C is moved to the root, and the attribute B becomes the label of the root of each sub-tree. These sub-trees can now be merged to generate a single decision tree with the appropriate root. This is shown diagrammatically in Figure 2.5.

This intelligent restructuring process makes implicit use of all the relevant training examples presented to the system whenever an attribute labelling a node is found to no longer be as important as another. Thus ID5 is able to more fully utilise all the training examples, unlike ID4 which must simply remove sub-trees when attribute changes are required. Further, ID5 with just a little more effort can produce exactly the same decision trees as generated by ID3.

The non-incremental nature of decision tree induction is of course not unique to ID3. Recent work has also addressed this same deficiency in the context of the CART algorithm. Crawford (1989) introduces a first attempt at an incremental CART algorithm by identifying those parts of a tree requiring regeneration and re-applying the algorithm to just those parts.

Theoretical considerations of decision tree induction are scarce. Goodman and Smyth (1990) provide some initial forays into a consideration of the theoretical aspects of the information-theory basis of decision tree induction algorithms like ID3. They present a model of decision tree induction very similar to that presented as the general algorithm above. This is used as a basis from which to analyse conjectures about decision tree induction (such as the utility of noisy data). Goodman and Smyth justify the use of decision tree induction algorithms and conclude that they are well-founded. Deeper theoretical analyses remain to be carried out.

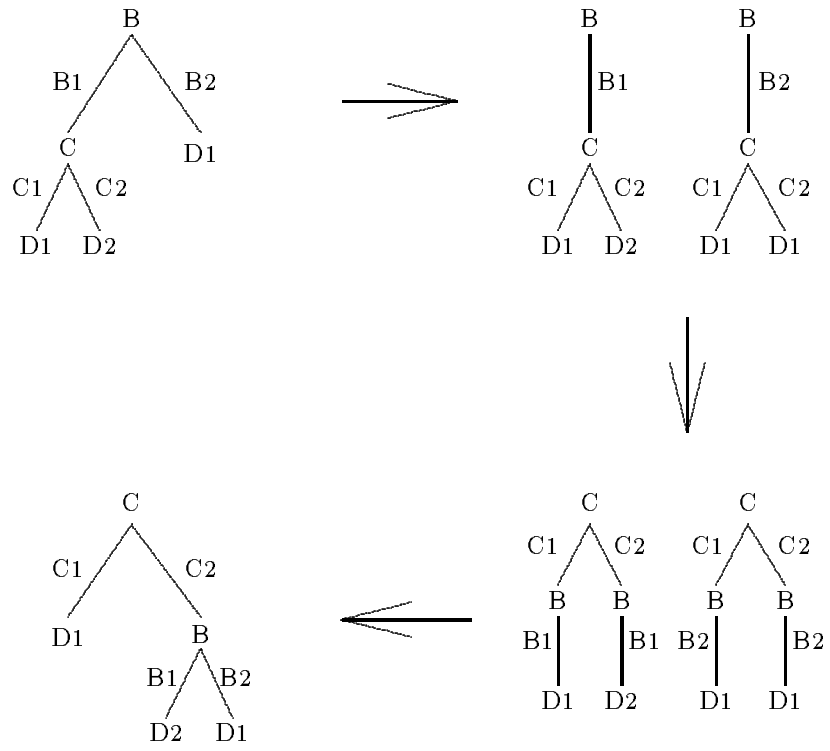


FIGURE 2.5: *ID5 (and ID5R) retains enough information about the training examples presented to it to be able to restructure a decision tree when a new example causes a change of attribute for the root node. In this example, the attribute associated with the root node is changed from B to C. For the leaf node of the tree, the values of those attributes not actually implicitly represented in the decision tree (by way of the attributes on the path to the leaf node) as they appear in past training examples must be recorded. These are not shown in this figure. Fashioned after Utgoff (1988).*

2.4.2 Issues

Decision tree induction algorithms, and algorithms for learning from examples in general, have certainly demonstrated their practicality. There are however a number of unresolved problem areas, and some are discussed below. Many of these problems relate to either shortcomings in the actual algorithm or in the

representation. Some have already been discussed above, and most have also been identified and considered by others.

Missing information. In practical applications examples often can not be completely specified, in the sense that some attributes may have missing values. Such examples may be found amongst those that are made available to a learning algorithm, or amongst those which are presented to a performance element for decision making. This is particularly the case case where the domain of the application is described in terms of a large number of attributes.

Missing attribute values can be handled in a number of ways by the learning element. For example, the distribution of values for an attribute, as found in a training set, can be used to fill in a value for an attribute with an unknown value. Similarly, the particular example can be broken up into fractional examples, each having one of the possible attribute values, and weighted according to the known distribution of values. This approach of using the known distribution of values was suggested by Kononenko, Bratko, and Roškar (1984) in the context of the ASSISTANT system. Quinlan (1986a) reports upon another approach suggested by Shapiro, in which a decision tree is built from those examples having values for the attribute in question, where the decision attribute of the decision tree is this same attribute. The original decision attribute is now regarded as just another attribute. Such a decision tree could then be used to determine a value for the attribute of the example having a missing value.

Quinlan (1986a) claims that these methods for determining unknown attribute values "give unconvincing results", and are particularly problematic when several attributes have missing values. Quinlan goes on to further consider the introduction of a new attribute value called "unknown" to handle such missing values, but observed anomalies. Quinlan then introduced the idea of "distributing" the example in proportion to the known distribution of values for that attribute. This approach requires only a simple modification to the cost function. Once an example with an unknown value for a chosen attribute has been used in this manner, it is disregarded and not passed down to any children

nodes. Such an approach has been found to provide quite satisfactory results, and has been implemented in, for example, TODAY-ES (Morley et al., 1988).

In terms of the performance element, missing attribute values can be handled by considering all paths emanating from the corresponding node in the decision tree. All decisions can be collected and some form of conflict resolution used to choose one. Quinlan (1986a) considers a similar approach and finds it quite appropriate. Alternatively, all decisions could be returned, with an indication that without knowledge of the missing attribute value, these decisions could not be distinguished.

An issue related to missing attribute values is the cost of obtaining attribute values. It might be the case that a missing attribute value can be determined, but has not yet been so, primarily because the cost of obtaining a value for the attribute is high. Such attributes should only be determined by a performance element if all other avenues for making a decision have been exhausted. Such knowledge has only occasionally been incorporated into the decision tree induction paradigm. CLS, for example, incorporated an attribute's measurement cost in its selection criterion, and a more recent example can be found in Tan and Schlimmer (1990).

Severe limitations on the use of a decision tree occur if such expensive attributes appear early in the tree structure. For example, if such an attribute labelled the root node of the tree, then a value for it must always be determined before that decision tree can be used. Such an observation leads one to question the adequacy of the decision tree for representing knowledge.

Yet another related issue is the problem of missing branches (which is also considered by Cheng et al. (1988)). During the step of partitioning a training set according to some categorical attribute, not all values of that attribute may be represented in the training set. Thus a decision tree based upon this partition will fail to provide a decision for relevant examples with these unrepresented values. In this sense, the decision tree induction algorithm will

generate trees which are too specific. Cheng et al. further identify the irrelevant values problem, which again leads to over-specialisation by requiring that a branch be created for every value of an attribute, irrespective of the relevance of the attribute values. Their solution is to consider the information content of attribute-value pairs, and then, for the selected attribute, create branches for those most informative values, lumping the rest of the values together to form a default branch. This algorithm, identified as GID3, empirically outperforms ID3 on a number of measures.

Adequacy of representation. One of the important goals of knowledge acquisition is to produce a knowledge base for use in an expert system, with the knowledge represented in an easily accessible form. The knowledge represents a model of the domain from which it was generated. A common criticism of decision tree induction algorithms is that they induce opaque tree structures. Whilst the algorithms generally guarantee to produce correct models (ignoring the issue of noise), it is observed that the resulting trees contain too little conceptual structure (Arbab and Michie, 1985). This manifests itself in the guise of an expert puzzling over the significance of various paths through the tree.

Limited generality. A decision tree is correct if it correctly classifies the training data. However, a correct decision tree may not necessarily reflect even some of the simplest generalisations that can be made from the training set. For example, consider a decision tree constructed from training examples involving just three attributes: A, B, and C. Just one attribute can be chosen for the root node of the tree—suppose it is attribute A. Suppose further that all examples in the training set for which attribute B has the value 2 have the same decision. This fact can not be easily represented in the decision tree. A solution to this problem is presented in C4, Quinlan's successor to ID3. In C4 a single ID3 decision tree is converted to rules, and then manipulated as rules in order to gain generality.

Converting a given decision tree to an equivalent set of rules is a straightforward task, and one which is considered in more detail in the following chapters. The inverse operation of transforming a given, arbitrary rule set into a decision tree is, in general, impossible. Rules are often found to be a more natural representation in terms of human perception. Further, individual rules can (ideally) be reviewed and modified in isolation. Thus, rules generated from a decision tree are more accessible (to the domain expert).

Size of decision tree. There is little doubt that smaller tree structures can be visually appealing. The larger structures that often arise when many attributes and many examples are involved can be daunting to the domain expert. Whilst the domain being modelled may be complicated, human understanding of, and appropriate functioning within, a particular domain begins with conceptually simple models. The structured induction approach addresses this problem by considering sub-models of a domain, which form just a part of the whole model. Also some of the decision tree induction algorithms presented here have addressed this problem by introducing heuristics based upon human understanding of the resulting trees (like linearity).

Replication. Pagallo (1989) also identifies a problem with the decision tree representation, referred to as the replication problem. With the basic decision tree structure there is no easy way to cache common sub-trees. Common sub-trees may arise quite naturally and correspond to the situation where a body of knowledge may be applicable in different situations. The different situations correspond to different combinations of attribute values, and may result in the common body of knowledge, represented as a sub-tree, being duplicated. This situation is alleviated in C4, for example, by the process of appropriately dropping conditions from the rules generated from a decision tree (Quinlan, 1987b). Many redundant conditions and rules can be effectively eliminated in this way. Pagallo presents a more flexible approach working with Boolean attributes and looking for opportunities to combine attributes, effectively allowing combinations of attributes to label tree nodes. Such combinations of attributes are in

fact treated as new attributes and the original training set is augmented with these new attributes.

A related restriction fundamental to the types of decision tree induction algorithms presented here is the association of single attributes with the nodes of the tree. Since each branch emanating from a node corresponds to a single value of the attribute which labels the node, the common sub-tree structures cannot be cached if the strict structure of the tree is to be maintained. Also, since only one attribute ever labels a node, we are restricted to simple tests.

Noisy data. The decision tree induction algorithm is compromised in the presence of noisy data. Noisy data is defined as data containing incorrect attribute values or incorrect decisions. A common characterisation of this problem is the training set containing examples having a common decision with a very small number of exceptions. These exceptions might be regarded as noise in the training set, and thus ignored. Such an approach to handling noise in this fashion is identified as tree pruning and is handled differently in C4 and CART, for example. A recent empirical study of decision tree pruning methods identifies a number of methods that perform well (Mingers, 1989a). This study also concludes that there is "no significant interaction between the creation [of a decision tree] and pruning methods".

A problem of inconsistent examples may also arise in constructing decision trees. This is identified as noise or as indicating a shortage of appropriate attributes. Under the assumption that it is noise, pruning is often used to eliminate it. For example, if 9 objects have decision D1 and a 10th has decision D2 then presumably we could say that the D2 decision is noise.

However, it is not always the case that inconsistencies in the training examples result from noise. It may be the case that the attributes used to describe the examples are inadequate in some cases, indicating that more attributes are required in order to produce appropriate decision trees. Most induction algorithms rely heavily upon the sufficiency of the supplied attributes to appropriately describe the examples.

Research addressing this and the related problem of the supplied attributes being inadequate, or inappropriate, for describing the final concepts, typically investigates techniques for constructing new attributes (see Matheus and Rendell (1989) and Pagallo (1989) as examples of recent efforts). Such innovation is extremely difficult to automate.

The problem of training subsets with apparent small exceptions to the general rule is related to the so called "problem of small disjuncts" (Holte, Acker, and Porter, 1989). An important observation here is that paths through a decision tree which correspond to relatively few of the training examples have higher mis-classification rates since they are based upon too few examples. Approaches to tree pruning generally target for removal those leaves of the decision tree corresponding to the fewest training examples. However, this may remove those paths through a decision tree corresponding to genuinely exceptional cases. This problem remains for further research.

Adequacy of selection criterion. Mingers (1989b) observed that the choice of selection criterion has little bearing upon the accuracy of the resulting decision trees. Thus, a selection criterion which tends to produce decision trees satisfying some heuristic requirements (like linearity or minimal depth) can be chosen to suit the needs of the particular application. This provides quite a degree of flexibility in constructing decision trees.

Another issue of concern in relation to selection criteria is their inability to always provide a single choice. Often, arbitrary decisions must be made. The following chapters further identify this problem and investigate how it may be addressed.

2.5 THE GENERIC ALGORITHM

A family of decision tree induction algorithms has been presented in this chapter. In the following chapters reference to the "decision tree induction algorithm" is to be interpreted as a reference to a generic algorithm which implements the divide-and-conquer technique introduced with the following assumptions, unless otherwise explicity stated. Candidate partitions are assumed to be constructed by considering single attributes as does ID3, and thus candidate descriptions consist of a test of an attributes value. Any selection criterion (or cost function) can be employed by this generic algorithm, but explicit reference will be made to the information-theoretic cost function. A homogeneous training set will be the assumed termination criteria.

The acronym DTIA will be used to refer to this generic decision tree induction algorithm.

3 Experiments in Decision Tree Induction

This chapter identifies a number of properties of a decision tree induction algorithm and provides experimental support for them. These experiments provide a basis from which a technique for improving upon the induced decision trees is developed (Chapter 4). An empirical approach is adopted so as to gain a measure of the performance of such learning systems when applied to actual data. Those aspects of the algorithm considered here have received scant attention in the decision tree induction literature.

For these experiments, use is made of the Australian Resources Information System (ARIS) database (Walker, Cocks, and Young, 1985), being a sizable collection of data, recording numerous environmental features for all regions of Australia. A training set of expertly classified examples, drawn from this data, has previously been used to construct a linear model for the prediction of cattle grazing viability in rangeland regions of Australia (Cocks, Young, and Walker, 1986). This same training set is used in the experiments described here to construct decision trees for the prediction of grazing viability.

Whilst the ARIS domain is used to guide the direction taken in these experiments, many of the experiments are repeated using data from a very different domain—that of credit approval. Such additional experiments provide a broader perspective.

The data used in these experiments is complete, in that no attribute values are missing. Whilst missing attribute values is an important issue itself, it is not considered in this thesis. The data here is also assumed to be, and believed to be, accurate. Once again, the important issue of noisy data is beyond the scope of this thesis. The focus is upon the use of decision tree induction with complete, noise free, but relatively small training sets.

Experiments, such as those presented here, provide insights into the sensitivity of the decision tree induction algorithms—different results are obtained from simple variations to the way in which the algorithm is used. Credence is given to the view of decision tree induction as a knowledge acquisition tool rather than a machine learning tool in its own right. That is, a decision tree induction algorithm, as a tool, can provide the knowledge engineer with a collection of prototype knowledge-bases. These knowledge-bases are not the end product, but rather, the starting point of the knowledge engineering process.

Three series of experiments will be described. The first concerns properties relating to the precision and the bias found in selection criterion. The second deals with the pruning of decision trees, and the final introduces the idea of combining decision trees. The two domains of application used in these experiments are first described.

3.1 THE RANGE DATABASE

The ARIS database is a continental-scale geographic information system developed for land use planners (Walker, Cocks, and Young, 1985). It consists of objects, each recording information about an approximately 700 square kilometre rectangular region. Australia divides into 11,109 regions, 8413 of which are in the rangeland areas of Australia. These rangeland regions form the Range database as used in the following experiments. For each object, values of some 40 attributes are maintained, including the dominant soil type, the type of vegetation, the distance to the nearest seaport, and a number of moisture indices.

3.1.1 Constructing a Linear Model

The Range database has been used for predicting the viability of the pastoral use of land (sheep or cattle grazing) in the Australian rangelands (Cocks, Young, and Walker, 1986). A linear model was constructed based upon 106 representative objects especially chosen from the Range database. The objects in this training set, collectively referred to as the T106 training set, were classified by an experienced agricultural scientist (the domain expert).

The domain expert initially chose as the training set 80 regions known well to him. They were selected so that the training set contained examples of a range of viability ratings. In order to ensure that the training set was representative, an exercise was carried out whereby the whole of the Range database was divided into 15 groups, each group sharing common features. Changes were then made to the membership of the training set in order to obtain approximately the same proportion of objects in each of the 15 groups in both the training set and the Range database. This augmented training set of 106 regions is the T106 training set.

Three primary attributes were devised by the domain expert to be used in the construction of the model: Return is the expected annual gross revenue; Cost is the expected annual total costs; and Variability is an indicator of the variability in the number of stock carried from year to year (Cocks, Young, and

Walker, 1986). The viability rating was expressed as a linear combination of these attributes:

$$\text{Viability} = a_0 + a_1\text{Return} + a_2\text{Cost} + a_3\text{Variability}.$$

These primary attributes were themselves expressed as linear combinations of other data attributes available in the database.

The expert provided values for the viability rating (a value between 1 and 100) and for each of the three primary attributes for each object in the T106 training set. A regression was carried out in order to determine the values of the coefficients of the above linear equation. The expert then had the opportunity to adjust those of his original values, both the viability rating and the three primary attributes, which he felt needed adjustment, after which another regression was carried out. The model was thus refined until it gave predictions for viability which corresponded (with 88% agreement) to the expert's opinion for all the objects in the training set. The model was then applied to the rest of the database to provide predictions for the grazing viability of each region in the Range database. These predictions were found to be acceptable to the domain expert.

Constructing a model using linear regression relies upon a number of assumptions, including the assumptions of ordered and continuous data. Data containing integer attributes and ordered categorical attributes are suitable but unordered categorical attributes are not. A further restriction on the use of linear modeling is the assumption of a continuous relationship between the order of each attribute and the decisions being made. Discontinuities in this relationship produce misleading results.

Decision tree induction provides an alternative approach to building models. In general, decision tree induction can be applied to less structured data than regression can. Decision trees can form models from integer and ordered categorical data, as regression can, in addition to unordered categorical data.

Also, decision tree induction does not require the assumption of a continuous relationship between attributes and decisions. In this sense, decision trees provide a richer representational paradigm than that afforded by linear regressions.

Decision tree induction takes advantage of continuity properties for appropriate data types (e.g., integers). Decision tree induction algorithms typically choose a point between represented values of an integer attribute when such an attribute is chosen by the selection criterion.

For the experiments described here the predictions of viability (i.e., the decisions) of the linear model (referred to as just the Model) form the basis on which decisions made by the induced decision trees are judged. The same training set, T106, is used for inducing decision trees in the experiments. The experiments illustrate the ability of decision tree induction to accurately capture the structure of this data.

This training set records values for the attributes selected by the expert as being relevant to the problem of predicting grazing viability. These are the predominant soil type (Soil), the form of the upper and lower stratum vegetation (UVeg and LVeg respectively), the distance in kilometres to the nearest seaport (DPort), and three moisture indicators: the average weekly moisture index for the wettest consecutive 13 weeks of the year (AWMIH); the average weekly moisture index from November to April inclusive, covering the Australian summer (AWMIS); and the average weekly moisture index from May to October inclusive, covering the Australian winter (AWMIW). The attributes Soil, UVeg, and LVeg are categorical attributes, with the symbolic categories replaced by numeric values. In the database the Soil attribute has 30 possible values, UVeg and LVeg have 50 and 41 possible values respectively. The moisture indicators take integer values in the range 0 to 100. The distance, regarded as an integer, ranges up to approximately 1500 kilometres. Thus, both categorical and integer attributes are represented.

Because of the possibly misleading accuracy implied when using real numbers for the viability ratings (as obtained from the Model), 4 decision classes

(corresponding to ranges of viability) have been used in the following experiments. The expert had the freedom to assign a value between 1 and 100 to the decision attribute of each object, and the resulting Model assigned values ranging from -8.1 to 90.3. For our purposes, the decisions are simply partitioned into the 4 decision classes of *Very Low* (abbreviated as *VLow*), *Low*, *Medium*, and *High*, using ranges supplied by the domain expert as given below. Agreement between the Model and a decision tree refers to agreement within these classes.

-8.1–14.9	⇒	*VLow*	30–49.9	⇒	*Medium*
15.0–29.9	⇒	*Low*	50–90.3	⇒	*High*

3.1.2 The T106 Training Set

The actual composition of the objects in the T106 training set is summarised below.

The training set exhibits a common deficiency of small training sets comprised of categorical attributes having large ranges of values: incompleteness in the various attribute values. The training set contains examples of only 9 of the 30 possible values for Soil, for example. There are only 17 of the possible 50 values for UVeg represented in T106, and only 15 of the possible 41 values for LVeg. As a consequence, the coverage of a resulting decision tree must be adversely affected. Integer attributes do not suffer from this anomaly, due to a known relationship between the values of such attributes. The branching associated with integer attributes are binary splits, covering all possible values of the attribute. Nevertheless, the values for DPort range from 121 km to 1225 km, there being 99 distinct values. AWMIH ranges from 9 to 95, with 46 distinct values in this range. Similarly the AWMIS attribute ranges from 8 to 83 with 30 distinct values, and AWMIW ranges from 4 to 43, with 25 distinct values. The decision associated with each object, indicating the grazing viability for the corresponding region, is represented by 14 objects classified as *VLow*, 16 as *Low*, 40 as *Medium* and 36 as *High*.

3.1.3 Constructing A Decision Tree: T106DC

A decision tree induction algorithm using the information-theoretic cost function was applied to the T106 training set. The values of the cost function, $E(A)$, associated with each of the attributes when choosing the initial root of the decision tree is given for illustration in Table 3.1. Note that the value of $E(A)$ for the integer attributes is the best obtainable by any binary split on the integers. For these best binary splits, the split point is also given in the table. Split points for categorical attributes correspond to each of the individual values of that attribute.

Attribute	Cost $E(A)$	Split Point
Soil	62.94	
UVeg	68.62	
LVeg	77.69	
DPort	102.80	925
AWMIH	116.24	20
AWMIS	119.83	31
AWMIW	105.62	11

TABLE 3.1: *Values for $E(A)$, the cost function, for each of the attributes in the T106 training set, together with the corresponding mid-point split for the integer attributes. Soil, having the minimum cost, will be chosen as the root of the decision tree.*

The complete decision tree constructed is shown in Figure 3.1. The attribute Soil, having the minimum value of $E(A)$, labels the root of the decision tree. There are 9 non-trivial branches from this root; the branches labelled with 2 and 12 have been combined as they lead to the same sub-tree. Each branch corresponds to one of the 9 values for the Soil attribute as found in the T106 training set. For those values of Soil which do not appear in T106 an implicit

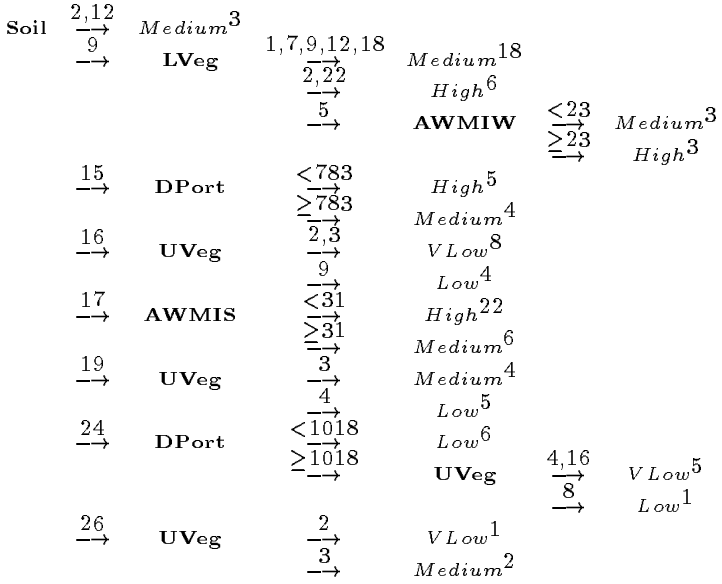


FIGURE 3.1: *Decision tree T106DC.*

branch leads to a sub-tree consisting of just the leaf node *Null*, indicating no decision.

Each leaf node of the decision tree is labelled with a decision and a superscript indicating the number of training set objects corresponding to the leaf node. For example, there are 3 objects in the T106 training set which have a value of 2 or 12 for the Soil attribute, each of which has an associated decision of *Medium*.

Each path through the decision tree to a leaf node corresponds to an If-Then type rule, and is readily convertible to this form for use in a rule-based system. An example of such a rule is:

If Soil=24 and DPort$\geq$1018 and UVeg$\in \{4, 16\}$,
Then Class = *VLow* (5 examples)

The decision tree so constructed, referred to as T106DC, can now be applied to all of the objects in the Range database. For each object, the application of the decision tree begins from the root node. A value for the attribute labelling the root node is obtained from the given object. The branch corresponding to this value is traversed, leading to another node. If this node is a leaf node, then the decision which labels this node results. Otherwise, the process is repeated,

Agree	Mild	Moderate	Strong
4237 (71.5%)	1581 (26.7%)	106 (1.8%)	0

TABLE 3.2: *A comparison of the decisions assigned to objects in the Range database by the T106DC decision tree and the Model. The Model and T106DC agree, or at worst mildly disagree in 98.2% of the cases in the database. The percentages exclude those objects not covered by T106DC.*

with the current node acting as the new root node, until a leaf node is reached, or until there is no branch corresponding to the particular attribute value. A decision of *Null* is made in the latter case.

T106DC is able to make a decision for only 5924 (or 70.4%) of the objects in the Range database (i.e., it has a coverage of 70.4%). The inability of this decision tree to cover all objects results from the absence of objects in the training set with particular values for the categorical attributes, as noted above.

A comparison of the decisions produced by the Model and T106DC is summarised in Table 3.2—it is observed that the decision tree agrees with the Model on 71.5% of the cases (excluding those objects not covered by T106DC). To assist in identifying the degree to which a decision tree and the Model disagree, three degrees of disagreement are introduced. With the ordering on the decisions being $VLow < Low < Medium < High$, two decisions for one object **mildly disagree** if the two decisions are neighbours in this ordering. A **moderate disagreement** occurs when the two decisions are one class apart, and a **strong disagreement** occurs when the two decisions are two classes apart (i.e., *VLow* and *High*). From Table 3.2 it can be seen that the Model and T106DC agree or only mildly disagree on 98.2% of the objects covered. This is quite a satisfactory result. As a practical application, decisions from this decision tree could be used to identify regions which should be considered for grazing.

3.2 THE CREDIT DATABASE

To gain a measure of the generality of the observations from the experiments which follow, each experiment is repeated using data drawn from a very different domain. The Credit database contains 602 records, each containing the information supplied by a person when applying for a loan for the purchase of a motor vehicle. Associated with each record is the decision made by a credit expert. This data has been supplied by an Australian financial institution.

The Credit data is more representative of the type of data actually available for induction. There is no clear underlying linear model as there was for the Range data. Since trees built from the Credit data are modelling the real world in a stronger sense than was the case for the Range data, we can expect more difficulty in obtaining high levels of accuracy.

For each applicant, five relevant items are recorded: The type of motor vehicle to be purchased (CarType); the applicant's occupation (Occup); the cash price of the purchase (CashPr); the amount the purchaser is willing to place as a deposit on the motor vehicle (CashDp); and the age of the applicant (Age). CarType and Occup are categorical attributes, having 6 and 9 distinct values respectively. CashPr is an integer ranging from 1450 to 65,000, CashDp is an integer ranging from 0 to 16,000, and Age is an integer ranging from 18 to 69. The decision attribute is a simple yes/no valued attribute.

For the purposes of the experiments presented here, random samplings of the Credit database have been generated, and used as training sets. These training sets range in size from 50 to 150. Reference will be made, for example, to the Credit 100a training set, which is one of the training sets containing 100 objects. Other training sets of size 100 are 100b and 100c. In all, 12 training

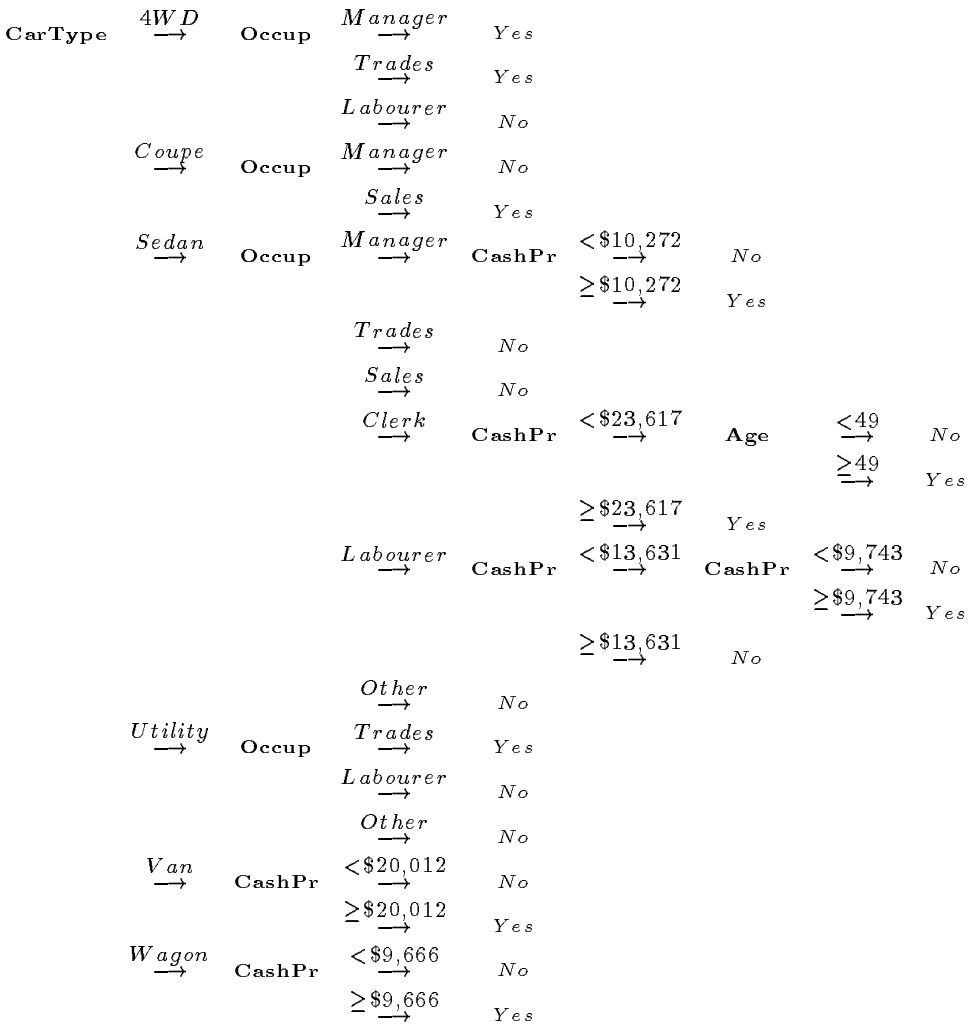


FIGURE 3.2: *Decision tree 050aDC, generated from the Credit database.*

sets were used: 050a, 050b, 050c, 100a, 100b, 100c, 125a, 125b, 125c, 150a, 150b, and 150c.

Decisions of Yes, No, or *Null* will be made when applying a Credit decision tree to the Credit database. Comparisons between the decisions produced by the decision tree and the actual decision made will thus consider only agreement and disagreement.

Figure 3.2 illustrates the type of decision tree induced (in this case, induced from the 050a training set). For example, managers applying for a loan to purchase a sedan will only have the loan approved if the sedan is a good

investment, as indicated by the cash price of the vehicle. Other interesting rules include the fact that sales persons purchasing a coupe will receive a loan, whereas if the sales person was to purchase a sedan, then the loan would be declined. The performance of this decision tree is measured as 87.9% coverage and 60.7% agreement.

For brevity, other Credit decision trees will not be illustrated. It is the performance of the decision trees, as measured by their coverage and accuracy, that is used as the criteria for assessment.

3.3 THE SELECTION CRITERION

The cost function (or selection criterion) is at the heart of a decision tree induction algorithm. Properties of the cost function concerning precision and bias are considered first. Identification of these properties and the supporting experimentation presented here leads to a better understanding of the behaviour of the induction algorithm.

3.3.1 Precision of the Cost Function

In using a particular cost function there is an expectation that the "best" choice will be made in deciding between alternative partitions of the training set. That is, a good cost function should have the property that it always makes the best choice. Whilst "best choice" is difficult to define, for our purposes this will be measured by how well the resulting decision tree models the domain data, in terms of coverage and accuracy.

If a cost function can be relied upon then overriding it with some other choice should lead to decision trees with a performance no better than, and in general worse than that of the original decision tree.

This hypothesis is first tested in the case of the Range data. In building the decision tree T106DC, the attribute Soil, being chosen to label the root node, has a cost associated with it of 62.94. The attribute UVeg has a cost of 68.62, making it a close second choice (see Table 3.1). The attribute UVeg was used instead of Soil in constructing a new decision tree: T106DU. Choices for all nodes other than this root node were made by the cost function. The application of this resulting decision tree to the Range database is summarised in Table 3.3. The decisions made by this decision tree are considerably worse

	T106DU	c.f. T106DC
Agreement:	58.5%	−635
Mild Disagreement:	34.0%	+515
Moderate Disagreement:	6.5%	+297
Strong Disagreement:	1.0%	+58
Coverage:	73.2%	+235

TABLE 3.3: *The decisions provided by the T106DU decision tree are compared to those given by the Model, and then to the performance of T106DC.*

than those made by T106DC, with a large decrease in agreement, and a large increase in disagreement.

Further experiments, replacing the root node of the decision tree with the third and fourth best choices, according to the cost function, lead to similarly poorer decision trees.

Repeated experimentation with the Credit data demonstrates similar behaviour. Training sets 050c, 125b, 125c, and 150b, for example, produced decision trees with poorer coverage and accuracy when the second best attribute was used as the choice for the root node. However, a number of other experiments with the Credit data (for example, 100b and 125a) produced decision trees of the same or slightly better performance when the second best attribute was used for the root node. The overall trend indicates that the choice of attribute made by the cost function is a good choice.

These results confirm that the choice made by the information-theoretic cost function may not always be the best. However, overriding the cost function can lead to decision trees with poorer performance or at best with small improvements.

3.3.2 Indeterminancy of the Selection Criterion

A second desirable property of a selection criterion is that it always a selection. It is inevitable though that a selection criteria based upon a numeric cost function may give rise to equal minimum costs, resulting in indecision. This is

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aDU	96.5	57.0	125aDU	92.2	67.4
050bDC	91.5	55.2	125bDC	95.7	63.9
050bDU	92.9	59.2	125bDU	94.4	62.0
050cDC	97.3	59.6	125cDC	92.2	64.0
050cDU	93.5	57.0	125cDU	89.2	62.8
100aDC	94.2	62.6	150aDC	96.7	64.4
100aDU	97.8	61.5	150aDU	94.5	66.6
100bDC	92.5	66.6	150bDC	97.7	66.5
100bDU	92.5	66.6	150bDU	97.2	62.6
100cDC	98.3	62.5	150cDC	96.5	68.8
100cDU	96.3	62.9	150cDU	97.7	68.2

TABLE 3.4: *The Credit experiments further illustrate the trend that the chosen attribute is usually the better choice. This is not always true though, as in the case of the 125a decision trees.*

confirmed in the case of the information-theoretic cost function where, in constructing T106DC, a number of attributes tied exactly for the minimum cost. Table 3.5 illustrates an occasion where 5 of the attributes under consideration were tied. In constructing decision trees from the Credit data, ties for best attribute were observed on each application of the induction algorithm to the various training sets. A selection criterion should have the property that when ties occur, any choice will lead to an equally good decision tree.

An arbitrary choice from amongst those partitions with equal minimum cost can be made. For the purposes of the following experiments, the choice will be based upon a ordering of the attributes. Two orderings are introduced,

Attribute	Cost $E(A)$	Split Point
UVeg	0.00	
LVeg	0.00	
DPort	0.00	766
AWMIH	0.00	27
AWMIS	7.21	9
AWMIW	0.00	20

TABLE 3.5: *An example of a minimum cost conflict arising during the construction of the T106DC decision tree. The cost function appears to be indicating that any one of the five zero-cost attributes could be chosen to produce an equally good decision tree.*

one listing categorical attributes before integer attributes, and the other listing integer attributes before categorical attributes.

By choosing an integer attribute over a categorical attribute it is expected that the coverage of the resulting decision tree will increase. A split on an integer attribute will account for all possible values of that attribute, whilst a split on a categorical attribute will only account for those attribute values found in the current training set. The ordering within the integer attributes, and within the categorical attributes, remains arbitrary.

In building decision tree T106DC, an attribute ordering of Soil, UVeg, LVeg, DPort, AWMIH, AWMIS, and then AWMIW, was used, giving preference to categorical attributes. A new decision tree, T106DI, can be constructed by always choosing integer attributes in preference to categorical attributes whenever their costs are the same. A preference ordering of DPort, AWMIH, AWMIS, AWMIW, Soil, UVeg, and then LVeg can be used.

T106DI, like T106DC, has Soil as the root node, as there is no other attribute with the same (minimum) cost. The final tree differs from T106DC in only 3 of the 9 branches emanating from the root (the starred (*) branches of Figure 3.3). Of these, 2 represent changes of choice from the categorical attribute UVeg to the integer attribute DPort, and the other from UVeg to AWMIH.

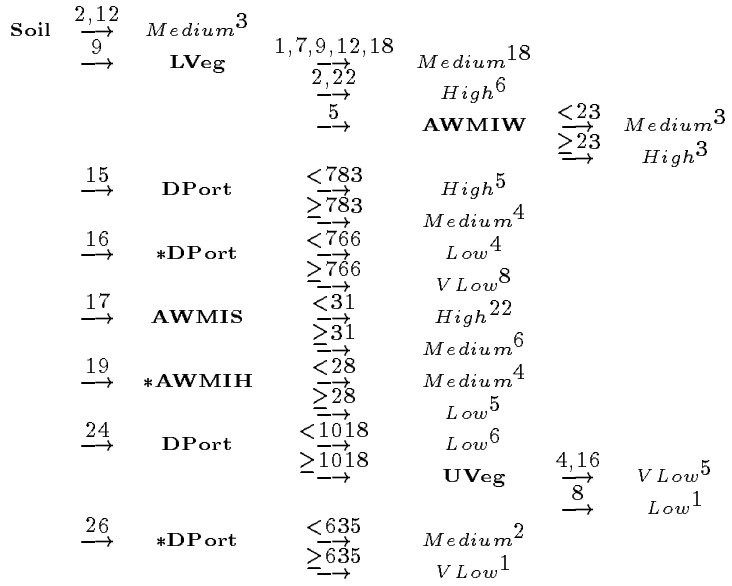


FIGURE 3.3: *The decision tree T106DI constructed by favouring integer attributes over categorical attributes. The differences from T106DC are identified with a *, where categorical attributes are replaced by apparently equally good integer attributes.*

Table 3.6 presents a comparison of the decisions made by T106DI with those made by the Model and T106DC. The coverage of T106DI has increased by almost 20% over the coverage of T106DC due to the preference given to integer attributes. The decisions made by T106DI for these extra objects, though, mostly disagree with the decisions made by the Model. Of the 1164 extra objects covered, only 358 (30.8%) are in agreement with the Model. Nevertheless, there are still no strong disagreements, and only 42 new moderate disagreements. The combined agreement/mild disagreement accounts for 97.9% of the decisions made by T106DI, comparable to T106DC.

The Credit experiments produce results which indicate that when integer attributes are chosen over categorical attributes, the coverage increases (as demonstrated in all but 2 cases—100c and 150c—where there is no change in coverage). Also, the accuracy of the decisions produced by the resulting decision trees has decreased, although in most cases only marginally. Refer to Table 3.7.

	T106DI	c.f. T106DC
Agreement:	64.8%	+358
Mild Disagreement:	33.1%	+764
Moderate Disagreement:	2.1%	+42
Coverage:	84.3%	+1164

TABLE 3.6: *The decisions made by T106DI are compared to those made by the Model. The first column details the performance of the decision tree, compared to the Model, whilst the second column indicates the changes in the actual coverage and decisions made from T106DC. Thus, whilst the actual agreement is now 64.8% of the coverage, the actual number of objects with agreeing decisions has increased by 358 over T106DC.*

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aDI	94.4	56.3	125aDI	93.7	65.8
050bDC	91.5	55.2	125bDC	95.7	63.9
050bDI	97.7	56.3	125bDI	96.7	62.2
050cDC	97.3	59.6	125cDC	92.2	64.0
050cDI	99.3	59.2	125cDI	93.4	63.9
100aDC	94.2	62.6	150aDC	96.7	64.4
100aDI	98.0	62.4	150aDI	98.3	65.7
100bDC	92.5	66.6	150bDC	97.7	66.5
100bDI	95.7	65.6	150bDI	97.8	65.4
100cDC	98.3	62.5	150cDC	96.5	68.8
100cDI	98.3	62.0	150cDI	96.5	68.0

TABLE 3.7: *The Credit experiments support the expectation that choosing integer attributes over categorical attributes, for equal costs, results in decision trees with greater coverage. Accuracy is less affected in the case of the Credit experiments than it was in the case of the Range experiment.*

Thus, where the cost function cannot choose between partitions, arbitrary choices can lead to decision trees with varied performances. Further, choosing integer attributes over categorical attributes generally results in a decision tree with greater coverage, but often with a decrease in the accuracy of the decisions made.

3.3.3 Bias Against Integer Attributes

A selection criterion should be fair in its selection of an attribute, with no arbitrary biases. However, a single cost function is typically applied to all attributes, regardless of whether they are categorical or integer attributes and regardless of the number of possible values associated with a categorical attribute. However, categorical and integer attributes are dealt with in quite different ways by the information-theoretic cost function. For integer attributes, only binary partitions are ever considered, whereas for categorical attributes, n-ary partitions are considered (with n generally greater than 2 in the data used here). Quinlan (1986a, page 100) has shown that the cost function favours partitions containing many cells—there is a strong bias against integer attributes. Quinlan (1985) deals with this bias by way of a Gain Ratio Criteria as described in Chapter 2 above.

The experiments described below confirm this bias in the cost function and further considers an empirically based alternative solution to this problem. This approach introduces the idea of treating integer attributes as if they were multi-valued categorical attributes (referred to here as pseudo-categorical attributes). This approach arose from the observation that in the Range training set the number of distinct values for particular integer attributes was of the order of the number of distinct values for the various categorical attributes. Thus, treating them as categorical will affect the bias.

Clustering or gap finding techniques can be used to discover appropriate subranges of integer values to use in categorising an integer attribute. Such methods, whilst attractive, require pre-processing of the data, and typically

work best with larger data sets. With the smaller data sets used in these experiments (typical of many real-world applications), an alternative approach was employed. The method developed here considers the distinct values of an integer attribute, as found in the training set, as the categories. The aim of the experiments presented below is to consider the value of this novel approach in handling the bias of the cost function.

3.3.3.1 Aa: Categorising Integer Attributes

The coverage of the resulting decision trees should decrease when integer attributes are treated as categorical—whenever a pseudo-categorical attribute appears in the tree, only a (possibly small) subset of all the possible values will be represented there. If the training set is representative, and the range of integer values is small (such as 0 to 100), then we would expect that only the less frequent values will be missing. Given these observations, an integer attribute should only be considered as a pseudo-categorical attribute when the number of distinct values of that attribute occurring in the training set is significantly less than the size of the training set. Such a restriction avoids decision trees which have large branching factors—having a (non-*Null*) branch for each of the possible values of the attribute.

The three AW attributes of the Range database can be treated as pseudo-categorical: attribute AWMIH has 46 distinct values, AWMIS has 30, and AWMIW has 25, in a training set of 106 objects. The integer attribute DPort is ruled out since it has 99 distinct values.

Decision tree T106Aa was constructed under such a scenario using the cost function for the choice of attribute at each node. An ordering on the attributes of Soil, UVeg, LVeg, AWMIH, AWMIS, AWMIW, and then DPort was used to remove ambiguity when ties for the minimum value of $E(A)$ arose. This ordering is the same as that used for T106DC, with the three new pseudo-categorical attributes moved ahead of DPort to maintain the bias towards categorical attributes, for comparison with T106DC.

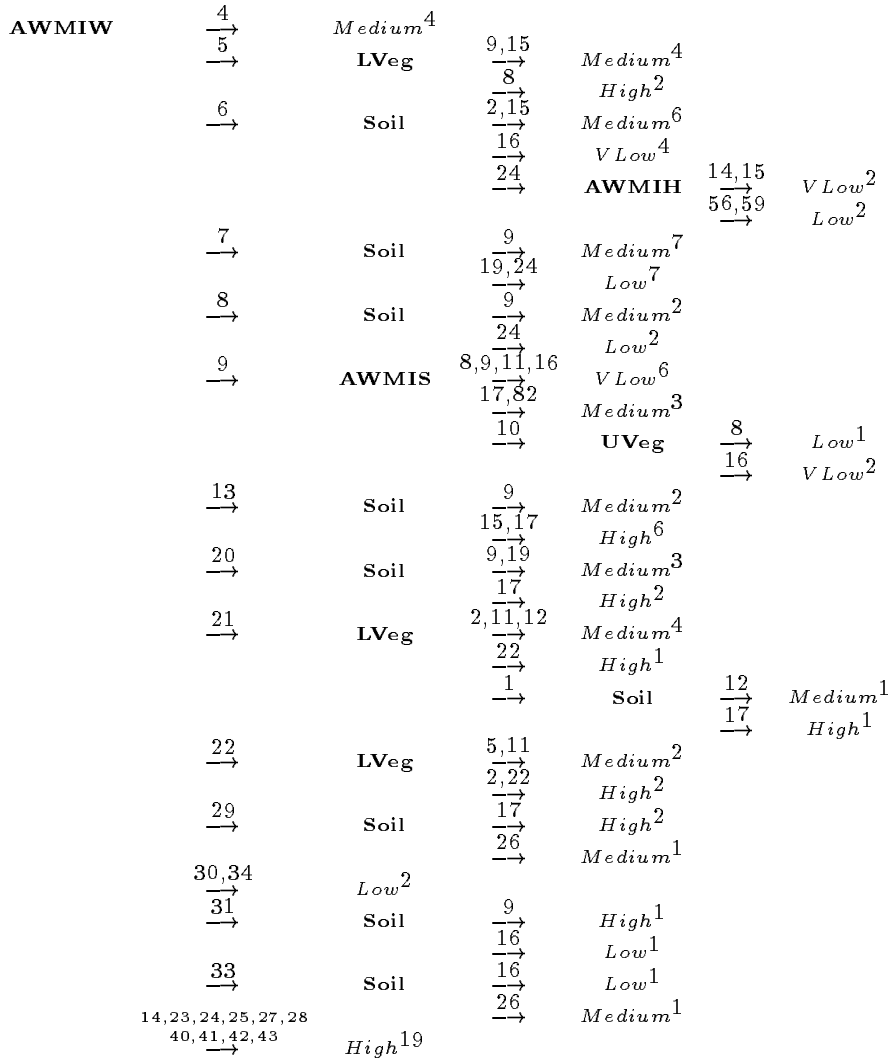


FIGURE 3.4: *Decision tree T106Aa, constructed by interpreting integer attributes as categorical attributes.*

The resulting decision tree, illustrated in Figure 3.4, has a root node labelled AWMIW, having a cost of 59.75. This is considerably smaller than the cost of 105.62 computed for AWMIW as an integer attribute in T106DC. For comparison, all the values for the costs computed in choosing the initial root node of the T106Aa decision tree are given in Table 3.8. The costs now associated with AWMIW and AWMIH are lower than the cost associated with Soil.

Applying T106Aa to the Range database results, as expected, in a dramatically reduced coverage of only 2963 objects (35.2%). Of this coverage, the

Attribute	Cost $E(A)$	Split Point
Soil	62.94	
UVeg	68.62	
LVeg	77.69	
AWMIH	60.91	
AWMIS	82.62	
AWMIW	59.75	
DPort	102.80	925

TABLE 3.8: *Values for $E(A)$, the cost function, for each of the attributes in the T106 training set with the AW attributes regarded as categorical rather than integer. AWMIW has the minimum cost and is chosen as the root of the T106Aa decision tree.*

	T106Aa	c.f. T106DC
Agreement:	48.4%	−2803
Mild Disagreement:	30.6%	−673
Moderate Disagreement:	18.6%	+446
Strong Disagreement:	2.3%	+69
Coverage:	35.2%	−2961

TABLE 3.9: *The decisions provided by T106Aa are compared to those given by the Model, and then to the performance of T106DC.*

ratio of agreement to disagreement with the Model is approximately 1:1. An analysis of the disagreements provides a little more encouragement. Of the three degrees of disagreement, 59% are mild (the Model making a decision of *High* and T106Aa a decision of *Medium*), whilst 36% are moderate (Model deciding *High*, T106Aa deciding *Low*), and only 5% are strong (Model deciding *High*, T106Aa deciding *VLow*). The combined agreement/mild disagreement accounts for almost 80% of the coverage.

Results from the Credit experiments agree with these observations. For these experiments only the attribute Age was considered as a pseudo-categorical attribute (both CashPr and CashDp have a large range of values). In all cases, the coverage has decreased, often quite dramatically (as in, for example, 050c where the coverage drops from 97.3% to 68.9%). Agreement is not so dramatically affected as it was for T106Aa. In half of the Credit experiments, percentage

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aAa	71.9	60.7	125aAa	81.7	64.8
050bDC	91.5	55.2	125bDC	95.7	63.9
050bAa	64.3	61.0	125bAa	80.4	65.1
050cDC	97.3	59.6	125cDC	92.2	64.0
050cAa	68.9	60.2	125cAa	87.9	61.4
100aDC	94.2	62.6	150aDC	96.7	64.4
100aAa	75.1	61.9	150aAa	80.7	67.3
100bDC	92.5	66.6	150bDC	97.7	66.5
100bAa	69.9	62.5	150bAa	80.1	69.3
100cDC	98.3	62.5	150cDC	96.5	68.8
100cAa	81.7	61.4	150cAa	78.1	67.7

TABLE 3.10: *The Credit experiments illustrate the dramatic affect upon the coverage of the decision trees when an integer attribute is treated as categorical. Accuracy has not suffered so much.*

agreement has decreased, the largest decrease being recorded for 100bA (from 66.6% to 62.5%). In a number of the experiments the percentage agreement has in fact increased, with 050b being an extreme example (from 55.2% to 61.0%). Refer to Table 3.10.

Thus, coverage and accuracy can be considerably affected by introducing pseudo-categorical attributes. Coverage in particular is dramatically reduced, as expected, with a general trend being demonstrated towards less accuracy from the resulting trees.

3.3.3.2 Ab: Growing Integer Branches

Treating an integer attribute as a multi-valued categorical attribute demonstrably decreases the coverage of the resulting decision tree—an undesirable

	T106Ab	c.f. T106Aa
Agreement:	47.3%	+783
Mild Disagreement:	30.4%	+519
Moderate Disagreement:	20.4%	+403
Strong Disagreement:	2.0%	+23
Coverage:	55.8%	+1728

TABLE 3.11: *The decisions provided by T106Ab are compared to those given by the Model, and then to the performance of T106Aa.*

consequence. However, a simple remedy is to now introduce the concept of subranges of attribute values, by way of growing the range of values associated with the branches of the decision tree.

Subranges are grown by defining a region surrounding the value labelling the branches. Mid-points will be used for this purpose: given two branches, one labelled with m and the other with n, where $m < n$ and there being no other branch labelled with l such that $m < l < n$, then compute the mid-point as $p = (m + n)/2$, and place an upper bound of $p - 1$ on the m branch, and a lower bound of p on the n branch. The branch labelled with the smallest value of the attribute is extended to cover all values down to the smallest possible value of the attribute. Similarly for the branch labelled with the largest value. Such an approach will restore the coverage of the decision tree. Its affect upon the accuracy of the decision tree is to be scrutinised.

As an example, the process of "growing" the set of values associated with particular branches in the decision tree can be applied to the root node of T106Aa (Figure 3.4). Single AWMIW values are grown to encompass ranges of values so that, for example, AWMIW$\leq$4 leads to a decision of Medium, and AWMIW in the range 34-36 leads to a decision of Low. The resulting decision tree is referred to as T106Ab.

This tree indeed has greater coverage than T106Aa. The extra decisions made, though, are not particularly accurate, with relatively large increases in all categories of agreement and disagreement (refer to Table 3.11). However, since

	T106Ac	c.f. T106Ab
Agreement:	46.8%	+353
Mild Disagreement:	31.9%	+322
Moderate Disagreement:	19.1%	+94
Strong Disagreement:	2.2%	+31
Coverage:	65.3%	+800

TABLE 3.12: *The decisions provided by the T106Ac decision tree are compared to those given by the Model, and then with the performance of T106Ab, demonstrating how growing the pseudo-categorical branches can lead to increased performance.*

percentage agreement has changed little from that of T106Aa, the increased coverage leads to the observation of an overall improvement in performance.

Other similar generalisations are possible. For example, the sub-tree in T106Aa with AWMIW=6 and Soil=24 is another case. The root of this sub-tree is labelled with AWMIH, and has two branches: one labelled with "14,15" leading to *VLow*; and the other labelled with "56,59"and leading to *Low*. A binary split, with a split point of 35, seems appropriate.

Another obvious candidate for growth in the decision tree is the sub-tree on the path AWMIW=9. If the AWMIS=10 branch of this sub-tree is ignored, then a binary split can be made with a split point of 17. Thence, for AWMIS less than 17, the decision is *VLow*, and for AWMIS greater than or equal to 17 the decision is *Medium*, with an exception when AWMIS=10, for which a decision of either *Low* or *VLow* will be made, depending upon the value of UVeg.

With these modifications to T106Aa, including those made in T106Ab, decision tree T106Ac is built. The coverage of the Range database obtained by the decision tree significantly improves upon that of T106Ab (Table 3.12). The accuracy, though, still lags behind that of T106DC.

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aAa	71.9	60.7	125aAa	81.7	64.8
050aAc	88.7	59.6	125aAc	86.2	64.4
050bAa	64.3	61.0	125bAa	80.4	65.1
050bAc	87.7	59.5	125bAc	85.5	65.0
050cAa	68.9	60.2	125cAa	87.9	61.4
050cAc	87.9	57.5	125cAc	88.9	61.9
100aAa	75.1	61.9	150aAa	80.7	67.3
100aAc	81.2	62.2	150aAc	85.2	66.3
100bAa	69.9	62.5	150bAa	80.1	69.3
100bAc	79.1	62.4	150bAc	81.9	69.8
100cAa	81.7	61.4	150cAa	78.1	67.7
100cAc	89.7	61.5	150cAc	80.9	67.6

TABLE 3.13: *Some of the coverage of the original decision tree is regained by growing the pseudo-categorical branches. The "Ac" trees are compared to the "Aa" trees.*

Once again, these findings are supported by the Credit experiments (see Table 3.13). Coverage, when compared to the "Aa" trees, has dramatically increased in 050aAc, 050bAc, and 050cAc, with less dramatic, yet still significant increases in the others. The percentage agreement has changed very little.

3.3.3.3 Ad: Changing The Root

The relatively poor performance of T106Ac can be attributed to the choice of AWMIW, a pseudo-categorical attribute, for the root node of the decision tree. Analysing the cost function in terms of its application to the pseudo-categorical AWMIW attribute and the categorical Soil attribute, it is seen that the bias inherent in the cost function has swung from being against AWMIW to being in favour of AWMIW over Soil. (Soil is an attribute with just 9 distinct values represented in the training set, compared to 25 for AWMIW). Thus, if

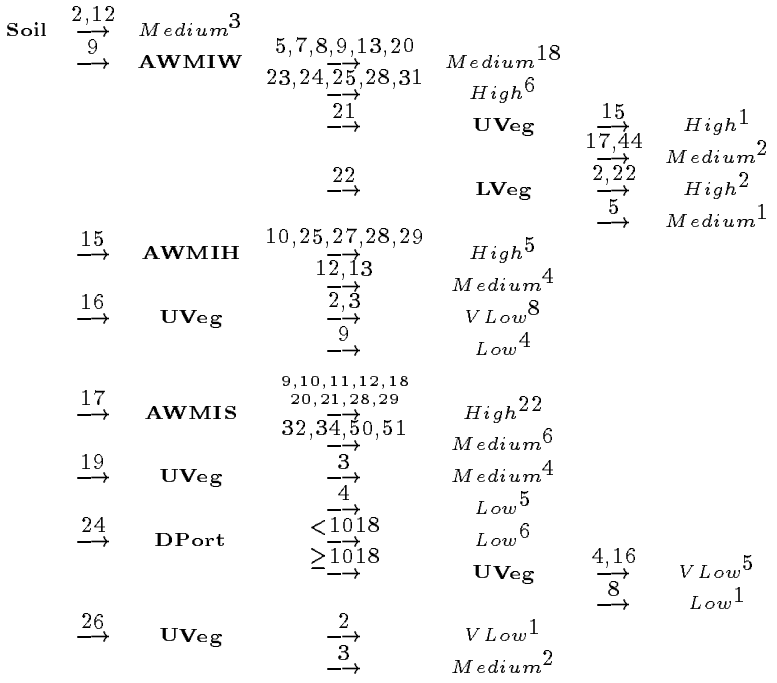


FIGURE 3.5: *Decision tree T106Ad was constructed by considering the AW integer attributes as pseudo-categorical, except that the choice for root node is overridden to be the same as that chosen for T106DC.*

Soil remained as the chosen attribute for the root of these trees, as identified originally, then better trees may result.

The next experiment repeats the process of building a decision tree, with the AW attributes treated as pseudo-categorical, but selecting Soil as the root of the tree. All other choices of attributes for the resulting T106Ad decision tree are based upon the cost function. The resulting decision tree is illustrated in Figure 3.5.

The performance of the resulting decision tree is indeed markedly improved. The coverage, being 4577 objects (54% coverage) is significantly better than that of T106Aa. It also has much greater percentage agreement (67.9%) with the Model, although it still short of T106DC's level of performance.

The pseudo-categorical branches associated with the Soil=9, Soil=15, and Soil=17 branches (Figure 3.5) can now be grown. The resulting decision tree,

	T106Ae	c.f. T106DC
Agreement:	71.0%	+285
Mild Disagreement:	27.3%	+158
Moderate Disagreement:	1.7%	+1
Strong Disagreement:	0.0%	no change
Coverage:	75.7%	+444

TABLE 3.14: *The decisions provided by T106Ae are compared to those given by the Model, and then to the performance of T106DC.*

T106Ae, in fact turns out to be almost identical in structure to T106DC. However, it improves upon T106DC in its coverage (by almost 8%), without lowering its accuracy (71% agreement), as summarised in Table 3.14.

A decision tree of greater coverage than T106DC has been produced, with the extra coverage primarily in agreement with the Model. This decision tree's coverage is approaching that of T106DI, whilst maintaining accuracy.

Less conclusive results were obtained from the Credit experiments. Each of the Credit training sets were used to construct a decision tree, where the Age attribute was treated as a pseudo-categorical attribute, but with the root node of the tree being chosen to be the same as that originally chosen in the first experiments (decision trees 005aDC, 005bDC, etc.). All Age branches were then grown to cover all possible values of that attribute. Coverage is already quite high for most of the Credit decision trees generated from the original application of the induction algorithm. In general though the "Ae" trees have slightly decreased coverage and accuracy, compared to both the original trees, and the "DI" trees. Refer to Table 3.15.

3.3.4 Summary

The series of experiments described here has illustrated some deficiencies of the selection criterion using the information-theoretic cost function. Two related issues were explored: the precision and the bias of the cost function. In inducing the decision tree T106DC it was observed that the costs calculated by the selection criterion are sometimes very similar. The issue of the precision

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aAe	91.7	58.0	125aAe	86.2	64.4
050aDI	94.4	56.3	125aDI	93.7	65.8
050bDC	91.5	55.2	125bDC	95.7	63.9
050bAe	92.7	53.6	125bAe	85.5	65.0
050bDI	97.7	56.3	125bDI	96.7	62.2
050cDC	97.3	59.6	125cDC	92.2	64.0
050cAe	93.2	60.8	125cAe	92.4	63.3
050cDI	99.3	59.2	125cDI	93.4	63.9
100aDC	94.2	62.6	150aDC	96.7	64.4
100aAe	89.9	60.4	150aAe	95.5	64.7
100aDI	98.0	62.4	150aDI	98.3	65.7
100bDC	92.5	66.6	150bDC	97.7	66.5
100bAe	77.6	63.6	150bAe	96.2	67.2
100bDI	95.7	65.6	150bDI	97.8	65.4
100cDC	98.3	62.5	150cDC	96.5	68.8
100cAe	90.2	62.1	150cAe	95.7	64.8
100cDI	98.3	62.0	150cDI	96.5	68.0

TABLE 3.15: *Coverage has been improved upon by changing the root node of the tree to that used in the original "DC" tree. In general, the "Ae" trees have lower coverage and accuracy than the "DI" trees, but improve upon the "DC" trees.*

of the numbers returned by the cost function was explored by way of choosing the second best attribute as the root node of the decision tree. It was demonstrated that inferior decision trees could be induced as illustrated with the "DU" decision trees.

A further observation made was that the cost function is not always capable of distinguishing between attributes. On many occasions, a set of attributes

Experiment	Description	Cover	Agree	Mild	Mod	Strong
T106DC	Application of the DTIA.	70.4	71.5	26.7	1.8	0.0
T106DU	Use second best attribute.	73.2	58.5	34.0	6.5	1.0
T106DI	Equally good attributes.	84.3	64.8	33.1	2.1	0.0
T106Aa	AW attributes as categorical.	35.2	48.4	30.6	18.6	2.3
T106Ab	Growing root branches.	55.8	47.3	30.4	20.4	2.0
T106Ac	Growing all branches.	65.3	46.8	31.9	19.1	2.2
T106Ad	Categorical attribute as root.	54.4	67.9	29.8	2.3	0.0
T106Ae	Growing branches in T106Ad.	75.7	71.0	27.3	1.7	0.0

TABLE 3.16: *Summary of results from the Range experiments dealing with the choices of attributes. Each decision tree is applied to the Range database, and the resulting decisions are compared to those obtained from the Model—all numbers are percentages.*

having a common, minimum cost, were identified. It was shown that decision trees differing both in terms of structure and performance can be induced, as illustrated with the "DI" decision trees.

The strong bias of the cost function against integer attributes was then identified, and a solution to this problem, turning the integer attributes into categorical attributes, was suggested (the "Aa" decision trees). For T106Aa, this resulted in one such integer attribute being chosen as the root node. When the fact that this attribute was actually continuous was used to then generalise the root (T106Ab), the results were disappointing. However, when other such generalisations were made in sub-trees (T106Ac) the results improved. If the original choice for the root node is used (Soil in T106DC), leaving the rest of the decisions to the cost function, treating the continuous attributes as pseudo-categorical, the good performance of T106Ad was achieved, although it still lacked coverage. Using the technique of growing the pseudo-categorical branches resulted in a decision tree (T106Ae) having the greater coverage without sacrificing accuracy. Its coverage is intermediate between that of T106DC and T106DI, with significantly better agreement than T106DI.

The results from the Range experiments are summarised in Table 3.16. Similar observations are made for the Credit experiments.

3.4 EXCEPTION SPLIT PRUNING

A second important concern of decision tree induction is determining just when to stop the process of partitioning the training set in generating the decision tree (the dividing and conquering). The basic approach may at first seem quite reasonable—stop when the objects in the training set have a common value for the decision attribute. However, a concern arises when a leaf of the resulting decision tree is associated with very few objects from the training set. Such a situation may indicate noise in the training set, and, if so, the resulting leaf node will be a cause of error.

Various techniques have been developed to deal with this situation. Chapter 2 includes descriptions of some of these. A common approach is to fully develop the decision tree and then to collapse the tree from its leaf nodes under certain conditions. This is often equivalent to introducing a stopping criterion to inhibit the development of leaf nodes corresponding to minorities in a training set in the first place.

The experiments presented below explore this type of tree pruning in the context of the Range and Credit domains. The concept of exception split pruning is introduced. An **exception split** is identified as one or more decision values found in a training set being represented by very few objects in that training set, in comparison to the size of the training set. The approach considered here for handling exception splits is to reclassify them in agreement with other objects in the corresponding training set. If these exceptions are due to noise, then the accuracy of the resulting trees is expected to increase with their removal. Coverage can also be affected. If the decision tree resulting from a training set containing an exception split has a categorical node as its root node, then replacing this tree with a single decision node can result in increased coverage.

The original decision tree T106DC contains a number of exception splits. One example is the set of training objects which have Soil=24 and DPort$\geq$1018

(see Figure 3.1). Of the 6 members of this set, 5 have decisions of *VLow* and 1 of *Low*. The sub-tree on the latter path, having UVeg as its root, could be replaced with a leaf node labelled *VLow*. This introduces a 16.7% error rate into this branch of the tree, or a 0.94% error rate when applied to the whole of the T106 training set. The resulting tree is no longer Tr-consistent.

The following experiments deal with pruning to a specific threshold error rate with respect to the current training set. Pruning to a threshold of 20% removes those exception splits in which the number of exceptions is no more than 20% of those objects contained in the current training set. These 20% or fewer objects will be treated as if they had the same decision value as the other 80% or more objects in the training set. The T106DC and T106DI decision trees are pruned, as are the "DC" Credit decision trees.

3.4.1 Ce: Pruning with 20% Threshold

This first experiment in pruning treats any training set with at least 80% of the objects having a single common decision as homogeneous with respect to this decision. Thus, they are not further divided. The purpose of this experiment is to empirically determine the utility of pruning.

Decision tree T106Ce results from removing the 20% exception splits from T106DC. There is only one such exception split, corresponding to the training set with Soil=24 and DPort≥1018. The sub-tree induced from this training set is replaced with the leaf node *VLow*.

The results of applying this new decision tree to the whole of the Range database are presented in Table 3.17, and are compared with T106DC. The greatest change is the increase in the number of agreements. Of the extra 351 objects now covered as a result of this pruning, more than half agree with the Model. A better performing decision tree results.

Applying this pruning technique to decision tree T106DI results in very similar improvements in overall performance. Decision tree T106Ie has greater coverage with only a minor decrease in the percentage agreement (see Table 3.23).

	T106Ce	c.f. T106DC
Agreement:	70.9%	+211
Mild Disagreement:	27.4%	+137
Moderate Disagreement:	1.7%	+3
Strong Disagreement:	0%	no change
Coverage:	74.6%	+351

TABLE 3.17: *The decision tree T106Ce removes exception splits up to the 20% level. The decisions provided by the T106Ce decision tree are compared to those given by the Model, and then to the performance of T106DC.*

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aCe	93.5	59.7	125aCe	94.0	66.8
050bDC	91.5	55.2	125bDC	95.7	63.9
050bCe	93.2	60.2	125bCe	95.7	64.2
050cDC	97.3	59.6	125cDC	92.2	64.0
050cCe	97.3	64.5	125cCe	97.2	64.4
100aDC	94.2	62.6	150aDC	96.7	64.4
100aCe	96.3	61.9	150aCe	98.8	64.0
100bDC	92.5	66.6	150bDC	97.7	66.5
100bCe	92.5	66.8	150bCe	97.7	66.0
100cDC	98.3	62.5	150cDC	96.5	68.8
100cCe	98.3	63.3	150cCe	97.2	66.5

TABLE 3.18: *Pruning a decision tree, with a threshold of 20%, improves the performance of the decision tree.*

The Credit experiments (Table 3.18) also demonstrate the improvement gained by pruning decision trees. In all cases, coverage has either increased or remained constant. The percentage of agreement has primarily been maintained with this extra coverage, with four cases demonstrating a slight decrease and two cases demonstrating significant increases in accuracy (050b and 050c).

	T106Cf	c.f. T106Ce
Agreement:	69.6%	+261
Mild Disagreement:	28.8%	+232
Moderate Disagreement:	1.6%	no change
Strong Disagreement:	0%	no change
Coverage:	80.4%	+493

TABLE 3.19: *Decision tree T106Cf removes exception splits up to the 30% level. The decisions provided by the T106Cf decision tree are compared to those given by the Model, and then to the performance of the T106Ce decision tree.*

3.4.2 Cf: Pruning with 30% Threshold

Further improvements are observed when a 30% threshold is introduced. Decision tree T106Cf is the same as T106DC with the 30% exception splits removed. Two further sub-trees of T106Ce are consequently removed. The first corresponds to the training set with Soil=17, which has a decision of *High* associated with 79% of the objects, and the second corresponds to the training set with Soil=9, which has a decision of *Medium* associated with 70% of the objects. This pruning results in a total error rate over the training set of 15.1%.

The coverage of the resulting decision tree has increased (see Table 3.19). The overall accuracy is maintained, with only a slight drop in the percentage of agreement, for an almost 15% increase in coverage, compared to T106DC.

The decision tree T106If exhibits a similar pattern, showing an increase in coverage of over 10% whilst maintaining accuracy. The Credit experiments underline the appropriateness of pruning, resulting in some quite dramatic improvements in performance (Table 3.20). In two cases, 100% coverage is achieved (in one case from an original coverage of 87%), with percentage agreement greater than that obtained with the "DC" trees.

3.4.3 Ch: Retaining Tr-Consistency

The removal of exception splits, as above, assumes that exception splits result from noise in the training set. The level of noise in a training set is not always significant. For the case when it is known or believed that the training set is

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aCf	100.0	60.8	125aCf	97.2	62.2
050bDC	91.5	55.2	125bDC	95.7	63.9
050bCf	97.7	59.7	125bCf	97.3	61.4
050cDC	97.3	59.6	125cDC	92.2	64.0
050cCf	100.0	63.8	125cCf	100.0	62.8
100aDC	94.2	62.6	150aDC	96.7	64.4
100aCf	99.5	62.3	150aCf	99.2	59.6
100bDC	92.5	66.6	150bDC	97.7	66.5
100bCf	96.8	66.4	150bCf	97.8	65.7
100cDC	98.3	62.5	150cDC	96.5	68.8
100cCf	99.2	63.1	150cCf	97.2	66.0

TABLE 3.20: *Pruning a decision tree, with a threshold of 30%, demonstrates further improvements in the performance of the decision tree.*

mostly noise free, a similar approach can be employed. The aim of this approach is to construct a Tr-consistent decision tree whilst handling exception splits in a more appropriate manner.

Again consider the Soil=24 and DPort≥1018 path through the T106DC decision tree. The training set associated with this path contains an exception split, as already shown. Instead of replacing the whole sub-tree emanating from this node with just a single value (*VLow*), a binary split is introduced, with one branch leading to the collection of exception objects. This branch is labelled with "= 8", whilst the other has the label "≠8", corresponding to an "otherwise" branch. It is only appropriate, however, to handle categorical attributes in this way. When an integer attribute is involved, coverage is already maximal.

	T106Ch	c.f. T106DC
Agreement:	70.8%	+552
Mild Disagreement:	27.7%	+291
Moderate Disagreement:	1.6%	+1
Strong Disagreement:	0%	no change
Coverage:	80.4%	+844

TABLE 3.21: *The decision tree T106Ch removes exception splits up to the 30% level but retains Tr-consistency. The decisions made by the T106Ch decision tree are compared to those given by the Model, and then to the performance of T106DC.*

By maintaining Tr-consistency it is expected that the resulting decision tree will maintain, or better still improve upon, the accuracy of the decision tree. Coverage is also expected to increase. These expectations are explored in the following experiment.

The decision tree T106Ch is built from applying this generalisation technique to the 30% threshold. The training set corresponding to Soil=17 is handled no differently from T106DC. The only generalisations introduced in T106Ch are associated with the path Soil=9 and the path Soil=24, DPort$\geq$1018. Any object in the Range database with Soil=9 and LVeg $\notin \{2, 5, 22\}$ will be decided as *Medium*, rather than just those with LVeg $\in \{1, 7, 9, 12, 18\}$. Likewise, any object with Soil=24, DPort$\geq$1018, and UVeg$\neq$8 will be *VLow* (refer to Figure 3.1).

Comparing T106Ch to T106DC (Table 3.21), it is seen that the coverage has indeed increased over that of T106DC to the same level as that of T106Cf. This extra coverage is generally in agreement with the Model, and in fact corrects some of the moderate disagreements which occurred in T106Cf. The resulting combined agreement and mild disagreement in fact accounts for 98.4% of the coverage obtained by T106Ch—the best obtained of any decision tree so far.

Both T106Ih (Table 3.23) and the Credit experiments (Table 3.22) confirm these observations, from which it is concluded that pruning together with maintaining Tr-consistency can offer significant benefits, especially if noise is known not to be a problem with the training set.

Experiment	Coverage	Agreement	Experiment	Coverage	Agreement
050aDC	87.9	60.7	125aDC	91.9	66.9
050aCh	100.0	60.5	125aCh	97.2	66.0
050bDC	91.5	55.2	125bDC	95.7	63.9
050bCh	97.7	55.8	125bCh	97.3	63.3
050cDC	97.3	59.6	125cDC	92.2	64.0
050cCh	100.0	59.6	125cCh	100.0	63.1
100aDC	94.2	62.6	150aDC	96.7	64.4
100aCh	98.0	62.7	150aCh	99.2	64.7
100bDC	92.5	66.6	150bDC	97.7	66.5
100bCh	96.8	66.9	150bCh	97.8	66.4
100cDC	98.3	62.5	150cDC	96.5	68.8
100cCh	99.2	62.5	150cCh	97.2	68.7

TABLE 3.22: *Pruning a decision tree whilst maintaining its Tr-consistency is an alternative to previous pruning methods.*

3.4.4 Summary

The series of three experiments presented here each confirm that pruning can improve the performance of a decision tree. Each improves upon the coverage achieved by the "DC" trees, with only minor decreases, if any at all, to the percentage agreement. The two exception split pruning experiments demonstrate the power of pruning with some quite impressive improvements. However, deciding when to prune and to what level to prune is extremely difficult, and remains an art.

Pruning relies upon the assumption that the training data is noisy. A novel, yet simple approach has been introduced which retains the Tr-consistency of the resulting decision trees. This approach has also been demonstrated to be effective in increase coverage, whilst maintaining agreement.

Table 3.23 summarises the results from this series of experiments as applied to the Range data. Results from the Credit experiments add support to the results obtained from the Range experiments.

Experiment	Description	Cover	Agree	Mild	Mod	Strong
T106DC	Application of the DTIA.	70.4	71.5	26.7	1.8	0.0
T106Ce	20% Exception splits.	74.6	70.9	27.4	1.7	0.0
T106Cf	30% Exception splits.	80.4	69.6	28.8	1.6	0.0
T106Ch	Retaining Tr-consistency.	80.4	70.7	27.8	1.6	0.0
T106DI	DTIA Favouring Integers.	84.3	64.8	33.1	2.1	0.0
T106Ie	20% Exception splits.	88.4	64.6	33.4	2.0	0.0
T106If	30% Exception splits.	94.3	63.9	34.2	1.9	0.0
T106Ih	Retaining Tr-consistency.	94.3	64.9	33.2	1.9	0.0

TABLE 3.23: *Summary of Range experiments dealing with exception splits. The Cover is the percentage of those objects in the Range database able to be handled by the decision tree. The Agreement and the three degrees of disagreement are with respect to the decisions assigned to the objects by the Model.*

3.5 COMBINING DECISION TREES

A deficiency of the decision tree structure becomes apparent when the issue of missing attribute values arises. If an object does not have a value for the root attribute of the decision tree, it can never be assigned a decision value. Suppose that in using T106DC an object to be classified has no value for the Soil attribute but has a value of 26 for LVeg. The training set shows that all objects with this value for LVeg have a decision of *Low*. But this object cannot be classified by the decision tree because it requires a value for Soil.

Quinlan (1986a) suggests exploring all branches from a node whenever the associated attribute has an unknown value in the object being considered. Each of these paths will lead to a leaf node eventually, and the leaf node (and therefore the decision) associated with the largest collection of objects from the training set is selected. Further work by Quinlan (1987a) has provided a more satisfactory solution to this problem by converting the decision trees into rules, and then applying statistics to test whether any of the conditions of the rules can be dropped.

The experiments presented earlier in this chapter suggest an alternative solution to this problem. This solution is to construct multiple decision trees, and to then combine them. Such a solution, however, introduces the problem of dealing with conflicting decisions. Each of the decision trees constructed will be Tr-consistent, and the combination of the decision trees is thus also Tr-consistent. When working on previously unseen objects however, different decision trees can make different decisions for the one object. Such conflicts arise whenever an object is similar to two objects of the training set which have different decisions associated with them, but for which there are no exact examples contained in the training set. Techniques will be required to deal with these conflicts.

This approach is explored in the following experiments by combining equally good decision trees; that is, decision trees which result from different choices

of equally good attributes, according to the cost function. Decision trees are combined more efficiently by considering each decision tree as a **rule set**. The rule set for a decision tree is constructed by simply traversing each path of the decision tree and recording the attributes and values encountered. These then form the If-part of a rule, with the Then-part being the decision found at the leaf node.

3.5.1 C1: Combining Equally Good Trees

When the information-theoretic cost function is applied to the training set containing just those objects with Soil=26, the value of $E(A)$ for each of the attributes UVeg, LVeg, DPort, AWMIH, AWMIS, and AWMIW is 0. UVeg was the chosen attribute because of the attribute ordering used in constructing T106DC.

In this experiment, a decision tree is constructed for each of the above 6 choices. These decision trees, treated as rule sets, can then be combined. For a given object, more than one rule may have its conditions met by the object. Thus, multiple, and possibly conflicting decisions may arise. The resolution of these conflicts follows the approach of choosing the decision with the most support from the training set. (Support is simply measured as the number of objects in the training set corresponding to the particular path through the decision tree which lead to the decision obtained, requiring the training set to be representative of the universe of objects.) If two different decisions have equal support from the training set, then one is arbitrarily chosen.

By effectively building multiple, equally good, decision trees, the coverage of the resulting system can be greater than that of the individual trees, especially when integer attributes are used. What needs to be assessed though is the accuracy of the resulting system.

The rule set T106C1 results from combining 6 equally good decision trees differing only in the Soil=26 branch. Any object to be classified, having Soil=26, will have a decision from at least four of the decision trees, corresponding to the

	T106C1	c.f. T106DC
Agreement:	67.9%	+203
Mild Disagreement:	29.1%	+320
Moderate Disagreement:	3.1%	+94
Strong Disagreement:	0%	no change
Coverage:	77.7%	+617

TABLE 3.24: *T106C1 is a collection of 6 "equally good" decision trees. The decisions provided by T106C1 are compared to those given by the Model, and then to the performance of T106DC.*

four continuous attributes which cover all possible values. The only conflicts that can occur are between a decision of *VLow* resulting from one decision tree and a decision of *Medium* from another. However, since there are twice as many objects in the training set corresponding to the *Medium* decision, *Medium* is chosen over *VLow* when a conflict arises.

As Table 3.24 indicates, T106C1 has a coverage of 77.7%, an improvement upon the coverage of T106DC, as expected. The percentage agreement has decreased from the 71.5% of T106DC to 67.9% for T106C1.

3.5.2 C2: Further Combinations

This approach can be applied to other portions of the T106DC decision tree where sets of equally good attributes arise. The sub-tree of T106DC with Soil=16 is one such case. From the corresponding training set the 5 attributes UVeg, LVeg, DPort, AWMIH, and AWMIW each have a value of 0 for $E(A)$. Five decision trees can be constructed which differ only in the choice of the attribute at this node.

The rule set T106C2 contains rules which can produce conflicting decisions, but only between decisions of *Low* and *VLow*. The choice when conflict does arise is not always as simple as it was for T106C1. If one of the rules corresponding to the continuous attributes makes a decision of *VLow*, then this will always have a support value of 8 objects in the training set, compared to 4 objects in the training set if *Low* was the successful decision. However, the two

	T106C2	c.f. T106DC
Agreement:	72.4%	+327
Mild Disagreement:	25.9%	+51
Moderate Disagreement:	1.7%	no change
Strong Disagreement:	0%	no change
Coverage:	74.9%	+378

TABLE 3.25: *T106C2 is a collection of 5 "equally good" decision trees. The decisions of T106C2 are compared to those from the Model, and then to those of T106DC.*

categorical attributes each lead to 3 rules, two concluding a decision of *VLow*, with the other resulting in *Low*. Each has a support value of 4 objects in the training set. If at least one of the two paths with Soil=16 which leads to *VLow* is followed, then the decision of *VLow* is not disputed. Otherwise, an arbitrary selection must be made.

The resulting rule set again demonstrates how coverage can be increased (at least when compared to T106DC), with an improved coverage of 74.9% (Table 3.25). Whilst its coverage is less than that of T106C1 (and less than that of T106DI), its accuracy is greater.

3.6 SUMMARY OF EXPERIMENTS

This chapter has identified desirable properties of decision tree induction and found, by way of experimentation, that not all hold. The experiments presented here further illustrate a number of techniques which enhance the decision tree induction algorithm. The techniques include the categorisation of integer attributes, split-generalisations on such attributes, exception split handling, and the combining of decision trees. A particular observation made in this chapter is that a decision tree induction algorithm is capable of producing more than just a single "best" decision tree from a given training set.

The principal results from the experiments are presented in Table 3.26. Tables 3.16 and 3.23 provide further summaries of the experiments. Below is a précis of the experiments.

- The precision of the cost function was explored in the "DU" series of experiments, where the choice of attribute for the root node of the decision tree was overridden with the second best attribute. The resulting improvement in coverage is offset by poorer accuracy. Similar results for other choices of close attributes were obtained. Where it did discriminate, the cost function was found to be a good discriminator. With the Credit data though, less marked changes in performance were observed, with some "second best" choices actually leading to trees with marginally improved coverage and accuracy. In general though, the choice made by the decision tree induction algorithm can be relied upon.
- Noting that the cost function was not always adequate in discriminating between attributes, an investigation of "equally good" decision trees was undertaken. T106DI was constructed by choosing an integer attribute over a categorical attribute whenever such a choice was available. This produced a decision tree which was able to cover many more objects from the Range database than T106DC, but at the cost of accuracy. The increase in the disagreements though is mostly confined to mild disagreements. The

Experiment	Description	Cover	Agree	Mild	Mod	Strong
T106DC	Application of the DTIA.	70.4	71.5	26.7	1.8	0.0
T106DI	Favour integer attributes.	84.3	64.8	33.1	2.1	0.0
T106Ae	AW as categorical, with generalisation.	75.7	71.0	27.3	1.7	0.0
T106Ch	30% Exception split and Tr-consistent.	80.4	70.8	27.7	1.6	0.0
T106C1	Combine Soil=26.	77.7	67.9	29.1	3.0	0.0
T106C2	Combine Soil=16.	74.9	72.4	25.9	1.7	0.0

TABLE 3.26: *Summary of decision tree applications, with comparisons to the Model—all figures are percentages.*

observation that "equally good" decision trees have significantly different performances is made. With preference given to integer attributes, decision trees with greater coverage but less accuracy generally result, compared to decision trees induced with preference given to categorical attributes.

- A deficiency of the information theoretic cost function used in many decision tree induction algorithms is that it has a bias toward many-valued categorical attributes. The T106A series of experiments considered this problem. A solution was considered whereby integer attributes were regarded as categorical. Dramatic decreases in the coverage of the decision trees were coupled with significant increases in the disagreements. However, the introduction of a generalisation technique improved the performance—pseudo-categorical attributes were generalised by growing the range of values associated with the branches, effectively restoring their "integer" nature.
- Further experiments showed that the choice of a categorical attribute as the root node of the decision tree, whilst permitting integer attributes (being treated as categorical) to appear elsewhere in the tree, improved the coverage and accuracy of the decision tree.
- A decision tree, T106Ae, formed by treating integer attributes as categorical, but with a categorical attribute as the root of the tree, and then

generalising, achieved greater coverage and accuracy compared to the original decision tree (T106DC). This tree also begins to approach the coverage obtained by T106DI, whilst maintaining high agreement with the Model.

- Three experiments dealt with the issue of pruning a decision tree. They showed how the coverage of a decision tree may be increased, with only minor degradation of the accuracy of the decisions produced, by removing exception splits. It was further found that retaining the Tr-consistency of the decision tree whilst employing this approach improved the performance.
- The final two experiments combined a number of equally good decision trees—in particular, decision trees which differ in only one sub-tree. A different attribute for the root of the sub-tree was chosen. Whilst both experiments increased the coverage, one was less accurate than the original decision tree, whilst the other was more accurate. The technique of combining decision trees promises a path to greater coverage and accuracy. From these preliminary experiments in combining decision trees it is argued that this approach is an interesting one, worthy of further study and development. This is the focus of Chapter 4.

These experiments have illustrated a number of uncertainties in using a decision tree induction algorithm. It is important for the user of these algorithms to be aware of such behaviour. The following chapter builds upon one of these observations by considering further the idea of combining decision trees.

4 The MIL Algorithm

The experiments presented in Chapter 3 demonstrated variations in the performance of multiple decision trees induced from a single training set. Choosing a single best decision tree is identified as a difficulty with decision tree induction algorithms. The MIL algorithm is developed in this chapter as an approach to handling this situation. Rather than choosing between decision trees, the approach adopted is to combine decision trees. The goal is to improve the accuracy and coverage of the resulting knowledge structures.

This chapter begins with an introduction to the idea of combining decision trees. The terminology and a number of properties relating to conflicts which arise when decision trees are combined are then presented. This is followed by an example which serves to motivate and illustrate the process of combining decision trees. A full specification of the MIL algorithm then follows. A series of experiments then illustrate how MIL can effectively combine decision trees.

4.1 COMBINING RULE SETS

Differences in performance (coverage and accuracy) were observed in the experiments of Chapter 3 where distinct decision trees were induced from a single training set. This observation is of concern to the knowledge engineer who must deal with these multiple (alternative) decision trees. A similar observation is also made by Mingers (1989b), reporting that different selection criteria can lead to multiple decision trees of similar accuracy. Again, a choice between alternative decision trees must be made.

This chapter develops a technique for combining decision trees. Rule sets, rather than decision trees, will be used to describe and implement this approach. Decision trees can be translated into equivalent sets of rules, representing knowledge in a more expressive and familiar form. A rule set derived from a decision tree will contain a rule for each leaf node, corresponding to the paths from the root of the decision tree to that leaf node.

A goal of combining rule sets is to improve performance by taking the best from the rule sets being combined. The resulting coverage and accuracy will depend upon the coverage and accuracy of the original rule sets. In terms of coverage, at best we can expect a combined rule set to have a coverage equal to that of the union of the coverage of the individual rule sets. In terms of accuracy the situation is less clear—rules from different rule sets may conclude different decisions for the same objects. How these conflicting decisions are resolved will affect accuracy.

Conflict indicates deficiencies in the knowledge base. The MIL algorithm identifies all conflicting rules, modifies them to remove the potential for conflict, and then suggests new rules to restore the coverage otherwise lost.

The concept of conflict resolution is not new (Davis and King, 1984). Typically, in a rule-based system when multiple rules are applicable in a particular situation, a single rule is chosen, and no conflicting decision are made. Heuristics for choosing a rule have been developed, and include data ordering (which

uses a rule involving certain attributes in preference to a rule involving other attributes), generality ordering (where the most specific rule is chosen), and rule precedence (where a precedence network is used to order the rules). Suwa, Scott, and Shortliffe (1984) introduce the concept of checking for conflicts in a rule base and then presenting these to a knowledge engineer for advice. No attempt is made to actually deal with the conflict. More recent work by Li, Barter, and Yu (1988), for example, has investigated the use of knowledge about the relationships between the values of an attribute in order to rationalise conflicts when they actually arise. The approach embodied in the MIL algorithm addresses conflict by identifying the rules causing the conflict, restricting the rules to avoid the conflict, and then suggesting new rules to make decisions for those objects previously resulting in conflict.

4.2 THE TERMINOLOGY OF CONFLICT

The terminology of conflict presented here builds upon the general terminology introduced in Chapter 2.

The concept of an object was introduced in Chapter 2 as a description of an entity. The set of **attributes A** $= \{A_1, \ldots, A_p\}$ is used to describe an object. Attributes are either categorical (with a domain consisting of an enumerated set of values) or integer. $A_i(o)$ will denote for the particular object o the value associated with the attribute A_i. The subset of the cartesian product of the attributes $A_1, \ldots, A_p$ corresponding to a particular domain of application will be denoted as $\mathcal{O}$. A **decision attribute** is the only attribute which may appear in the conclusion of a rule (and for convenience it is not included in **A**). Elements from the domain of the decision attribute are identified as *Class*.

A **rule set** is a set of rules denoted by **R**. The individual rules of this rule set are denoted as R, R', R'', etc. The prime notation is used to distinguish the rules within a particular rule set. A subscript will be used to distinguish rule sets. Each rule has the syntactic form:

$$R\text{:} \quad \mathit{Cond} \quad \Longrightarrow \quad \mathit{Class},$$

where *Cond* specifies a condition under which the decision *Class* can be deduced for an object. If an object o satisfies the conditions specified by *Cond*, then the rule R is said to **trigger** on the object o.

Cond consists of a simple logical conjunction:

$$\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n,$$

$\mathcal{C}_k$ has the form $A_i < v$ or $A_i \geq v$ when A_i is an integer attribute and the form $A_i \in \{v_1, \ldots, v_n\}$ or $A_i \notin \{v_1, \ldots, v_n\}$ when A_i is a categorical attribute. The forms $A_i = v$ and $A_i \neq v$ will be used for categorical attributes as an abbreviation for singleton sets of values.

The values v and v_j are drawn from the domain of A_i. The form $\mathcal{C}_k$ is referred to as a **condition triplet** and consists of an attribute, a relational operator, and

a value or set of values. A second subscript will be used when it is important to identify the rule set from which the rule containing this condition triplet originates ($\mathcal{C}_{jk}$ for rule set $\mathbf{R}_j$).

If the object o triggers the rule R, then $R(o)$ is taken to be the set containing just the decision of this rule, $\{Class\}$. If o does not trigger this rule, then $R(o)$ will be the empty set. $\mathbf{R}(o)$ is the union of $R(o)$ over all rules in the rule set $\mathbf{R}$.

The term **scope** refers to those objects from $\mathcal{O}$ which trigger a particular rule. The scope of the rule R is the set of objects for which $Cond$ is true.

A rule set is **derived directly** from a decision tree by constructing a single rule for each path through the decision tree from the root node to a leaf node. The concept of a directly derived rule set is introduced to distinguish these rule sets from modified rule sets. A directly derived rule set is equivalent to the decision tree. Reference will be made to a **path** through a decision tree which, unless otherwise stated, will refer to the direct path from the root node to a leaf.

A **conflict** consists of a pair of rules R_1 and R_2 (members of the rule sets $\mathbf{R}_1$ and $\mathbf{R}_2$ respectively) having **consistent conditions** yet inconsistent conclusions. That is, there exists objects which can trigger both rules, but the rules conclude different values for the decision attribute. The symbol Q will be used to refer to a conflict between two rules. Often, it is convenient to also associate with a conflict the subset of the training set containing those objects which trigger either of the rules in conflict. The two rules have the form:

$$\begin{aligned} R_1\text{:} \quad & Cond_1 \implies Class_1, \\ R_2\text{:} \quad & Cond_2 \implies Class_2, \end{aligned}$$

where $Class_1 \neq Class_2$, recalling that the subscript is used to distinguish between the two conditions and to distinguish between the two conclusions. It also ties R_1 to the rule set $\mathbf{R}_1$ and R_2 to the rule set $\mathbf{R}_2$.

The set of objects for which two given rules trigger is referred to as the **common scope** of those two rules, and denoted by **Cs**:

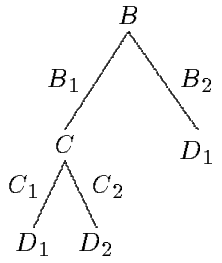
$$\mathbf{Cs} = \{o \,|\, o \in \mathcal{O},\ R_1(o) \neq \{\},\ R_2(o) \neq \{\}\}$$

Associated with each rule in a directly derived rule set is a non-empty subset of the training set containing those objects which trigger the rule. For the rules R_1 and R_2, these corresponding training subsets are denoted by $\mathbf{Tr}_1$ and $\mathbf{Tr}_2$:

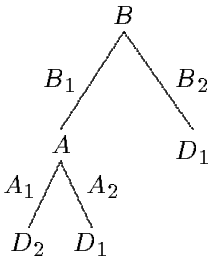
$$\mathbf{Tr}_1 = \{o \,|\, o \in \mathbf{Tr},\, R_1(o) \neq \{\}\,\}$$
$$\mathbf{Tr}_2 = \{o \,|\, o \in \mathbf{Tr},\, R_2(o) \neq \{\}\,\}$$

A training set from which a decision tree of depth 1 (a decision tree having a root node, and one or more branches leading only to leaf nodes) is induced will be referred to as a **terminating training set**. Such a training set is partitioned by the induction algorithm so that each cell satisfies the termination criterion. In ID3, for example, with partitions based upon a single attribute and a termination criterion of decision homogeneity, a terminating training set, is one containing at least two different decision values and for which a cost of 0 is computed for a partition.

The experiments of Chapter 3 illustrated the possibility of equally good partitionings of a terminating training set. Only one of these partitionings is chosen, leading to a collection of rules which differ in only a single condition triplet (corresponding to the different branches from the common node of the tree). A different choice leads to a collection of rules differing from any other rules induced from this training set in only a single condition triplet. Figure 4.1 illustrates this in terms of both decision trees and rules.



If $B = B_1$ and $C = C_1$ Then D_1
If $B = B_1$ and $C = C_2$ Then D_2
If $B = B_2$ Then D_1



If $B = B_1$ and $A = A_1$ Then D_2
If $B = B_1$ and $A = A_2$ Then D_1
If $B = B_2$ Then D_1

FIGURE 4.1: *Multiple decision trees (rule sets) can be induced when equally good partitions of a terminating training set arise. In this illustration the training set containing those objects of the original training set having the value B_1 for attribute B is a terminating training set. Partitioning this training set using either attribute C or attribute A results in each cell of the partition containing objects with a single common decision. A terminating training set will generate a decision tree of depth 1.*

Conflicting rules are derivable by choosing alternative partitions of a terminating training set. Such a terminating training set will be denoted as $\mathbf{Tr}_c$. Both $\mathbf{Tr}_1$ and $\mathbf{Tr}_2$ are subsets of $\mathbf{Tr}_c$. Whilst conflict can arise in other circumstances, it is only those conflicts arising in the context of terminating training sets that will be considered here.

4.3 PROPERTIES OF CONFLICT

The properties below formalise some of the intuitions underlying the MIL algorithm. The relationship between rules from pairs of decision trees and the conflicts which result are examined. The context is that of inducing multiple decision trees from a single training set.

> **Property 4.1:** Suppose $\mathbf{R}_k$ is a directly derived rule set and that R_k and R'_k are distinct rules contained in this rule set. Then $Cond_k$ and $Cond'_k$ cannot both be true of a single object.
>
> **The collection of rules belonging to a rule set directly derived from a decision tree are mutually exclusive: at most one rule can trigger for any object.**

Branches emanating from a node of a decision tree correspond to mutually exclusive choices and thus direct paths from the root node to a leaf node of the decision tree are mutually exclusive. Since the rules in $\mathbf{R}_k$ and the paths through the decision tree from which $\mathbf{R}_k$ was derived have a one-to-one correspondence, the conditions of each rule correspond to these mutually exclusive choices.

> **Property 4.2:** Suppose $\mathbf{R}_1$ and $\mathbf{R}_2$ are distinct rule sets and that (R_1, R_2) and (R'_1, R'_2) are two different pairs of rules from the respective rule sets. If the common scope of the pair of rules R_1 and R_2 is $\mathbf{Cs}$, and the common scope of the pair of rules R'_1 and R'_2 is $\mathbf{Cs}'$, then $\mathbf{Cs} \cap \mathbf{Cs}' = \emptyset$.
>
> **The common scope of any pair of rules drawn from two distinct rule sets has no overlap with the common scope of any other different pair of rules drawn the same two rule sets.**

If it were possible for one object to be in both common scopes then two rules from the one rule set must have triggered, contradicting Property 4.1. Whilst this property holds in general, it is of particular interest for rules in conflict.

The above two properties consider the scope of rules within a rule set, and the common scope of rules drawn from different rule sets. The following properties consider the actual structure of rules in conflict.

> **Property 4.3:** Suppose R_1 and R_2 are distinct rules (from directly derived rule sets $\mathbf{R}_1$ and $\mathbf{R}_2$ respectively) which conflict and the set of condition triplets contained in R_1 is $\{\mathcal{C}_{11}, \ldots, \mathcal{C}_{1n}\}$ and the set of condition triplets contained in R_2 is $\{\mathcal{C}_{21}, \ldots, \mathcal{C}_{2m}\}$. Then $n = m$ and the size of the set resulting from the union of the two sets of condition triplets is $m + 1$.

Two rules in conflict have all but one condition triplet in common.

This property is a direct consequence of considering only those conflicts which arise when alternative choices for the partitioning of a terminating training set exist. Rules generated from a terminating training set have all but one condition triplet in common. The rules in conflict will be written as:

$$R_1 : \quad \mathit{Cond} \wedge \mathcal{C}_1 \quad \Longrightarrow \quad \mathit{Class}_1$$
$$R_2 : \quad \mathit{Cond} \wedge \mathcal{C}_2 \quad \Longrightarrow \quad \mathit{Class}_2$$

A consequence of this property is that rules in conflict are derived from a common terminating training subset.

> **Property 4.4:** Suppose R_1 and R_2 are distinct rules (from directly derived rule sets $\mathbf{R}_1$ and $\mathbf{R}_2$ respectively) which conflict, and that $\mathbf{Tr}_c$ is the corresponding terminating training subset. Then there exists $R_1' \in \mathbf{R}_1$, $R_2' \in \mathbf{R}_2$ a their corresponding terminating training subset $\mathbf{Tr}_c'$ such that R_1' and R_2' are in conflict and $\mathbf{Tr}_c = \mathbf{Tr}_c'$.

Conflicts arise in groups. Each group of conflicts share a common terminating training set from which they were derived.

Suppose R_1 and R_2 have the form as above. $\mathbf{Tr}_c$ will consist of those objects in $\mathbf{Tr}$ for which *Cond* is true. The induction algorithm will have partitioned $\mathbf{Tr}_c$ to generate R_1 and at least one other rule. One of these other rules will conclude Class_2:

$$R'_1: \quad Cond \wedge \mathcal{C}'_1 \quad \Longrightarrow \quad Class_2$$

The case of R_2 is symmetrical, being generated from an alternative partition of $\mathbf{Tr}_c$. A rule of the following form will also be generated:

$$R'_2: \quad Cond \wedge \mathcal{C}'_2 \quad \Longrightarrow \quad Class_1$$

Just as $\mathcal{C}_1$ and $\mathcal{C}_2$ are consistent, $\mathcal{C}'_1$ and $\mathcal{C}'_2$ are consistent, and thus R'_1 and R'_2 are also in conflict.

For a terminating training set containing two distinct values for the decision attribute at most two conflicts result. For a terminating training set with three distinct values for the decision attribute at most six conflicts result. In general, for a terminating training set with n distinct values for the decision attribute at most $n * (n-1)$ conflicts result. Conflicts arising from a common terminating training set will be referred to as **complementary conflict pairs**, and will be denoted as the set $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, where the notation Q_i is used to denote a pair of rules in conflict together with the corresponding terminating training set.

> **Property 4.5:** Suppose R_1 and R_2 are in conflict and R_1 and R'_2 are also in conflict, where $R_1 \in \mathbf{R}_1$, $R_2, R'_2 \in \mathbf{R}_2$, and $R_2 \neq R'_2$. Then R_2 and R'_2 are derived from the same terminating training set.
>
> **Two conflicts involving a common rule are derived from a common terminating training set.**

This follows directly from Property 4.4.

The above properties pinpoint the key features used to identify and remove conflict. The following example demonstrates how these properties can be used.

4.4 EXAMPLE

The decision trees T106DC and T106DI of Chapter 3 were constructed using the information-theoretic cost function to select an attribute at each step. They differ only in the attribute chosen to partition a terminating training set. For T106DC a categorical attribute was always selected in preference to an "equally good" integer attribute. For T106DI, integer attributes were favoured.

Whilst T106DC and T106DI are equally good decision trees with respect to the cost function, their performances differ. T106DI has a higher coverage than T106DC, but is less accurate in the decisions it makes.

The rule sets corresponding to these two decision trees will be used here to introduce the issues which must be addressed in combining decision trees. The process of resolving the conflicts which arise will be described. This example is intended to provide an overview of the MIL algorithm before the full details are provided.

4.4.1 Treating Conflicts as Null Decisions

Disagreement between the decisions made by the two rule sets occurs in only 416 cases, of which 162 (39%) are mild disagreements and 254 (61%) are moderate disagreements. The coverage of T106DI is a superset of the coverage of T106DC, and so the combined coverage is the same as that of T106DI, which is 1164 more objects than T106DC.

The simplest approach to handling conflict is to ignore it by returning a *Null* decision. The combined coverage of T106DC and T106DI is then decreased by 416 objects. Table 4.1 presents a summary of the performance of this combined rule set (CombDCDI) for comparison with T106DC and T106DI. Considering the measure of performance as a two dimensional space with agreement and coverage as the axes, the combined rule set represents a compromise between T106DC and T106DI.

Experiment	Description	Cover	Agree	Mild	Mod	Strong
CombDCDI	Incomplete Combined Rule Set.	79.3	67.4	30.8	1.8	0.0
T106DC	Application of the DTIA.	70.4	71.5	26.7	1.8	0.0
T106DI	DTIA Favouring Integers.	84.3	64.8	33.1	2.1	0.0

TABLE 4.1: *Summary of results from applying the (conflict ignored, and therefore incomplete) combined rule set to the 8413 objects of the Range database. The combined rule set combines T106DC and T106DI. Any conflicting decisions are replaced by Null decisions. All figures are percentages.*

Of interest is the observation that of the 416 objects given conflicting decisions, 76% are given decisions by T106DI which disagree with the Model. The conflicts indicate deficiencies in the knowledge.

This deficiency is to be addressed. The approach here analyses conflicts in the context of the information contained in the training set, attempting to increase the coverage of a combined rule set, whilst maintaining its accuracy.

4.4.2 Resolving a Conflict Between Two Rules

Two of the rules in conflict will be identified as DC_{25} from T106DC and DI_{24} from T106DI:

$$DC_{25}: \quad \text{Soil} = 26, \text{UVeg} = 2 \quad \Longrightarrow \quad \mathit{VLow},$$
$$DI_{24}: \quad \text{Soil} = 26, \text{DPort} < 635 \quad \Longrightarrow \quad \mathit{Medium}.$$

These are in conflict for any object which satisfies the condition: Soil = 26, UVeg = 2, and DPort < 635. In removing this potential for conflict, it must be noted that an object which does not satisfy this condition, but does trigger one of these rules, must still have the same decision assigned to it. One approach is to strengthen the condition of one of the rules, leaving the other rule as it is. This involves restricting the chosen rules' applicability by adding conditions to it. The rules here may be strengthened by adding the negation of the condition of the other rule in the conflict. For example, rule DC_{25} could become (after simple modifications):

$$\text{Soil} = 26, \text{UVeg} = 2, \text{DPort} \geq 635 \quad \Longrightarrow \quad \mathit{VLow}.$$

This approach would meet our goal of removing the potential for conflict whilst maintaining coverage. It is not clear though how to decide which rule should be strengthened. The MIL algorithm effectively delays this decision by strengthening both rules in the above manner:

$$DC'_{25}: \quad \text{Soil} = 26, \text{UVeg} = 2, \text{DPort} \geq 635 \quad \Longrightarrow \quad \mathit{VLow},$$
$$DI'_{24}: \quad \text{Soil} = 26, \text{DPort} < 635, \text{UVeg} \neq 2 \quad \Longrightarrow \quad \mathit{Medium}.$$

A new rule can then be introduced to handle the conflict. It will consist of the conjunction of the conditions of the two conflicting rules and will make a decision of $\mathit{Conflict}_1$.

$$DCc_{25}: \quad \text{Soil} = 26, \text{UVeg} = 2, \text{DPort} < 635 \quad \Longrightarrow \quad \mathit{Conflict}_1.$$

If $\mathit{Conflict}_1$ were replaced by the decision *Medium* the effect would be that of strengthening the single rule DC_{25} as above.

In the absence of other knowledge, the training set is called upon to provide guidance in assigning decisions to objects triggering rule DCc_{25}. The terminating training set ($\mathbf{Tr}_c$) from which these two rules were generated is listed in

Region	Soil	UVeg	LVeg	DPort	AWMIH	AWMIS	AWMIW	Decision
30503	26	2	4	801	16	16	09	*VLow*
21423	26	3	2	467	52	09	33	*Medium*
21424	26	3	2	469	49	09	29	*Medium*

TABLE 4.2: *The terminating training set* $\mathbf{Tr}_c$ *associated with the two rules* DC_{25} *and* DI_{24} *is listed. The first object listed constitutes* $\mathbf{Tr}_1$ *and corresponds to* DC_{25} *(*Soil $= 26$ *and* UVeg $= 2$*) whilst the final two constitute* $\mathbf{Tr}_2$ *and correspond to* DI_{24} *(*Soil $= 26$ *and* DPort < 635*).*

Table 4.2. Condition triplets which distinguish between the objects of $\mathbf{Tr}_1$ and $\mathbf{Tr}_2$ are sought by considering each attribute. Low values of AWMIH, for example, are associated with $\mathbf{Tr}_1$ (a decision of *VLow*), whilst higher values are associated with $\mathbf{Tr}_2$ (a decision of *Medium*). A midpoint split of this integer attribute, with the mid-point of $\frac{49+16}{2}$ or 33, partitions this training set homogeneously. The integer attribute AWMIS can similarly be used with a mid-point value of 13, as can AWMIW with a midpoint of 19. The categorical attribute LVeg similarly partitions on LVeg $= 2$ and LVeg $= 4$. The other attributes are not considered since they already appear in DCc_{25}. Taking AWMIS, for example, two new rules may be proposed:

$$\mathrm{DCc}'_{25}\text{: Soil} = 26, \text{UVeg} = 2, \text{DPort} < 635, \text{AWMIS} < 13 \implies \textit{Medium},$$
$$\mathrm{DIc}'_{24}\text{: Soil} = 26, \text{UVeg} = 2, \text{DPort} < 635, \text{AWMIS} \geq 13 \implies \textit{VLow}.$$

The rules DC_{25} and DI_{24} can be replaced by DC'_{25}, and DI'_{24}, and the rules DCc'_{25}, and DIc'_{24} can be suggested for inclusion in the combined rule set. Of those objects in the Range database for which the condition Soil $= 26$, UVeg $= 2$, and DPort < 635 is true, only 6% of these trigger rule DCc'_{25}, whilst the rest trigger DIc'_{24}. 63% of these objects were previously in moderate disagreement between T106DI and the Model, and now in agreement with the Model. There are no other changes in terms of agreement and disagreement with the Model. The performance of this modified combined rule set is summarised in Table 4.3, indicating increased coverage whilst maintaining the accuracy. Whilst the increase in coverage is slight, only one conflict involving just 0.4%

Experiment	Description	Cover	Agree	Mild	Mod	Strong
CombDCDI	Incomplete Combined Rule Set.	79.3	67.4	30.8	1.8	0.0
CombDCDIa	Modified Combined Rule Set.	79.7	67.4	30.8	1.8	0.0

TABLE 4.3: *Comparison of the results from using the modified combined rule set (replacing* DC_{25} *and* DI_{24} *with* DC'_{25}, DI'_{24}, *and introducing* DCc'_{25} *and* DIc'_{24}*) and the incomplete combined rule set. Conflicts which have yet to be dealt with are still regarded as Null decisions. Figures are percentages.*

of all the objects in the Range database for which conflict arises has been dealt with.

4.4.3 Resolving All Conflicts

All remaining pairs of conflicting rules can be similarly treated. However, Property 4.4 observes that conflicts occur in groups based upon terminating training sets. The MIL algorithm resolves conflict in the context of complementary conflict pairs.

The first step is to identify these sets of complementary conflict pairs. There are three such sets in the combined T106DC, T106DI rule set, each involving four rules.

The first set contains the following rules, where DC_{14} and DI_{14} are in conflict and DC_{16} and DI_{15} are in conflict:

$$\begin{aligned}
DC_{14}&: \quad \text{Soil} = 16,\ \text{UVeg} \in \{2,3\} \quad \Longrightarrow \quad \mathit{VLow},\\
DI_{14}&: \quad \text{Soil} = 16,\ \text{DPort} < 766 \quad \Longrightarrow \quad \mathit{Low},\\
DC_{16}&: \quad \text{Soil} = 16,\ \text{UVeg} = 9 \quad \Longrightarrow \quad \mathit{Low},\\
DI_{15}&: \quad \text{Soil} = 16,\ \text{DPort} \geq 766 \quad \Longrightarrow \quad \mathit{VLow}.
\end{aligned}$$

As with the earlier example, each rule is strengthened to eliminate the potential for conflict:

$$\begin{aligned}
DC'_{14}&: \quad \text{Soil} = 16,\ \text{UVeg} \in \{2,3\},\ \text{DPort} \geq 766 \quad \Longrightarrow \quad \mathit{VLow},\\
DI'_{14}&: \quad \text{Soil} = 16,\ \text{DPort} < 766,\ \text{UVeg} \notin \{2,3\} \quad \Longrightarrow \quad \mathit{Low},\\
DC'_{16}&: \quad \text{Soil} = 16,\ \text{UVeg} = 9,\ \text{DPort} < 766 \quad \Longrightarrow \quad \mathit{Low},\\
DI'_{15}&: \quad \text{Soil} = 16,\ \text{DPort} \geq 766,\ \text{UVeg} \neq 9 \quad \Longrightarrow \quad \mathit{VLow}.
\end{aligned}$$

However, the rules DC'_{14} and DC'_{16} are redundant, since they are specialisations of DI'_{15} and DI'_{14} respectively. They can be removed, with the consequence that the four original rules are replaced with just two.

The actual conflicts are now explicitly identified with the following rules:

DCc_{14}: Soil = 16, UVeg $\in \{2,3\}$, DPort < 766 $\Longrightarrow$ $Conflict_2$,

DCc_{16}: Soil = 16, UVeg = 9, DPort ≥ 766 $\Longrightarrow$ $Conflict_3$.

The next set of complementary conflicts is handled similarly. From the four rules:

DC_{19}: Soil = 19, UVeg = 3 $\Longrightarrow$ $Medium$,

DI_{19}: Soil = 19, AWMIH ≥ 28 $\Longrightarrow$ Low,

DC_{20}: Soil = 19, UVeg = 4 $\Longrightarrow$ Low,

DI_{18}: Soil = 19, AWMIH < 28 $\Longrightarrow$ $Medium$,

the following four rules are generated:

DI'_{19}: Soil = 19, AWMIH ≥ 28, UVeg $\neq 3$ $\Longrightarrow$ Low,

DI'_{18}: Soil = 19, AWMIH < 28, UVeg $\neq 4$ $\Longrightarrow$ $Medium$,

DCc_{19}: Soil = 19, UVeg = 3, AWMIH ≥ 28 $\Longrightarrow$ $Conflict_4$,

DCc_{20}: Soil = 19, UVeg = 4, AWMIH < 28 $\Longrightarrow$ $Conflict_5$.

Similarly, the complementary conflicts:

DC_{25}: Soil = 26, UVeg = 2 $\Longrightarrow$ $VLow$,

DI_{24}: Soil = 26, DPort < 635 $\Longrightarrow$ $Medium$,

DC_{26}: Soil = 26, UVeg = 3 $\Longrightarrow$ $Medium$,

DI_{25}: Soil = 26, DPort ≥ 635 $\Longrightarrow$ $VLow$,

generate:

DI'_{25}: Soil = 26, UVeg $\neq 3$, DPort ≥ 635 $\Longrightarrow$ $VLow$,

DI'_{24}: Soil = 26, UVeg $\neq 2$, DPort < 635 $\Longrightarrow$ $Medium$,

DCc_{25}: Soil = 26, UVeg = 2, DPort < 635 $\Longrightarrow$ $Conflict_1$,

DCc_{26}: Soil = 26, UVeg = 3, DPort >= 635 $\Longrightarrow$ $Conflict_6$.

The resulting combined rule set contains 6 new conflict-free rules and 6 conflict identifying rules in place of the original 12 rules in conflict. The next step is to deal with these rules which explicitly identify conflict.

The rules DCc_{14} and DCc_{16} were generated from the four rules associated with a terminating training subset consisting of 12 objects. These objects have a decision of either *VLow* or *Low*. Rule DCc_{14} triggers on 124 objects in the whole of the Range database, whilst DCc_{16} triggers on only 13 objects. With the benefit of knowing the decisions made by the Model it is observed that for these 137 objects, the Model also makes a decision of either *VLow* or *Low*.

An attribute is now chosen to differentiate between the two training subsets. The attributes Soil, UVeg, and DPort are not considered, since they have already been employed in the induction process in generating the rules under consideration here. The potential candidates are LVeg, AWMIH, and AWMIW (AWMIS does not distinguish between those objects in the training subsets). The split points for each of these attributes, together with the associated decisions are listed below. Any of these pairs can be introduced to the conflict rule, generating two conflict-free rules.

$$\begin{aligned}
\text{LVeg} \in \{1,4\} &\Rightarrow \mathit{VLow}, \quad \text{LVeg} = 2 \Rightarrow \mathit{Low};\\
\text{AWMIH} < 27 &\Rightarrow \mathit{VLow}, \quad \text{AWMIH} \geq 27 \Rightarrow \mathit{Low};\\
\text{AWMIW} < 20 &\Rightarrow \mathit{VLow}, \quad \text{AWMIW} \geq 20 \Rightarrow \mathit{Low}.
\end{aligned}$$

Since the decisions made by the Model are known, the performance of the two resulting conflict-free rules can be determined. In using the first pair of conditions above (those involving LVeg) to generate two rules from DCc_{14}, 66% (82) of the associated objects (124 in all) will be covered. Of these, 99% will be in agreement with the decisions made by the Model, whilst 1% is in mild disagreement. If the pair involving AWMIH is chosen instead, then full coverage of the objects results, with 79% in agreement and 21% in mild disagreement. The final choice, using AWMIW, again results in full coverage with 99% agreement and only 1% mild disagreement.

The conflict rule DCc_{16} is associated with exactly the same terminating training subset, and thus the same choices exist. The following two tables

summarise the performances of the new conflict-free rules for each of the possible choices, for conditions to be added to the rules DCc_{14} and DCc_{16} respectively.

Attribute	Split	Agree	Mild	Cover
LVeg	1,4 2	99%	1%	+82
AWMIH	27	79%	21%	+124
AWMIW	20	99%	1%	+124

Attribute	Split	Agree	Mild	Cover
LVeg	1,4 2	92%	8%	+13
AWMIH	27	46%	54%	+13
AWMIW	20	85%	15%	+13

The conflict rules DCc_{19} and DCc_{20} are similarly handled. The associated terminating training set is the subset of T106 for which the condition Soil = 19 holds. It consists of 9 objects having decisions of either *Medium* or *Low*. DCc_{19} triggers on 17 objects from the Range database, for which the Model makes decisions of either *Medium* or *High*. DCc_{20} triggers on 8 objects from the Range database, for which the Model decides *Low*.

Candidate attributes are AWMIS and AWMIW, both with split points of 14, and LVeg with the sets of values $\{2, 11\}$ and $\{3, 26\}$. The performance of the conflict-free rules which can replace DCc_{19} and DCc_{20} respectively is summarised as:

Attribute	Split	Agree	Mild	Cover
LVeg	2,11 3,26	88%	12%	+17
AWMIS	14	88%	12%	+17
AWMIW	14	88%	12%	+17

Attribute	Split	Agree	Mild	Cover
LVeg	2,11 3,26	100%	0%	+6
AWMIS	14	88%	12%	+8
AWMIW	14	100%	0%	+8

Experiment	Description	Cover	Agree	Mild	Mod	Strong
Best	Select best rules.	84.3	67.4	30.9	1.7	0.0
Worst	Select worst rules.	82.7	66.6	31.4	2.2	0.0
CombDCDI	Combined Rule Set.	79.3	67.4	30.8	1.8	0.0
T106DC	Application of the DTIA.	70.4	71.5	26.7	1.8	0.0
T106DI	DTIA Favouring Integers.	84.3	64.8	33.1	2.1	0.0

TABLE 4.4: *Summary of results from choosing the extra condition for each conflict rule which leads to the best performance, and then the choice which leads to the worst performance, compared to previous rule sets.*

The conflict rules DCc_{25} and DCc_{26} generate rules with the following performances:

Attribute	Split	Agree	Disagree	Cover
LVeg	2 4	92%	8%	+12
AWMIH	33	63%	37%	+35
AWMIS	13	63%	37%	+35
AWMIW	19	63%	37%	+35

Attribute	Split	Agree	Disagree	Cover
LVeg	2 4	2%	98%	+154
AWMIH	33	45%	55%	+219
AWMIS	13	30%	70%	+219
AWMIW	19	45%	55%	+219

These results provide a bound on the performance of MIL in this example. For each conflict MIL will choose one of the alternatives. Table 4.4 compares the best possible and worst possible conflict-free combined rule sets. The best rule set has coverage equal to that of T106DI, the best of any coverage obtained, and has greater agreement than that of T106DI. Even the worst rule set has similar coverage and percentage agreement. In comparison to T106DC, both the best and worst offer more coverage, at a cost to the accuracy.

This example has demonstrated how two rule sets can be combined such that the potential for conflict is removed from the combined rule set. It also empirically demonstrates that the strategy of using a third attribute which also partitions the corresponding terminating training set to assist in resolving conflict recovers coverage with accuracy.

4.5 RESOLVING CONFLICT

The introductory example illustrated improvement to a particular combined rule set by resolving conflict. A general framework for resolving the type of conflict which arises as a result of combining decision trees is now presented. The task of reconciling two rules in conflict is packaged up in a conflict resolver (MILcr). The conflict resolver removes the conflict whilst attempting to retain coverage, as illustrated in the example above.

4.5.1 Specifications

MIL has three inputs: two rule sets to be combined ($\mathbf{R}_1$, and $\mathbf{R}_2$ say) and the original training set from which both rule sets were induced ($\mathbf{Tr}$). A combined rule set is returned ($\mathbf{R}$). The following requirements are placed on the combined rule set $\mathbf{R}$:

1. For any object $o \in \mathcal{O}$, if $\mathbf{R}_1(o) = \{Class_i\}$ and $\mathbf{R}_2(o) = \{Class_j\}$ and $Class_i \neq Class_j$, then one and only one of the following holds: either $\mathbf{R}(o) = \{Class_i\}$, or $\mathbf{R}(o) = \{Class_j\}$, or $\mathbf{R}(o) = \{\}$.
2. a) For any training set object $o \in \mathbf{Tr}$, if $\mathbf{R}_1(o) = \{Class_i\}$ then $\mathbf{R}(o) = \{Class_i\}$.
2. b) For any training set object $o \in \mathbf{Tr}$, if $\mathbf{R}_2(o) = \{Class_j\}$ then $\mathbf{R}(o) = \{Class_j\}$.
3. For any object $o \in \mathcal{O}$, if $\mathbf{R}_1(o) \cup \mathbf{R}_2(o) = \{Class_i\}$ then $\mathbf{R}(o) = \{Class_i\}$.

The first statement requires that all conflicts between $\mathbf{R}_1$ and $\mathbf{R}_2$ be resolved: $\mathbf{R}$ is a conflict-free rule set. The second and third require that $\mathbf{R}$, like $\mathbf{R}_1$ and $\mathbf{R}_2$, is $\mathbf{Tr}$-consistent. That is, $\mathbf{R}$ makes the same decisions for objects in $\mathbf{Tr}$ as those associated with the training set. The fourth requires the combined rule set to cover those objects covered by the rule set being combined, where conflict does not arise.

The conflict resolver MILcr has three inputs: a pair of rules in conflict (R_1 and R_2) and the corresponding terminating training set ($\mathbf{Tr}_c$). Up to four

rules are returned by MILcr, identified as Cmb_1, Cmb_2, Cmb_3, and Cmb_4. The following requirements are specified for the conflict resolver. Recall that the set **Cs** is the subset of all objects from $\mathcal{O}$ for which both R_1 and R_2 trigger.

1. For any object for which both R_1 and R_2 do not trigger ($o \notin$ **Cs**), if $R_1(o) = \{Class_i\}$ then $Cmb_1(o) = \{Class_i\}$.
2. For any object for which both R_1 and R_2 do not trigger ($o \notin$ **Cs**), if $R_2(o) = \{Class_j\}$ then $Cmb_2(o) = \{Class_j\}$.
3. The rules Cmb_3 and Cmb_4 should trigger only on objects in **Cs**, and then, at most one should trigger.

The rules Cmb_1 and Cmb_2 are replacements for (or specialisations of) R_1 and R_2 and will be called the **replacement rules**. The two new rules Cmb_3 and Cmb_4 are introduced to handle objects in conflict and are called **suggested rules**. The first two statements indicate that the replacement rules must make the same decisions for those objects in $\mathcal{O}$ but not in **Cs** as previously made by R_1 and R_2. The third requirement restricts the scope of the suggested rules to only those objects in **Cs**. No object other than thos contained in **Cs** can satisfy the conditions of either rule. Also, the suggested rules may not simultaneously trigger.

4.6 THE CONFLICT RESOLVER

The conflict resolver is at the core of the MIL algorithm and will be described first. Each step is briefly introduced, followed by a detailed description of the operations involved. To summarise, MILcr begins by modifying R_1 and R_2 so that their conditions are mutually exclusive, generating conflict-free Cmb_1 and Cmb_2. It then considers candidate descriptions, based on a difference between the two training subsets $\mathbf{Tr}_1$ and $\mathbf{Tr}_2$. One of these candidates is selected and used to construct the suggested rules.

4.6.1 Eliminate Conflict

Step 1. Construct replacement rules: Strengthen R_1 and R_2 such that both cannot trigger for the same object from the universe, and such that they still make correct decisions for objects in $\mathbf{Tr}_c$.

The replacements rules introduced in the example (Section 4.4) satisfy these constraints and are of the form:

$$\begin{array}{lll} Cmb_1\text{:} & Cond_1 \wedge \neg Cond_2 & \Longrightarrow \quad Class_1 \\ Cmb_2\text{:} & Cond_2 \wedge \neg Cond_1 & \Longrightarrow \quad Class_2 \end{array}$$

The conjunction $Cond_1 \wedge \neg Cond_2$ describes all those objects in **Tr** that are also described by $Cond_1$ alone, and similarly for $Cond_2 \wedge \neg Cond_1$. Hence Cmb_1 and Cmb_2 will cover the same objects in **Tr** as covered by R_1 and R_2. Further, $Cond_1 \wedge \neg Cond_2$ excludes objects in **Cs**. Cmb_1 will make the same decisions for all the objects not in **Cs** as made by R_1 previously, and similarly for Cmb_2. Thus, by replacing R_1 and R_2 by Cmb_1 and Cmb_2, **Tr**-consistency is maintained, and decisions made for objects not in **Cs** are unchanged, satisfying the requirements.

4.6.2 Constructing Candidate Descriptions

The coverage of the rules R_1 and R_2 has been reduced by specialising them so that their conditions are mutually exclusive. Coverage is regained by introducing new rules built from the conjunction of the conditions of R_1 and R_2

($Cond_1 \wedge Cond_2$) but further specialised by the addition of condition triplets derived from the associated training subsets.

The second step of the conflict resolver searches for a distinguishing characterisation of the two training subsets, by considering as candidate descriptions pairs of condition triplets which differentiate between the objects in $\mathbf{Tr}_1$ and $\mathbf{Tr}_2$.

Step 2. For each attribute in **A** attempt to construct a candidate description, consisting of a pair of condition triplets, ($Cons_1$, $Cons_2$), such that $Cons_1$ is true for every object in $\mathbf{Tr}_1$, but not for any object in $\mathbf{Tr}_2$, and $Cons_2$ is true for every object in $\mathbf{Tr}_2$, but not for any object in $\mathbf{Tr}_1$.

(Any attribute appearing as $A_i = v$ in either of $Cond_1$ or $Cond_2$, can be removed from consideration. No description involving this attribute alone can be used to distinguish those objects in $\mathbf{Tr}_1$ from those in $\mathbf{Tr}_2$.)

Binary splits are considered in building candidate descriptions. Integer and categorical attributes will be considered separately. V_1 and V_2 will denote the sets of values of an attribute associated with the two training sets. For attribute A_i, $V_1 = \{A_i(o) \,|\, o \in \mathbf{Tr}_1\}$, and $V_2 = \{A_i(o) \,|\, o \in \mathbf{Tr}_2\}$.

Case 2.1. When A is an integer attribute: Find some value of A, γ say, such that all the values of A in $\mathbf{Tr}_1$ are less than (or alternatively greater than or equal to) γ and all values in $\mathbf{Tr}_2$ are greater than or equal to (or less than) γ. If no such γ can be found, then no candidate description is constructed for this attribute.

If a candidate description can be constructed, then either $\max(V_1) < \min(V_2)$ or $\max(V_2) < \min(V_1)$. The former will be assumed, with the latter case covered by symmetry. Let

$$\alpha = \max_{o \in \mathbf{Tr}_1} A(o), \text{ and } \beta = \min_{o \in \mathbf{Tr}_2} A(o).$$

That is, α is the maximum value of A in V_1, and β is the minimum value in V_2. If $\alpha < \beta$ then we can begin looking for a suitable γ such that $\alpha < \gamma \leq \beta$. One such γ is $\frac{\alpha+\beta}{2}$, rounded up to the nearest integer. If such a γ exists, then $Cons_1$ as $A < \gamma$ and $Cons_2$ as $A \geq \gamma$ are a candidate pair of condition triplets.

Case 2.2. When A is a categorical attribute: Find two sets of values of the attribute A such that every object in $\mathbf{Tr}_1$ has a value for A which is in one of the sets, and every object in $\mathbf{Tr}_2$ has a value which is in the other set. If appropriate disjoint sets can't be found then no candidate description is constructed for this attribute.

V_1 and V_2 are the obvious choices. If the intersection of V_1 and V_2 is empty, then $Cons_1$ as $A \in V_1$ and $Cons_2$ as $A \in V_2$ are a candidate pair of condition triplets. Otherwise, no candidate description based on A is constructed.

4.6.3 Description Selection

If no descriptions have been constructed in Step 2, then no new rules can be suggested and this and the next step will not apply.

The approach taken in choosing from amongst the candidate descriptions is based on the heuristic: *descriptions using integer attributes are preferred to those using categorical attributes.* Results from Chapter 3 indicate that such a heuristic can lead to increased coverage.

Step 3. If the set of candidate descriptions is non-empty, then choose the description which accounts for the most objects in $\mathbf{Tr}_c$.

If choice still remains, a pre-specified ordering of the attributes is relied upon.

4.6.4 Rule Construction

Rules which cover those objects removed from the coverage by the replacement of R_1 and R_2 by Cmb_1 and Cmb_2 are now introduced.

Step 4. If a candidate description has been chosen, construct the suggested rules from $Cons_1$ and $Cons_2$.

The suggested rules are:

$$Cmb_3\text{:} \quad Cond_1 \wedge Cond_2 \wedge Cons_1 \implies Class_1$$
$$Cmb_4\text{:} \quad Cond_1 \wedge Cond_2 \wedge Cons_2 \implies Class_2$$

There are no objects in **Tr** for which $Cond_1 \wedge Cond_2$ holds (Property 4.1), and so **R** remains **Tr**-consistent. Further, **Cs** contains all those objects in $\mathcal{O}$ for which $Cond_1 \wedge Cond_2$ holds, thus the effect of the above two rules is restricted to **Cs**.

The final step returns the new rules to MIL:

Step 5. Return $\{Cmb_1, Cmb_2, [Cmb_3, Cmb_4]\}$, where Cmb_3 and Cmb_4 are significant only if candidate descriptions were found.

4.6.5 The MILcr Algorithm

Pseudo-code is used to specify the complete MILcr algorithm.

MILcr$(R_1, R_2, \mathbf{Tr}_c)$:–

Descr := $\emptyset$; {Set of candidate descriptions}
$\mathbf{Tr}_1 := \{o \mid o \in \mathbf{Tr}_c, o \text{ triggers } R_1\}$; {Training set associated with R_1}
$\mathbf{Tr}_2 := \{o \mid o \in \mathbf{Tr}_c, o \text{ triggers } R_2\}$; {Training set associated with R_2}
{Step 1—Replacement Rules}
$Cmb_1 := Cond_1 \wedge \neg Cond_2 \Rightarrow Class_1$;
$Cmb_2 := Cond_2 \wedge \neg Cond_1 \Rightarrow Class_2$;
{Step 2—Construct Candidate Descriptions}
For $A_i := A_1, \ldots, A_p$ **Do**
  $V_1 := \{A_i(o) \mid o \in \mathbf{Tr}_1\}$; {Values of A_i in $\mathbf{Tr}_1$ and $\mathbf{Tr}_2$ respectively}
  $V_2 := \{A_i(o) \mid o \in \mathbf{Tr}_2\}$;
  Case type of A_i:
    Integer:
      If $\max(V_1) < \min(V_2)$ **Then**
        $\alpha := \max(V_1)$; $\beta := \min(V_2)$; $\gamma := \text{round}(\frac{\alpha+\beta}{2})$;
        $\mathcal{C}_1 := A_i < \gamma$; $\mathcal{C}_2 := A_i \geq \gamma$;
        Descr := Descr + $(\mathcal{C}_1, \mathcal{C}_2)$;
      ElseIf $\min(V_1) > \max(V_2)$ **Then**
        $\alpha := \max(V_2)$; $\beta := \min(V_1)$; $\gamma := \text{round}(\frac{\alpha+\beta}{2})$;
        $\mathcal{C}_1 := A_i \geq \gamma$; $\mathcal{C}_2 := A_i < \gamma$;
        Descr := Descr + $(\mathcal{C}_1, \mathcal{C}_2)$;
      End;
    Categorical:
      If $V_1 \cap V_2 = \emptyset$ **Then**
        $\mathcal{C}_1 := A_i \in V_1$; $\mathcal{C}_2 := A_i \in V_2$;
        Descr := Descr + $(\mathcal{C}_1, \mathcal{C}_2)$;
      End;
  End;
Done;
{Step 3—Choose a Candidate Description}
If Descr $\neq \emptyset$ **Then**
  $(\mathcal{C}_1, \mathcal{C}_2) := (\mathcal{C}_1, \mathcal{C}_2) \in$ Descr, such that
    $|\{o \mid o \in \mathbf{Tr}_c, \mathcal{C}_1 \text{ or } \mathcal{C}_2 \text{ is true}\}|$ is maximal; {Choose one with largest coverage}
{Step 4—Suggested Rules}
If Descr $\neq \emptyset$ **Then**
  $Cmb_3 := Cond_1 \wedge Cond_2 \wedge \mathcal{C}_1 \Rightarrow Class_1$;
  $Cmb_4 := Cond_1 \wedge Cond_2 \wedge \mathcal{C}_2 \Rightarrow Class_2$;
Else $Cmb_3 := Cmb_4 :=$ null;
{Step 5—Return Rules}
Return { Cmb_1, Cmb_2, Cmb_3, Cmb_4 };

4.7 USING THE CONFLICT RESOLVER

The input to MIL consists of two rule sets derived directly from different decision trees, together with the corresponding training set. Potential conflicting pairs of rules are easily identified by a pair-wise comparison of rules in the two rule sets. All rules found not to be in conflict can be immediately added to the combined rule set. Each pair of conflicting rules, together with their corresponding training subset, $\mathbf{Tr}_c$, is then passed on to the conflict resolver, with the resulting conflict-free rules added to the combined rule set.

The example presented in Section 4.4 illustrated that conflicts do not arise in isolation—sets of complementary conflict can be identified. When dealing with sets of conflict, redundant rules were readily identified and removed. The example contained conflicts for which the corresponding training subsets contained only two distinct values of the decision attribute. When more than two distinct values are found, the interaction between the complementary conflicts is somewhat more complex. To avoid unnecessarily complicating the conflict resolver, MILcr is applied separately to each pair of conflicting rules. The resulting redundant rules must be reconciled and opportunities to combine rules, where sensible, are sought.

4.7.1 Identifying Conflicts

The first step is to identify all potential conflicts. The rules which do not give rise to conflict in the context of the two rule sets being combined are included immediately in the combined rule set.

Step 1. Add each rule R in the intersection of $\mathbf{R}_1$ and $\mathbf{R}_2$ to the combined rule set $\mathbf{R}$.

Each of the remaining rules are compared, and all pairs that have all but one condition triplet in common are collected (see Property 4.3). Each pair is identified as being in conflict when their decisions differ and their single differing condition triplets are not mutually exclusive.

Step 2. For each rule R_1 in $\mathbf{R}_1$ with the form $Cond_1 \wedge \mathcal{C}_1 \Longrightarrow Class_i$, and for each rule R_2 in $\mathbf{R}_2$ with the form $Cond_1 \wedge \mathcal{C}_2 \Longrightarrow Class_j$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are independent and $Class_i \neq Class_j$, record R_1 and R_2 as a conflict.

Independence here means that both conditions could hold for a single object.

The order in which condition triplets appear in a rule is not important—they can thus be re-ordered to ensure that condition triplets common to the two rules appear first. In practice, because the rules are generated from decision trees, re-ordering is not necessary.

4.7.2 Application of the Conflict Resolver

Conflicts are grouped into sets of complementary conflicts, and the conflict resolver independently applied to each conflict. The rules returned by the conflict resolver are grouped according to the set of complementary conflicts from which they were derived.

Step 3. For each Q_i in the complementary set of conflicts $\{Q_1, \ldots, Q_k\}$ call upon MILcr. Repeat this for each set of complementary conflicts.

4.7.3 Rule Set Reconciliation

The rules generated by the conflict resolver must now be modified to remove redundancies and to correct any overly general rules. The resulting combined rule set must eliminate all conflict. As is demonstrated below, the conflict resolver can potentially generate new conflicts and redundant rules may also be generated. The task of reconciling the rules returned by MILcr is treated case-by-case based upon the number of conflicts contained in the complementary conflict set.

4.7.3.1 Binary splits of a terminating training set

The simplest case involves a terminating training set leading to a set of complementary conflicts containing just two conflicts (a total of four rules). Choosing one partition of this training set leads to rules of the form:

$$R_1: \quad Cond \wedge \mathcal{C}_1 \implies Class_i$$
$$R'_1: \quad Cond \wedge \mathcal{C}'_1 \implies Class_j$$

An alternative choice leads to rules having the same form, differing only in the final condition triplet:

$$R_2: \quad Cond \wedge \mathcal{C}_2 \implies Class_j$$
$$R'_2: \quad Cond \wedge \mathcal{C}'_2 \implies Class_i$$

Rules R_1 and R_2 are in conflict as are R'_1 and R'_2. The conflict resolver will generate the following replacement rules:

$$Cmb_1: \quad Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}_2 \implies Class_i$$
$$Cmb'_1: \quad Cond \wedge \mathcal{C}'_1 \wedge \neg\mathcal{C}'_2 \implies Class_j$$
$$Cmb_2: \quad Cond \wedge \mathcal{C}_2 \wedge \neg\mathcal{C}_1 \implies Class_j$$
$$Cmb'_2: \quad Cond \wedge \mathcal{C}'_2 \wedge \neg\mathcal{C}'_1 \implies Class_i$$

No conflict is introduced by these rules. R_1 and R'_1 are the only rules in $\mathbf{R}_1$ which could trigger for a given object satisfying $Cond$, and similarly for $\mathbf{R}_2$. Excluding R_1, R'_1, R_2, and R'_2 from the combined rule set removes the coverage of those objects satisfying $Cond$. If an object triggers any one of the above replacement rules then that object will not trigger any other rule in the combined rule set. Further, since $\mathcal{C}_1$ and $\mathcal{C}'_1$ can not both be true (and similarly $\mathcal{C}_2$ and $\mathcal{C}'_2$), there is no possibility of conflicting decisions being made for a single object from these replacement rules.

Redundancies are identified in the replacement rules when at least one of $\mathcal{C}_1$ and $\mathcal{C}'_1$ is true for any object in $\mathcal{O}$. This is the case when $\mathcal{C}_1$ involves an integer attribute, where $\neg\mathcal{C}'_1$ is simply $\mathcal{C}_1$ and vice versa. Further suppose that $\mathcal{C}_2$ and $\mathcal{C}'_2$ have the form $A_i \in \{v_1, \ldots, v_n\}$ with non-intersecting sets of values. Then $\neg\mathcal{C}_2$ is more general than $\mathcal{C}'_2$, and $\neg\mathcal{C}'_2$ is more general than $\mathcal{C}_2$, so that two rules of the form:

$$Cond \wedge \mathcal{C}_2 \quad \Longrightarrow \quad Class_j$$
$$Cond \wedge \neg\mathcal{C}_2' \quad \Longrightarrow \quad Class_j$$

can be collapsed into the single rule:

$$Cond \wedge \neg\mathcal{C}_2' \quad \Longrightarrow \quad Class_j$$

Thus, when $\mathcal{C}_1$ involves an integer attribute, and $\neg\mathcal{C}_2'$ is more general than $\mathcal{C}_2$, the four replacement rules are reduced to just two:

$$Cmb_1'': \quad Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}_2 \quad \Longrightarrow \quad Class_i$$
$$Cmb_2'': \quad Cond \wedge \neg\mathcal{C}_1 \wedge \neg\mathcal{C}_2' \quad \Longrightarrow \quad Class_j$$

If $\mathcal{C}_2$ involves an integer attribute, then $\neg\mathcal{C}_2'$ is simply $\mathcal{C}_2$, resulting in symmetrical rules.

Any suggested rule will also contain $Cond$, and so the potential for conflict with the replacement rules must be considered. Whilst the MILcr requirements state that there can be no conflict between any of the rules generated by one application of MILcr, no such guarantee has been made for rules generated by the multiple application of MILcr to rules in a complementary set of conflicts. The suggested rules have the form:

$$Cmb_3: \quad Cond \wedge \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \mathcal{C}_3 \quad \Longrightarrow \quad Class_i$$
$$Cmb_3': \quad Cond \wedge \mathcal{C}_1' \wedge \mathcal{C}_2' \wedge \mathcal{C}_3' \quad \Longrightarrow \quad Class_j$$
$$Cmb_4: \quad Cond \wedge \mathcal{C}_2 \wedge \mathcal{C}_1 \wedge \mathcal{C}_3' \quad \Longrightarrow \quad Class_j$$
$$Cmb_4': \quad Cond \wedge \mathcal{C}_2' \wedge \mathcal{C}_1' \wedge \mathcal{C}_3 \quad \Longrightarrow \quad Class_i$$

There is no potential for conflict here since $Class_i$ is concluded only if $\mathcal{C}_3$ (in addition to other conditions) holds, and $Class_j$ is concluded only if $\mathcal{C}_3'$ holds. But $\mathcal{C}_3$ and $\mathcal{C}_3'$ are mutually exclusive. Further, cross checking each of Cmb_1, Cmb_2, Cmb_1', Cmb_2' with each of Cmb_3, Cmb_4, Cmb_3', Cmb_4' identifies no possibility of conflict.

This step of reconciliation then is appropriate for the case of a terminating training set containing just two distinct values for the decision attribute. If $\mathcal{C}_1$ and $\neg\mathcal{C}_1'$ (or $\mathcal{C}_2$ and $\neg\mathcal{C}_2'$) are logically equivalent, then the four replacement rules generated by MILcr can be reconciled into just two.

4.7.3.2 Ternary splits of a terminating training set

The second case concerns those terminating training sets which contain three distinct values for the decision attribute. Any choice of attribute will lead to three distinct rules. Combining two rule sets leads to a set of complementary conflicts consisting of six conflicts (and six rules). The independent application of the conflict resolver to these six conflicts leads to a collection of rules which, as shown below, have conflict and must be reconciled. Only categorical attributes will be dealt with since integer attributes give rise only to binary splits, and can not be chosen for the training sets considered here.

The set of rules in conflict have the form:

$$\begin{array}{llllllll} R_1: & Cond \wedge \mathcal{C}_1 & \Longrightarrow & Class_i & R_2: & Cond \wedge \mathcal{C}_2 & \Longrightarrow & Class_j \\ R'_1: & Cond \wedge \mathcal{C}'_1 & \Longrightarrow & Class_j & R'_2: & Cond \wedge \mathcal{C}'_2 & \Longrightarrow & Class_k \\ R''_1: & Cond \wedge \mathcal{C}''_1 & \Longrightarrow & Class_k & R''_2: & Cond \wedge \mathcal{C}''_2 & \Longrightarrow & Class_i \end{array}$$

with the six conflicts:

$$\begin{array}{ll} Q_1\text{: } R_1, R_2 & Q_2\text{: } R_1, R'_2 \\ Q_3\text{: } R'_1, R'_2 & Q_4\text{: } R'_1, R''_2 \\ Q_5\text{: } R''_1, R''_2 & Q_6\text{: } R''_1, R_2 \end{array}$$

The complete list of replacement rules generated by MILcr as applied to each of the conflicts above respectively (pairwise across) is:

$$\begin{array}{llll} Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}_2 & \Longrightarrow & Class_i & Cond \wedge \mathcal{C}_2 \wedge \neg\mathcal{C}_1 \Longrightarrow Class_j \\ Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}'_2 & \Longrightarrow & Class_i & Cond \wedge \mathcal{C}'_2 \wedge \neg\mathcal{C}_1 \Longrightarrow Class_k \\ Cond \wedge \mathcal{C}'_1 \wedge \neg\mathcal{C}'_2 & \Longrightarrow & Class_j & Cond \wedge \mathcal{C}'_2 \wedge \neg\mathcal{C}'_1 \Longrightarrow Class_k \\ Cond \wedge \mathcal{C}'_1 \wedge \neg\mathcal{C}''_2 & \Longrightarrow & Class_j & Cond \wedge \mathcal{C}''_2 \wedge \neg\mathcal{C}'_1 \Longrightarrow Class_i \\ Cond \wedge \mathcal{C}''_1 \wedge \neg\mathcal{C}''_2 & \Longrightarrow & Class_k & Cond \wedge \mathcal{C}''_2 \wedge \neg\mathcal{C}''_1 \Longrightarrow Class_i \\ Cond \wedge \mathcal{C}''_1 \wedge \neg\mathcal{C}_2 & \Longrightarrow & Class_k & Cond \wedge \mathcal{C}_2 \wedge \neg\mathcal{C}''_1 \Longrightarrow Class_j \end{array}$$

The first rules on lines one and two are both replacements for R_1. They both restrict the application of R_1 to avoid conflict with rules R_2 and R'_2. However, since $\mathcal{C}_2$ and $\mathcal{C}'_2$ involve distinct sets of values, these two rules taken together are equivalent to the original rule (R_1). By applying the conflict resolver independently to the two corresponding conflicts we have effectively re-introduced

the conflict: the first rule on the first line and the second rule on the third line are, for example, in conflict. In fact, all of these replacement rules give rise to conflict.

MIL reconciles these conflicts by combining pairs of replacement rules so that they conform with the original intent—to produce a single replacement rule. Two rules are combined by conjoining their conditions.

Considering again the first rule of the first line, and the first rule of the second line the more appropriate replacement rule is:

$$Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}_2 \wedge \neg\mathcal{C}'_2 \implies Class_i$$

Repeating this exercise for all replacement rules eliminates all conflict, whilst reducing the number of suggested rules from 12 to 6:

$$\begin{array}{llll} Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}_2 \wedge \neg\mathcal{C}'_2 & \implies & Class_i & Cond \wedge \mathcal{C}_2 \wedge \neg\mathcal{C}_1 \wedge \neg\mathcal{C}''_1 \implies Class_j \\ Cond \wedge \mathcal{C}'_1 \wedge \neg\mathcal{C}'_2 \wedge \neg\mathcal{C}''_2 & \implies & Class_j & Cond \wedge \mathcal{C}'_2 \wedge \neg\mathcal{C}_1 \wedge \neg\mathcal{C}'_1 \implies Class_k \\ Cond \wedge \mathcal{C}''_1 \wedge \neg\mathcal{C}_2 \wedge \neg\mathcal{C}''_2 & \implies & Class_k & Cond \wedge \mathcal{C}''_2 \wedge \neg\mathcal{C}'_1 \wedge \neg\mathcal{C}''_1 \implies Class_i \end{array}$$

Further simplification is performed whenever $\mathcal{C}_2 \vee \mathcal{C}'_2 \vee \mathcal{C}''_2$, for example, is true for every object in $\mathcal{O}$. Any pair of these condition triplets appearing negated in a rule can be replaced by the remaining condition triplet. For example, $Cond \wedge \mathcal{C}_1 \wedge \neg\mathcal{C}_2 \wedge \neg\mathcal{C}'_2$ can be replaced by $Cond \wedge \mathcal{C}_1 \wedge \mathcal{C}''_2$. If $\mathcal{C}_1$ also meets these requirements, then the 6 replacement rules can be reduced to just 3.

Using similar arguments as for the case of a binary split of the terminating training set, the twelve suggested rules do not lead to any further conflict. The conditions of the suggested rules exclude any object which triggers any of the six replacement rules, and the only possibility for two of the suggested rules to trigger for a given object lies with the rules which conclude the same decision.

4.7.3.3 Generalised heuristics

The two simplest cases have illustrated how a collection of rules generated by MILcr by its application to a collection of conflicts belonging to a set of complementary conflicts can be reconciled. These cases will be summarised and general heuristics for reconciliation will be introduced below.

For a complementary set of conflicts involving just two conflicts, each rule set contributes two rules, and each rule appears in just one conflict. For each conflict, two replacement and two suggested rules are generated. Thus, we have $2 \times 1 \times 2$ $(= 4)$ replacement rules, and $2 \times 1 \times 2$ $(= 4)$ suggested rules.

Neither the replacement rules nor the suggested rules generated by MILcr result in conflict. A procedure of reconciliation rationalises the replacement rules when complementary condition triplets logically cover all possibilities. The negation of one condition triplet is then logically equivalent to the other condition triplet. This is trivially true for integer attributes. Under such circumstances, the 4 replacement rules can be reduced to 2.

For a complementary set of conflicts involving just three conflicts, each rule set contributes three rules, and each rule appears in two conflicts. Thus, we have $3 \times 2 \times 2$ $(= 12)$ replacement rules, and $3 \times 2 \times 2$ $(= 12)$ suggested rules.

The replacement rules returned by MILcr give rise to conflict. MIL reconciles these conflicts by restricting the coverage of the rules by conjoining the appropriate condition triplets. The number of replacement rules was reduced from 12 to 6 (there are just 6 rules being replaced). Further simplification was considered when both groups of complementary condition triplets (involving $\mathcal{C}_1$ and $\mathcal{C}_2$) covered all possibilities. As with the binary case, the number of rules can be halved, leading to 3 rather than 6 rules. If all values of only one of the attributes are represented in the condition triplets, then some simplification can still be carried out, but no rules can be removed.

For a complementary set of conflicts involving n conflicts, $n \times (n-1) \times 2$ replacement rules, and $n \times (n-1) \times 2$ suggested rules are generated by MILcr. There are n rules from each rule set involved, and each rule conflicts with $n-1$ rules from the other rule set, and two replacement and two suggested rules are returned by MILcr for each pair of conflicts

The replacement rules contain conflict (for $n > 2$). Groups of $n-1$ rules are conjoined appropriately, leaving just $n \times 2$ replacement rules, for the original

$n \times 2$ rules in conflict. When all attribute values for each of the attributes are represented in the $n \times 2$ condition triplets, then the number of suggested rules is reduced to just n. If all values of only one of the attributes are represented in the condition triplets, then simplification can be carried out, but no rules can be removed. The suggested rules do not lead to any conflict.

The remaining steps of the MIL algorithm are thus:

Step 4. For each set of complementary conflicts, add the suggested rules to the combined rule set **R**.

Step 5. For each set of complementary conflicts, combine excess replacement rules, leaving a single replacement rule for each rule being replaced. Rationalise these rules further, where appropriate.

4.7.4 The MIL Algorithm

MIL$(\mathbf{R}_1, \mathbf{R}_2, \mathbf{Tr})$:–

{Step 1—Add Common Rules to the Combined Rule Set}
$\mathbf{R} := \{R \mid R \in \mathbf{R}_1 \cap \mathbf{R}_2\}$;
{Step 2—Identify Conflicts}
For $R_1^i \in \mathbf{R}_1$ **and** $R_1^i \notin \mathbf{R}$ **Do**
  R_1^i is of the form $Cond_1^i \wedge \mathcal{C}_1^i \Rightarrow Class_1^i$;
  $\mathcal{C}_1^i$ is of the form $A_1^i \mathrm{Rel}_1^i \mathrm{Values}_1^i$;
  For $R_2^j \in \mathbf{R}_2$ **and** $R_2^j \notin \mathbf{R}$ **Do**
    R_2^j is of the form $Cond_2^j \wedge \mathcal{C}_2^j \Rightarrow Class_2^j$;
    $\mathcal{C}_2^j$ is of the form $A_2^j \mathrm{Rel}_2^j \mathrm{Values}_2^j$;
    If $Cond_1^i = Cond_2^j$ **and** $A_1^i \neq A_2^j$ **and** $Class_1 \neq Class_2$ **Then**
      $\mathbf{Tr}_c := \{o \in \mathcal{O} \mid Cond_1^i(o)\}$;
      $\mathcal{Q}_{\mathrm{Cond}_1^i} := \mathcal{Q}_{\mathrm{Cond}_1^i} \cup \{(R_1^i, R_2^j, \mathbf{Tr}_c)\}$;
    End; {Complementary conflict sets have $Cond$ in common}
  Done;
Done;
For $\mathcal{Q} \in \{\mathcal{Q}_{\mathrm{Cond}}, \ldots\}$ **Do** {For each set of complementary conflicts}
  {Step 3—Apply the Conflict Resolver}
  Repl $:= \emptyset$; **Intr** $:= \emptyset$;
  For $Q \in \mathcal{Q}$ **Do** {For each particular conflict in the set}
    $(R_1, R_2, \mathbf{Tr}_c) \in Q$;
    $\{Repl_1, Repl_2, Intr_1, Intr_2\} := \mathrm{MILcr}(R_1, R_2, \mathbf{Tr}_c)$; {Apply the conflict resolver}
    Repl $:=$ **Repl** $\cup \{Repl_1, Repl_2\}$;
    Intr $:=$ **Intr** $\cup \{Intr_1, Intr_2\}$;
  Done;
  {Step 4—Add Suggested Rules to Combined Rule Set}
  $\mathbf{R} := \mathbf{R} \cup$ **Intr**;
  {Step 5—Reconcile the Replacement Rules}
  $\mathbf{R} := \mathbf{R} \cup \mathrm{Reconcile}($**Repl**$)$;
Done
Return$(\mathbf{R})$;

The function `Reconcile` does the work of amalgamating multiple replacement rules and carrying out further simplifications if possible, as describe in detail in the preceding section.

4.8 THE UNIPER EXPERIMENTS

A series of experiments is now described, providing results from the MIL algorithm. The data for these experiments comes from a database recording values for 4 attributes and a decision attribute. Two of the attributes are categorical and the other two are integer. The categorical attributes E and S have one of the values 0, 1, 2, 3, and 4. The integer attributes Y and A range from 0-10 and 15-35 respectively. The decision attribute has three possible values, C1, C2, and C3. The decisions were constructed using a linear model. Once again, the data are complete and noise-free.

The complete database consists of 5775 records, being the complete enumeration of all possible values of each attribute. In inducing decision trees, random samplings of the database were taken, generating training sets of sizes 20, 30, and 40. Sixty training sets were generated, twenty at each size. These training sets are identified as r20a, r20b, ..., r20t, r30a, r30b, ..., r40t. From each training set two decision trees were induced: one using the attribute ordering of S, E, A, and Y (a categorical bias), the other using an attribute ordering of Y, A, E, and S (an integer bias). These two decision trees were then combined using MIL. For example, the decision trees r30aC and r30aI were combined to give the r30aX rule set.

The following three tables record the performance of each of the 120 decision trees and the 60 combined rule sets.

Training Set	Categorical		Integer		Combined	
	Accuracy	Coverage	Accuracy	Coverage	Accuracy	Coverage
r20a	92.1	72.0	78.7	92.0	86.7	92.0
r20b	81.2	80.0	80.4	100.0	83.0	98.7
r20c	84.9	92.0	90.1	96.0	89.6	96.0
r20d	88.5	84.0	94.6	92.0	93.4	92.0
r20e	95.0	92.0	84.8	92.0	91.4	92.0
r20f	88.6	92.0	79.6	100.0	81.1	100.0
r20g	95.7	84.0	76.5	88.0	81.8	87.6
r20h	89.4	88.0	79.3	96.0	81.5	96.0
r20i	90.2	96.0	94.2	100.0	94.7	100.0
r20j	81.4	84.0	75.4	96.0	77.3	96.0
r20k	94.5	84.0	83.2	92.0	85.9	90.3
r20l	88.9	84.0	84.9	96.0	88.6	92.0
r20m	86.1	84.0	74.2	100.0	79.3	96.0
r20n	83.4	100.0	83.4	100.0	83.4	100.0
r20o	67.9	80.0	67.9	80.0	67.9	80.0
r20p	88.4	76.0	84.5	92.0	86.5	90.1
r20q	86.2	68.0	78.3	72.0	80.7	71.2
r20r	90.3	72.0	72.2	100.0	74.9	98.5
r20s	90.0	76.0	81.4	88.0	81.6	88.0
r20t	89.1	96.0	73.3	100.0	76.7	100.0

TABLE 4.5: *Combining decision trees using MIL with training sets of size 20. Each row provides performance details for the two decision trees induced from a common training set, together with the performance of the MIL generated combined rule set. All figures are percentages.*

For training sets of size 20 (Table 4.5), in all cases the coverage of the combined rule set is equal to or slightly less than the greater of the coverages of the decision trees being combined. In two cases, (r20b and r20i) the accuracy of the combined rule set is greater than that of the individual decision trees being combined. In nine cases the accuracy of the rule set is equal to or slightly less than that of the better of the two decision trees. In all cases, the accuracy is greater than that of the poorer of the two decision trees being combined.

Training Set	Categorical		Integer		Combined	
	Accuracy	Coverage	Accuracy	Coverage	Accuracy	Coverage
r30a	77.6	88.0	80.6	96.0	81.7	96.0
r30b	100.0	84.0	100.0	84.0	100.0	84.0
r30c	81.2	80.0	84.3	100.0	84.9	96.0
r30d	95.0	80.0	79.7	96.0	84.0	96.0
r30e	95.7	92.0	86.1	100.0	90.8	96.0
r30f	90.3	100.0	90.3	100.0	90.3	100.0
r30g	90.7	92.0	83.8	92.0	87.1	92.0
r30h	65.9	80.0	69.2	100.0	69.4	100.0
r30i	95.0	92.0	95.0	92.0	95.0	92.0
r30j	88.7	88.0	84.9	100.0	89.2	93.5
r30k	85.1	96.0	85.1	96.0	85.1	96.0
r30l	81.8	92.0	88.0	100.0	86.9	100.0
r30m	91.8	84.0	81.9	96.0	86.4	92.0
r30n	93.8	96.0	86.6	96.0	90.6	96.0
r30o	87.4	92.0	87.8	96.0	91.4	92.0
r30p	94.6	92.0	86.1	100.0	88.3	100.0
r30q	84.2	80.0	93.0	80.0	89.4	80.0
r30r	86.8	88.0	94.0	92.0	93.5	92.0
r30s	91.0	100.0	91.0	100.0	91.0	100.0
r30t	76.2	92.0	92.0	100.0	87.8	93.7

TABLE 4.6: *Combining decision trees using MIL—training sets of size 30.*

For training sets of size 30 (Table 4.6), coverage of the combined rule set is mostly equal to or slightly less than the greater of the coverage of the two decision trees. In only one case (r30o) is the coverage equal to that of the lesser coverage of the two decision trees, but the accuracy is greater than either decision tree. There are five instances of a combined rule set having greater accuracy than either of the individual decision trees (r30a, r30c, r30h, r30j, and r30o). Two of these (r30a and r30h) have coverage equal to the greater coverage of the decision trees being combined. Overall, both the coverage and the accuracy of the combined rule set is closer to that of the better coverage and

Training Set	Categorical		Integer		Combined	
	Accuracy	Coverage	Accuracy	Coverage	Accuracy	Coverage
r40a	100.0	88.0	92.9	96.0	94.5	96.0
r40b	95.0	92.0	95.0	92.0	95.0	92.0
r40c	93.0	92.0	97.1	96.0	97.9	96.0
r40d	96.2	96.0	96.2	96.0	96.2	96.0
r40e	94.8	88.0	94.8	88.0	94.8	88.0
r40f	93.8	92.0	93.8	92.0	93.8	92.0
r40g	99.0	92.0	99.0	92.0	99.0	92.0
r40h	92.1	88.0	82.5	96.0	85.2	96.0
r40i	91.8	92.0	83.4	96.0	87.2	92.0
r40j	97.9	88.0	97.9	88.0	97.9	88.0
r40k	91.8	92.0	91.8	92.0	91.8	92.0
r40l	98.0	92.0	98.0	92.0	98.0	92.0
r40m	98.0	96.0	90.8	96.0	98.6	96.0
r40n	99.0	92.0	91.2	92.0	98.5	92.0
r40o	94.1	92.0	87.3	100.0	91.1	96.0
r40p	96.8	88.0	96.8	88.0	96.8	88.0
r40q	90.5	96.0	92.7	100.0	94.5	97.1
r40r	88.9	96.0	97.1	96.0	96.2	96.0
r40s	100.0	84.0	100.0	84.0	100.0	84.0
r40t	96.0	92.0	88.1	100.0	91.9	95.8

TABLE 4.7: *Combining decision trees using MIL—training sets of size 40.*

accuracy of the two decision trees being combined. In a number of instances (r30b, r30f, r30i, and r30k) the same decision tree is induced, irrespective of the attribute ordering, indicating the absence of any choice for any terminating training set. In such cases MIL will have no affect.

For training sets of size 40 (Table 4.7), the incidence of the induction of the same decision tree, irrespective of attribute ordering, is higher (ten cases) and thus MIL is effective in only half of the experiments carried out here. For these, three cases demonstrate greater accuracy in the combined rule set than in the decision trees being combined, with only a small, if any, decrease in

Training Set Size	Accuracy						Coverage					
	Categorical			Integer			Categorical			Integer		
	<	=	>	<	=	>	<	=	>	<	=	>
20	14	2	4	3	2	15	0	3	17	8	12	0
30	6	5	9	4	5	11	0	9	11	8	12	0
40	6	10	4	1	10	9	0	14	6	4	16	0

TABLE 4.8: *Summary of the relationships between the performance of the combined rule sets and that of the decision trees being combined. For example, 15 of the 20 combined rule sets in the experiments using training sets of size 20 have an accuracy greater than that of the "Integer" decision tree. And 17 of these combined rule set had coverage greater than that of the "Categorical" decision tree. Recall that the "=" category for training sets of size 40 includes 10 entries for which both decision trees, and consequently the combined rule, are identical.*

coverage (r40c, r40m, and r40q). The remaining combined rule sets once again have accuracy and coverage bounded by the accuracy and coverage of the two decision trees, with these measures generally being closer to the greater of the two. These observations conform with those for training sets of sizes 20 and 30.

These results provide support for the use of MIL as an effective approach to combining decision trees. In each case the coverage and accuracy of the combined rule set was found to be bound below by the respective minimums from the decision trees being combined. In most cases the coverage and accuracy was found to be closer to, or equal to, that of the maximum of the decision trees being combined rather than to the minimum. In a few cases, coverage was maintained whilst actually increasing the accuracy to produce a combined rule set of greater accuracy than either of the decision trees being combined.

Table 4.8 provides a summary of the relationships between the performance of the combined rule set and that of the decision trees being combined. Various

trends can be discerned, including the trend that the combined rule set is at least as accurate, but often more accurate than the "Integer" decision tree. Also, the coverage of the combined rule set is mostly equal to that of the "Integer" decision tree, and mostly greater than that of the "Categorical" decision tree.

4.9 SUMMARY

This chapter has presented in its entirety the MIL algorithm. This algorithm addresses the situation of multiple decision trees induced from a single training set. Combining decision trees leads to the potential for conflicting decisions to be made. A study of these conflicts gave rise to a number of observations which were presented as a collection of properties of conflict. An example was presented, illustrating the actual process of identifying and reconciling conflicts. The full specification of this process was then provided. The algorithm was then applied to a large sample of decision trees, and the results confirm the effectiveness of MIL in combining decision trees.

A discussion of issues raised by the approach to combining decision trees presented in this chapter follows in Chapter 5. Particular attention is paid to alternatives in implementing the algorithm.

5 Alternatives in Implementing MIL

The MIL algorithm was developed in Chapter 4 as an approach to combining induced decision trees. The motivation for combining decision trees comes from the observation of Chapter 3 that decision tree induction algorithms can induce multiple decision trees from a single training set. If a categorical attribute is always chosen over an integer attribute, whenever the algorithm identifies multiple choices, the resulting decision tree tends to have less coverage, but often greater accuracy, than when integer attributes are always chosen.

The example of combining the decision trees T106DC and T106DI presented in Chapter 4 both motivated and illustrated the MIL algorithm. The example also demonstrated a best and worst performance. A series of experiments in Chapter 4 confirmed the effectiveness of MIL's approach to combining decision trees.

This chapter provides a discussion of further features and issues related to MIL. These include consideration of alternatives to the implementation of MIL and various approaches in using MIL. Issues relating to the "suggested rules" are covered. Sequential and parallel models for conflict resolution in MIL are considered and shown to produce the same combined rule set. An approach implementing MIL directly in the decision tree induction algorithm is described. And the applicability of MIL to more than two rule sets is considered.

5.1 RESTORING COVERAGE

Conflict between pairs of rules in a combined rule set is removed by the MIL algorithm by adding conditions to each rule, thereby reducing coverage. MIL regains this coverage by building "suggested" rules, which apply only to objects in the common scope of the pair of rules in conflict. MIL searches for a further attribute to use to resolve the conflict. The effectiveness of this approach has been demonstrated in the examples of Chapter 4. Limitations and alternative approaches are considered here.

In the context of the type of conflicts described in Chapter 4, conflicting rules arise when alternative attributes, each capable of partitioning a terminating training set equally well, are chosen. A pre-specified ordering of the attributes is typically used by a decision tree induction algorithm to choose just one attribute.

The experiments of Chapter 3 demonstrated that using an ordering which favours integer attributes over categorical attributes generally results in decision trees with greater coverage but reduced accuracy, when compared to decision trees induced using an attribute ordering which favours categorical attributes. These "integer" decision trees and "categorical" decision trees represent a trade-off between coverage and accuracy. Although other equally good decision trees might be induced from the training set, these two trees will be considered as "representative" decision trees.

The MIL algorithm is viewed as a tool facilitating the task of combining "representative" decision trees into a single rule set. Further domain knowledge may be required if MIL is to maintain the coverage otherwise lost when removing conflict. In particular, domain knowledge is useful when MIL needs to choose a further attribute to use to resolve the conflict.

The current implementation of MIL searches for a resolution of any conflict by considering the terminating training set from which a pair of rules in conflict was derived. Already, two (equally good) attributes have been used,

each distinguishing between those objects having one decision or another. If other attributes also distinguish between these objects, then MIL proposes the use of one of these attribute for resolving the conflict (and only the conflict). Domain knowledge can usefully be employed to assist in the choice of this attribute. Such domain knowledge might be a partial ordering of the attributes, based on the difficulty of determining a value for each attribute. In choosing a third attribute to resolve the conflict, a less attractive attribute will be used. The resulting "suggested rules" can be presented to the knowledge engineer for approval, disapproval, or as a starting point for further work on the knowledge base.

A difficulty with this approach arises when multiple (more than two) such rule sets are to be combined. Under the scenario described in this thesis, a third rule set will be induced using an alternative ordering of the attributes, resulting in different rules only where there are at least three choices for partitioning a terminating training set. This third attribute is chosen by MIL for use in the suggested rules when resolving the conflicts between the first two decision trees. Thus, further care must be taken when incorporating a third, fourth, etc. rule set into the combined rule set. Such issues have not been fully considered in the work described here, and are identified as important areas for further work.

Several alternative approaches to generating "suggested" rules are possible. Three are introduced here, and illustrate various directions that could be taken in future research.

An alternative is to employ all equally good attributes, rather than just one. Thus, from a terminating training set for which a number of attributes define equally good partitions, a composite rule could be introduced. Sub-components of this rule will test a single attribute's value, and make a decision. All sub-components could be considered, and the decision with the most support taken. A similar approach was introduced in Chapter 3 where for any conflict the decision with the most support in the terminating training set is taken. Alternatively, an ordering of the sub-components (based on attribute

ordering) could be used, leading to a scenario similar to that used in MIL. All of these approaches make more use of the knowledge that is contained in the corresponding terminating training set.

Conflict has been defined in terms of rules having consistent conditions concluding inconsistent decisions. The above approaches attempt to resolve this conflict using the information contained in the corresponding terminating training set. A suggested alternative approach to deciding between two decisions uses a larger sample of objects having either of these decisions. This larger sample can be the subset of the full training set containing just those objects having either of the decisions in question. Intuitively, such a larger sample would contain more information about the types of objects associated with each decision value, without having the complication of other decision values. Used as a new training set itself, a new decision tree can be induced. This decision tree can then be employed when it is known that conflict would otherwise result.

A final alternative borrows ideas from clustering to assist in the resolution of conflict. Clustering techniques rely upon distance measures to determine class membership. For objects that give rise to conflict, clustering could be introduced to associate the objects with the appropriate decision. This would entail the introduction of distance measures for attributes, a concept employed in research elsewhere (De Ferrari, 1990).

5.2 SEQUENTIAL VERSUS PARALLEL

Another issue involves the commutativity of the conflict resolver, and the consequence that MILcr can be employed either sequentially (incrementally) or in parallel. The properties introduced below summarise observations made in the development of the algorithm in Chapter 4.

As presented, MIL is essentially a parallel algorithm. Once all pairs of rules in conflict have been identified, the conflict resolver can be applied to each pair independently and in parallel. MIL then assimilates the results into the combined rule set (removing redundancies and over-generalisations). Whilst this parallel approach has advantages in the context of parallel computing, MIL was originally developed as a sequential process not requiring any post-processing of the rules. A rule common to a number of conflicts will be modified by successive invocations of the conflict resolver. This serial approach is demonstrated here to produce the same collection of rules as the parallel approach. The key to this is the commutativity of MILcr.

Recall the terminology and notation of Chapter 4. The examples which make up the training set and which are presented to the performance element come from the set of objects $\mathcal{O}$. $\mathbf{R}$ is a rule set which results from the process of combining the two rule sets $\mathbf{R}_1$ and $\mathbf{R}_2$, each of which is derived directly from decision trees induced from the same training set. Suppose the rules R_1 and R_2, from the rule sets $\mathbf{R}_1$ and $\mathbf{R}_2$ respectively, give rise to conflict (i.e., they have consistent conditions and inconsistent decisions). The replacement rules Cmb_1 and Cmb_2 are generated by MILcr as specialisations of R_1 and R_2, having the conflict removed. The suggested rules Cmb_3 and Cmb_4 are generated by MILcr for objects which trigger both rules R_1 and R_2. These objects are said to belong to the common scope $\mathbf{Cs}$ of the two rules.

The following discussion will concentrate on the case of a set of complementary conflicts containing just four rules.

Property 5.1: When MILcr replaces R_1 and R_2 in **R** with Cmb_1 and Cmb_2, and introduces Cmb_3 and Cmb_4, only those objects in **Cs** are affected.

Decisions made by R for those objects not in Cs are unchanged by the replacement of the two rules in conflict with the replacement and suggested rules.

This can be verified by considering the structure of the replacement and suggested rules as in Chapter 4. In summary the replacement rules Cmb_1 and Cmb_2 trigger on exactly the same objects as R_1 and R_2 respectively, except for those objects in the common scope of R_1 and R_2: neither Cmb_1 nor Cmb_2 trigger on these objects. Furthermore, Cmb_1 and Cmb_2 make the same decisions as R_1 and R_2 respectively. Thus, replacing R_1 and R_2 in **R** with Cmb_1 and Cmb_2 reduces the coverage of **R** by exactly those objects in **Cs**. Next, the conditions of Cmb_3 and Cmb_4, the suggested rules, are met only by those objects in the common scope of R_1 and R_2, and thus have no affect upon the objects in $\mathcal{O}$ but not in **Cs**.

Consequently, the effects of the application of the MILcr algorithm are localised to the objects which are given conflicting decisions.

5.2.1 Commutativity of MILcr

Assume that MILcr has been applied to a conflict Q, involving the rules R_1 and R_2, replacing R_1 and R_2 with Cmb_1 and Cmb_2, and adding the suggested rules Cmb_3 and Cmb_4. Call this modified combined rule set $\mathbf{R}'$. $\mathbf{R}'$ differs from **R** only in that R_1 and R_2 have been replaced by Cmb_1, Cmb_2, Cmb_3, and Cmb_4. Under a sequential implementation, following the application of MILcr to Q, the form of the rules involved in some other different conflict Q', involving the rules R_1' and R_2' say, may have changed. There are only two possibilities:

Case 1: $R'_1 \neq R_1$ and $R'_2 \neq R_2$.

The rules in conflict in Q' are unchanged by the application of MILcr to Q. The set of objects from $\mathcal{O}$ for which $Cond_1 \wedge Cond_2$ is true is disjoint from the set of objects for which $Cond'_1 \wedge Cond'_2$ is true (Property 4.2). The replacement and suggested rules resulting from the application of MILcr to Q affect only the former set of objects and no other objects in $\mathcal{O}$ (Property 5.1). With $R'_1 \neq R_1$ and $R'_2 \neq R_2$, the application of MILcr to Q' is unaffected by the modification made by the application of MILcr to Q.

Case 2: Either $R'_1 = R_1$ and $R'_2 \neq R_2$, or $R'_1 \neq R_1$ and $R'_2 = R_2$.

This is the case where a rule from one rule set conflicts with two rules from the other rule set. The alternatives are symmetric, and so only the former is considered: $R'_1 = R_1$ and $R'_2 \neq R_2$. The application of MILcr to Q will replace R_1 by Cmb_1, and Q' will then involve the rules Cmb_1 and R'_2. The replacement rules and the suggested rules are considered separately. Recall that the common scope of the rules in Q and the common scope of the rules in Q' are disjoint (Property 4.2).

On applying MILcr to Q, R_1 and R_2 are replaced by Cmb_1 and Cmb_2 respectively. MILcr applied to Q' replaces Cmb_1 and R'_2 by Cmb'_1 and Cmb'_2. The following three rules thus replace R_1, R_2, and R'_2, respectively.

$$\begin{aligned}
Cmb'_1\text{:} \quad & Cond_1 \wedge \neg Cond_2 \wedge \neg Cond'_2 && \Longrightarrow \quad Class_1,\\
Cmb_2\text{:} \quad & Cond_2 \wedge \neg Cond_1 && \Longrightarrow \quad Class_2,\\
Cmb'_2\text{:} \quad & Cond'_2 \wedge \neg(Cond_1 \wedge \neg Cond_2) && \Longrightarrow \quad Class'_2.
\end{aligned}$$

The third rule is equivalent to the two rules:

$$\begin{aligned}
Cond'_2 \wedge \neg Cond_1 \quad &\Longrightarrow \quad Class'_2,\\
Cond'_2 \wedge Cond_2 \quad &\Longrightarrow \quad Class'_2.
\end{aligned}$$

The second of these rules can be removed, since it can never succeed (Property 4.1) as $Cond'_2$ and $Cond_2$ come from the same decision tree. The final three replacement rules are then:

$$\begin{aligned} Cond_1 \wedge \neg Cond_2 \wedge \neg Cond'_2 &\implies Class_1, \\ Cond_2 \wedge \neg Cond_1 &\implies Class_2, \\ Cond'_2 \wedge \neg Cond_1 &\implies Class'_2. \end{aligned}$$

These rules are symmetric with respect to $Cond_2$ and $Cond'_2$. Hence, this same set of replacement rules will result if conflict Q' were dealt with before Q.

For the suggested rules, the same candidate description must be chosen by MILcr when working on Q' irrespective of whether MILcr has already been applied to Q. Applying MILcr to Q results in the suggested rules:

$$\begin{aligned} Cmb_3\text{:} \quad Cond_1 \wedge Cond_2 \wedge Cons_1 &\implies Class_1, \\ Cmb_4\text{:} \quad Cond_1 \wedge Cond_2 \wedge Cons_2 &\implies Class_2, \end{aligned}$$

where $Cons_1$ and $Cons_2$ are the constructed descriptions. Applying MILcr to Q' where R_1 has been replaced by Cmb_1 results in the suggested rules:

$$\begin{aligned} Cmb'_3\text{:} \quad Cond_1 \wedge \neg Cond_2 \wedge Cond'_2 \wedge Cons'_1 &\implies Class_1, \\ Cmb'_4\text{:} \quad Cond_1 \wedge \neg Cond_2 \wedge Cond'_2 \wedge Cons'_2 &\implies Class'_2. \end{aligned}$$

By Property 4.1, these two rules are equivalent to the simpler rules:

$$\begin{aligned} Cmb'_3\text{:} \quad Cond_1 \wedge Cond'_2 \wedge Cons'_1 &\implies Class_1, \\ Cmb'_4\text{:} \quad Cond_1 \wedge Cond'_2 \wedge Cons'_2 &\implies Class'_2. \end{aligned}$$

Suppose that MILcr were applied to Q' before it were applied to Q. The only possible difference will be the constructed conditions $Cons'_1$ and $Cons'_2$ in the rules Cmb'_3 and Cmb'_4. Since the training subsets corresponding to R_1 and Cmb_1 are the same, the basis upon which MILcr constructs the candidate descriptions is unchanged. Hence, $Cons'_1$ and $Cons'_2$ will be used in either case, resulting in the same rules being generated.

Thus, the order in which MILcr is applied to a pair of conflicts, assuming sequential application, is not important, with the same rules being generated irrespectively.

5.2.2 Sequential versus Parallel

We can now show that MILcr will generate the same final set of rules irrespective of whether it is applied sequentially or in parallel coupled with rule reconciliation. The rules produced by way of a sequential application, and by way of a

parallel application, are considered in the case of two conflicts, involving three rules (i.e., having one rule in common). The case of two conflicts having no rules in common is not considered, since with either approach the conflicts are handled independently and thus identically.

Consider the two conflicts Q, involving the rules R_1 and R_2, and Q', involving the rules R_1 and R_2'. These rules are of the form:

$$\begin{aligned} R_1\text{:} \quad & Cond_1 \implies Class_1, \\ R_2\text{:} \quad & Cond_2 \implies Class_2, \\ R_2'\text{:} \quad & Cond_2' \implies Class_2'. \end{aligned}$$

Under the sequential application of MILcr to these two conflicts, the resulting replacement rules are:

$$\begin{aligned} Cond_1 \wedge \neg Cond_2 \wedge \neg Cond_2' &\implies Class_1, \\ Cond_2 \wedge \neg Cond_1 &\implies Class_2, \\ Cond_2' \wedge \neg Cond_1 &\implies Class_2'. \end{aligned}$$

Under the parallel application of MILcr to these same two conflicts, the resulting replacement rules are:

$$\begin{aligned} Cond_1 \wedge \neg Cond_2 &\implies Class_1, \\ Cond_2 \wedge \neg Cond_1 &\implies Class_2, \\ Cond_1 \wedge \neg Cond_2' &\implies Class_1, \\ Cond_2' \wedge \neg Cond_1 &\implies Class_2'. \end{aligned}$$

which, after reconciliation (as described in Chapter 4), become the same three rules as in the sequential case.

Likewise, under a sequential application of MILcr to the conflicts, the suggested rules become:

$$
\begin{aligned}
Cond_1 \wedge Cond_2 \wedge Cons_1 &\implies Class_1, \\
Cond_1 \wedge Cond_2 \wedge Cons_2 &\implies Class_2, \\
Cond_1 \wedge \neg Cond_2 \wedge Cond_2' \wedge Cons_1' &\implies Class_1, \\
Cond_1 \wedge \neg Cond_2 \wedge Cond_2' \wedge Cons_2' &\implies Class_2'.
\end{aligned}
$$

The last two rules can be simplified by removal of the redundant $\neg Cond_2$, due to the presence of $Cond_2'$ (Property 4.1), resulting in exactly the same set of suggested rules generated under a parallel application of MILcr.

The MIL algorithm thus generates the same combined rule set using either a parallel or sequential approach.

5.3 IMPLEMENTING MIL DIRECTLY

MIL represents a general paradigm for handling multiple decision trees, and is independent of the decision tree induction algorithm employed. The ideas embodied in MIL could be directly implemented by modifying the decision tree induction algorithm to handle the case where the selection criterion is unable to identify a single attribute.

Such an approach has been developed by augmenting the decision tree structure to allow alternative sub-trees to be associated with a node. A performance element is then required to consider all alternative sub-trees. One (conservative) approach is to return a decision only if all sub-trees agree. A second approach is to extend the decision tree representation by allowing richer logical expressions. These new logical expressions will replace the single attribute-value test of the node. This approach is considered here, and its relationship to the MIL algorithm is shown.

In decision trees, the tests corresponding to individual branches can be enhanced. Consider the simple case of two alternative sub-trees, one with attribute A, the other with attribute B (both being binary-valued attributes, with values A_1, A_2, and B_1, B_2, respectively), as in Figure 5.1. These two sub-trees are merged into one, with the corresponding tests as shown in the figure. Just as with the replacement rules generated by MILcr, the expression associated with the decision D_1 (or D_2), cannot simply be $A_1 \vee B_1$ (or $A_2 \vee B_2$) as this can lead to indeterminacy and conflict. This occurs in the case, for example, where $A = A_1$ and $B = B_2$.

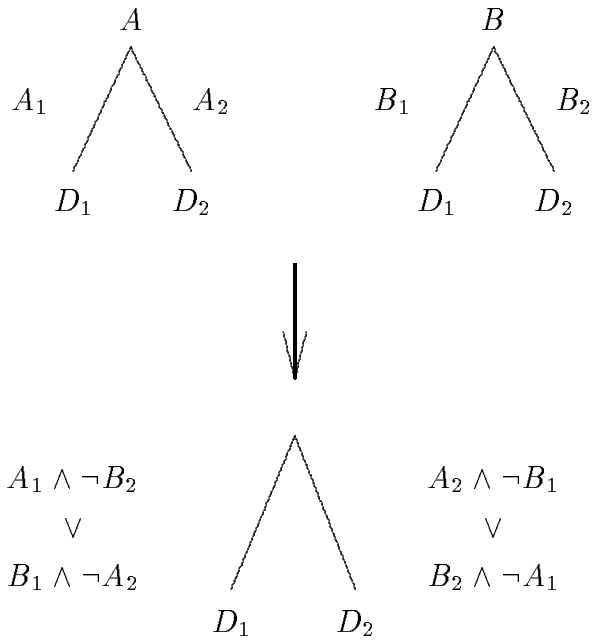


FIGURE 5.1: *A simple illustration of the implications of implementing* MIL *by modifying the decision tree algorithm, and consequently, the structure of the decision tree tests. The expression* $A_1 \wedge \neg B_2$*, for example, is to be read as "attribute A has value* A_1 *and attribute B does not have the value* B_2*". The logical symbols* $\wedge$*,* $\vee$*,* $\neg$ *represent conjunction, disjunction, and negation, respectively.*

This simple scenario does not handle conflicts, considering only the replacement rules of the combined rule set. Such a merged decision tree is equivalent to the combined rule set containing the replacement rules, but not the suggested rules. To verify this, consider the "combined" decision tree of Figure 5.1. Considering only this part of the decision tree, the corresponding 4 rules are:

$$
\begin{aligned}
R_1 : \quad & A = A_1 \quad \Longrightarrow \quad D_1, \\
R_2 : \quad & A = A_2 \quad \Longrightarrow \quad D_2, \\
R_1' : \quad & B = B_1 \quad \Longrightarrow \quad D_1, \\
R_2' : \quad & B = B_2 \quad \Longrightarrow \quad D_2.
\end{aligned}
$$

MIL will resolve the two conflicts here by replacing these four rules with:

$$\begin{aligned} Cmb_1 : &\quad A = A_1 \wedge B \neq B_2 &\Longrightarrow\ & D_1, \\ Cmb_2 : &\quad A = A_2 \wedge B \neq B_1 &\Longrightarrow\ & D_2, \\ Cmb'_1 : &\quad B = B_1 \wedge A \neq A_2 &\Longrightarrow\ & D_1, \\ Cmb'_2 : &\quad B = B_2 \wedge A \neq A_1 &\Longrightarrow\ & D_2, \end{aligned}$$

which is precisely the same set of rules as represented in the combined decision tree.

The suggested rules, which are generated by MIL to restore coverage, can also be implemented directly by way of decision tree manipulation. The same process as carried out by MILcr can be employed, resulting in the addition of further branches emanating from this new node, with appropriate expressions attached, leading to leaf nodes labelled with the appropriate decisions.

Thus, the MIL algorithm can be expressed (and implemented) in terms of either combining rule sets, or combining decision trees. In terms of decision trees, the representation must be enhanced, allowing richer expressions to be associated with branches of the tree. The consequences of this approach are not further developed here, and remain an interesting area for further work.

5.4 MULTIPLE RULE SETS

Another important issue is how MIL can be employed to combine three or more rule sets, each derived directly from decision trees induced from the same training set.

The process of combining multiple decision trees (rule sets) can be described using a simple case. Figure 5.2 illustrates three decision trees (sub-trees) which are, as far as the selection criterion is concerned, alternatives. The process of merging these to produce a single sub-tree simply generalises the approach demonstrated in Figure 5.1. Ignoring "suggested rules", this process of combining decision trees simply restricts the coverage to those objects for which no conflict arises. The objects that give rise to conflicts are problematical, and could be regarded as outside the scope of the decision trees, as discussed earlier. To introduce "suggested rules", new branches may be added to the node to cover other possibilities, with such branches either generated by MIL or by the knowledge engineer or domain expert. In general, the set of expressions associated with the branches emanating from a node will be kept mutually exclusive, so that branching is minimised (using disjunction where appropriate), and the possibility for conflict is removed.

In terms of rules, each of the original sub-trees in Figure 5.2 corresponds to two rules, leading to six rules in the combined rule set. The six rules lead to six conflicts, which MIL will resolve appropriately, producing six replacement rules, equivalent to the six disjuncts illustrated in the figure. The MIL algorithm will combine two of the rule sets, and then combine this combined rule set with the third rule set. Obvious efficiencies are gained by extending MIL to work on more than two rule sets at a time, handling the larger set of complementary conflicts at once. Similarly, direct modifications to the decision tree induction algorithm may also improve efficiency.

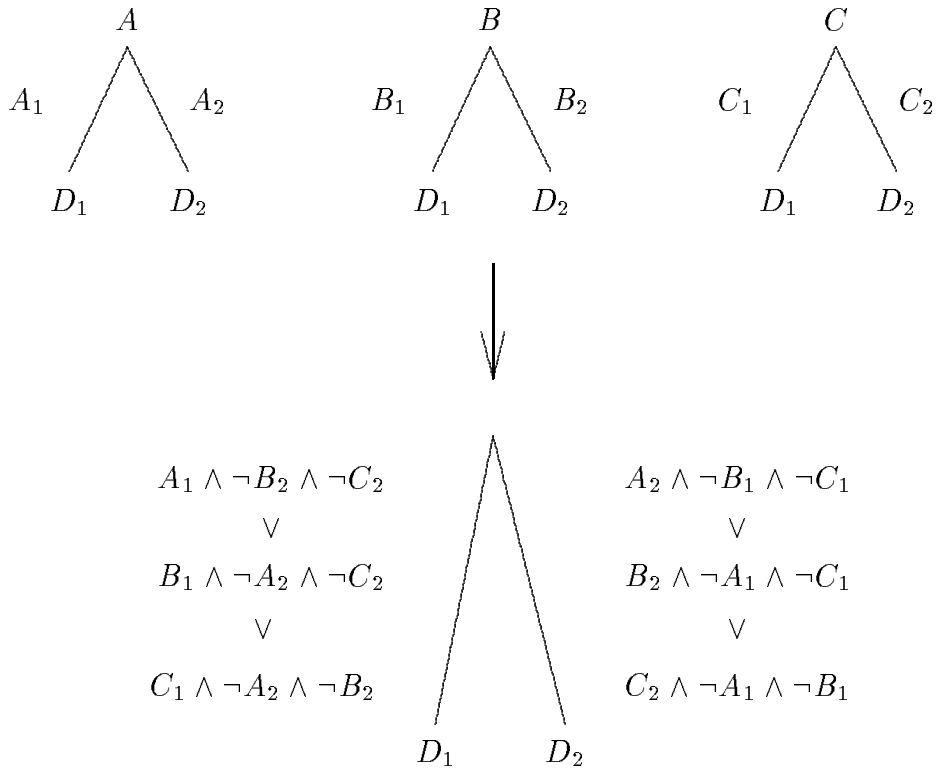


FIGURE 5.2: *A simple illustration of the process of combining three decision trees. This can be implemented either directly upon the decision trees, or within the rule set paradigm.*

The MIL paradigm, then, is not restricted to combining just two rule sets, but the issue of the suggested rules, as discussed earlier, is an important area that must be addressed by further research.

5.5 ALTERNATIVE REPLACEMENT RULES

In generating replacement rules MILcr takes the logical negation of the conditions of one rule, and conjoins this to the conditions of another rule. In the simplest case, as considered here, all but one of the conjuncts in this negated conjunction appear un-negated in the condition to which it is being conjoined. Thus, all but one of these conjuncts in the negated condition are logically unnecessary, leaving just a single term to be negated and conjoined. The usual semantics of logical negation is meant, where the negated expression is true whenever the expression being negated is false.

An earlier version of MIL, as reported in Williams (1988) was somewhat more cautious in its approach to negating the conditions of another rule. The approach taken in this earlier work was to express the negation in terms of only those objects in the training set. Thus, a condition of the form $A = A_i$, when negated under this scheme, becomes $A = A_j$ if the only values of the attribute A found in the associated training set were A_i and A_j. Such an approach does indeed lead to conflict-free replacement rules, but their generality is significantly reduced. With the goal of producing rule sets with acceptable coverage and accuracy, the more general concept of negation was found appropriate.

Other options are available in producing replacement rules. The process of modifying the conditions of just one rule of a pair of rules in conflict, conflict pair was introduced in Chapter 4. For example, given the rules:

$$\begin{aligned} R_1: &\quad A = A_1 &&\Longrightarrow\ D_1 \\ R_2: &\quad B = B_1 &&\Longrightarrow\ D_2 \end{aligned}$$

suitable replacement rules might be:

$$\begin{aligned} Cmb_1: &\quad A = A_1 &&\Longrightarrow\ D_1 \\ Cmb_2: &\quad B = B_1 \wedge A \neq A_1 &&\Longrightarrow\ D_2 \end{aligned}$$

Under such a simple scheme, all conflict is removed, and generality is maintained. This approach can be compared to the informal approach of resolving conflict by always choosing one decision in favour of another. This approach was not chosen for MIL as it was argued to be more appropriate to specifically

identify examples for which the rule sets can not give a consistent decision, and then to handle these separately.

Further alternative means of generating suggested rules could be considered, and is again an area identified for further research.

5.6 SUMMARY

This chapter has discussed a number of alternatives that have been considered in the implementation of the MIL algorithm. Areas for further research have been identified and preliminary discussion of these provide initial insights.

Alternatives to the generation of the "suggested rules" have been considered, with a number of interesting new directions being proposed. Different approaches to the implementation of the MIL algorithm were discussed, covering the issues of sequential and parallel execution, and the direct implementation of MIL within a decision tree induction algorithm. Alternatives to the "introduced rules" were discussed briefly, and the issue of combining many rule sets was considered.

6 Summary and Conclusions

This thesis has presented a research effort investigating and developing upon the problem of inducing decision structures for use in expert systems. The first stage involved a series of experiments using a decision tree induction algorithm (Chapter 3). The second stage (Chapter 4) developed the MIL algorithm which builds upon a decision tree induction algorithm by combining decision trees into a unified collection of rules with all potential for conflict removed. This work is summarised below and the conclusions are drawn together.

Decision tree induction algorithms have been a demonstrably successful knowledge-acquisition tool for building knowledge-based expert systems (Chapter 2). Whilst such algorithms have been used to build knowledge bases for immediate deployment, intervention by experienced knowledge engineers is still, in general, necessary. An understanding of the nature of the algorithms, and of the types of decision structures they produce is essential for this technology to be appropriately employed.

The series of experiments described in Chapter 3 explored a number of aspects of the decision tree induction algorithm. Whilst the algorithm was found to produce good decision trees, a significant inadequacy was identified whereby apparently equally good decision trees could be induced from a single training set. These decision trees differed in terms of their coverage and accuracy, sometimes quite significantly. The concept of tree pruning was then considered, and found to be a useful technique for improving the performance of a decision tree. An approach to pruning which retained the agreement of the resulting decision tree with the decisions contained in the training set was introduced, and found to produce improvements.

The MIL algorithm was developed here upon the observations that a decision tree induction algorithm can generate multiple decision trees and that

alternative induction algorithms can provide alternative decision trees. The MIL algorithm combines decision trees, providing a tool for use by a knowledge engineer, providing rapid prototyping and guidance in the development of a knowledge-based expert system.

The MIL algorithm identifies rules common to the decision trees being combined as constituting an initial rule set. The remaining rules, which potentially give rise to conflict, are specialised so that any potential for conflict is removed. New rules are suggested to cover any object which previously would have given rise to conflict. The experiments indicated that MIL was often able to produce a combined rule set with maximal, or near maximal, coverage whilst maintaining accuracy.

Overall, my research, as reported upon in this thesis, has investigated issues relating to the usage of decision tree induction algorithms. I provide insights into the performance of these algorithms with real data, drawn from several domains. I have developed a technique for combining multiple decision trees and have identified the potential advantages of this technique over the single application of a decision tree induction algorithm to a training set.

A References

Abrett, G. and Burstein, M. H. (1988). The KREME knowledge editing environment. In J. H. Boose and B. R. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*. New York, NY: Academic Press.

Aikins, J. S. (1983). Prototypical knowledge for expert systems. *Artificial Intelligence*, *20* (2), 163–210.

Arbab, B. (1985). Building expert systems by generating rules from examples. (G320-2763.) Los Angeles Scientific Center: IBM.

Arbab, B. and Michie, D. (1985). Generating rules from examples. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence, IJCAI-85* (pp. 631–633). Los Angeles, CA: Morgan Kaufmann.

Bramer, M. A. (1982). A survey and critical review of expert systems. In D. Michie (Ed.), *Introductory Readings in Expert Systems*. London: Gordon and Breach Science Publishers.

Bratko, I. (1983). Generating Human-Understandable Decision Rules. (Working paper.) Liubljana, Yugoslavia: E. Keardelj University.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Buchanan, B. G. (1982). New research on expert systems. In J. E. Hayes, D. Michie, and Y. H. Pao (Eds.), *Machine Intelligence 10*. Chichester, England: Ellis Horwood.

Buchanan, B. G., Barstow, D., Bechtal, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D. A. (1983). Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (Eds.), *Building Expert Systems*. Reading, MA: Addison Wesley.

Cheng, J., Fayyad, U. M., Irani, K. B., and Qian, Z. (1988). Improved decision trees: A generalised version of ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 100–106). Ann Arbor, MI: Morgan Kaufmann.

Clancey, W. J. (1983). The epistemology of a rule-based expert system — a framework for explanation. *Artificial Intelligence*, *20*, 215–251.

Clancey, W. J. (1985). Book review: A practical guide to designing expert systems. *AI Magazine*, *5* (4), 84–86.

Cocks, K. D., Young, M. D., and Walker, P. A. (1986). Mapping relative viability prospects for pastoralism in Australia. *Agricultural Systems*, *20*, 175–193.

Crawford, S. L. (1989). Extensions to the CART algorithm. *International Journal of Man Machine Studies*, *31*, 197–217.

Davis, R. and King, J. J. (1984). The origin of rule-based systems in AI. In B. G. Buchanan and E. H. Shortliffe (Eds.), *Rule-Based Expert Systems*. Reading, MA: Addison Wesley.

De Ferrari, L. G. (1990). *Combining conceptual clustering and explanation-based learning*. Honours Thesis, Department of Computer Science, Australian National University.

Delbridge, A., (Ed.) (1982). *The Macquarie Dictionary*. Doubleday Australia.

Duda, R., Gaschnig, J., and Hart, P. (1979). Model design in the PROSPECTOR consultant system for mineral exploration. In D. Michie (Ed.), *Expert Systems in the Micro-Electronic Age*. Edinburgh: Edinburgh University Press.

Forsyth, R. and Rada, R. (1986). *Machine Learning: Applications in Expert Systems and Information Retrieval*. Artificial Intelligence, Ellis Horwood.

Gaines, B. R. and Boose, J. H. (1988a). Knowledge acquisition tools for expert systems. In J. H. Boose and B. R. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*. New York, NY: Academic Press.

Gaines, B. R. and Boose, J. H. (1988b). Knowledge acquisition for knowledge-based systems. In B. R. Gaines and J. H. Boose (Eds.), *Knowledge Acquisition for Knowledge-Based Systems*. New York, NY: Academic Press.

Goodman, R. M. and Smyth, P. (Mar. 1990). Decision tree design using information theory. *Knowledge Acquisition*, *2* (1), 1–19.

Hart, P. E. (1980). What's preventing the widespread use of expert systems. (Position paper.) Expert Systems Workshop, San Deiga, California.

Hertz, D. B. (1990). The knowledge engineering basis of expert systems. *Expert Systems With Applications*, *1* (1), 79–84.

Holte, R. C., Acker, L. E., and Porter, B. W. (1989). Concept learning and the problem of small disjuncts. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89* (pp. 813–818). Detroit, MI: Morgan Kaufmann.

Hunt, E. B., Marin, J., and Stone, P. J. (1966). *Experiments in Induction*. New York, NY: Academic Press.

Kitto, C. M. and Boose, J. H. (1988). Heuristics for expertise transfer: an implementation of a dialog manager for knowledge acquisition. In J. H. Boose and B. R. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*. New York, NY: Academic Press.

Klein, M. and Methlie, L. B. (1990). *Expert Systems: A Decision Support Approach*. Reading, MA: Addison Wesley.

Kononenko, I., Bratko, I., and Roškar, E. (1984). Experiments in automatic learning of medical diagnostic rules. (Technical report.) Ljubljana, Yugoslavia: Jozef Stefan Institute.

Langley, P. (1989). Unifying themes in empirical and explanation-based learning. *Proceedings of the Sixth International Workshop on Machine Learning, ML89* (pp. 2–4). Cornell University, Ithaca, NY.

Lenat, D. B. and Guha, R. V. (1989). *Building large knowledge-based systems*. Reading, MA: Addison Wesley.

Lenat, D. B., Hayes-Roth, F., and Klahr, P. (1983). Cognitive economy in a fluid task environment. *Proceedings of the 1983 International Machine Learning Workshop* (pp. 133–146). University of Illinois at Urbana-Champagne.

Li, T., Barter, C. J., and Yu, C. (1988). MULTIPLE: a rule-based design tool with output conflict resolution. In J. S. Gero and R. B. Stanton (Eds.), *Artificial Intelligence Developments and Applications*. Amsterdam: North Holland.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980). *Applications of artificial intelligence for organic chemistry*. McGraw-Hill.

Mackenzie, H. G. (1984). Expert Systems—A Case Study. (TR-11.) Division of Computing Research, CSIRO, Canberra, Australia.

Matheus, C. J. and Rendell, L. A. (1989). Constructive induction on decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89* (pp. 645–650). Detroit, MI: Morgan Kaufmann.

Messier Jr., W. F. and Hansen, J. V. (1988). Inducing rules for expert system development: An example using default and bankruptcy data. *Management Science*, *34* (12), 1403–1415.

Michalski, R. S. and Chilausky, R. L. (1980). Learning by being told and learning from examples: An experimental comparison of two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, *4* (2), 125–160.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (Eds.) (1983). *Machine Learning: An Artificial Intelligence Approach.* Palo Alto, CA: Tioga Publishing Company.

Michalski, R. S. and Stepp, R. E. (1983). Learning From Observation: Conceptual Clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Tioga Publishing Company.

Michie, D. (1987). Current developments in expert systems. In J. R. Quinlan (Ed.), *Applications of expert systems.* Maidenhead: Addison Wesley.

Michie, D., Muggleton, S., Riese, C., and Zubrick, S. (1984). RuleMaster—A second generation knowledge engineering facility. *Proceedings of the First Conference on Artificial Intelligence Applications.* Denver, CO.

Michie, D. (1983). Inductive rule generation in the context of the fifth generation. *Proceedings of the 1983 International Machine Learning Workshop* (pp. 65–70). University of Illinois at Urbana-Champagne.

Mingers, J. (1989a). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, *4* (2), 227–243.

Mingers, J. (1989b). An empirical comparison of selection measures for decision tree induction. *Machine Learning*, *3* (4), 319–342.

Moret, B. M. E. (1982). Decision trees and diagrams. *Computing Surveys*, *14* (4), 593–623.

Morley, D., Harper, D., Nethercote, S., and Williams, G. J. (1988). *Knowledge engineering and TODAY-ES*. Melbourne, Australia: BBJ Computers International.

Mostow, J. (1985). IEEE workshop on principles of knowledge-based systems: A personal review. *SIGART Newsletter*, *92*, 15–27.

Musen, M. A., Fagan, L. M., Combs, D. M., and Shortliffe, E. H. (1988). Use of a domain model to drive an interactive knowledge-editing tool. In J. H. Boose and B. R. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*. New York, NY: Academic Press.

Newell, A. and Simon, H. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Company.

O'Neill, J. L. (1986). Knowledge Acquisition for Radar Classification. *Proceedings of the Conference on Applications of Expert Systems* (pp. 215–229). Sydney, Australia.

Pagallo, G. (1989). Learning DNF by decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89* (pp. 639–644). Detroit, MI: Morgan Kaufmann.

Paterson, A. and Niblett, T. (1982). *ACLS User Manual*. Edinburgh: Intelligent Terminals Limited.

Quinlan, J. R. (1979a). Discovering rules from large collections of examples: A case study. In D. Michie (Ed.), *Expert Systems in the Micro-Electronic Age*. Edinburgh: Edinburgh University Press.

Quinlan, J. R. (1979b). Induction over large data bases. (Memo HPP-79-14.) Heuristic Programming Project, Stanford University.

Quinlan, J. R. (1982). Semi-autonomous acquisition of pattern-based knowledge. In D. Michie (Ed.), *Introductory Readings in Expert Systems*. London: Gordon and Breach Science Publishers.

Quinlan, J. R. (1983a). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto, CA: Tioga Publishing Company.

Quinlan, J. R. (1983b). Learning from noisy data. *Proceedings of the 1983 International Machine Learning Workshop* (pp. 58–64). University of Illinois at Urbana-Champagne.

Quinlan, J. R. (1985). Decision trees and multi-valued attributes. (Technical Report 85.4.) Sydney, Australia: Computer Sciences, New South Wales Institute of Technology.

Quinlan, J. R. (1986a). Induction of decision trees. *Machine Learning*, *1* (1), 81–106.

Quinlan, J. R. (1986b). Simplifying decision trees. *Proceedings of the AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems*. Banff, Canada.

Quinlan, J. R. (1987a). Simplifying decision trees. *International Journal of Man Machine Studies*, *27*, 221–234.

Quinlan, J. R. (1987b). Generating production rules from decision trees. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, IJCAI-87* (pp. 304–307). Milan, Italy: Morgan Kaufmann.

Quinlan, J. R., Compton, P. J., Horn, K. A., and Lazarus, L. (1986). Inductive Knowledge Acquisition: A Case Study. *Proceedings of the Conference on Applications of Expert Systems* (pp. 183–204). Sydney, Australia.

Schlimmer, J. C. and Fisher, D. (1986). A case study of incremental concept induction. *Proceedings of the Fifth National Conference on Artificial Intelligence, AAAI-86* (pp. 496–501). Philadelphia, PA: Morgan Kaufmann.

Shapiro, A. D. (1983). *The role of structured induction in expert systems.* PhD Thesis, University of Edinburgh: Machine Intelligence Research Unit.

Shapiro, A. D. (1987). *Structured induction in expert systems.* Workingham, England: Addison-Wesley.

Shepherd, B. A. (1983). An appraisal of a decision tree approach to image classification. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI-83* (pp. 473–475). Karlsruhe, West Germany: Morgan Kaufman.

Shortliffe, E. (1976). *Computer-Based Medical Consultation: Mycin.* Amsterdam: North Holland.

Suwa, M., Scott, A. C., and Shortliffe, E. H. (1984). Completeness and consistency in a rule-based system. In B. G. Buchanan and E. H. Shortliffe (Eds.), *Rule-Based Expert Systems.* Reading, MA: Addison Wesley.

Tan, M. and Schlimmer, J. C. (1990). Two case studies in cost-sensitive concept acquisition. *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90* (pp. 854–860). Cambridge, MA: MIT Press.

Turban, E. (1990). *Decision Support and Expert Systems*. New York: Macmillan Publishing Company.

Utgoff, P. E. (1988). ID5: An incremental ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 107–120). Ann Arbor, MI: Morgan Kaufmann.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning*, *4* (2), 161–186.

Walker, P. A., Cocks, K. D., and Young, M. D. (1985). Regionalising continental data sets. *Cartography*, *14* (1), 66–73.

Weiss, S., Kulikowski, C., Amarel, S., and Safir, A. (1978). A model-based method for computer-aided medical decision making. *Artificial Intelligence*, *11* (1,2), 145–172.

Weiss, S. M. and Kulikowski, C. A. (1984). *A Practical Guide to Designing Expert Systems*. Totowa, New Jersey: Rowman and Allanheld.

Williams, G. J. (1986). Deficiencies of expert systems and attempts at improvement: A survey. *Proceedings of the Ninth Australian Computer Science Conference* (pp. 63–72). Canberra, Australia: Australian National University.

Williams, G. J. (1987). Some experiments in decision tree indution. *Australian Computer Journal*, *19* (2), 84–91.

Williams, G. J. (1988). Combining decision trees: Initial results from the MIL algorithm. In J. S. Gero and R. B. Stanton (Eds.), *Artificial Intelligence Developments and Applications*. Amsterdam: North Holland.

B Publications

Below is a list of publications by the author which have played either a direct or indirect role in the development of the work described in this thesis.

Davis, J. R., Nanninga, P. M., and Williams, G. J. (1985). Geographic expert systems for resource management, the *First Australian Conference on Applications of Expert Systems*, May, 1985, Sydney, Australia.

Williams, G. J. (1985). GEM users' manual, Technical Memorandum 85/16, Division of Water and Land Resources, Commonwealth Scientific and Industrial Research Organisation, Canberra, Australia.

Williams, G. J. (1986). Deficiencies of expert systems and attempts at improvement: A survey, *Proceedings of the Ninth Australian Computer Science Conference*, February, 1986, Canberra, Australia.

Williams, G. J. (1986). FrameUP: A frames formalism for expert systems, Technical Report number TR-CS-86-04, Department of Computer Science, Australian National University, Canberra, Australia. Accepted for publication in The Australian Computer Journal.

Williams, G. J., Nanninga, P. M., and Davis, J. R. (1986). GEM: A microcomputer based expert system for geographic domains, *Proceedings of the 6th International Workshop and Conference on Expert Systems and Their Applications*, April, 1986, Avignon, France. This paper received the award for best paper at the conference, and was given as an invited paper to the *Australian Computer Conference—ACC87*, 1987, Melbourne. This paper is also available as Technical Report number TR-CS-86-01, Department of Computer Science, Australian National University.

Davis, J. R., Nanninga, P. M., and Williams, G. J. (1986). The design of an expert system for environmental management, in *Readings in Australian Geography*: Proceedings of the 21st IAG Geography Conference, May, 1986, Perth, Australia.

Williams, G. J. (1987). Some experiments in decision tree induction. *Australian Computer Journal, 19(2),* 84-91. This was originally released as Technical Report number TR-CS-87-01, Department of Computer Science, Australian National University. This paper was chosen to be published in the Australian Computer Journal from amongst those presented at the *Tenth Australian Computer Science Conference*, February, 1987, Geelong, Victoria.

Williams, G. J. (1988). Combining decision trees: Initial results from the MIL algorithm. In Gero, J. S. and Stanton, R. B. *Artificial Intelligence Developments and Applications,* North-Holland, 1988. This paper was selected to be published from the *Proceedings of the Australian Joint Artificial Intelligence Conference*, November, 1987, Sydney, Australia.

Williams, G. J. (1989). FrameUP: A frames formalism for expert systems, The *Australian Computer Journal*, 21(1), 33-40.